Oracle XML DB 11gR1 Basic Features

An Oracle White Paper
October 2007

ORACLE

Introduction

The Oracle XML DB basic demonstration high-lights the core features of Oracle XML DB including

- Storing XML content in Oracle XML DB using standard internet protocols
- Using XML Schema to optimizing XML processing
- Querying and updating XML content stored in Oracle XML DB
- Working with XML content using traditional relational SQL
- Publishing XML from relational content and working with XML views of relational content
- Accessing and updating XML content stored in Oracle XML DB with standard desktop software
- XML Schema Evolution
- Accessing the Oracle XML DB repository from SQL and PL/SQL
- Using full-text search with XML content
- Accessing XML and relational data directly from a Browser.
- Transforming XML content with XSL stylesheets

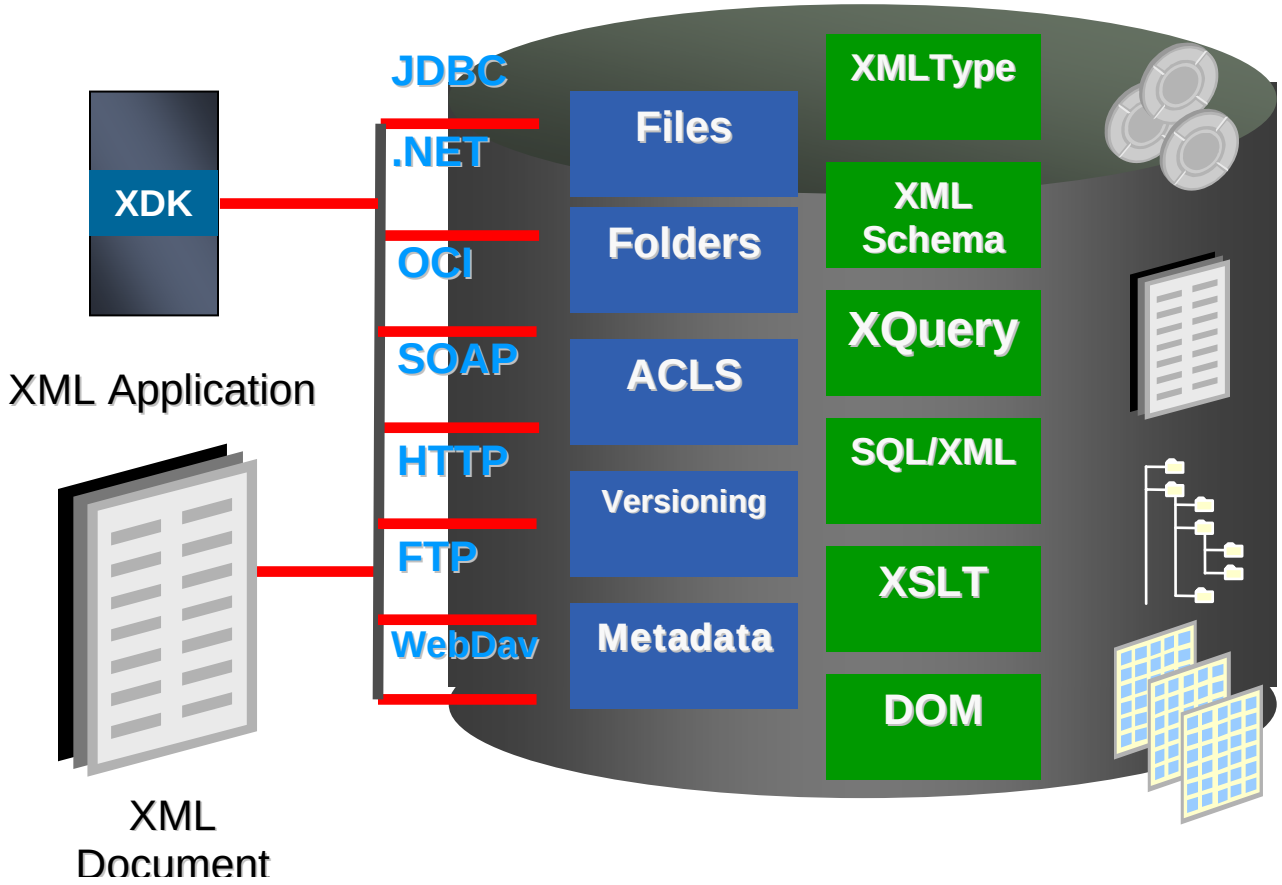
Oracle XML DB Architecture

The overall architecture of Oracle XML DB is explained in the following diagram:

XML content can be accessed programmatically using common API's included JDBC, .NET, OCI. It can also be accessed using the SOAP protocol. Documents in the XML DB repository can be accessed programmatically, or via File level protocols, such as HTTP/HTTPS, FTP and WebDAV. The Oracle XML DB repository can also be used with JSR-170.

The Oracle XML DB repository allows XML (and other kinds of) content to organized using a file folder hierarchy. The repository is based on the IETF DAV standard, file and folders in the repository are referred to as resources. Resources are protected by ACLS, which allow permissions to be granted or revoked from database users (principles). The repository maintains metadata for each resource it manages, basic meta-data is maintained automatically and the basic metadata can be augmented with user-defined metadata. The repository provides a simple, linear versioning model which can used to keep track of different versions of a resource. One major advantage of the Oracle XML DB repository is that it can be

accessed and updated directly from SQL. This allows any application capable of send SQL to an Oracle Database to work with content stored in the Oracle XML DB repository.



All XML content is managed using the XMLType data type. The XMLType and its associated operators allow the database to store and manage XML in the same way that is stores and manages other extended data types. XML specific operators enable validation, transformation, fragment extraction, node-level update and path-based searching. Oracle XML DB also provides support for the most import of the W3C's XML related standards.

XML Schema is supported a mechanism for validating content and optimizing storage and query.

XQuery is standard query language for XML. Oracle allows XQuery expressions to be evaluated as part of a SQL Statement via the standard operators XMLQuery, XMLTable and XMLExists.

SQL/XML based XML publishing allows for efficient, scalable generation of XML documents directly from relational data. The SQL/XML operators XMLElement, XMLAttributes, XMLAgg and XMLForest make it very easy to use a SQL statement to generate complex XML documents.

A powerful XSLT engine, the Oracle XSL VM allows XSL stylesheet based transformation to be performed in the database. Performing XSL transformation inside the database, next to the data can lead to significant performance benefits, especially when performing sparse transformations.

The packages DBMS_XMLDOM and DBMS_XMLPARSER provide a full implementation of the W3C DOM API, which can be used by a PL/SQL programmer. The Oracle XDK also provides Java and C implementations of these API that are optimized to work directly with an XMLType stored in the database.

The XMLType itself is an abstraction over multiple storage models. This include

- XML-Schema optimized object-relational XMLType: This option derives an optimized object-relational storage model from a W3C XML Schema. The XML is stored as a set of objects that conform to the storage model. The storage model can guarantee DOM fidelity, ensuring the DOM representation of XML stored in the database is identical to the DOM representation of the original document. When object-relational storage is selected XQuery expressions that perform fragment extraction, node-level update or path-based searching are automatically re-written into relational operations on the underlying storage model. This allows the full power of the relational engine to be used to evaluate the XQuery expression. Traditional relational B-Tree and Bitmapped indexing techniques can be used to optimize operations on object-relational XMLType.

- CLOB-based XMLType: This storage model stores the XML as a CLOB. It guarantees 100% document fidelity, ensuring that the XML stored in the database is byte-for-byte identical to the original document. CLOB based storage is useful when Document fidelity is mandatory, however it should be avoided when there is a requirement to access or update fragments of the XML. With CLOB-based XMLType fragment extraction and update operations are performed functionally. This requires each XML document to be parsed and operated on use DOM based APIs. If the document is updated the entire CLOB must be re-written when the changes are saved. This is acceptable when the operation is performed on a small number of documents but the overhead associated with parsing and the DOM traversal leads to poor-performance when the operation is performed on a large number of documents. Path-based searches on CLOB-based XML can be optimized by using the new XML Index feature of Oracle Database 11g.

- Binary XMLType. This storage model stores the XML as a BLOB. It stores the XML in post-parsed format that is understood by the database and by all of the components in the Oracle XDK. Binary XML storage should be chosen when object-relational storage is not appropriate. Good reasons for not choosing object-relational storage include

    - No XML Schema is available.

    - Need to store different kinds of XML document in a single table or column

    - The XML Schema changes rapidly and unpredictable or the changes are outside of the scope of what is manageable using the Schema Evolution features of Oracle XML DB.

- The XML Schema defines an object model that does map well to a SQL object model. This is true of document-centric schemas that tend to define very loose, recursive object models or XML Schemas where large parts of the content are mapped to the XML Schema any construct. The object model defined by this kind of XML Schema does not lend itself to the SQL based optimization.

Binary XML offers an extremely high degree of XML fidelity. All of the information in the DOM is preserved by the Binary XML format; however insignificant white-space will not be preserved by the encoding and decoding process. Binary XML supports streaming XPath evaluation. This enables efficient fragment extraction operations to be performed on Binary XML. The Binary XML model uses the sliding insert feature of Oracle Secure Files to support node-level updates. Sliding inserts enable partial updates of the underlying storage when the content of the document is updated. Binary XML was architected to be indexed by XML Index. This allows for efficient path-based search of the XML content. Creating an XML Index on Binary XML storage also helps optimize fragment extraction, since the index contains a node offset which is used to navigate directly to the part of the LOB that contains the required node.

- XMLType views: XMLType views define the content of an XMLType using SQL/XML operators. An XMLType view can be operated on just like any other XMLType, using XQuery and XSLT and DOM. XQuery operations on XMLType views are re-written into SQL operations on the underlying tables. XMLType views are updated using instead of triggers.

The purpose of this demonstration is to introduce the basic features of Oracle XML DB. Some of the more advanced XML-related features of Oracle Database 11g are not included in this demonstration. Features such as Binary XML, XML Indexing, Repository Events, Repository Metadata, Database Native Web Services, XML Type partitioning, in-place schema evolution are covered by other demonstrations that can be downloaded from the Oracle XML DB OTN page.

Installation

Server Pre-requisites

The following software is required to run the Oracle XML DB basic demonstration

Oracle Software

- Oracle Database 11g release 11.1.0.6.0 or later, with the XML DB, Oracle Text and Oracle JVM features installed.

Client Pre-requisites

The installation process uses an HTML application, VB Scripting and the HTTP protocol to upload the source code into the Oracle XML DB repository, SQL*PLUS scripts are used to re-configure the Oracle XML DB repository to support the demonstration.

The following software is required to install the Oracle XML DB Basic features demonstration.

Oracle Software

- Oracle Client (SQL*PLUS and Oracle Net Services) 11.1.0.6.0 (Production) or later. The application can be installed into a remote database, however both SQL*PLUS and Oracle Net Services must be installed on the client machine in order to perform a remote install. Currently remote installs are only supported on the Windows platform.

- Oracle XML DB X-Files application. The basic demonstration runs inside an AJAX-framework that is included as part of the Oracle XML DB X-Files application. Starting with Oracle Database 11g the X-Files demonstration must be downloaded and installed before installing the basic features demonstration.

Non Oracle Software

- Microsoft Internet Explorer 7.0 with the latest service packs. The X-Files application has not been tested with any release of other browsers including Firefox, Mozilla, Netscape, Safari or Opera.

- Microsoft Windows Scripting Technologies version 5. Windows Scripting is used by the installation process. You can verify the version of Windows Scripting installed on your machine by opening a command prompt and typing the command cscript.

- Microsoft Core XML Services (MSXML) versions 4.0sp2 and 6.0 are required in-order install the demonstration. It is also required by the AJAX based framework that is used to run the demonstration.

  At the time of writing the latest version of this software can be downloaded from http://www.microsoft.com/downloads/Search.aspx?displaylang=en

- XMLSPY: XMLSPY is an IDE for XML from Altova Corporation. An evaluation copy of this product can be obtained from Altova's website at http://www.altova.com.
- Mapforce: Mapforce is a XML mapping tool from Altova Corporation. If you do not have a license for this product you can download an evaluation copy from http://www.altova.com.
- Microsoft Windows XP Professional with Service Pack 2.
- Microsoft Word and Microsoft Excel from Microsoft Office 2000, 2003 or 2007.

Database Configuration Changes

Installing the Basic Features demonstration will make the following changes to the configuration of the target database.

- **Oracle XML DB HTTP and FTP Server:** Installing the basic demonstration will enable the database's native HTTP and FTP Servers. Ensure that you have read the XML DB documentation regarding the use of the XML DB HTTP Server before installing this application into a database that contains production data. This information can be found in the Oracle XML DB Developers guide.

- **Database Native Web Services:** The basic demonstration is executed using an AJAX-based framework. Ensure that you have read the XML DB documentation regarding the use of the Database Native Web Services before installing this application into a database that contains production data. This information can be found in the Oracle XML DB Developers guide.

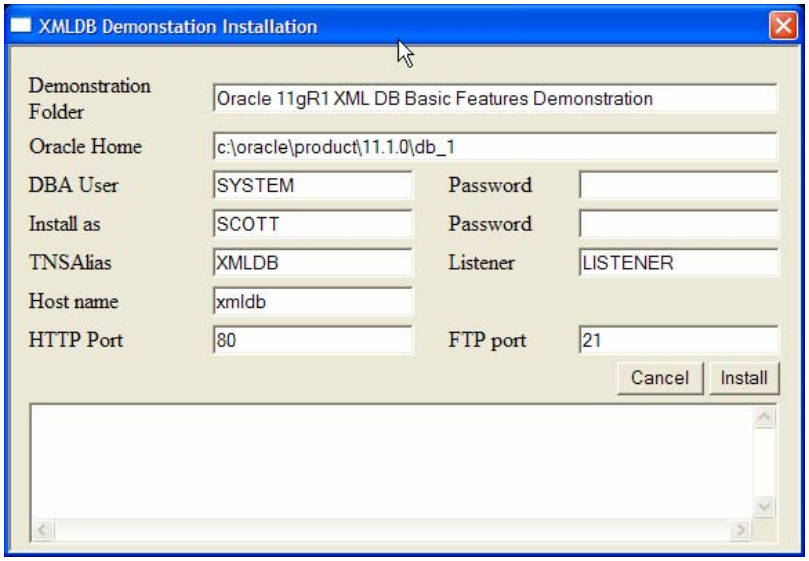
Installing the Application

To install the Oracle XML DB 11g Release 1 basic features demonstration unzip the contents of the file XMLDB_11gR1_Basic.zip into a folder of your choice. Ensure that there are no spaces in any of the parent folder names. After unzipping this file the target folder should contain a subfolder called basicFeatures. This folder will contain subfolders SQL, setup and Install.

To start the installation, execute the file install.hta found in the install subfolder.

```
C:\basicFeatures\install>install.hta
```

This will launch the installer dialog. The installation process is an HTML application.



The default values for the dialog are obtained from the file InstallationParameters.xml. The contents of this file are as follows:

```
<installationParameters>
  <shortCutFolderName>Oracle 11gR1 XML DB Basic Features Demonstration</shortCutFolderName>
  <oracleHome>c:\oracle\product\11.1.0\db_1</oracleHome>
  <dba>SYSTEM</dba>
  <oracleUser>SCOTT</oracleUser>
  <oraclePassword/>
  <tnsAlias>XMLDB</tnsAlias>
  <listener>LISTENER</listener>
  <sqlPort/>
  <hostName>xmldb</hostName>
  <httpPort>80</httpPort>
  <ftpPort>21</ftpPort>
  <parameter name="%BASEFOLDER%" value="/publishedContent/basicFeatures" />
</installationParameters>
```

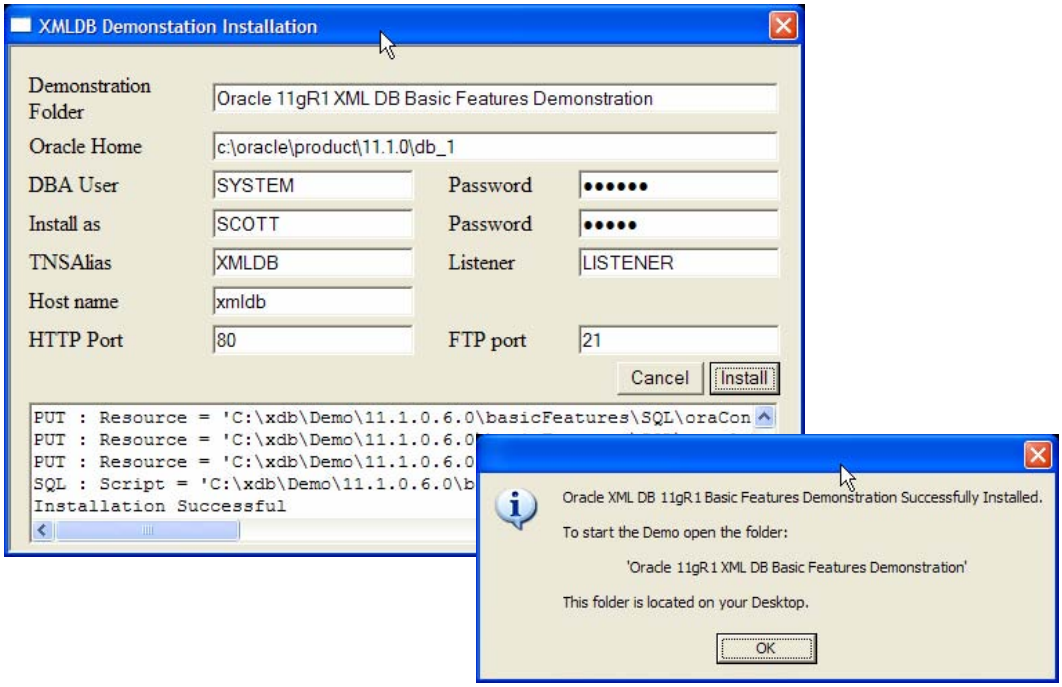
You can modify the default values by modifying the contents of InstallationParameters.xml before starting the installation. To install the demonstration, start the installer, modify any values that are not correct for your environment, enter the DBA and installation user's passwords and click install. Remember that in Oracle Database 11g passwords are case-sensitive. To cancel the installation click cancel.

The meaning of each parameter / field is given in the following table

< shortCutFolderName >	The name of the folder that will contain the set of Icons used to run the demo. This folder will be placed on the user's desktop.
<oracleHome>	The location of the Oracle Home on the local computer. An Oracle client installation is required to run the installation process.
<dba>	The name of a user with DBA capabilities which can be used to install the demonstration. Normally this will be SYSTEM, but any DBA is acceptable. The password for the DBA user can only be entered using the installation dialog.
<oracleUser>	The database user that will be used to run the demo. This user should already exists and be able to connect to the database. The installation process will grant this user the following privileges: session, unlimited tablespace, create table, create view, create any directory, drop any directory. These privileges are required to run the demonstration,
<oraclePassword>	The password for the demonstration user.
<tnsAlias>	The tnsAlias that can be used to connect to the target database instance
<listener>	The name of the listener associated with the database instance. A listener should not service more than one database when Oracle XML DB protocols are in use.
<hostname>	The name of the machine running the Listener.
httpPort	The port used by the Oracle XML DB HTTP Service. The port must not already be in use by any other service. The installation process will configure the database to use this HTTP port. If this port is a privileged port on a unix system the listener.ora must be configured appropriately.
ftpPort	The port used by the Oracle XML DB FTP Service. The port must not already be in use by any other service. The installation process will configure the database to use this HTTP port. If this port is a privileged port on a unix system the listener.ora must be configured appropriately.

Clicking Install will start the installation. The installation will verify that it can connect as the DBA, and as the demonstration user using SQL and HTTP. Once connectivity has been verified the demonstration will be installed. If the connectivity tests fail the installation will not proceed.

The progress of the installation will be shown in the status winder at the bottom of the installer dialog. When the installation is complete the following message will be displayed.



Click OK to dismiss the dialog and then cancel to exit the installer.

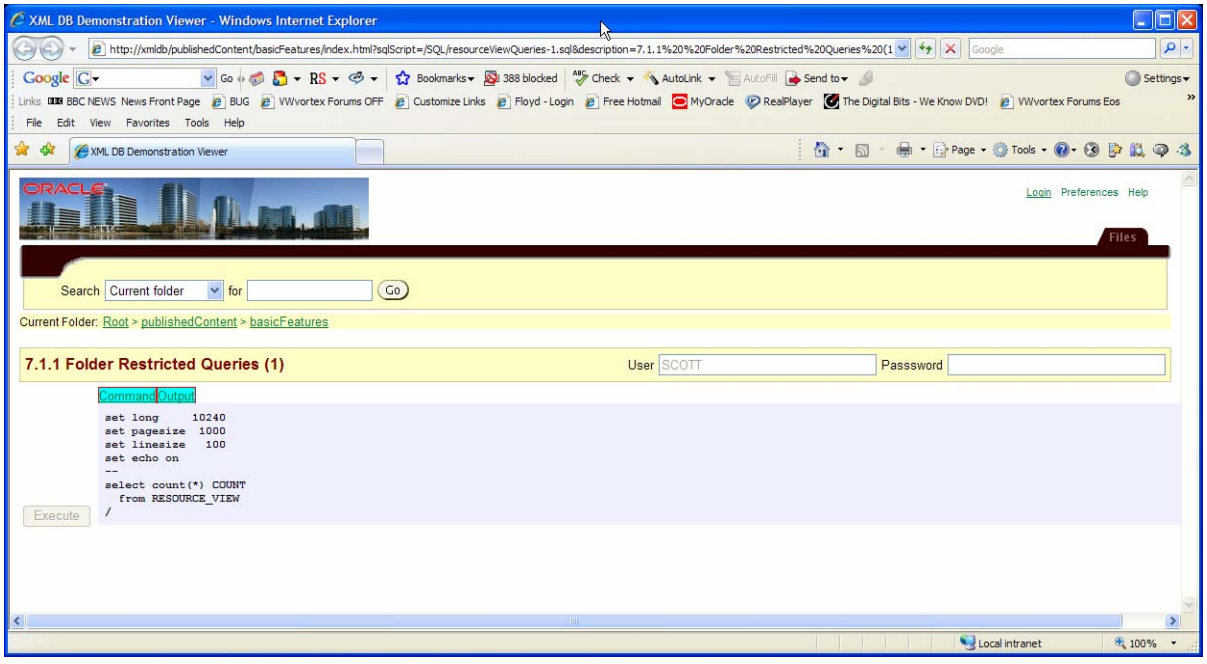
Using the demonstration framework

The SQL based portions of the demo are presented using the Oracle XML DB demonstration framework. This is an AJAX-based application uses the Database Native Web Services (DBNWS) feature of Oracle Database 11g to execute SQL scripts. The demonstration framework is installed as part of the X-Files application, which can be downloaded from the XML DB page on OTN. Please make sure you have the latest version of the X-Files application installed before running this demonstration.

The XML DB demonstration framework is launched by clicking the icons in the demonstration folder. If the browser currently owns an authenticated HTTP connection to the Oracle XML DB Database, the framework will automatically execute the SQL script. If the browser does not own an authenticated HTTP connection to the Oracle Database, or the current session does not belong to the correct user, the framework will prompt for a password before running the script. Entering the correct password will execute the script.

If the demonstration framework encounters a pause command, the CONTINUE button will be enabled. Click the CONTINUE button to continue executing the script. When the script is complete the CLOSE button will be enabled. Click the close button will close the framework session. If at least one framework session is left open, subsequent windows will be able to inherit the HTTP connection, avoiding the need to enter a password each time a new framework session is opened.

The following screen shot shows the demonstration framework ready to run a script. The framework is waiting for the password to be entered. The first SQL command in the script is displayed in the command area. The cursor is positioned in the Password field and the EXCUTE button is disabled.



Entering a valid password will automatically execute the SQL statement.

The framework consists of interleaved command and output areas.

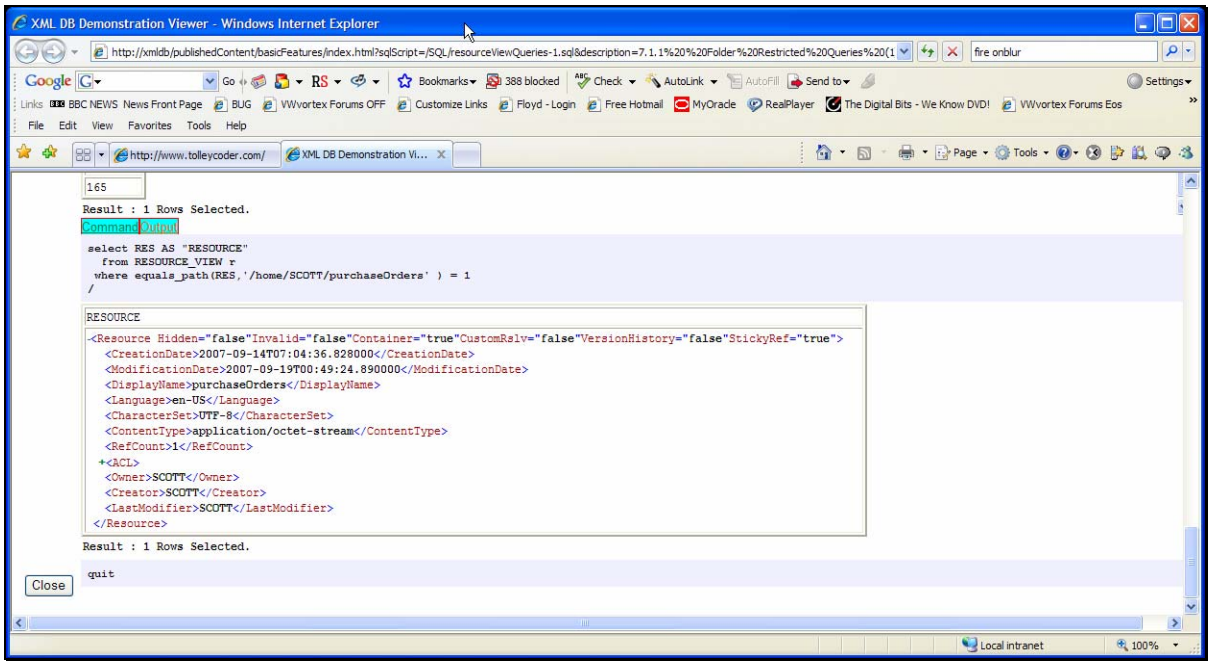
The command area shows the current SQL command. If the SQL command contain more lines than can be displayed in the default command area a vertical scroll bar will appear. The scroll bar can be used to scroll the contents of the command area. When the scroll bar is present clicking the Command tab will expand the command area to show the entire command. Once the command area is expanded clicking the Command tab again will revert to the default size.

The output area shows the results of the query. If the script includes a **set autotrace on explain** command the output area can also show the query plan for the current query. When the query plan is available two additional tabs, Result and Plan will be displayed. Click Result to see the query output, Click Plan to see the query plan.

If the command generates more output than can be displayed by the default output area a vertical scroll bar will appear. The scroll bar can be used to scroll the contents of the output area. When the scroll bar is present clicking the Output tab will expand the output area to show as much of the output as possible. Once the output area has been expanding clicking the Output tab again will revert to the default size. This behavior also occurs when the query plan is displayed in the output area.

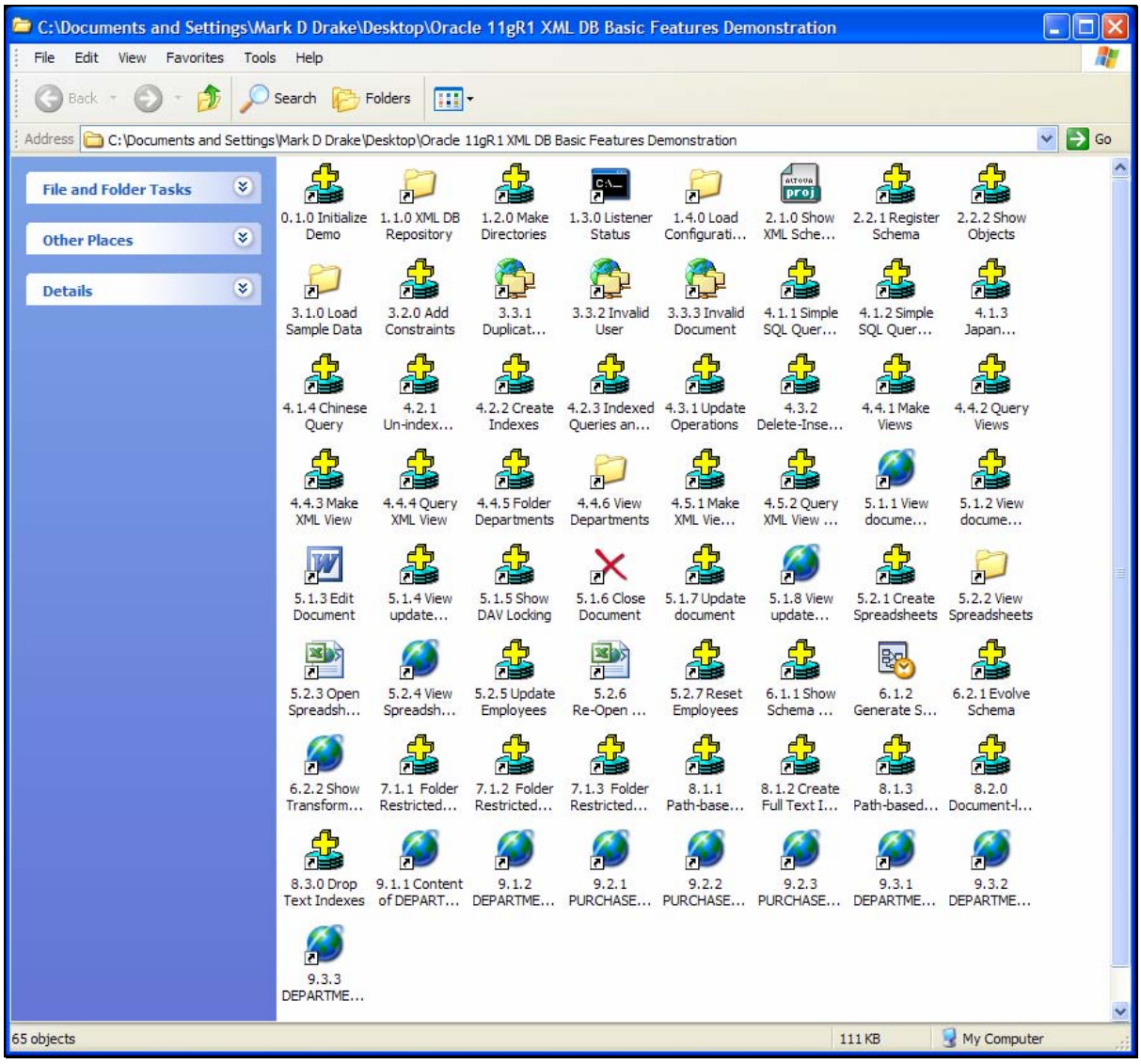
Once major advantage of the demonstration framework is that is completely XML aware. When a query executes XML, the XML will be displayed complete will all of its tag information intact. Elements with complex content will be marked with a **-** icon. Clicking the **-** icon will close up the children of the element, and replace the **-** icon with a **+** icon. Clicking the **+** icon will expand the sub-tree for the element.

The following screen show shows the demonstration framework after completion of a script. The output area contains an XML document. Since the script is complete the CLOSE button is enabled.



Performing the demonstration

The installation creates the folder "Oracle 11gR1 XML DB Basic Features Demonstration". This folder is located on the Desktop. The folder contains the following Icons.



The icons are numbered. To run the demonstration, click each icon in turn.

0.1.0 Initialize Demo

This step initializes the demonstration.

Click the icon to launch the XML DB demonstration framework and run the SQL script. The username field will be pre-filled with the name of the demonstration user and the password field will be empty. Type the password and hit enter or click outside of the password field. If the password is entered correctly the form will refresh and the script will execute. If an HTTP authentication dialog appears after entering the correct password then Database Native Web Services have not been correctly configured for the user.

The script undoes all the actions that are performed by the demonstration, including:

Command Output

```
set echo on
--
-- Reset the changes to HR_SCHEMA
--
update hr.employees
   set DEPARTMENT_ID = NULL
 where EMPLOYEE_ID = 178
/

1 row Updated.
```

Command Output

```
update hr.employees
   set DEPARTMENT_ID = 50
 where EMPLOYEE_ID = 199
/

1 row Updated.
```

Command Output

```
delete from hr.job_history
 where (employee_id = 178 or employee_id = 199)
/

1 row Deleted.
```

- Reverse all changes made to the HR schema

Command Output

```
--
-- Delete XML Views
--
declare
  non_existant_view exception;
  PRAGMA EXCEPTION_INIT( non_existant_view , -942 );
begin
--
  begin
    execute immediate 'DROP VIEW DEPARTMENT_WORKBOOK_XML';
  exception
    when non_existant_view then
      null;
  end;
  begin
    execute immediate 'DROP VIEW DEPARTMENT_XML';
  exception
    when non_existant_view then
      null;
  end;
end;
/

PL/SQL procedure successfully completed.
```

- Drop the views created by running the demonstration

Command Output

```
--
-- DROP the context index DESCRIPTION_TEXT_INDEX before dropping the Schema
--
declare
  non_existant_index exception;
  PRAGMA EXCEPTION_INIT( non_existant_index , -1418 );
begin
--
  begin
    execute immediate 'drop index iDESCRIPTION_FULL_TEXT';
  exception
    when non_existant_index then
      null;
  end;
  begin
    execute immediate 'drop index iPURCHASEORDER_FULL_TEXT';
  exception
    when non_existant_index then
      null;
  end;
--
end;
/
```

```
PL/SQL procedure successfully completed.
```

- Drop the full-text indexes created by running the demonstration

Command Output

```
declare
  unregistered_schema exception;
  PRAGMA EXCEPTION_INIT( unregistered_schema , -31000 );
begin
  dbms_xmlschema.deleteSchema
  (
     'http://xmldb:80/home/SCOTT/poSource/xsd/purchaseOrder.xsd',
     dbms_xmlschema.DELETE_CASCADE_FORCE);
exception
  when unregistered_schema then
    null;
end;
/
```

```
PL/SQL procedure successfully completed.
```

- Drop the PurchaseOrder XML schema and all dependant objects and resources

Command Output

```
--
--
-- Create home directory for target user...
--
call xdb_utilities.createHomeFolder()
/
```

```
PL/SQL procedure successfully completed.
```

Command Output

```
begin
  if (dbms_xdb.existsResource('/home/SCOTT/purchaseOrders')) then
    dbms_xdb.deleteResource('/home/SCOTT/purchaseOrders',dbms_xdb.DELETE_RECURSIVE_FORCE);
  end if;
  if (dbms_xdb.existsResource('/home/SCOTT/poSource')) then
    dbms_xdb.deleteResource('/home/SCOTT/poSource',dbms_xdb.DELETE_RECURSIVE_FORCE);
  end if;
  if (dbms_xdb.existsResource('/home/SCOTT/Departments')) then
    dbms_xdb.deleteResource('/home/SCOTT/Departments',dbms_xdb.DELETE_RECURSIVE_FORCE);
  end if;
  if (dbms_xdb.existsResource('/home/SCOTT/Workbooks/Departments')) then
    dbms_xdb.deleteResource('/home/SCOTT/Workbooks/Departments',dbms_xdb.DELETE_RECURSIVE_FORCE);
  end if;
end;
/
```

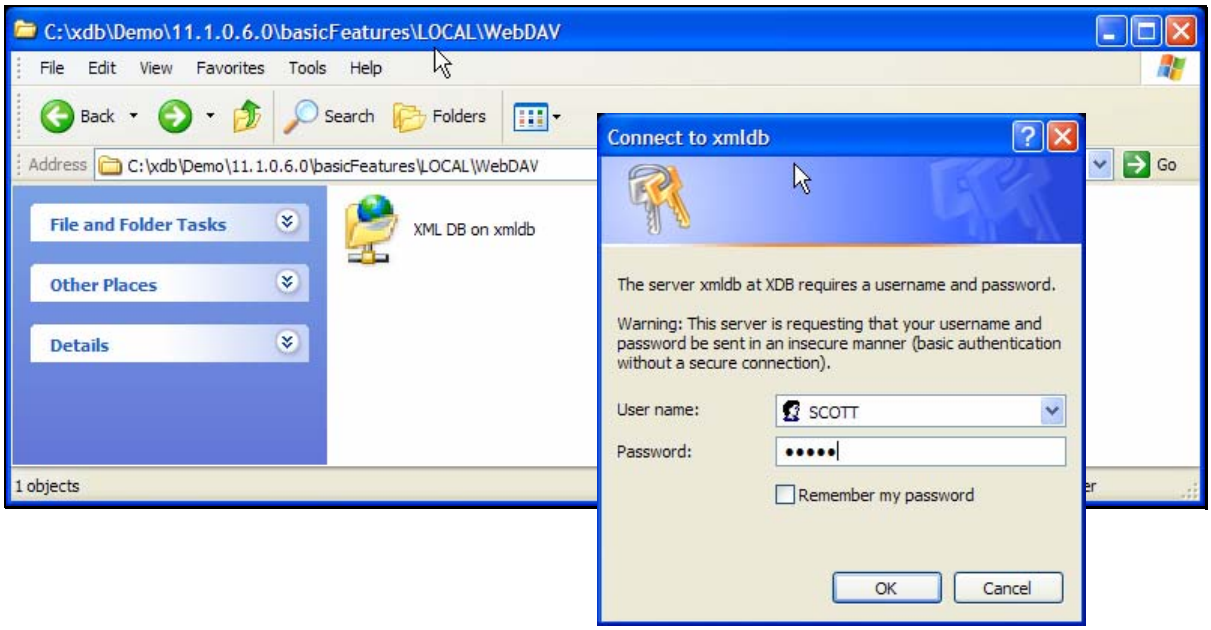
```
PL/SQL procedure successfully completed.
```

- Reset the status and content of all documents in the demonstration user's home folder
- Recreate the library of styles used when generating Excel spreadsheets.

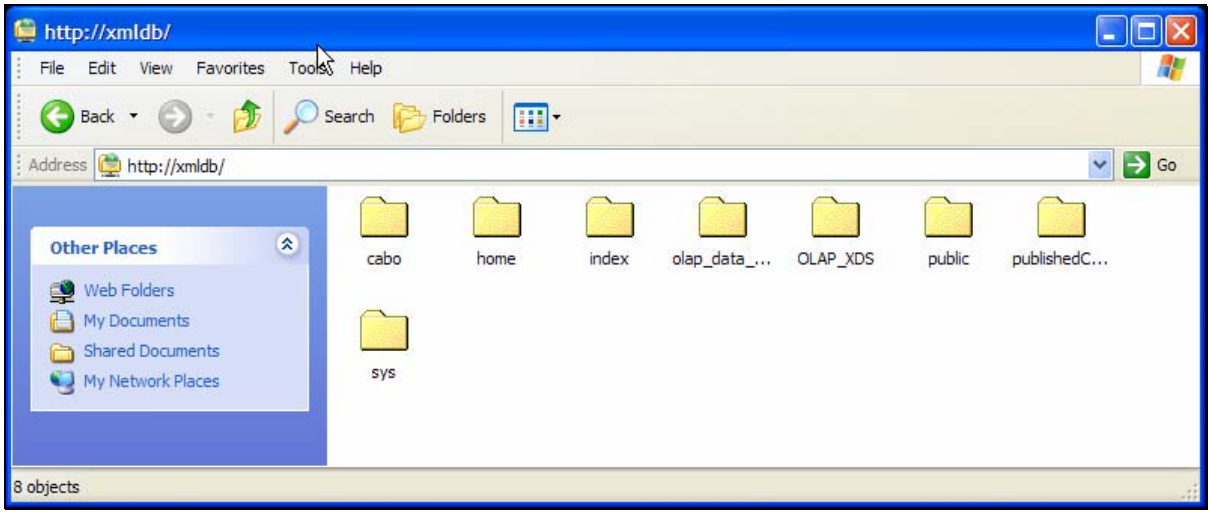
1.1.0 XML DB Repository

This step uses Microsoft Windows Explorer to access the Oracle XML DB repository. Right click the icon and select explore. This will open a new window containing the local folder WebDAV. This folder contains a shortcut called "XML DB on ***hostname***".

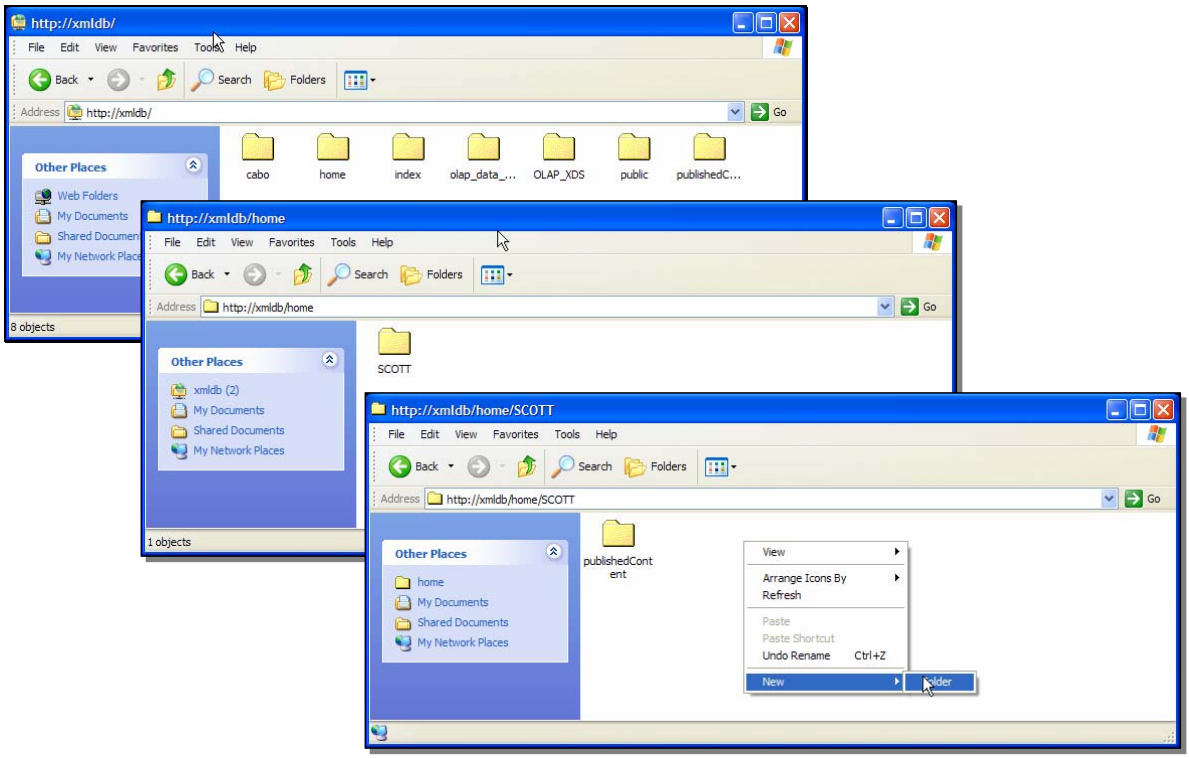
Click the shortcut to establish a WebDAV connection between Microsoft Windows and the Oracle XML DB repository. Since the repository requires an authenticated connection, Windows Explorer will prompt for a username and password. Enter the demonstration user's username and password and click OK.



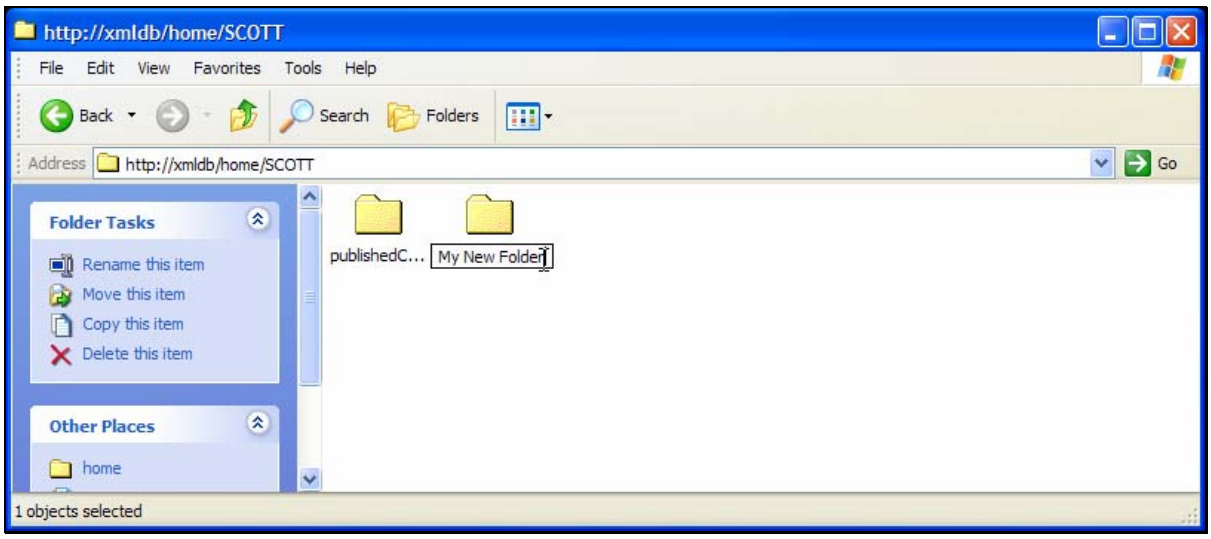
Windows Explore will open a window showing the contents of the Oracle XML DB repository root folder



Double click the home folder icon to view the demonstration user's home folder. In this example the demonstration user is SCOTT so the home folder is called SCOTT. Double click the home folder to view its content. Right click anywhere in the home folder and select new, folder.



Windows Explorer will create a new folder in the demonstration user's home folder.



Give the new folder a clearly identifiable name.

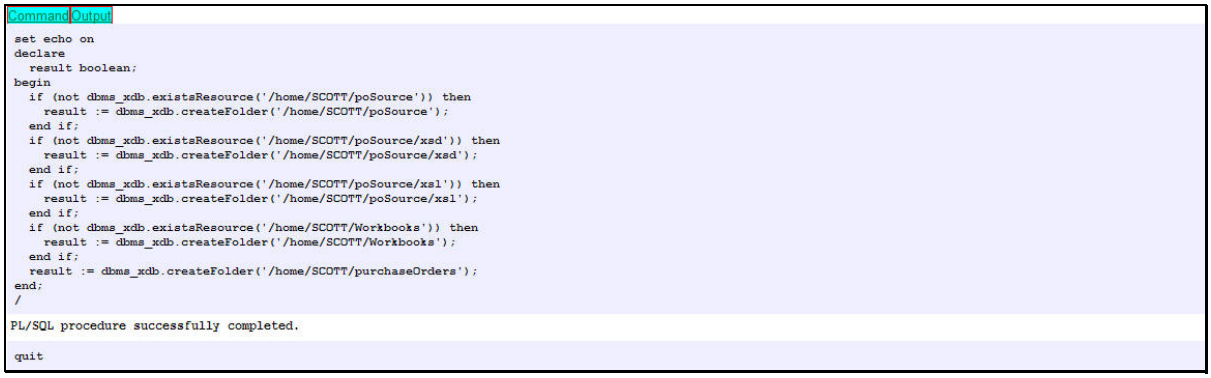
The Oracle XML DB repository allows XML and other kinds of content to be organized and managed using a familiar, intuitive file/folder metaphor. The repository supports the FTP, HTTP, HTTPS and the WebDAV extensions to HTTP. This allows the content stored in the repository to be accessed using standard tools like Microsoft Windows Explorer and Microsoft Office. No additional plug-ins, adaptors or drivers need to be installed to use this feature.

WebDAV support allows end users to access the Oracle XML DB repository using familiar tools and interfaces. WebDAV is an open standard, defined by the IETF. The standard defines a set of extensions to the HTTP protocol that allow an HTTP Server to appear as a file server to a DAV enabled client. Many vendors, including Microsoft, Adobe and Macromedia, now provide support for WebDAV in their products. Consequently popular desktop applications such as Word, Excel, Acrobat and Dreamweaver are able to work directly with content stored and managed by the Oracle XML DB repository.

The WebDAV standard uses the term resource to describe a file or a folder. Every resource managed by a WebDAV server is identified by a URL.

1.2.0 Make Directories

This step uses PL/SQL to create folders in the Oracle XML DB repository. It also demonstrates the interaction between changes made in PL/SQL and Windows Explorer. Click the icon to launch the XML DB demonstration framework and run the SQL script.

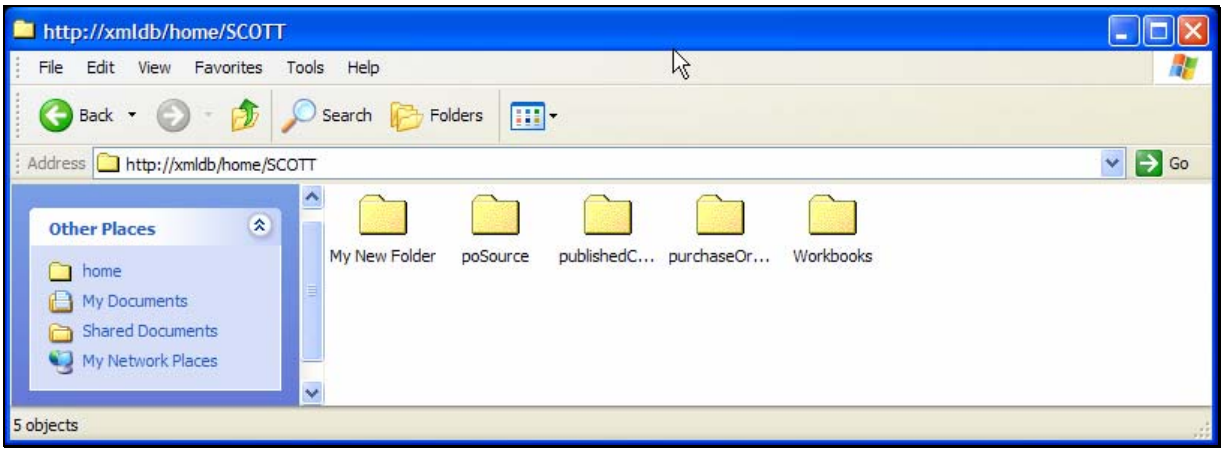

Command Output

```
set echo on
declare
  result boolean;
begin
  if (not dbms_xdb.existsResource('/home/SCOTT/poSource')) then
    result := dbms_xdb.createFolder('/home/SCOTT/poSource');
  end if;
  if (not dbms_xdb.existsResource('/home/SCOTT/poSource/xsd')) then
    result := dbms_xdb.createFolder('/home/SCOTT/poSource/xsd');
  end if;
  if (not dbms_xdb.existsResource('/home/SCOTT/poSource/xsl')) then
    result := dbms_xdb.createFolder('/home/SCOTT/poSource/xsl');
  end if;
  if (not dbms_xdb.existsResource('/home/SCOTT/Workbooks')) then
    result := dbms_xdb.createFolder('/home/SCOTT/Workbooks');
  end if;
  result := dbms_xdb.createFolder('/home/SCOTT/purchaseOrders');
end;
/

PL/SQL procedure successfully completed.

quit
```

The script uses procedure createFolder, defined by package DBMS_XDB, to create new folders in the demonstration user's home folder. Once the SQL script is complete refresh the explorer window that contains the demonstration user's home folder.



The Package DBMS_XDB enables operations on the repository from SQL and PL/SQL. This means that any program capable of calling a PL/SQL procedure can work with the Oracle XML DB repository.

Operations performed via protocols are atomic. E.g. each operation, such as creating a resource, renaming a resource, deleting a resource, deleting a folder, etc is a transaction in its own right. Each change is immediately visible to all other users.

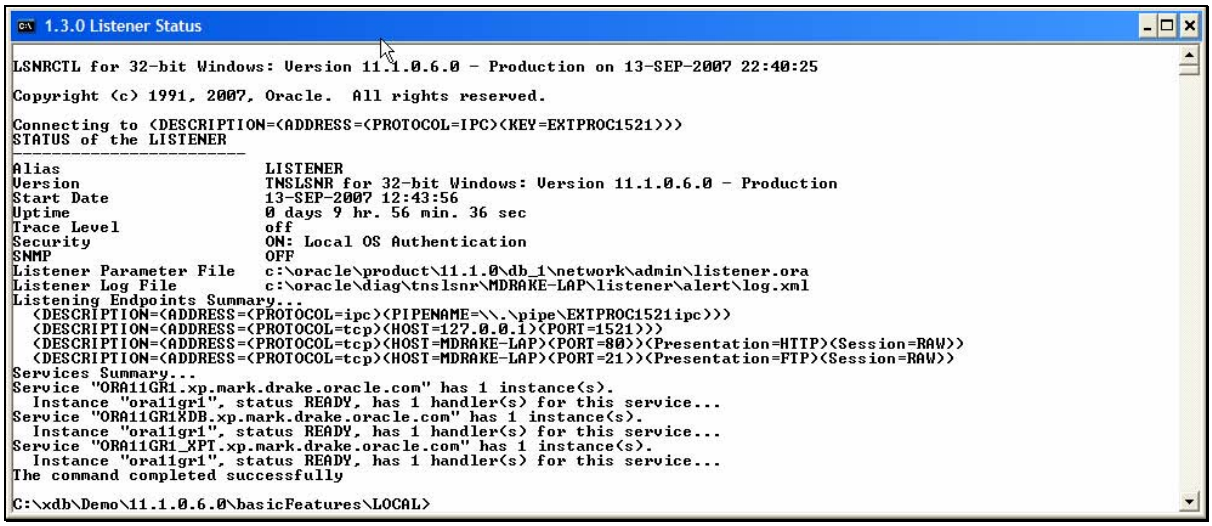
Operations performed via SQL are transactional in nature. Changes made from a SQL session are not visible to other users until they are committed. In SQL, a set of repository operations can be performed as a single transaction which can be rolled-back if an error is encountered.

Oracle XML DB Database Native Web Services are currently stateless; each invocation of the Web Service is treated as an atomic transaction. Since the demonstration framework uses Database Native Web Services to execute SQL the changes made by the running the demonstration scripts are visible to others users as soon as each service invocation completes.

1.3.0 Listener Status

This step shows that the Oracle XML DB protocols leverage the existing networking infrastructure provided by the Oracle Listener. Click the icon to display the current status of the local Listener.

This step only works when the demonstration is being run against a local database instance. If a remote instance is being used, log into the remote server and manually execute the command "lsnrctl status".



The Oracle XML DB protocol implementation is tightly integrated with the standard Oracle networking infrastructure. The Oracle Listener supports the HTTP, WebDAV and FTP protocols in addition to Oracle Net Services (SQL*NET). The Listener listens for HTTP and FTP requests in the same way that is listens for Oracle Net Services requests. HTTP and FTP requests are handed off to an Oracle Shared-server. When the request is processed the response is sent back to the client. The database should be configured with enough Shared-server processes to handle the desired number of concurrent users.

Use the command lsnrctl status to determine whether or not the HTTP, HTTPS and FTP protocols are active. A listener instance can only provide HTTP and FTP support for one database instance. In the above example the database is configured to listen for HTTP on port 80, and FTP or port 21.

The protocols are disabled by default. They are enabled by specifying the desired port number for each protocol. The port numbers are stored in the database instance's xdbconfig.xml file. This file is found in the root folder of the Oracle XML DB repository. XDBADMIN is required to be able to view or update this file. The port numbers can be supplied using Oracle Database Control or by calling procedures setHTTPPort and setFTPPort in package DBMS_XDB.

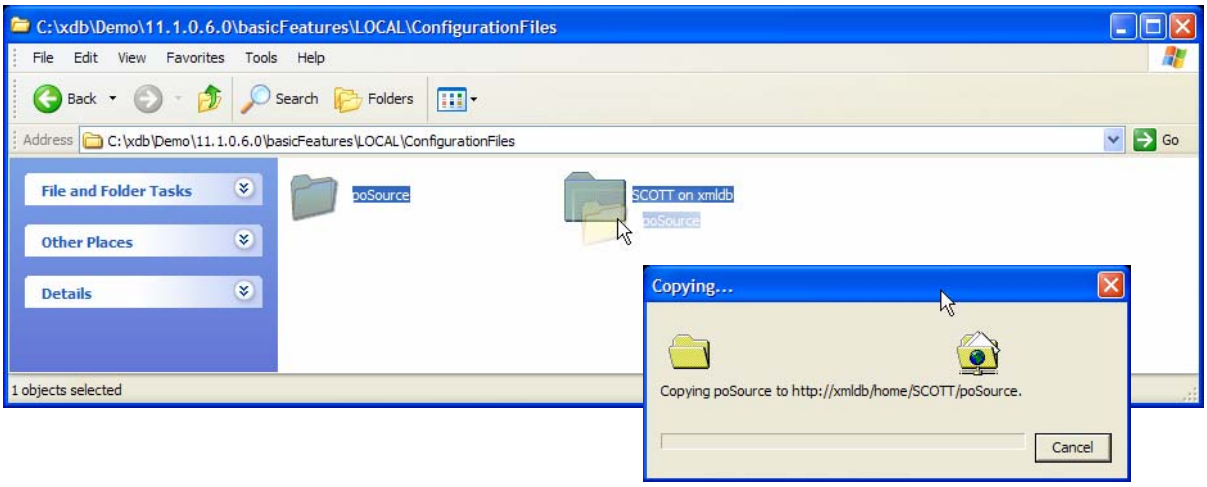
Special considerations exist when using privileged (port numbers < 1024) ports for HTTP or FTP in a UNIX or Linux environment.

1.4.0 Load Configuration Files

This step uses Windows Explorer to load files into the Oracle XML DB repository. Many Oracle XML DB features operate on XML files. Examples include registering an XML Schema and performing XSL transformations. These files must be accessible to the database instance. The easiest way to make the files available is to load them into the Oracle XML DB repository. The WebDAV protocol allows files to be loaded into the Oracle XML DB repository using a simple drag and drop operation.

Right click the icon and select explore. This will open a new window containing the local folder ConfigurationFiles. This folder contains a folder called poSource and a shortcut called "***user*** on ***hostname***". The poSource folder is simply a folder on the local file system., the shortcut is a link to demonstration user's home folder in the Oracle XML DB repository.

Click on the poSource folder and drag into on the shortcut. When prompted, enter the demonstration user's username and password and click OK.



Windows explorer copies the folder and its contents from the local hard-drive into the Oracle XML DB repository. This folder contains XML Schema and XSLT style sheets.

The Oracle XML DB repository can also manage non XML content, such as HTML files, JPEG images, word documents etc.

2.1.0 Show XML Schema

This step provides an introduction to XML Schema and the way in which Oracle XML DB use's XML Schema to optimize storage and process of highly structured XML content.

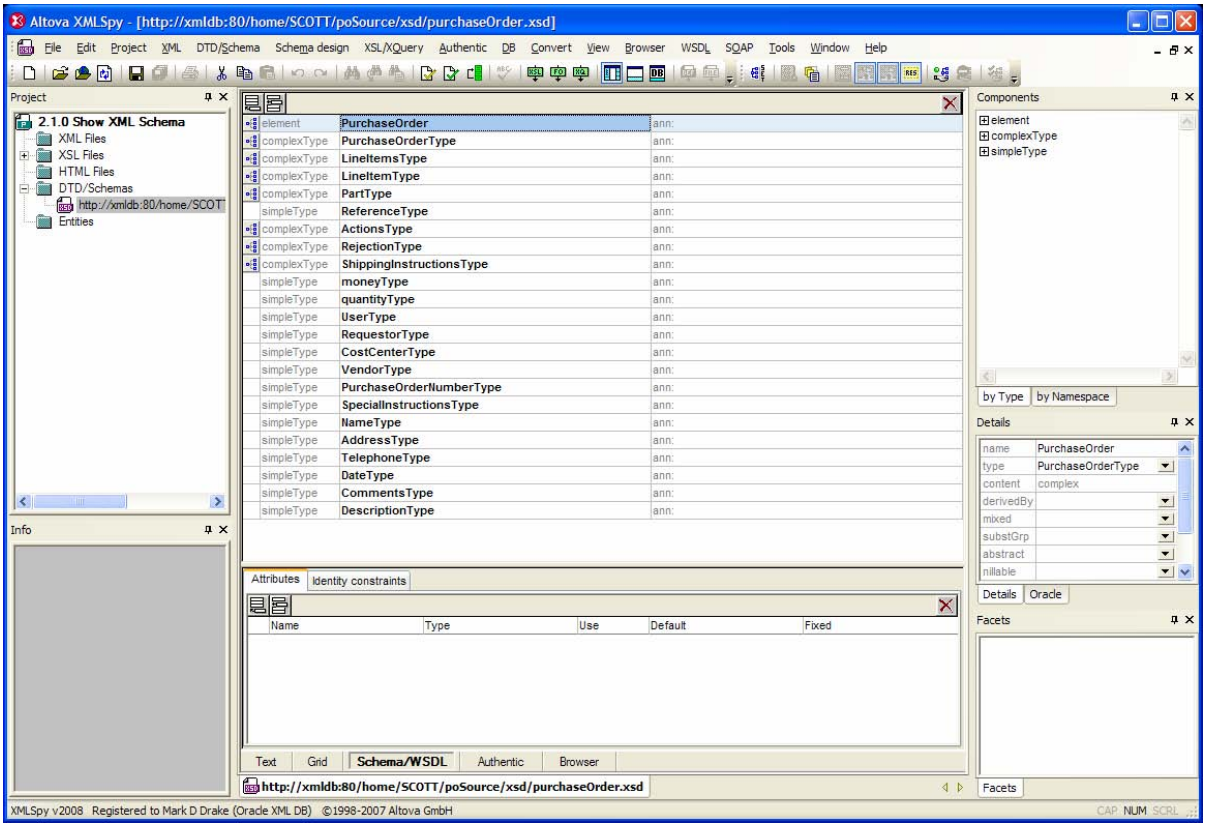
Creating and Editing an XML Schema

There are number of tools that can be used to create and edit an XML Schemas, including Oracle's JDeveloper. These tools provide a graphical, easy to use, interface for creating and editing an XML Schema. Most of these tools also include support for FTP and WebDAV, allowing them to work directly with the Oracle XML DB repository.

The current market leader is XMLSPY from Altova. XMLSPY includes Oracle XML DB specific functionality that simplifies the process of creating an XML Schema for use with Oracle XML DB. See http://www.altova.com for more details about XMLSPY.

Click the icon to open XMLSPY. When the application opens click the + sign next to the DTD/Schemas entry in the Project Panel. This branch will contain a single XML Schema, accessible via the following URL: http://hostname:httpPort/home/USER/poSource/xsd/purchaseOrder.xsd.

Double-click this item. XMLSPY will prompt for a username and password. Enter the demonstration user's username password and click OK. XML Spy will open the XML Schema using HTTP and display a list of the elements and types defined by the XML Schema.

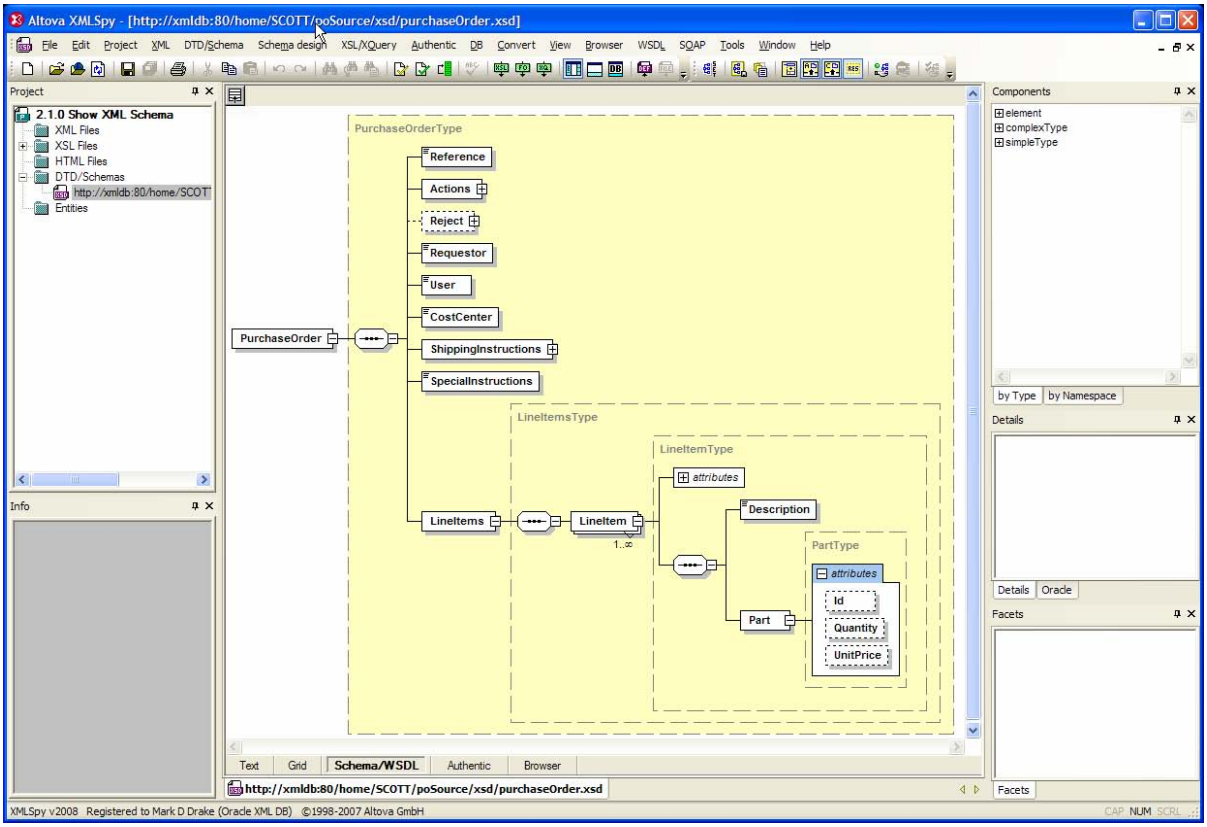


The XML Schema standard defines a language that can be used to define the structure of an XML document. An XML Schema is an XML document, compliant with the Schema for Schemas. This Schema is defined by the W3C.

XML Schema allows for strong typing of the elements and attributes in a document. It defines 47 scalar data types. The base set of types can be extended using object orientated techniques like inheritance and extension to define more complex types. XML Schemas are typically used a mechanism for validating instance documents. Oracle XML DB can use XML Schema in this manner.

XMLSPY provides a powerful, graphical, easy to use interface for creating and editing XML Schemas. XMLSPY supports both the WebDAV and FTP protocols allowing it direct access to content stored in Oracle XML DB.

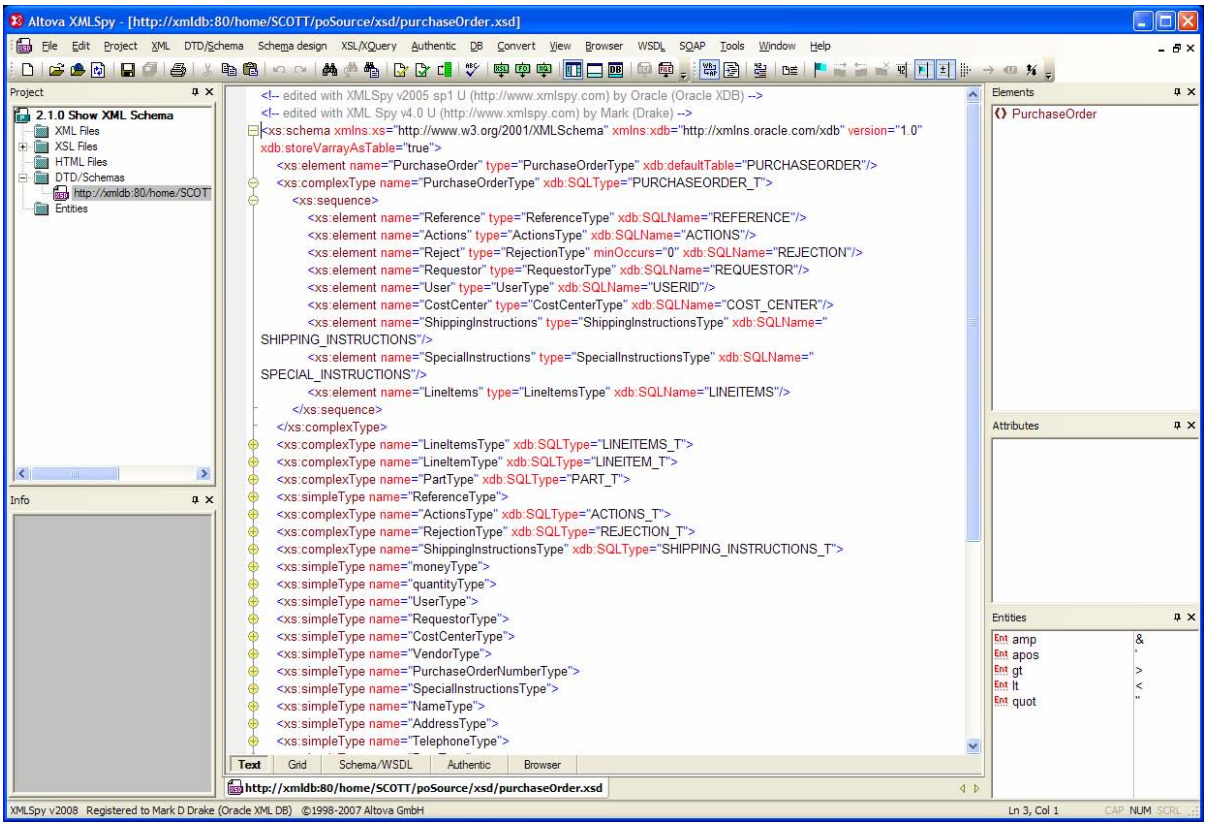
Click the  control next to element PurchaseOrder. Click the + sign next to the element LineItems, followed by the + sign next to element LineItem. Finally click on the + sign next to element Part. XMLSPY displays a graphical representation of the XML Schema.



The PurchaseOrder schema is a relatively simple XML Schema that demonstrates the key features of a typical XML document:

- The global element PurchaseOrder is an instance of the complexType PurchaseOrderType.
- PurchaseOrderType defines the set of nodes which make up a PurchaseOrder element.
- The LineItems element consists of a collection of LineItem elements.
- Each LineItem elements consists of two elements, Description and Part.
- The Part element has attributes Id, Quantity and UnitPrice.

Click the Text tab to switch into text mode. This will display the XML Schema as an XML document.



XMLSPY allows the XML Schema editor to work directly with the XML Schema in its native form This schema defines two namespaces:

- http://www.w3c.org/2001/XMLSchema, is the namespace reserved by the W3C for the Schema for Schemas. This namespace is used to define the structure of the XML document.

- http://xmlns.oracle.com/xdb is the namespace reserved by Oracle for the Oracle XML DB schema annotations. This namespace is used to add Oracle XML DB specific information to the XML Schema that control how the instance documents will be stored in the database.

- The annotation mechanism is the W3C approved mechanism for adding vendor specific information to a W3C XML Schema.

Annotating the XML Schema

Annotations allow database administrators and application developers to influence the way Oracle XML DB processes the XML Schema. Using annotations provides two basic benefits.

- Fine-Tuning of the storage model to ensure it meets the needs of the application.

- Enforcing site-specific naming conventions for any SQL objects that are generated from the XML Schema.

Oracle XML DB can register an XML Schema which contains no annotations. A set of default rules are used to generate valid SQL names for the objects generated from the XML Schema. Annotations are commonly used to control:

- The naming of the XMLType table that is associated with each global element defined by the XML Schema.

- The naming of SQL Types and Attributes

- The mapping between the XML Schema data types and SQL data types.

- How repeating elements are managed in the SQL object model.

- The degree of XML fidelity that is required

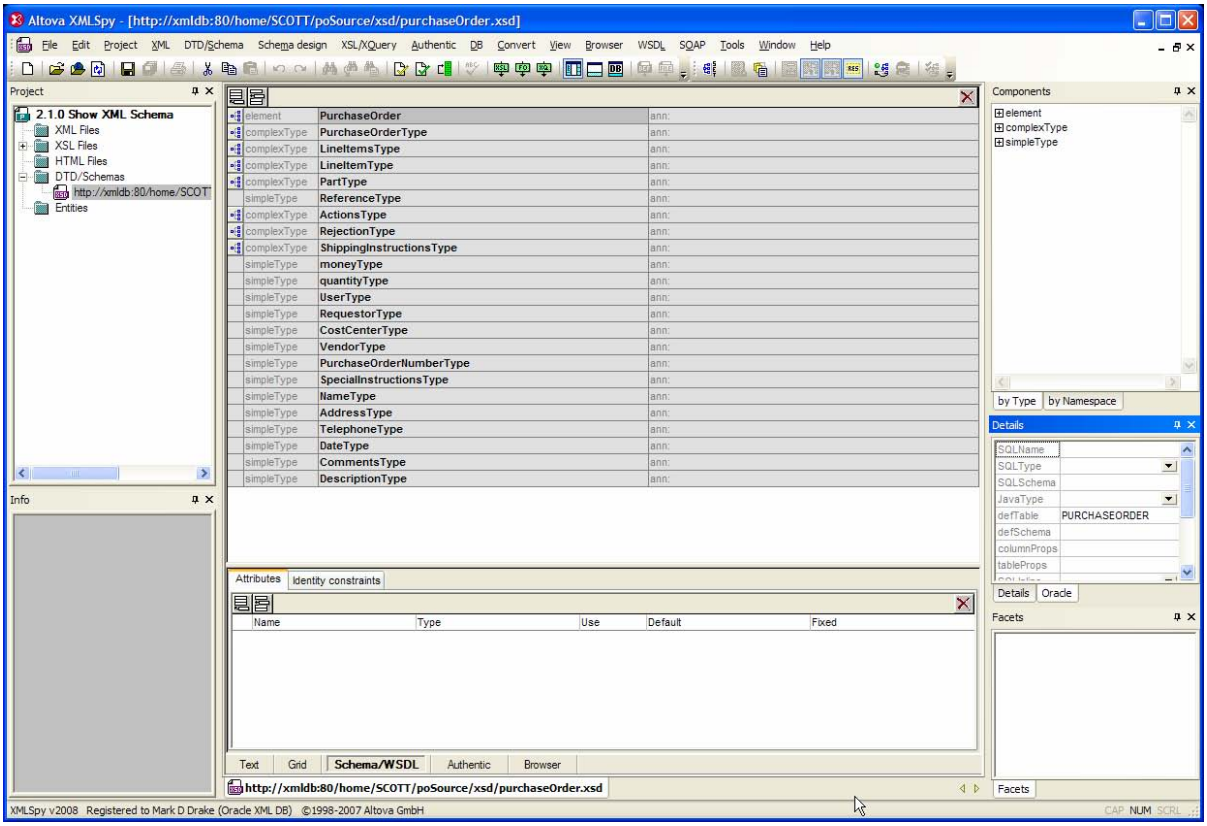
In the example shown above the following simple annotations are used

- xdb:defaultTable specifies the name of the default table for PurchaseOrder documents will be PURCHASEORDER..

- xdb:SQLType annotation specifies the name of the SQL object type corresponding to the complexType PurchaseOrderType, will be PURCHASEORDER_T.

- xdb:SQLName specifies the name of the SQL Attribute corresponding to element Reference will be REFERENCE.

Click the Schema/WDSL tab to switch the XMLSPY editor back in graphical mode and click the Oracle tab in the Details window.

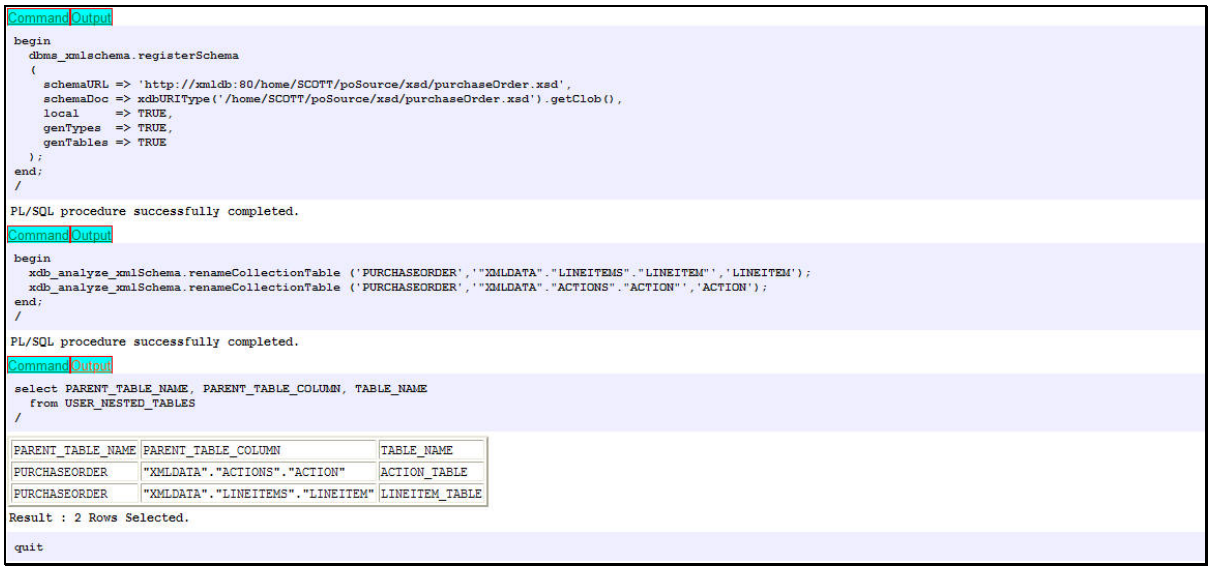


This shows the XMLSPY panel that allows entry of Oracle specific annotations while editing the XML Schema. This feature is enabled when ever an XML Schema declares the Oracle XML DB namespace or by selecting the Enable Oracle Schema Extensions option from the Schema design menu.

2.2.1 Register Schema

This step registers the PurchaseOrder XML Schema with Oracle XML DB. Registering an XML Schema with the database enables optimized processing for the instance documents that belong to the XML Schema.

Click the icon to launch the XML DB demonstration framework and run the SQL script.



Command Output

```
begin
  dbms_xmlschema.registerSchema
  (
    schemaURL => 'http://xmldb:80/home/SCOTT/poSource/xsd/purchaseOrder.xsd',
    schemaDoc => xdbURIType('/home/SCOTT/poSource/xsd/purchaseOrder.xsd').getClob(),
    local     => TRUE,
    genTypes  => TRUE,
    genTables => TRUE
  );
end;
/

PL/SQL procedure successfully completed.
```

Command Output

```
begin
  xdb_analyze_xmlSchema.renameCollectionTable ('PURCHASEORDER','"XMLDATA"."LINEITEMS"."LINEITEM"','LINEITEM');
  xdb_analyze_xmlSchema.renameCollectionTable ('PURCHASEORDER','"XMLDATA"."ACTIONS"."ACTION"','ACTION');
end;
/

PL/SQL procedure successfully completed.
```

Command Output

```
select PARENT_TABLE_NAME, PARENT_TABLE_COLUMN, TABLE_NAME
  from USER_NESTED_TABLES
/
```

PARENT_TABLE_NAME	PARENT_TABLE_COLUMN	TABLE_NAME
PURCHASEORDER	"XMLDATA"."ACTIONS"."ACTION"	ACTION_TABLE
PURCHASEORDER	"XMLDATA"."LINEITEMS"."LINEITEM"	LINEITEM_TABLE

```
Result : 2 Rows Selected.

quit
```

XML Schemas are registered using procedure registerSchema defined in package DBMS_XMLSCHEMA. IN Oracle Database 11g XML Schemas can be registered for use with either XML Schema-optimized object-relational storage or with Binary XML storage. The default is to register the XML Schema for use with object-relational storage. Binary XML storage is selected by setting parameter OPTIONS to DBMS_XMLSCHEMA.REGISTER_BINARYXML when calling registerSchema().

The use of an XML Schema is mandatory for XML Schema-optimized storage. The object-relational SQL Type model used to manage the instance documents is derived from the XML Schema's object model. Starting with Oracle Database 11g, collections (elements defined with maxOccurs > 1 or unbounded) are automatically mapped into heap organized nested tables when default tables are generated. This results in a storage model tuned for operations on collections. Prior to Oracle Database 11g, the schema annotation xdb:storeVarrayAsTable="true" needed to be added to the XML Schema in order to generate a collection optimized storage model.

The use of an XML Schema is optional with Binary XML storage. Registering an XML Schema for use with Binary XML storage enables automatic schema-validation as well as a more accurate mapping between XML and SQL data types and optimized use of XML index.

If parameter GENTABLES is true registerSchema generates a set of defaults tables that can store the instance documents.

The XML Schema is uniquely identified using parameter schemaURL. Instance documents that include a schemaLocation or noNamespaceSchemaLocation attribute matching the schemaURL will automatically be associated with the XML Schema. This method of identifying an instance document is defined by the W3C XMLSchema-Instance specification.

In this example the XML Schema is registered for object-relational storage.

- The SchemaURL value is used an internal identifier for the XML Schema. Oracle XML DB will never attempt to access the location identified by the schemaURL.

- The operator xdbUriType provides access to a document that has been stored in the XML DB repository.

- Nested tables are generated for the collection of Action elements and the collection of LineItem elements. Procedure registerSchema uses system generated names for these tables. Procedure renameCollectionTable, defined by the package XDB_ANALYZE_SCHEMA, can be used to specify meaningful names for these tables.

- Procedure renameCollectionTable takes three arguments: the name of the parent table; the name of the SQL attribute corresponding to the repeating element; and the new name for the nested table. The new table name is automatically be suffixed with _TABLE. Renaming nested tables makes it much easier to create indexes and interpret query plans.

Table LINEITEM_TABLE is used to store the collection of LineItem elements for each document in table PURCHASEORDER. Each LineItem element is stored as a separate row in this table. This makes it possible to re-write path expressions that reference the LineItem element into SQL operations on LINEITEM_TABLE.

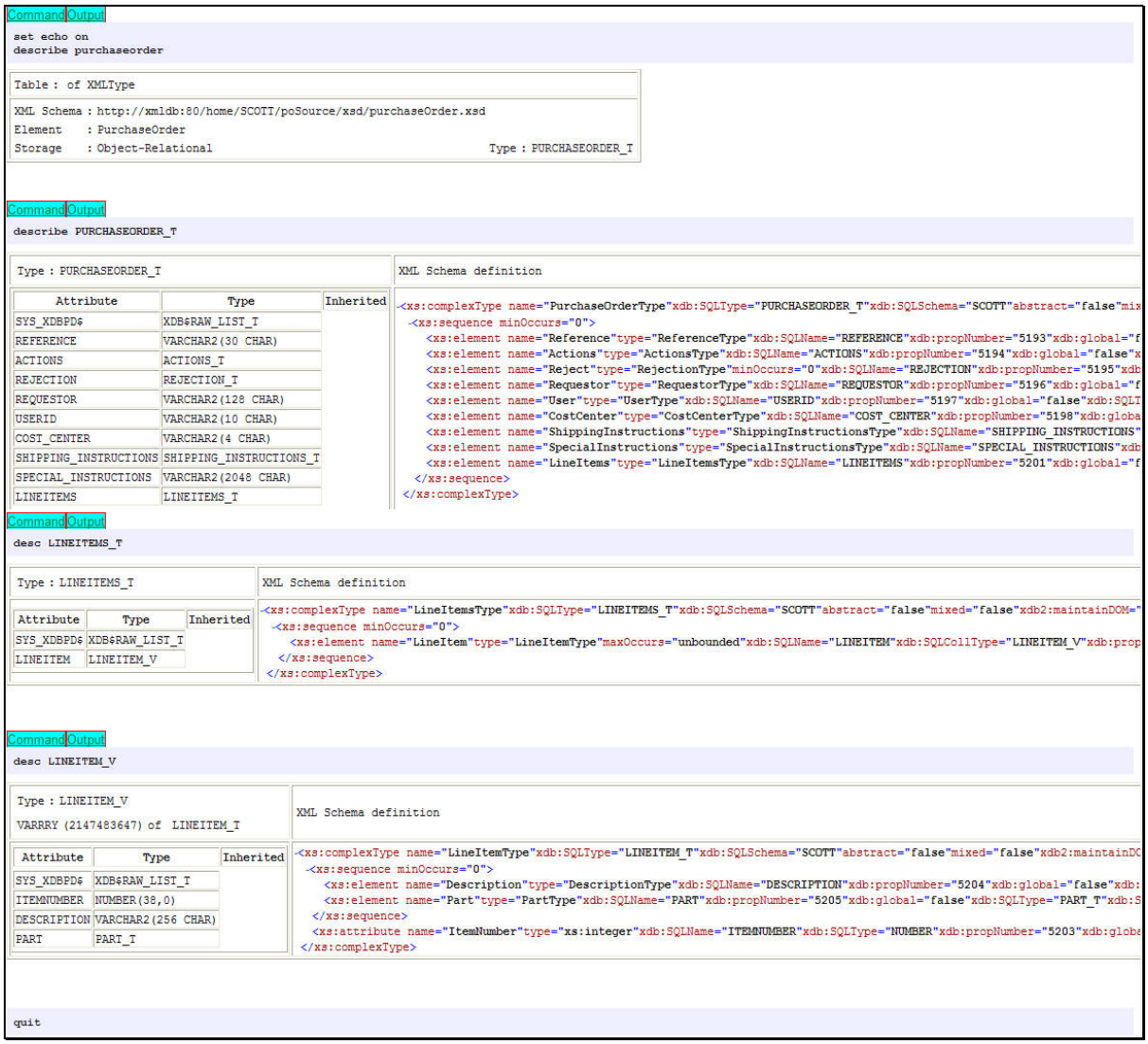
There is an implicit primary-key foreign-key relationship between table PURCHASEORDER and table LINEITEM_TABLE. The foreign-key is the hidden column NESTED_TABLE_ID. A system generated index is created on the foreign-key column.

LINEITEM_TABLE is a nested table and cannot be accessed directly using SQL DML statements. It can be accessed using DDL statements, allowing operations like create index to be performed. In Oracle Database 11g the table is a normal heap-organized table. In previous versions of Oracle XML DB, LINEITEM_TABLE was an Index Organized Table (IOT).

Table ACTION_TABLE is used to store the collection of Action elements for each document in table PURCHASEORDER.

2.2.2 Show Objects

This step describes some of the tables and types generated by registering the PurchaseOrder XML Schema. Click the icon to launch the XML DB demonstration framework and run the SQL script



```
Command Output
set echo on
describe purchaseorder

Table : of XMLType

XML Schema : http://xmldb:80/home/SCOTT/poSource/xsd/purchaseOrder.xsd
Element    : PurchaseOrder
Storage    : Object-Relational                                  Type : PURCHASEORDER_T

Command Output
describe PURCHASEORDER_T

Type : PURCHASEORDER_T

Attribute              Type                        Inherited
SYS_XDBPD$             XDB$RAW_LIST_T
REFERENCE              VARCHAR2(30 CHAR)
ACTIONS                ACTIONS_T
REJECTION              REJECTION_T
REQUESTOR              VARCHAR2(128 CHAR)
USERID                 VARCHAR2(10 CHAR)
COST_CENTER            VARCHAR2(4 CHAR)
SHIPPING_INSTRUCTIONS  SHIPPING_INSTRUCTIONS_T
SPECIAL_INSTRUCTIONS   VARCHAR2(2048 CHAR)
LINEITEMS              LINEITEMS_T

XML Schema definition

<xs:complexType name="PurchaseOrderType"xdb:SQLType="PURCHASEORDER_T"xdb:SQLSchema="SCOTT"abstract="false"mix
 <xs:sequence minOccurs="0">
   <xs:element name="Reference"type="ReferenceType"xdb:SQLName="REFERENCE"xdb:propNumber="5193"xdb:global="f
   <xs:element name="Actions"type="ActionsType"xdb:SQLName="ACTIONS"xdb:propNumber="5194"xdb:global="false"x
   <xs:element name="Reject"type="RejectionType"minOccurs="0"xdb:SQLName="REJECTION"xdb:propNumber="5195"xdb
   <xs:element name="Requestor"type="RequestorType"xdb:SQLName="REQUESTOR"xdb:propNumber="5196"xdb:global="f
   <xs:element name="User"type="UserType"xdb:SQLName="USERID"xdb:propNumber="5197"xdb:global="false"xdb:SQLT
   <xs:element name="CostCenter"type="CostCenterType"xdb:SQLName="COST_CENTER"xdb:propNumber="5198"xdb:globa
   <xs:element name="ShippingInstructions"type="ShippingInstructionsType"xdb:SQLName="SHIPPING_INSTRUCTIONS"
   <xs:element name="SpecialInstructions"type="SpecialInstructionsType"xdb:SQLName="SPECIAL_INSTRUCTIONS"xdb
   <xs:element name="LineItems"type="LineItemsType"xdb:SQLName="LINEITEMS"xdb:propNumber="5201"xdb:global="f
 </xs:sequence>
</xs:complexType>

Command Output
desc LINEITEMS_T

Type : LINEITEMS_T

Attribute   Type          Inherited
SYS_XDBPD$  XDB$RAW_LIST_T
LINEITEM    LINEITEM_V

XML Schema definition

<xs:complexType name="LineItemsType"xdb:SQLType="LINEITEMS_T"xdb:SQLSchema="SCOTT"abstract="false"mixed="false"xdb2:maintainDOM="
 <xs:sequence minOccurs="0">
   <xs:element name="LineItem"type="LineItemType"maxOccurs="unbounded"xdb:SQLName="LINEITEM"xdb:SQLCollType="LINEITEM_V"xdb:prop
 </xs:sequence>
</xs:complexType>

Command Output
desc LINEITEM_V

Type : LINEITEM_V
VARRAY (2147483647) of LINEITEM_T

Attribute    Type              Inherited
SYS_XDBPD$   XDB$RAW_LIST_T
ITEMNUMBER   NUMBER(38,0)
DESCRIPTION  VARCHAR2(256 CHAR)
PART         PART_T

XML Schema definition

<xs:complexType name="LineItemType"xdb:SQLType="LINEITEM_T"xdb:SQLSchema="SCOTT"abstract="false"mixed="false"xdb2:maintainDO
 <xs:sequence minOccurs="0">
   <xs:element name="Description"type="DescriptionType"xdb:SQLName="DESCRIPTION"xdb:propNumber="5204"xdb:global="false"xdb:
   <xs:element name="Part"type="PartType"xdb:SQLName="PART"xdb:propNumber="5205"xdb:global="false"xdb:SQLType="PART_T"xdb:S
 </xs:sequence>
 <xs:attribute name="ItemNumber"type="xs:integer"xdb:SQLName="ITEMNUMBER"xdb:SQLType="NUMBER"xdb:propNumber="5203"xdb:globa
</xs:complexType>

quit
```

Schema registration generates a SQL Type from each complexType defined by the XML Schema. It also generates a default table for each global element defined by the XML Schema. The following objects were created by registering the example Schema.

- Table PURCHASEORDER. This is an XMLType table. Each row in the table will contain an XML document. The documents must conform to the definition of the element PurchaseOrder in the XML Schema http://localhost:8080/home/SCOTT/posource/xsd/purchaseOrder.xsd. The table uses object-relational storage, based on the SQL type PURCHASEORDER_T.

- Type PURCHASEORDER_T. The definition this object is derived from the complexType PurchaseOrderType. The attributes of the SQL type correspond to the elements and attribute defined by the complexType. Internally a PurchaseOrder document will stored as an instance of PURCHASEORDER_T.

- Type LINEITEMS_T. The definition this object is derived from the complexType LineItemsType. LineItemsType is defined as being a collection of LineItem elements. Each LineItem is an instance of the complexType LineItemType. Internally any element based on complexType LineItemsType will be stored as an instance of LINEITEMS_T.

- Type LINEITEM_T. The definition this object is derived from the complexType LineItemType. Internally any element based on complexType LineItemType will be stored as an instance of LINEITEM_T.

- Varray LINEITEM_V. This is a collection of LINEITEM_T objects. The varray type is required since element LineItem is allowed to occur multiple times within a LineItems element. Internally collections of LINEITEM_T objects will be stored as instances of LINEITEM_V.

- All of the SQL Types contain a SYS_XDBPD$ attribute. Oracle XML DB uses this attribute to track instance level meta data that allows it to provide DOM fidelity for the documents it manages. DOM Fidelity ensures that all the significant information in the document is maintained during the conversion between the internal and external representations of the XML.

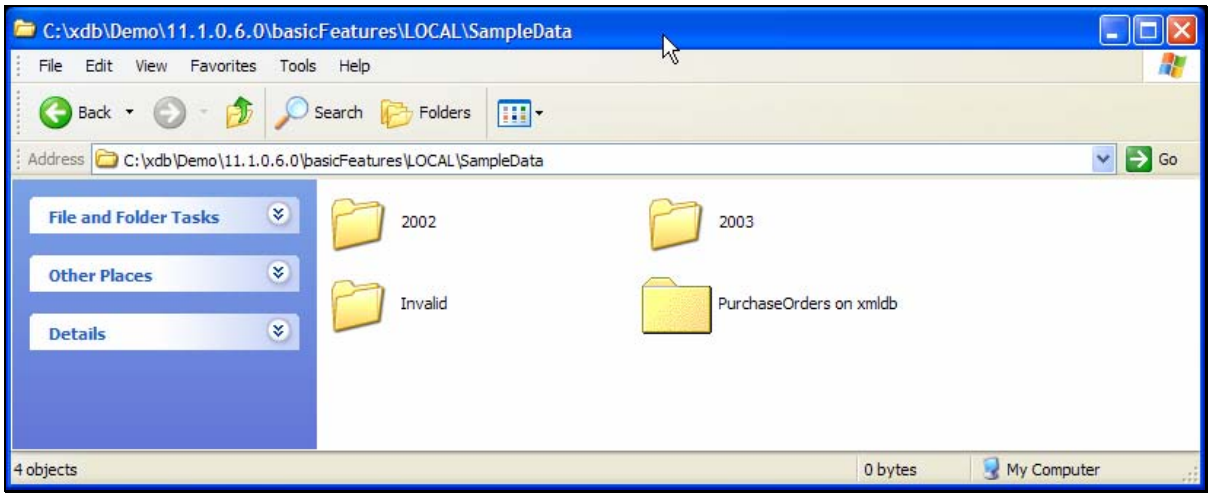
Maintaining DOM Fidelity can result in significant storage and processing overheads.

- It is common for DOM Fidelity to be required by document-centric applications where white-space is significant, and the documents are likely to contain processing instructions, comments and mixed-text.

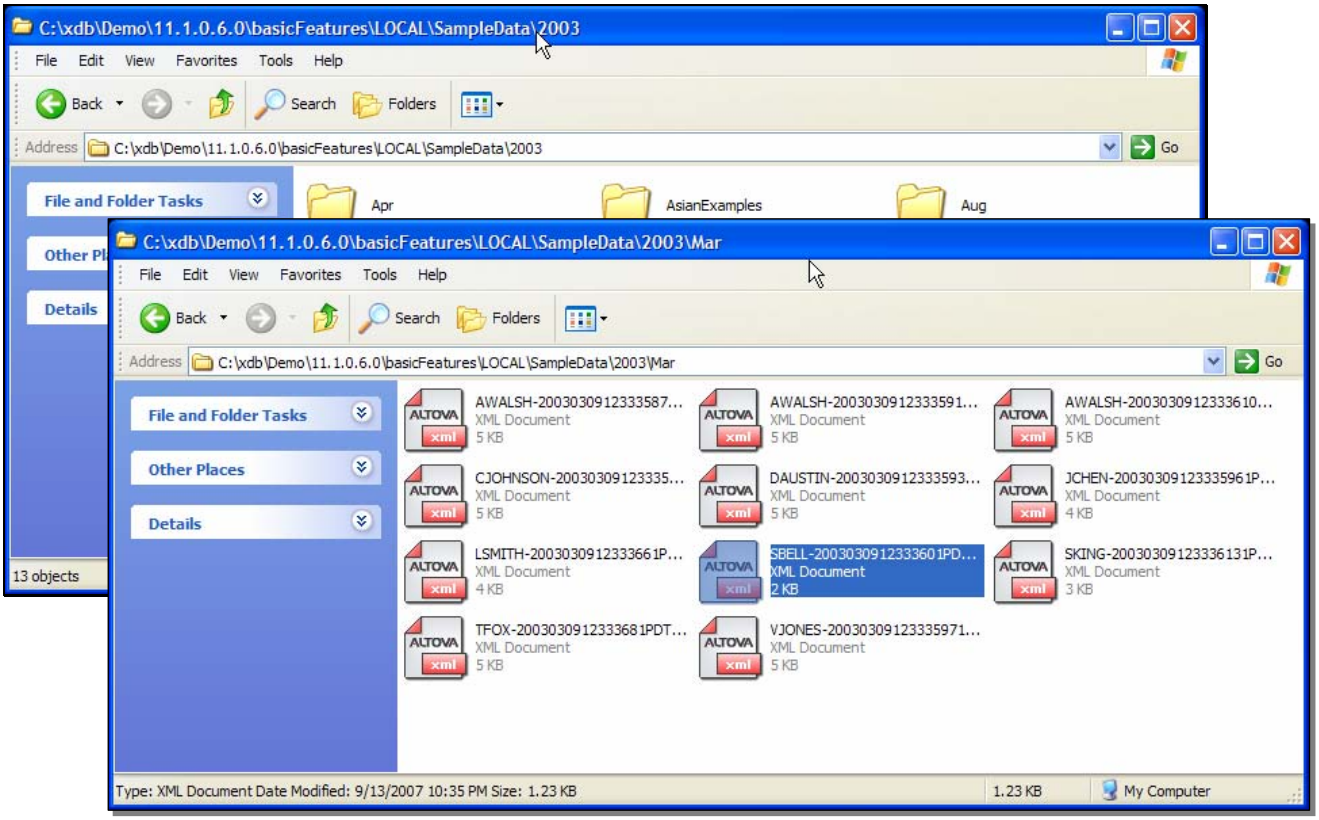
- It is uncommon for DOM Fidelity to be required by data-centric applications, where all of the information in the document is represented as elements or attributes. For these applications, disabling DOM Fidelity by adding the annotation xdb:maintainDOM=”false” to each complexType, leads to significant reductions in storage and improvements in throughput.

3.1.0 Load Sample Data

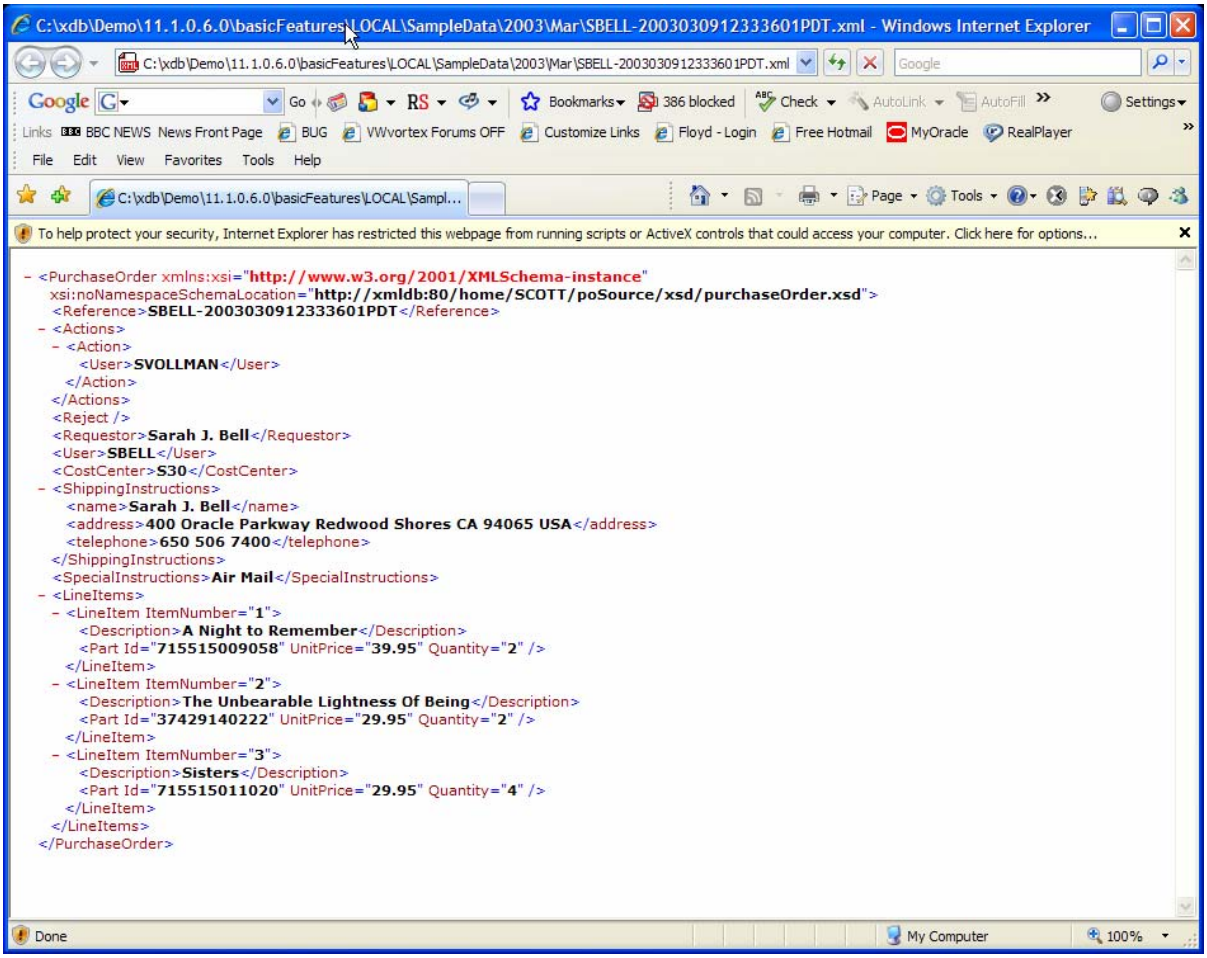
This step uses Microsoft Windows Explorer to load XML files into table PurchaseOrder. Right click the icon and select explore. This will open a new window containing the local SampleData.



The window contains four items, a shortcut called "PurchaseOrders on ***hostname***" and folders called 2002, 2003 and Invalid. This shortcut is a link to PurchaseOrders folder in the demonstration user's home folder in the Oracle XML DB repository. Double-click folder 2003. Double-click folder Mar.



Double click SBELL-20030309123336O1PDT.xml. This will launch Internet Explorer and display the document. If some other application is launched, use the Open With option of Windows Explorer's right mouse button menu to open the file with Internet Explorer.



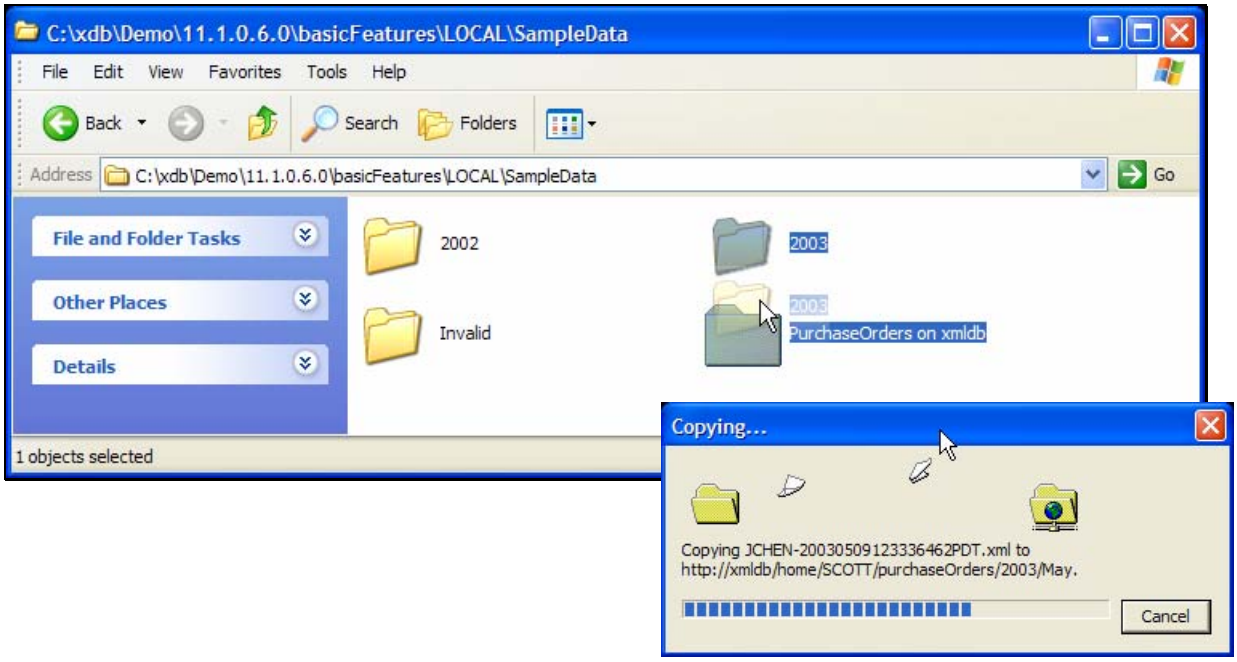
This document is valid instance of the XML Schema. The file contains a noNamespaceSchemaLocation attribute that correctly identifies it as an instance of the class defined by the XML Schema. Close the browser and navigate back to the folder SampleData.

Oracle XML DB provides multiple ways of storing XML documents in the database.

From SQL and PL/SQL normal insert statements can be used The XML can be supplied as a constant or bind variable. Before the XML can be stored as an XMLType is must first be converted from the source form into an XMLType instance using one of the XMLType constructors. Variants of the XMLType constructor allow the XML to be supplied as a VARCHAR2 or CLOB. If the XML is contained in a file that is accessible from the database's local file system a BFILE can be used to construct the XMLType directly from the contents of the file. Oracle XML DB also supports the SQL/XML standard operator XMLParse.

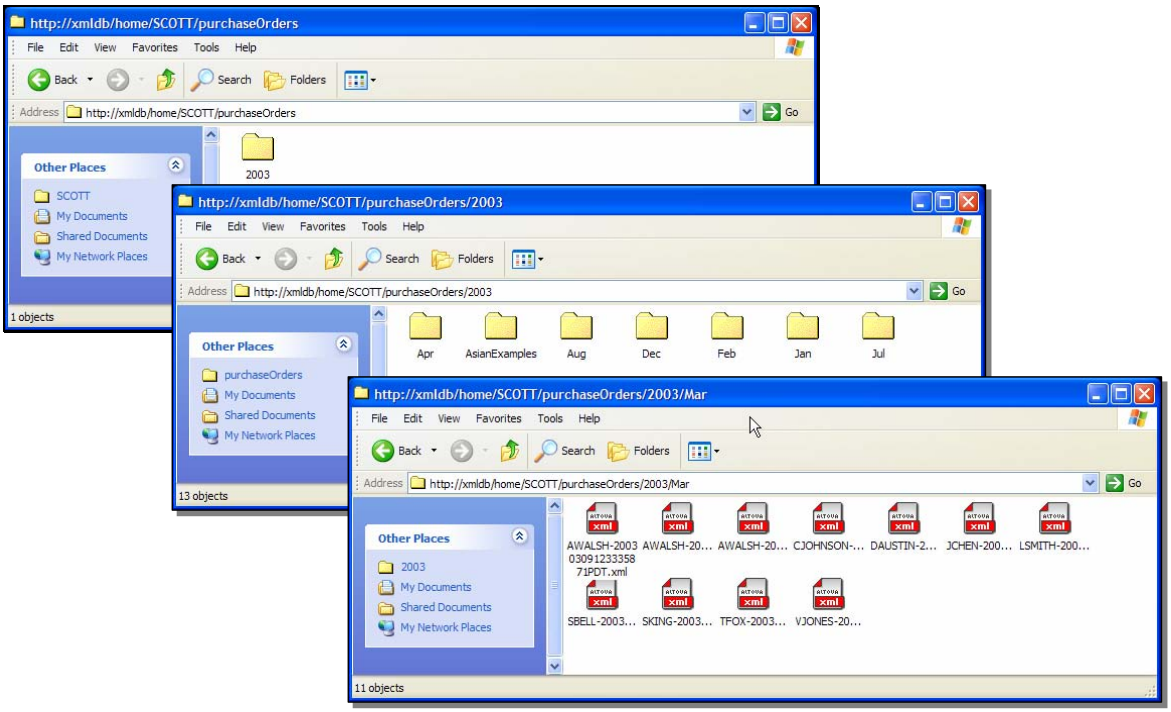
The Oracle XML DB repository can be used to load XML files into an XMLType table. The XML must be schema-based and it can only be loaded into the default table defined by the XML Schema. Documents can be loaded using FTP, HTTP or WebDAV or by calling procedure createResource, defined by package DBMS_XDB. When a schema-based XML document, associated with a known XML Schema, is loaded into the Oracle XML DB repository XML DB automatically stores the content of the document in the default table defined by the XML Schema.

Drag the folder 2003 on to the PurchaseOrders shortcut. When prompted, enter the demonstration user's username and password and click OK.



Windows Explorer will copy the contents of the folder into the Oracle XML DB repository. Since the documents in this folder are instances of the PurchaseOrder XML Schema, the content of each document becomes a row in table PURCHASEORDER. Since Microsoft Windows and Oracle XML DB support the WebDAV protocol no software needs to be installed on the desktop to enable this feature.

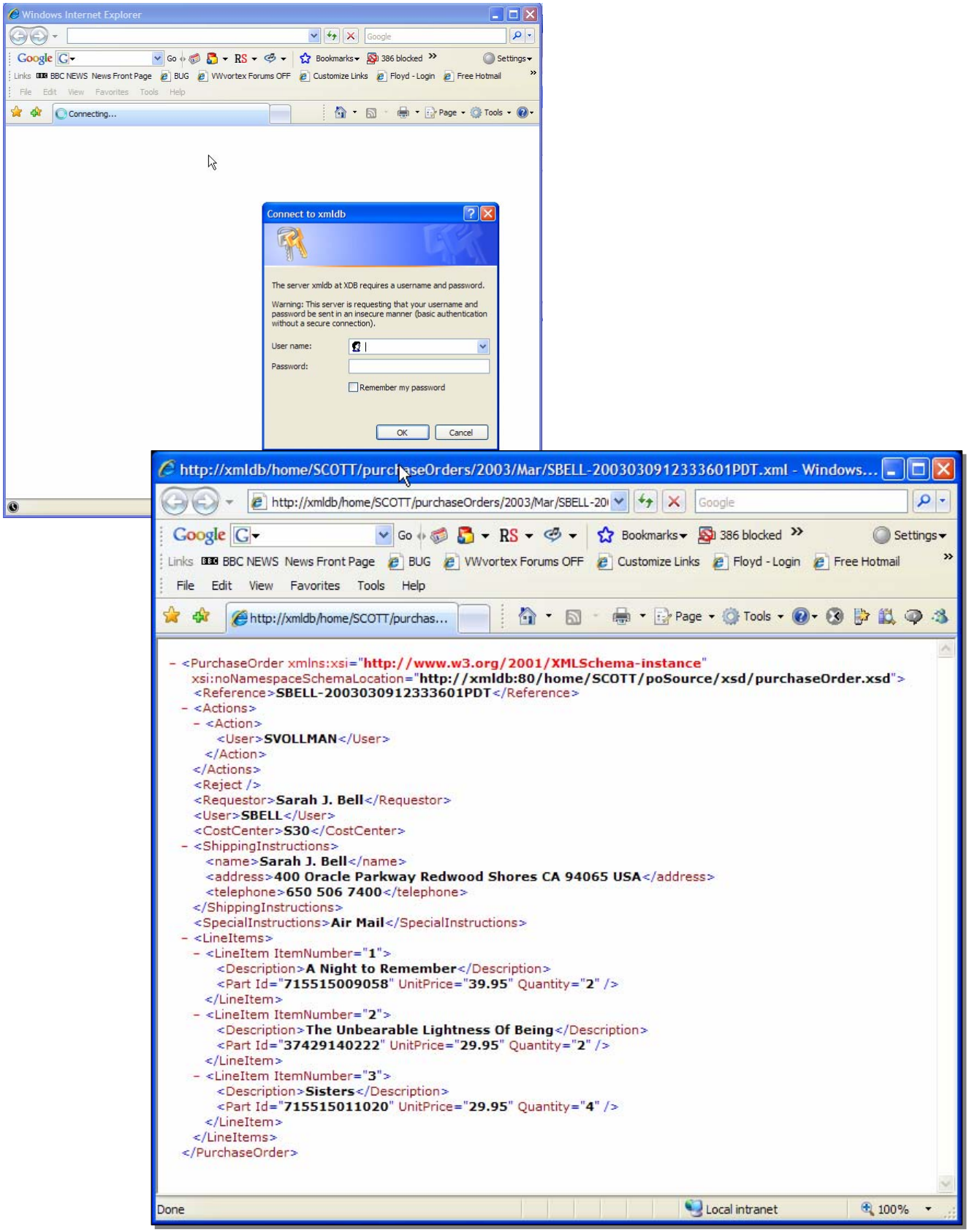
When the copy is complete right click the PurchaseOrders shortcut and select explore. This will open a new window containing the Oracle XML DB folder /home/***USER***/PurchaseOrders. The content of this window comes from the Oracle XML DB repository. The window will contain a single folder called 2003. Double click to open folder 2003. This folder will contain folders Jan thru Dec. Double click folder Mar. The folder will contain the same set of XML documents as the local folder SampleData\2003\Mar. Windows Explorer has used the WebDAV protocol to replicate the structure and content of the local folder 2003 in the Oracle XML DB repository.



Double click SBELL-20030309123336O1PDT.xml.

This launches Internet Explorer and displays the document. The document is be fetched from the Oracle XML DB repository using an HTTP GET. If the browser does not own an active, authenticated HTTP connection to the repository, Internet Explorer will prompt for a username and password before accessing the document. If this happens enter the demonstration user's username and password and click OK.

If the URL in the browser appears to reference the local file system this means the document has been fetched from browsers cache. If this happens, clear the browser cache and re-try the operation.


Windows Internet Explorer
Connecting...
Connect to xmldb
The server xmldb at XDB requires a username and password.
Warning: This server is requesting that your username and password be sent in an insecure manner (basic authentication without a secure connection).
User name:
Password:
Remember my password
OK
Cancel
http://xmldb/home/SCOTT/purchaseOrders/2003/Mar/SBELL-20030309123333601PDT.xml - Windows...
http://xmldb/home/SCOTT/purchaseOrders/2003/Mar/SBELL-20
http://xmldb/home/SCOTT/purchas...
- <PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://xmldb:80/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
<Reference>SBELL-20030309123333601PDT</Reference>
- <Actions>
- <Action>
<User>SVOLLMAN</User>
</Action>
</Actions>
<Reject />
<Requestor>Sarah J. Bell</Requestor>
<User>SBELL</User>
<CostCenter>S30</CostCenter>
- <ShippingInstructions>
<name>Sarah J. Bell</name>
<address>400 Oracle Parkway Redwood Shores CA 94065 USA</address>
<telephone>650 506 7400</telephone>
</ShippingInstructions>
<SpecialInstructions>Air Mail</SpecialInstructions>
- <LineItems>
- <LineItem ItemNumber="1">
<Description>A Night to Remember</Description>
<Part Id="715515009058" UnitPrice="39.95" Quantity="2" />
</LineItem>
- <LineItem ItemNumber="2">
<Description>The Unbearable Lightness Of Being</Description>
<Part Id="37429140222" UnitPrice="29.95" Quantity="2" />
</LineItem>
- <LineItem ItemNumber="3">
<Description>Sisters</Description>
<Part Id="715515011020" UnitPrice="29.95" Quantity="4" />
</LineItem>
</LineItems>
</PurchaseOrder>
Done
Local intranet
100%

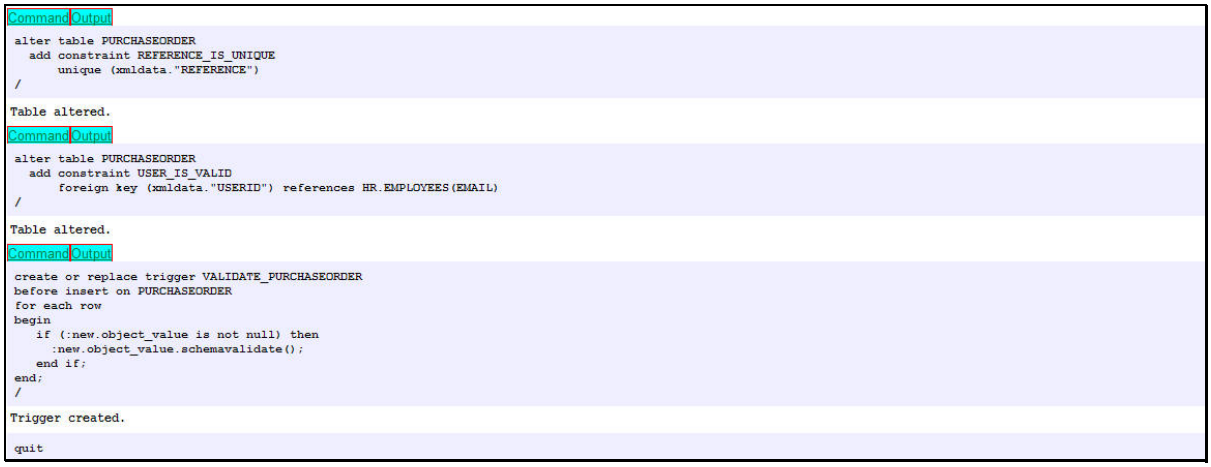
3.2.0 Add Constraints

The XML Schema specification is an extremely powerful standard. It allows XML Schema designers to define very complex rules about what is valid in an instance document. However there are some fairly common data management concepts that cannot be easily expressed using the current version of the XML Schema standard. These include the ability to define SQL style constraints and referential integrity rules.

For instance, in XML Schema it very easy to specify that a value must be unique within a document but quite difficult to express that a value must be unique across a collection of documents (a UNIQUE constraint). Schema makes it easy to express that the value of an element or attribute must match the value of some other attribute or element within the document, but difficult to express the fact the value must match a value found in a some other data source (a FOREIGN KEY constraint).

Oracle XML DB allows SQL type integrity rules to be applied to XML documents. Simple rules, like enforcing uniqueness and foreign key relationships, are applied by defining SQL constraints on the tables that manage the XML data. More complex rules are enforced using database triggers.

This step uses SQL constraints to impose SQL-style referential integrity rules and full schema validation on documents stored in the Oracle XML DB repository. Click the icon to launch the XML DB demonstration framework and run the SQL script



Command Output

```
alter table PURCHASEORDER
  add constraint REFERENCE_IS_UNIQUE
      unique (xmldata."REFERENCE")
/

Table altered.
```

Command Output

```
alter table PURCHASEORDER
  add constraint USER_IS_VALID
      foreign key (xmldata."USERID") references HR.EMPLOYEES(EMAIL)
/

Table altered.
```

Command Output

```
create or replace trigger VALIDATE_PURCHASEORDER
before insert on PURCHASEORDER
for each row
begin
   if (:new.object_value is not null) then
      :new.object_value.schemavalidate();
   end if;
end;
/

Trigger created.

quit
```

The script enforces the following rules on PurchaseOrder documents.

- Constraint RERENCE_IS_UNIQUE enforces the rule that the value of the text node for element /PurchaseOrder/Reference must be unique for all documents stored in the PURCHASEORDER table

- Constraint USER_IS_VALID enforces the rule that the value of the text node for element /PurchaseOrder/User must match a value in the EMAIL column of table HR.EMPLOYEES.

- Trigger VALIDATE_PURCHASEORDER forces a full schema validation to be performed for each document stored in table PURCHASEORDER.

Unique and Foreign Key constraints are defined in terms of the underlying object model when object-relational storage is used. Object-relational SQL has to be used to identify the target of the constraint. The virtual column name object_value is used to reference the content of an XMLType table.

In Oracle Database 11g, virtual columns can be used to define constrains for binary XML storage.

Schema Validation

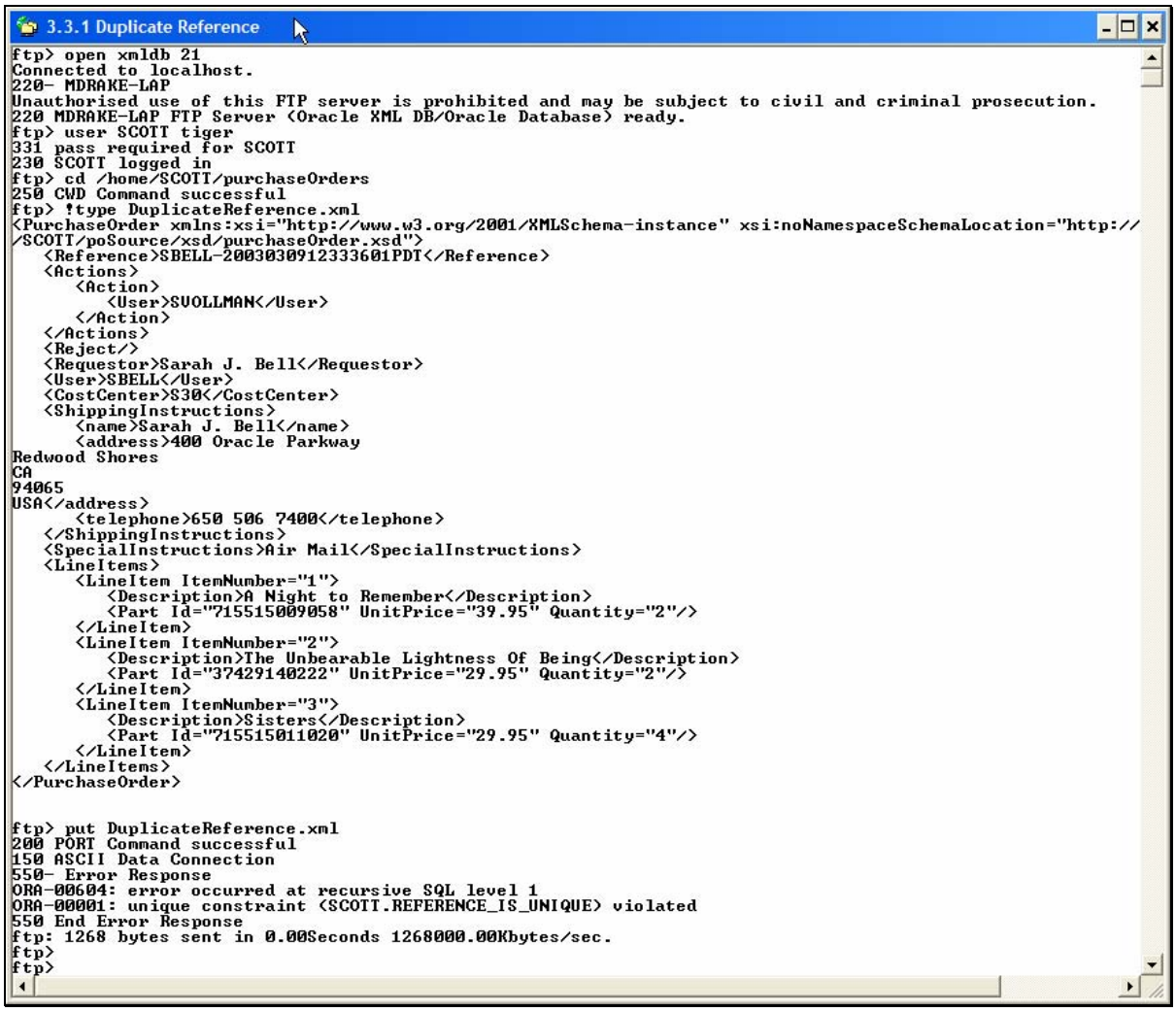
With object-relational storage Oracle XML DB performs a simple 'lax' validation of the incoming XML Documents before they are stored. This validation ensures that all mandatory information is present and there are no unexpected elements or attributes in the document. It does not check all of the rules defined by the XML Schema. Full schema validation is performed by invoking the schemaValidate method on the value of the XMLType.

Full XML Schema validation is an expensive in terms of memory and cpu. Making schema-validation optional allows database administrators and application designers to choose whether or not the overhead of performing a full schema-validation inside the database is appropriate. If the application inserting data can be trusted to ensure that the data is valid, the overhead of performing a full schema-validation can be avoided. If it not possible to trust the applications inserting the data, then a simple trigger is all that is required to enforce schema-validation. One common use-case for having database enforced schema-validation is when the protocols are used to load content into the database. In this scenario there is no way to know whether or not the application loading the data has checked that the data is schema-valid.

Binary XML always perfumes a full-schema validation. It uses a very efficient streaming validator to minimize the overhead associated with XML Schema validation. Since binary XML relies on the XML Schema for type-checking etc, the additional overhead imposed by full schema-validation is minimal.

3.3.1 Duplicate Reference

This step shows a constraint ensuring that the value of element /PurchaseOrder/Reference is unique for all the documents stored in table PURCHASEORDER. The constraint is enforced when the document is inserted into the XMLType table. It does not matter whether the insert is done directly from SQL or indirectly via the Oracle XML DB repository. Double click the icon to launch FTP and load a PurchaseOrder document into the XML DB repository.



The XML document is schema-valid, however the value of the node /PuchaseOrder/Reference/text(), SBELL-2003030912333601PDT.xml , is a duplicate of one of the documents already in table PURCHASEORDER. This causes the unique constraint REFERENCE_IS_UNIQUE to be violated.

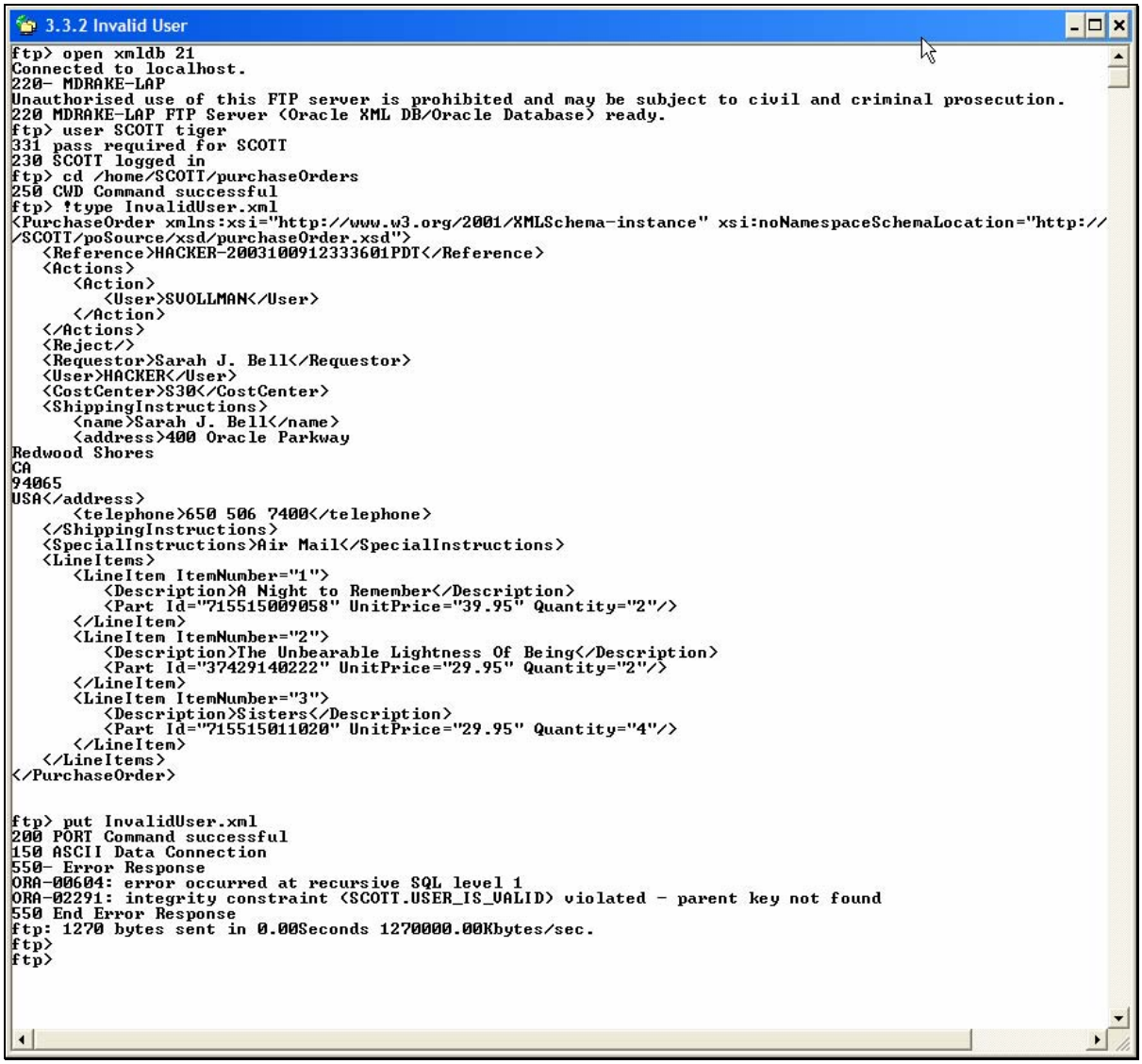
The upload operation is aborted, the ftp put operation fails, and the following error is reported:

ORA-00604: error occurred at recursive SQL level 1
ORA-00001: unique constraint (SCOTT.REFERENCE_IS_UNIQUE) violated

Type bye or quit to exit the FTP session

3.3.2 Invalid User

This step shows a foreign-key constraint ensuring that the value of element /PurchaseOrder/User can be found in the EMAIL column of table HR.EMPLOYEES. The constraint is enforced when the document is inserted into the XMLType table. It does not matter whether the insert is done directly from SQL or indirectly via the Oracle XML DB repository. Double click the icon to launch FTP and load a PurchaseOrder document into the XML DB repository.



The XML document is schema-valid, however the value of elemente /PuchaseOrder/User, HACKER, does not match any of the values contained in the EMAIL column of table HR.EMPLOYEES. This causes the foreign-key constraint USER_IS_VALID to be violated.

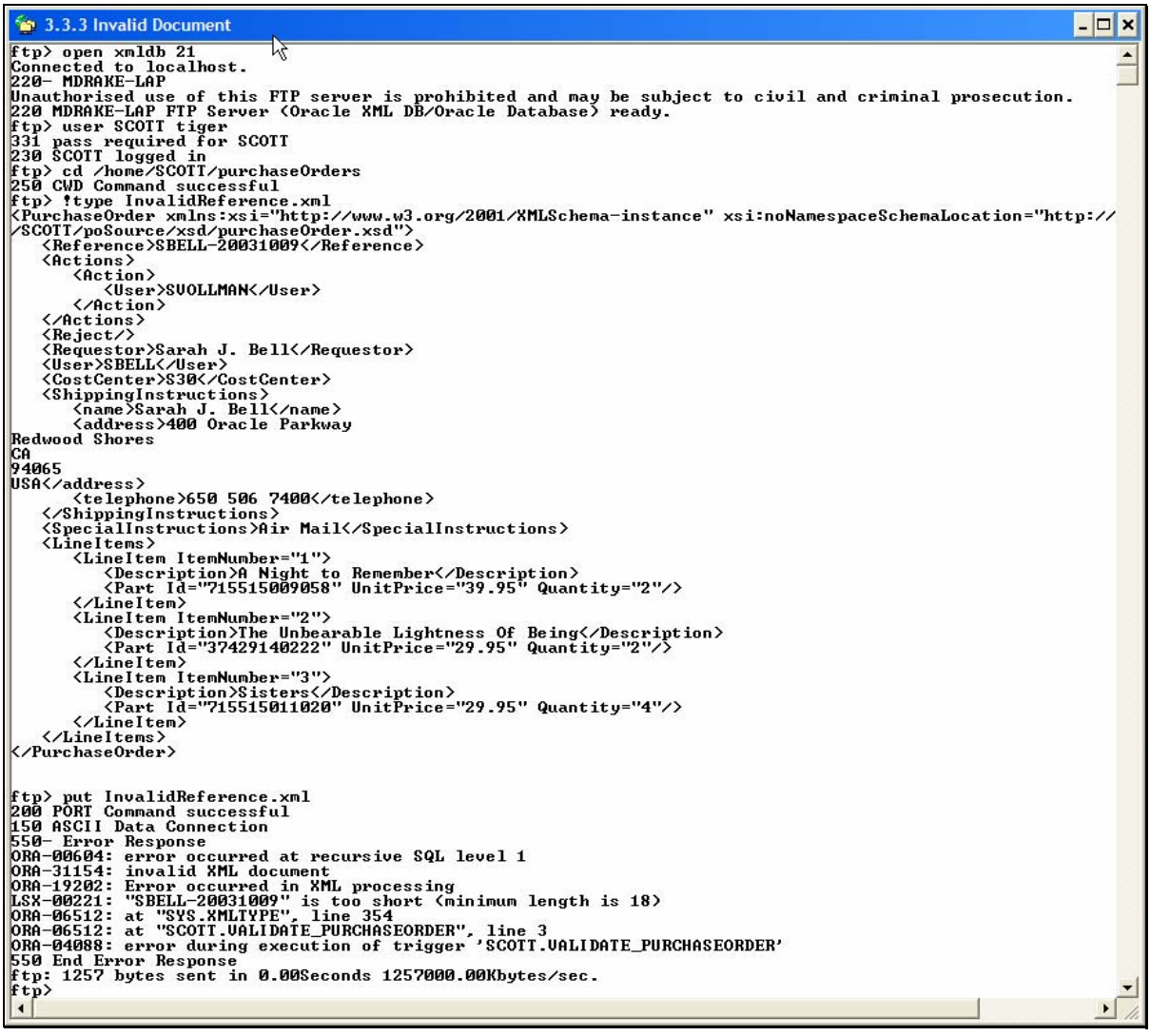
The upload operation is aborted, the ftp put operation fails, and the following error is reported:

ORA-00604: error occurred at recursive SQL level 1
ORA-02291: integrity constraint (SCOTT.USER_IS_VALID) violated - parent key not found

Type bye or quit to exit the FTP session

3.3.3 Invalid Document

This step shows a database trigger ensuring that all documents stored in table PURCHASEODER are fully schema-valid. The trigger is executed when the document is inserted into the table. It does not matter whether the insert is done directly from SQL or indirectly via the Oracle XML DB repository.



The PurchaseOrder XML Schema defines that the minimum length for the content of the Reference element is 18 characters. In the document being uploaded the length of the Reference element is only 15 characters long. Consequently the document is not schema-valid. The full schema-validation performed by trigger VALIDATE_PURCHASEORDER detects that the document is not schema-valid and throws an exception. Since the exception is not caught the upload operation is aborted, the ftp put operation fails, and the following error is reported:

ORA-00604: error occurred at recursive SQL level 1
ORA-31154: invalid XML document
ORA-19202: Error occurred in XML processing
LSX-00221: "ADAMS-20011127PST" is too short (minimum length is 18)
ORA-06512: at "SYS.XMLTYPE", line 0
ORA-06512: at "SCOTT.VALIDATE_PURCHASEORDER", line 5
ORA-04088: error during execution of trigger 'SCOTT.VALIDATE_PURCHASEORDER'

Since the document satisfies the lax-validation rules that are enforced when a document is inserted into object-relational storage, it would be possible to store it in table PURCHASEORDER despite the fact that it is not schema-valid if the trigger were not present

Whenever an error occurs while using protocols to upload a document into the Oracle XML DB repository, a complete SQL error stack is returned to the client. How the error is interpreted and reported is determined by the error handling of the client application. Some clients, such as the command line FTP utility, report the error returned by Oracle XML DB. Others, such as Windows Explorer, swallow the error reported by Oracle XML DB and simply report a generic error message.

Choosing to use Oracle XML DB to store and manage XML makes it possible to combine the flexibility of XML with the power of SQL and the Reliability, Availability, Scalability and Security of the Oracle Database 11g. Click the icon to launch the XML DB demonstration framework and run the SQL script

Type bye or quit to exit the FTP session

4.1.1 Simple SQL Queries (1)

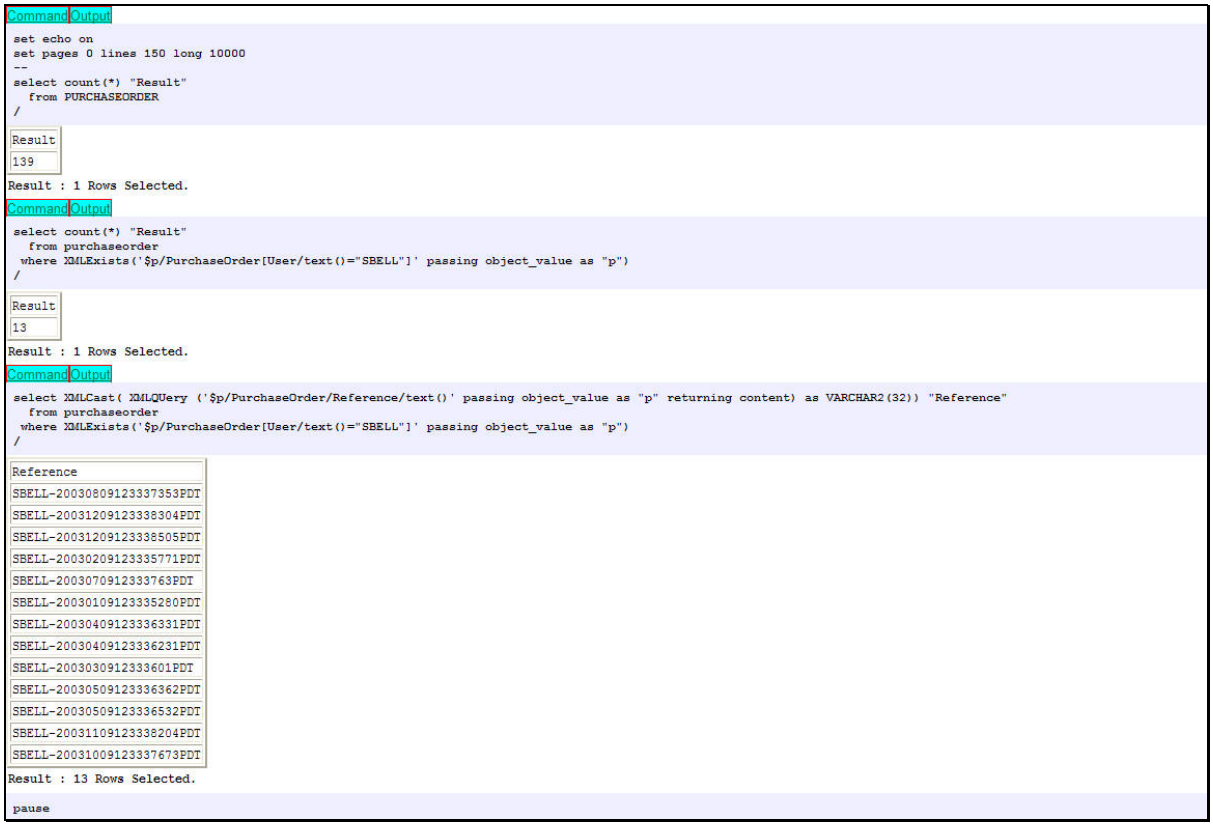
This step shows using simple Path expressions to query XML content. The Path expressions reference nodes that only occur once in each document and include very basic predicates.

The SQL/XML standard defines a set of operators that allow XQuery operations to be performed on XML content as part of a SQL statement. XQuery is the W3C standard for querying and accessing XML. XQuery flower and path expressions are familiar to XML programmers and authors. Oracle XML DB is able to execute XQuery Flower and Path expressions very efficiently.

Operator XMLQuery takes an XQuery expression as a string literal, an optional context item, and other bind variables and returns the result of evaluating the XQuery expression using these input values. The result is returned as an XMLType, even if the XQuery evaluates to a sequence containing a single scalar value.

Operator XMLExists evaluates an XQuery expression against an XML document. It returns true when the document contains a node which matches the XQuery expression, false otherwise It is typically used in the where clause of a SELECT statement to select which rows are processed by rest of the statement.

Click the icon to launch the XML DB demonstration framework and run the SQL script



Command Output

```
set echo on
set pages 0 lines 150 long 10000
--
select count(*) "Result"
  from PURCHASEORDER
/
```

Result
139

Result : 1 Rows Selected.

Command Output

```
select count(*) "Result"
  from purchaseorder
 where XMLExists('$p/PurchaseOrder[User/text()="SBELL"]' passing object_value as "p")
/
```

Result
13

Result : 1 Rows Selected.

Command Output

```
select XMLCast( XMLQUery ('$p/PurchaseOrder/Reference/text()' passing object_value as "p" returning content) as VARCHAR2(32)) "Reference"
  from purchaseorder
 where XMLExists('$p/PurchaseOrder[User/text()="SBELL"]' passing object_value as "p")
/
```

Reference
SBELL-20030809123337353PDT
SBELL-20031209123338304PDT
SBELL-20031209123338505PDT
SBELL-20030209123335771PDT
SBELL-2003070912333763PDT
SBELL-20030109123335280PDT
SBELL-20030409123336331PDT
SBELL-20030409123336231PDT
SBELL-20030309123333601PDT
SBELL-20030509123336362PDT
SBELL-20030509123336532PDT
SBELL-20031109123338204PDT
SBELL-20031009123337673PDT

Result : 13 Rows Selected.

```
pause
```

The first query simply counts the number of rows in table PURCHASEORDER. This shows that a row was created for each of the PurchaseOrder documents loaded into the repository.

- The query counts the total number of documents in table PURCHASEORDER. The rows in table PURCHASEORDER were created when the PurchaseOrder documents were loaded into the XML DB repository. Each row in table PURCHASEORDER corresponds to one PurchaseOrder document.

- Counting the number of rows in table PURCHASEORDER gives the total number of PurchaseOrder documents stored in the repository.

The second query uses a path expression containing a simple predicate to access a subset of the documents in a table. The predicate is on a node that can only occur once in each document. The query is "Find the number of documents where the user is **SBELL**".

- The query counts the number of documents in table PURCHASEORDER where the text node belonging to element /PurchaseOrder/User contains the value **SBELL**.

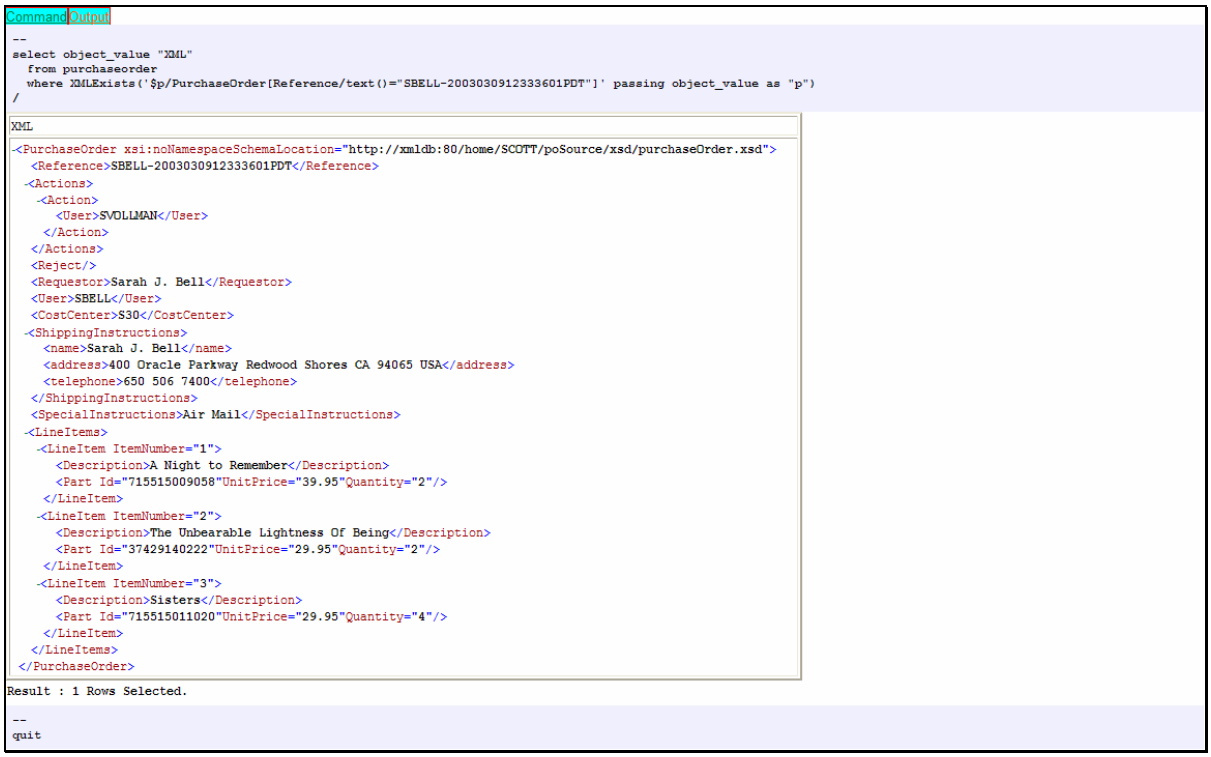
- XMLExists identifies which rows contain a node that satisfies the path expression.

The third query performs a fragment extraction on a subset of the documents in a table. The query is "Get the reference of all documents where the user is **SBELL**".

- The query returns element /PurchaseOrder/Reference for each document in table PURCHASEORDER where the value of element /PurchaseOrder/User is **SBELL**.

- XMLExists identifies which rows contain a node that satisfies the path expression.

Click continue to execute the remaining queries



The last query returns the content of an XML document. The query is "Get the document where the reference is **SBELL-20030309123336O1PDT**".

- The query returns the XML document where element /PurchaseOrder/Reference contains the value **SBELL-20030309123336O1PDT**. The pseudo column object value provides access to the entire document. XMLExists is used to determine which document to return. Since element reference is subject to a unique constraint at most one document can match the supplied value.

4.1.2 Simple SQL Queries (2)

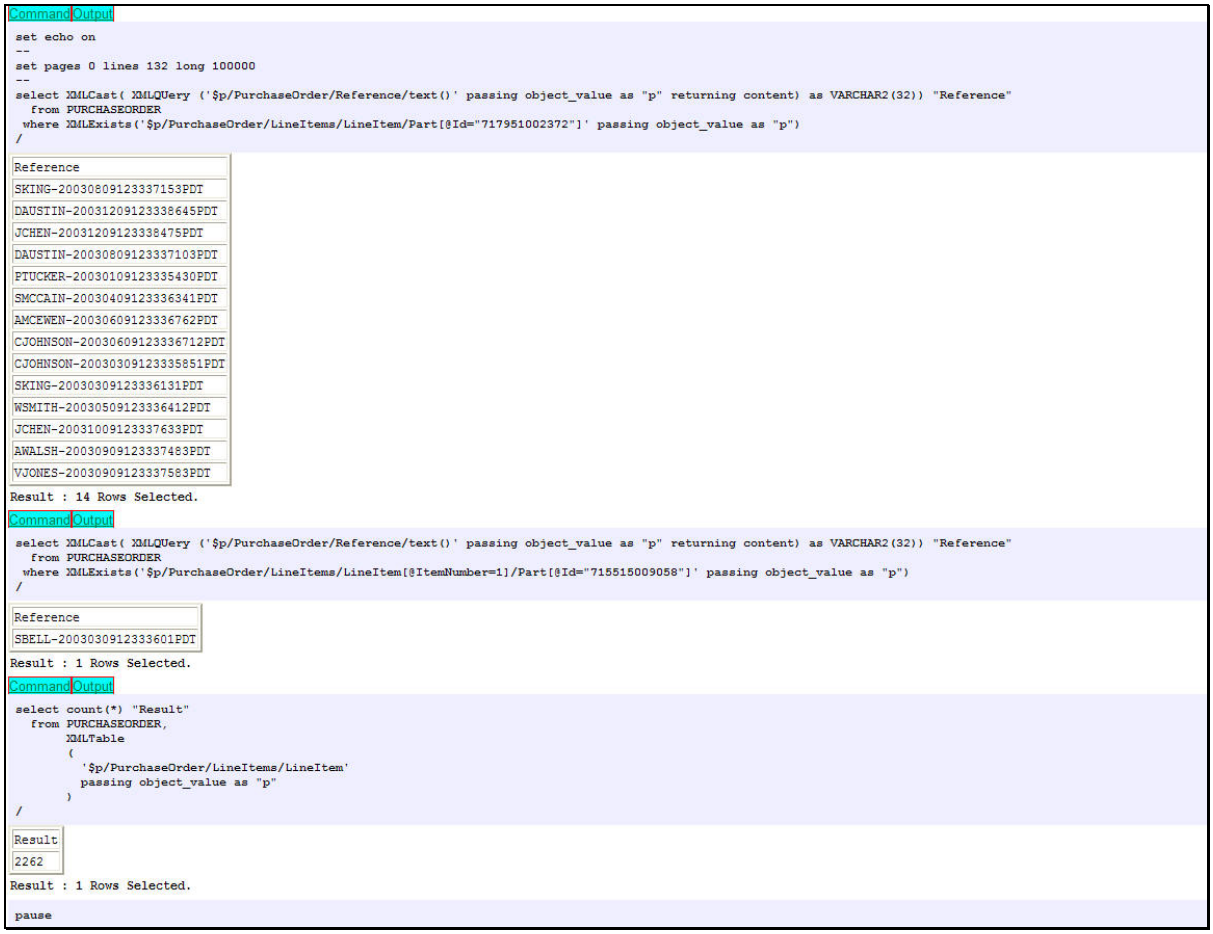
This step shows using slightly more complex Path expressions. The Path expressions reference nodes that occur more than once in each document and include some complex predicates.

Oracle XML DB is able to quickly and efficiently evaluate complex XQuery path expressions. The path expressions can contain multiple predicates and reference nodes that occur multiple times within a document.

XQuery uses the XML Schema type model. Operator XMLCast takes the result of an XQuery expression and converts it from an XQuery data type into a SQL data type.

Operator XMLTable maps the result of an XQuery evaluation into relational rows and columns. You can query the result returned by the function as a virtual relational table using SQL.

Click the icon to launch the XML DB demonstration framework and run the SQL script



Command Output

```
set echo on
--
set pages 0 lines 132 long 100000
--
select XMLCast( XMLQUery ('$p/PurchaseOrder/Reference/text()' passing object_value as "p" returning content) as VARCHAR2(32)) "Reference"
  from PURCHASEORDER
 where XMLExists('$p/PurchaseOrder/LineItems/LineItem/Part[@Id="717951002372"]' passing object_value as "p")
/
```

Reference
SKING-20030809123337153PDT
DAUSTIN-20031209123338645PDT
JCHEN-20031209123338475PDT
DAUSTIN-20030809123337103PDT
PTUCKER-20030109123335430PDT
SMCCAIN-20030409123336341PDT
AMCEWEN-20030609123336762PDT
CJOHNSON-20030609123336712PDT
CJOHNSON-20030309123335851PDT
SKING-20030309123336131PDT
WSMITH-20030509123336412PDT
JCHEN-20031009123337633PDT
AWALSH-20030909123337483PDT
VJONES-20030909123337583PDT

```
Result : 14 Rows Selected.
```

Command Output

```
select XMLCast( XMLQUery ('$p/PurchaseOrder/Reference/text()' passing object_value as "p" returning content) as VARCHAR2(32)) "Reference"
  from PURCHASEORDER
 where XMLExists('$p/PurchaseOrder/LineItems/LineItem[@ItemNumber=1]/Part[@Id="715515009058"]' passing object_value as "p")
/
```

Reference
SBELL-20030309123336012PDT

```
Result : 1 Rows Selected.
```

Command Output

```
select count(*) "Result"
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
       )
/
```

Result
2262

```
Result : 1 Rows Selected.

pause
```

The first query uses a path expression containing a simple predicate to access a subset of the documents in a table. The predicate is on a node that can occur more than once in each document. The query is "Get the reference of all documents that contain part **717951002372**".

- The path expression does not explicitly state which instance of the node to process so the predicate will be evaluated for all nodes that match the path expression.

- The query returns the value of element /PurchaseOrder/Reference where at least one instance of attribute /PurchaseOrder/LinteItems/LineItem/Part/@Id contains the value **717951002372**.

- XMLExists identifies which rows contain a node that matches the supplied path expression.

- XMLCast converts the contents of XQuery sequence generated by the XMLQuery operator from the XML Schema data type xs:string into the SQL data type VARCHAR2.

The second query uses a path expression containing multiple predicate to access a subset of the documents stored in a table. The predicates are on nodes that can occur more than once in the document. The nodes occur at different levels of the document. The query is "Get the reference of all documents where the first line item is part **717951002372**".

- The path expression does not explicitly state which instance of the node to process so the predicate will be evaluated for all nodes that match the path expression.

- The query returns the value of element /PurchaseOrder/Reference for each document where at least one instance of element /PurchaseOrder/LinteItems/LineItem contains attribute @ItemNumber with a value of 1 and the attribute Part/@Id with a of value 717951002372.

- XMLExists identifies which rows contain a node that matches the supplied path expression.

- XMLCast converts the contents of XQuery sequence generated by the XMLQuery operator from the XML Schema data type xs:string into the SQL data type VARCHAR2..

The third query uses a path expression to count the number of occurrences of a repeating element for all the documents stored in a table. The query is "Find the total number of line items for all documents".

- The query counts the number of occurrences of element LineItem for all documents in table PURCHASEORDER.

- XMLTable processes each of the documents in the table, generating one row for each instance of element LineItem.

- Counting the number of rows generated by XMLTable gives the total number of LineItem elements in the table.

Click continue to execute the remaining queries

```
Command Output
--
select l.*
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
         columns
         DESCRIPTION path 'Description'
       ) l
 where XMLExists('$p/PurchaseOrder[Reference="SBELL-20030309123336O1PDT"]' passing object_value as "p")
/

DESCRIPTION
A Night to Remember
The Unbearable Lightness Of Being
Sisters

Result : 3 Rows Selected.

--
quit
```

The last query uses a path expression to expose a collection of nodes from a set of documents as a virtual table. This table can be operated on using standard SQL statements. The query is "Get the description for all line items where the reference is **SBELL-20030309123336O1PDT**".

- The query generates rows from XML documents where element /PurchaseOrder/Reference contains the value **SBELL-20030309123336O1PDT**.
- The query creates a table with a single column, DESCRIPTION. The table will contain one row for each node that matches the path expression /PurchaseOrder/LineItems/LineItem.
- Column DESCRIPTION will contain the value of element Description.
- XMLExists identifies which rows are input to the XMLTable operator

4.1.3 Japanese Query

This step shows using a simple Path expression to query XML documents that contain multi-byte content. The Path expression contains a predicate with Japanese characters. The document returned also contains Japanese content.

Click the icon to launch the XML DB demonstration framework and run the SQL script

```
Command Output
set echo on
--
select REFERENCE, DESCRIPTION
  from PURCHASEORDER,
       xmlTable
       (
         '/PurchaseOrder'
         passing object_value
         columns
         REFERENCE           path 'Reference',
         LINEITEMS  xmltype  path 'LineItems'
       ),
       xmlTable
       (
          '/LineItems/LineItem'
          passing LINEITEMS
          columns
          DESCRIPTION         path 'Description'
       )
 where XMLExists('/PurchaseOrder/LineItems/LineItem[Description="生きる (Ikiru)"]' passing object_value)
/
```

REFERENCE	DESCRIPTION
CDAVIES-20031008092436 18PDT	東京物語 (Tokyo Story)
CDAVIES-20031008092436 18PDT	生きる (Ikiru)
CDAVIES-20031008092436 18PDT	天空の城ラピュタ (Castle in the Sky)

```
Result : 3 Rows Selected.

quit
```

The query uses nested XMLTable operators to create a virtual table containing data from different levels of the XML document. XMLExists identifies which rows will supply input to XMLTable.

- The query results in a table with two columns, REFERENCE and DESCRIPTION.
- The first XMLTable will create a table with two columns, REFERENCE and LINEITEMS. The table will contain one row for each document in table PURCHASEORDER where element /PurchaseOrder/LineItems/LineItem/Description contains the value 生きる (Ikiru). The value of column REFERENCE will come from element Reference. Column LINETIEMS will consist of an XML fragment containing the set of LineItem elements.
- The second XMLTable will take the fragment contained in LINEITEMS and create an table with one column, DESCRIPTION. The value of the column will come from element Description.
- The query result will be generated by selecting from the result of a join between the outputs of the two XMLTable operators. Since the output of the first XMLTable was the input to the second XMLTable, the join is a correlated join, meaning that each row generated by the inner XMLTable is automatically joined with the corresponding row from the outer XMLTable.

4.1.4 Chinese Query

This step shows using a simple Path expression to query XML documents that contain multi-byte content. The Path expression contains a predicate with Chinese characters. The document returned also contains Chinese content.

Click the icon to launch the XML DB demonstration framework and run the SQL script

Command Output

```
set echo on
--
select REFERENCE, DESCRIPTION
  from PURCHASEORDER,
       xmlTable
       (
         '/PurchaseOrder'
        passing object_value
        columns
        REFERENCE             path 'Reference',
        LINEITEMS  xmltype    path 'LineItems'
       ),
       xmlTable
       (
          '/LineItems/LineItem'
         passing LINEITEMS
         columns
         DESCRIPTION          path 'Description'
       )
 where XMLExists('/PurchaseOrder/LineItems/LineItem[Description="花样年华"]' passing object_value)
/
```

REFERENCE	DESCRIPTION
AMCEWEN-20040409123336271PDT	贵妇失踪案
AMCEWEN-20040409123336271PDT	烈爱风云
AMCEWEN-20040409123336271PDT	三十九级台阶
AMCEWEN-20040409123336271PDT	第三者
AMCEWEN-20040409123336271PDT	喋血双雄
AMCEWEN-20040409123336271PDT	辣手神探
AMCEWEN-20040409123336271PDT	沉默的羔羊
AMCEWEN-20040409123336271PDT	世界末日
AMCEWEN-20040409123336271PDT	斯巴达克斯
AMCEWEN-20040409123336271PDT	蝴蝶梦
AMCEWEN-20040409123336271PDT	美人计
AMCEWEN-20040409123336271PDT	花样年华
AMCEWEN-20040409123336271PDT	勇闯夺命岛

```
Result : 13 Rows Selected.

quit
```

Oracle strongly recommends the use of the AL32UTF8 database character set when managing XML content that contains a mix of single-byte and multi-byte content.

4.2.1 Un-indexed Queries and Plans

This step uses the output of the SQL explain plan utility to understand how path expressions are evaluated. In the following examples are based on an object-relational storage that has been optimized for collection processing.

Oracle XML DB is designed to deliver highly performant and scaleable processing of XQuery. With object-relational storage, this is achieved by re-writing the XQuery expressions contained in XMLQuery, XMLTable, XMLExists and XMLCast operators into the same internal algebra used for SQL statements. Re-writing XQuery this way allows the optimizer to process XQuery expressions in the same way that it processes other SQL operations.

XQuery re-write makes also makes it possible to use standard SQL query analysis and tuning techniques to optimize XQuery processing. The output of the explain plan utility can be used to understand how an XQuery will be executed and what indexes could be created to tune its performance.

Click the icon to launch the XML DB demonstration framework and run the SQL script. This script will re-run the queries from steps 4.1.1 and 4.1.2, but this time the explain plan output will be collected for each query. Once the queries have executed Click the Plan and Output tabs to see the full explain plan out for each query.

The first query counts the number of rows in table PURCHASEORDER.

```
select count(*) "Result"
  from PURCHASEORDER
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	22	5	00:00:01		
1	SORT AGGREGATE		1	22				
2	TABLE ACCESS FULL	PURCHASEORDER	1	22	5	00:00:01		SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read-properties/><read-contents/></privilege>'))=1

- The query plan is based on a full scan of table PURCHASEORDER
- A filter based on function SYS_CHECKACL function ensures that only documents the user has permission to access are included in the result set. In this case the minimum permission required is read-contents.
- This is the optimal plan for this query given that ACL based security is being enforced.

The second query counts the rows in table PURCHASEORDER where element /PurchaseOrder/User contains the value **SBELL**.

```
select count(*) "Result"
  from purchaseorder
 where XMLExists('$p/PurchaseOrder[User/text()="SBELL"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	29	5	00:00:01		
1	SORT AGGREGATE		1	29				
2	TABLE ACCESS FULL	PURCHASEORDER	1	29	5	00:00:01		"PURCHASEORDER"."SYS_NC00022 $"='SBELL' AND SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read- contents/></privilege>'))=1

- XQuery-rewrite allows the optimizer to process an XQuery expression exactly the same way that a SQL statement is processed.
- The query plan is based on a full scan of table PURCHASEORDER
- A filter selects the rows that satisfy the predicate. The filter is on the internal column name corresponding to element User
-
- This plan is probably reasonable when table PURCHASEORDER contains a small number of documents or when there are a limited number of distinct values for element /PurchaseOrder/User. If there are a large number of rows in table PURCHASEORDER or many distinct values for element /PurchaseOrder/User this is not a good plan.
- The plan can be optimized by creating an index on the content of element /PurchaseOrder/User.

The third query returns the contents of element /PurchaseOrder/Reference from table PURCHASEORDER where element /PurchaseOrder/User contains the value **SBELL**.

```
select XMLCast( XMLQuery ('$p/PurchaseOrder/Reference/text()' passing object_value as "p" returning content) as VARCHAR2(32)) "Reference"
  from purchaseorder
 where XMLExists('$p/PurchaseOrder[User/text()="SBELL"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	57	5	00:00:01		
1	TABLE ACCESS FULL	PURCHASEORDER	1	57	5	00:00:01		"PURCHASEORDER"."SYS_NC00022 $"='SBELL' AND SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read- contents/></privilege>'))=1

- The plan is the essentially the same as the plan for the previous query. The same comments apply.

The forth query returns the content of the document where element /PurchaseOrder/Reference contains the value **SBELL-2003030912333601PDT**.

```
select object_value "XML"
  from purchaseorder
  where XMLExists('$p/PurchaseOrder[Reference/text()="SBELL-2003030912333601PDT"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	226	1	00:00:01		
1	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	226	1	00:00:01		SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read-contents/></privilege>'))=1
2	INDEX UNIQUE SCAN	REFERENCE_IS_UNIQUE	1		0	00:00:01	"PURCHASEORDER"."SYS_NC00009$"='SBELL-2003030912333601PDT'	

- The query plan uses index REFERENCE_IS_UNIQUE to find the required row. This is the index that enforces the unique constraint on element /PurchaseOrder/Reference.

- Even though the constraint had to be specified using object-relational SQL syntax, query-rewrite of the path expression allows the optimizer to determine that the index can be used to optimize the query.

- This is the optimal plan for this query given that ACL based security is being enforced.

The fifth query returns element /PurchaseOrder/Reference for XML documents where at least one instance of attribute /PurchaseOrder/LinteItems/LineItem/Part/@Id contains the value **717951002372**.

```
select XMLCast( XMLQuery ('$p/PurchaseOrder/Reference/text()' passing object_value as "p" returning content) as VARCHAR2(32)) "Reference"
  from PURCHASEORDER
 where XMLExists('$p/PurchaseOrder/LineItems/LineItem/Part[@Id="717951002372"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	99	8	00:00:01		
1	NESTED LOOPS SEMI		1	99	8	00:00:01		
2	TABLE ACCESS FULL	PURCHASEORDER	1	67	5	00:00:01		SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read-contents/></privilege>'))=1
3	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	15	416	3	00:00:01		"SYS_NC00011$"='717951002372' AND "SYS_NC00009$" IS NOT NULL
4	INDEX RANGE SCAN	SYS_C0010104	16		1	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC00034000356$"	

- The query plan is based on a full scan of table PURCHASEORDER

- An index range scan selects the rows in table LINEITEM_TABLE that correspond to each row in table PURCHASEORDER. A filter selects rows that satisfy the predicate. The filter is on the internal column corresponding to attribute Id in element Part.

- This plan is probably reasonable when table PURCHASEORDER contains a small number of documents and each document only contains a small number of LineItem elements. If there are a large number of documents in table PURCHASEORDER or if the documents contain large numbers of LineItem elements this is not a good plan.

- The plan can be optimized by creating an index on the content of attribute /PurchaseOrder /LineItems/LineItem/Part/@Id. Since LineItem elements are stored in LINEITEM_TABLE, the index will need to be created directly on this table.

The sixth query returns element /PurchaseOrder/Reference for XML documents where at least one instance of element /PurchaseOrder/LinteItems/LineItem contains attribute @ItemNumber with a value of **1** and the attribute Part/@Id with a of value **717951002372**.

```
select XMLCast( XMLQuery ('$p/PurchaseOrder/Reference/text()' passing object_value as "p" returning content) as VARCHAR2(32)) "Reference"
  from PURCHASEORDER
 where XMLExists('$p/PurchaseOrder/LineItems/LineItem[@ItemNumber=1]/Part[@Id="717951009058"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	104	8	00:00:01		
1	NESTED LOOPS SEMI		1	104	8	00:00:01		
2	TABLE ACCESS FULL	PURCHASEORDER	1	67	5	00:00:01		SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read-contents/></privilege>'))=1
3	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	1	37	3	00:00:01		"SYS_NC00011$"='717515009058' AND "ITEMNUMBER"=1 AND "SYS_NC00009$" IS NOT NULL AND "SYS_NC_TYPEID$" IS NOT NULL
4	INDEX RANGE SCAN	SYS_C0010104	16		1	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC00034000358"	

- The plan is the essentially the same as the plan for the previous query. In general the same comments apply.

- An index range scan selects the rows in table LINEITEM_TABLE that correspond to each row in table PURCHASEORDER. A filter selects the rows that satisfy the predicate. The filter is on column ITEMNUMBER, which corresponds to attribute ItemNumber in element LineItem, and the internal column corresponding to attribute Id in element Part.

- This plan can be optimized by creating an index on the content of attribute /PurchaseOrder /LineItems/LineItem/Part/@Id. A compound index on the content of /PurchaseOrder /LineItems/LineItem/@ItemNumber and /PurchaseOrder/LineItems/LineItem/Part/@Id would be more effective. A compound index is possible since the query plan shows that both nodes are stored in LINEITEM_TABLE.

The seventh query counts the number of occurrences of element LineItem for all documents in table PURCHASEORDER.

```
select count(*) "Result"
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
       )
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	50	8	00:00:01		
1	SORT AGGREGATE		1	58				
2	NESTED LOOPS							
3	NESTED LOOPS		23	1334	8	00:00:01		
4	TABLE ACCESS FULL	PURCHASEORDER	1	39	5	00:00:01		SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read-contents/></privilege>'))=1
5	INDEX RANGE SCAN	SYS_C0010104	16		1	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC00034000356"	
6	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	16	304	3	00:00:01		"SYS_NC_TYPEID$" IS NOT NULL

- The query plan is based on a full scan of table PURCHASEORDER
- An index range scan selects the rows in table LINEITEM_TABLE that correspond to each row in table PURCHASEORDER.
- The query drives from table PURCHASEORDER since ACL based security is in effect.
- This is the optimal plan for this query given that ACL based security is being enforced.

The last query creates a virtual table from the collection of Description elements in the document where element Reference contains the value **SBELL-20030309123336O1PDT**.

```
--
select l.*
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
         columns
         DESCRIPTION path 'Description'
       ) l
 where XMLExists('$p/PurchaseOrder[Reference="SBELL-20030309123336O1PDT"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		15	1605	4	00:00:01		
1	NESTED LOOPS		15	1605	4	00:00:01		
2	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	66	2	00:00:01		SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read-contents/></privilege>'))=1
3	INDEX UNIQUE SCAN	REFERENCE_IS_UNIQUE	1		1	00:00:01	"PURCHASEORDER"."SYS_NC000094"='SBELL-20030309123336O1PDT'	
4	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	15	615	3	00:00:01		"SYS_NC_TYPEID$" IS NOT NULL
5	INDEX RANGE SCAN	SYS_C0010104	15		1	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC00034000356"	

- The query plan uses index REFERENCE_IS_UNIQUE to find the required row.
- An index range scan selects the rows in table LINEITEM_TABLE that correspond to the row in table PURCHASEORDER.
- This is the optimal plan for this query given that ACL based security is being enforced.

4.2.2 Create Indexes

Analysis of the explain plan output generated in the previous section identified three indexes that would improve query performance. This step creates the three indexes

Creating Indexes is a accepted way of improving the performance of SQL queries on relational data. Oracle XML DB allows the same technique to be used to improve the performance of XQuery operations on XML content. With object-relational storage, B-TREE and BITMAP indexes can be created on any node in the document. Compound indexes can be created when the nodes being indexed are mapped into the same storage table. Functional and Text based indexes can be created on XML content, regardless of how the content is stored. A new XML Index is available in Oracle Database 11g. This index is designed to optimize XQuery operations on XML content is being managed using Binary or CLOB based XMLType.

Oracle XML DB allows path expressions to be used to create B-Tree or BITMAP indexes on nodes that are mapped directly into an XMLType table or column. Normally only nodes that are not members of a collection are the mapped directly into an XMLType table or column. Indexing nodes that are members of a collection requires creating the index directly on the nested table that manages the collection using object-relational SQL. It is not possible to index nodes that are members of a collection if the collection is not managed by a nested table.

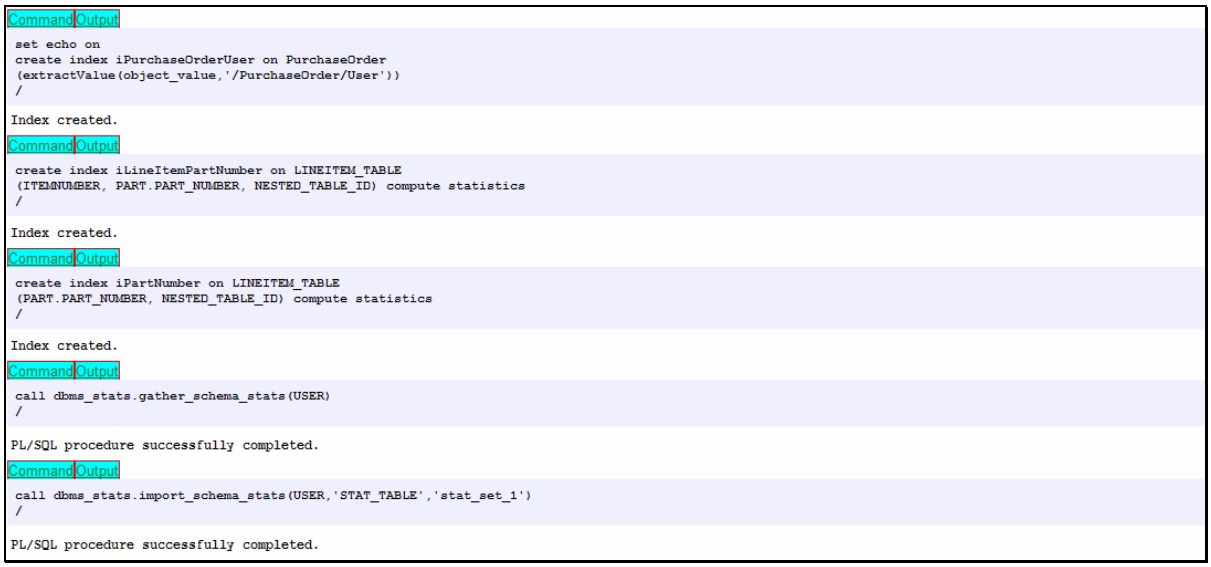
If the nodes are stored in an out-of-line table then the node is indexed by creating the index directly on the out-of-line table, using a path expression that is relative to the root node of the out-of-line table.

When an index is created using a path expression Oracle XML DB uses XQuery-rewrite to determine the internal column that contains the node referenced by the path expression. For re-write to occur the path expression must map to exactly one node in the document. If the path-expression cannot be re-written a functional index will be created.

To check whether XQuery re-write was able to successfully process the path expression check table USER_IND_EXPRESSIONS. If there is an entry in table USER_IND_EXPRESSIONS then the path expression was not rewritten.

Currently when creating an index on object-relational storage the path expression has to be supplied using the legacy extractValue operator. This operator pre-dates SQL/XML's XMLCast operator.

Click the icon to launch the XML DB demonstration framework and run the SQL script.



```
Command Output

set echo on
create index iPurchaseOrderUser on PurchaseOrder
(extractValue(object_value,'/PurchaseOrder/User'))
/

Index created.

Command Output

create index iLineItemPartNumber on LINEITEM_TABLE
(ITEMNUMBER, PART.PART_NUMBER, NESTED_TABLE_ID) compute statistics
/

Index created.

Command Output

create index iPartNumber on LINEITEM_TABLE
(PART.PART_NUMBER, NESTED_TABLE_ID) compute statistics
/

Index created.

Command Output

call dbms_stats.gather_schema_stats(USER)
/

PL/SQL procedure successfully completed.

Command Output

call dbms_stats.import_schema_stats(USER,'STAT_TABLE','stat_set_1')
/

PL/SQL procedure successfully completed.
```

- The first index created is on element /PurchaseOrder/User. The node is not a member of collection. The node is identified using a Path expression supplied using operator extractValue. XQuery-rewrite uses information in the XML Schema to determine the SQL Object and Attribute that corresponds to node identified in the path expression. A conventional B-Tree index is created on the appropriate column of the PURCHASEORDER table.

- The second index is created on attribute /PurchaseOrder/LineItems/LineItem/@ItemNumber and attribute /PurchaseOrder/LineItems/LineItem/Part/@Id. Element LineItem can occur more than once in each document, so the nodes are members of a collection. Hence the nodes cannot be identified using path expressions. The index is created on directly on LINEITEM_TABLE using object-relational SQL syntax to identify the attribute. The index is a compound index.

- The third index is created on attribute /PurchaseOrder/LineItems/LineItem/Part/@Id. Element LineItem can occur more than once in each document, so the node is a member of a collection. Hence the node cannot be identified using a path expression. The index is created on directly on LINEITEM_TABLE using object-relational SQL syntax to identify the attribute.

- After creating the indexes, database schema level statistics are gathered using package DBMS_STATS.

- The statistics for current set of 133 documents are over-written with statistics generated using a much larger sample of 10,000 PurchaseOrder documents. With only 133 documents the cost-based optimizer tends to select table-scans as the most efficient access method. Importing the statistics for 10,000 rows forces the optimizer to choose plans appropriate for large data sets.

- The set of 10,000 PurchaseOrder documents can be for downloaded from OTN.

4.2.3 Indexed Queries and Plans

This step uses the output of the SQL explain plan utility to show that the new indexes are being used to optimize XQuery execution. No changes are required to the XQuery expressions for the indexes to be used.

This step also shows the improvements obtained by disabling ACL based security. ACL based security should only be disabled when the default tables generated by schema registration will not be used in conjunction with the Oracle XML DB repository.

Set parameter ENABLEHIERARCHY to DBMS_XMLSCHEMA.ENABLE_HIERARCHY_NONE to disable ACL based security during XML Schema registration. ACL based security can also be disabled post XML Schema registration using method disable_hierarchy method in package DBMS_XDBZ. It can be enabled using method enable_hierarchy

Click the icon to launch the XML DB demonstration framework and run the SQL script.

Before the first query is executed procedure disable_heirarchy disables ACL based security on the PURCHASEORDER table.

```
call dbms_xdbz.disable_hierarchy(USER,'PURCHASEORDER')
/

PL/SQL procedure successfully completed.
```

The query plan for the first query is not affected by the new indexes. Performance will improve as the filter on function SYS_CHECKACL is eliminated when ACL based security is disabled.

```
select count(*) "Result"
  from PURCHASEORDER
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1		204	00:00:03		
1	SORT AGGREGATE		1					
2	TABLE ACCESS FULL	PURCHASEORDER	10000		204	00:00:03		

The query plan for the second query is updated to take the new indexes into consideration. The new plan is driven off index IPURCHASEORDERUSER.

```
select count(*) "Result"
  from purchaseorder
 where XMLExists('$p/PurchaseOrder[User/text()="SBELL"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	8	1	00:00:01		
1	SORT AGGREGATE		1	8				
2	INDEX RANGE SCAN	IPURCHASEORDERUSER	86	688	1	00:00:01	"PURCHASEORDER"."SYS_NC00022$"='SBELL'	

- The index allows documents that satisfy the predicate to be located without accessing all the documents in table PURCHASEORDER.

- In the presence of large amounts of data this is a much more efficient plan for this query.

The query plan for the third query is updated to take the new indexes into consideration. The new plan is driven off index IPURCHASEORDERUSER.

```
select XMLCast( XMLQUery ('$p/PurchaseOrder/Reference/text()' passing object_value as "p" returning content) as VARCHAR2(32)) "Reference"
  from purchaseorder
 where XMLExists('$p/PurchaseOrder[User/text()="SBELL"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		86	3182	17	00:00:01		
1	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	86	3182	17	00:00:01		
2	INDEX RANGE SCAN	IPURCHASEORDERUSER	86		1	00:00:01	"PURCHASEORDER"."SYS_NC00022$"='SBELL'	

- The index allows documents that satisfy the predicate to be located without accessing all the documents in table PURCHASEORDER.

- In the presence of large amounts of data this is a much more efficient plan for this query.

The query plan for the fourth query is not affected by the new indexes.

```
select object_value "XML"
  from purchaseorder
  where XMLExists('$p/PurchaseOrder[Reference/text()="SBELL-20030309123336O1PDT"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	231	2	00:00:01		
1	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	231	2	00:00:01		
2	INDEX UNIQUE SCAN	REFERENCE_IS_UNIQUE	1		1	00:00:01	"PURCHASEORDER"."SYS_NC00009$"='SBELL-20030309123336O1PDT'	

The query plan for the fifth query is updated to take the new indexes into consideration. The new plan is driven off index IPARTNUMBER.

```
select XMLCast( XMLQUery ('$p/PurchaseOrder/Reference/text()' passing object_value as "p" returning content) as VARCHAR2(32)) "Reference"
  from PURCHASEORDER
 where XMLExists('$p/PurchaseOrder/LineItems/LineItem/Part[@Id="717951002372"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		26	2028	44	00:00:01		
1	NESTED LOOPS							
2	NESTED LOOPS		26	2028	44	00:00:01		
3	SORT UNIQUE		26	832	30	00:00:01		
4	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	26	832	30	00:00:01		"SYS_NC00009$" IS NOT NULL
5	INDEX RANGE SCAN	IPARTNUMBER	26		3	00:00:01	"SYS_NC00011$"='717951002372'	
6	INDEX UNIQUE SCAN	LINEITEM_LIST	1		0	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC0003400035$"	
7	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	46	1	00:00:01		

- The query uses index IPARTNUMBER to locate the rows in LINEITEM_TABLE that satisfy the predicate. It then finds the corresponding rows in table PURCHASEORDER using the foreign-key primary-key relationship.

- In the presence of large amounts of data this is a much more efficient plan for this query.

The query plan for the sixth query is updated to take the new indexes into consideration. The new plan is driven off the compound index ILINEITEMPARTNUMBER.

```
select XMLCast( XMLQUery ('$p/PurchaseOrder/Reference/text()' passing object_value as "p" returning content) as VARCHAR2(32)) "Reference"
  from PURCHASEORDER
 where XMLExists('$p/PurchaseOrder/LineItems/LineItem[@ItemNumber=1]/Part[@Id="715515009058"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		2	166	7	00:00:01		
1	NESTED LOOPS							
2	NESTED LOOPS		2	166	7	00:00:01		
3	SORT UNIQUE		2	74	5	00:00:01		
4	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	2	74	5	00:00:01		"SYS_NC00009$" IS NOT NULL AND "SYS_NC_TYPEID$" IS NOT NULL
5	INDEX RANGE SCAN	ILINEITEMPARTNUMBER	2		3	00:00:01	"ITEMNUMBER"=1 AND "SYS_NC00011$"='715515009058'	
6	INDEX UNIQUE SCAN	LINEITEM_LIST	1		0	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC0003400035$"	
7	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	46	1	00:00:01		

- The query uses index ILINEITEMPARTNUMBER to locate the rows in LINEITEM_TABLE that satisfy the predicate. It then finds the corresponding rows in table PURCHASEORDER using the foreign-key primary-key relationship.

- In the presence of large amounts of data this is a much more efficient plan for this query.

The query plan for the seventh query is updated by disabling ACL based security.

```
select count(*) "Result"
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
       )
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	36	731	00:00:09		
1	SORT AGGREGATE		1	36				
2	NESTED LOOPS		148814	5357304	731	00:00:09		
3	TABLE ACCESS FULL	LINEITEM_TABLE	148814	2827466	718	00:00:09		"SYS_NC_TYPEID$" IS NOT NULL
4	INDEX UNIQUE SCAN	LINEITEM_LIST	1	17	0	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC0003400035$"	

.

- The new plan is driven off LINEITEM_TABLE.

- With ACL based security disabled there is no need to touch table PURCHASEORDER while executing the query.

- With ACL based disabled, this is a much more efficient plan for this query.

The query plan for the last query is not affected by the new indexes. Performance will improve as the filter on function SYS_CHECKACL is eliminated when ACL based security is disabled.

```
--
select l.*
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
         columns
         DESCRIPTION path 'Description'
       ) l
 where XMLExists('$p/PurchaseOrder[Reference="SBELL-20030309123336O1PDT"]' passing object_value as "p")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		15	1305	4	00:00:01		
1	NESTED LOOPS		15	1305	4	00:00:01		
2	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	46	2	00:00:01		
3	INDEX UNIQUE SCAN	REFERENCE_IS_UNIQUE	1		1	00:00:01	"PURCHASEORDER"."SYS_NC00009$"='SBELL-20030309123336O1PDT'	
4	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	15	615	2	00:00:01		"SYS_NC_TYPEID$" IS NOT NULL
5	INDEX RANGE SCAN	SYS_C0010104	15		1	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC0003400035$"	

After the last query is executed procedure enable_heirarchy in package DBMS_XDBZ re-enables ACL based security on the PURCHASEORDER table.

```
call dbms_xdbz.enable_hierarchy(USER,'PURCHASEORDER')
/

PL/SQL procedure successfully completed.
```

No changes to the XQuery expressions are required to use the new indexes, or to take advantage of disabling ACL based security. XQuery-rewrite allows the optimizer to process an XQuery expression just like it processes a SQL statement. The optimizer is able to determine these indexes provide a more efficient way of executing the queries and selects query plans that make use of the new indexes.

XQuery and the XMLType abstraction allow programmers to develop applications in a way that is independent of the underlying storage model and indexing. Database administrators can tune the performance of applications by creating and dropping indexes. No changes are required to the application code as indexes are created or dropped.

Nothing changes from the database administrator's perspective. The skills and tools required to optimize and manage databases that use XQuery and XMLType are the same as those required for database with traditional relational tables: Analyze query plans carefully; create indexes to optimize predicate evaluation; monitor index usage; drop indexes that are not contributing to query performance.

4.3.1 Update Operations

This step updates XML documents using path expressions to perform node and fragment level updates. Node and fragment level updates are much more efficient than a document level update when only a small part of the document is changing. Node and Fragment level updates are not only more efficient than document level updates, in a database environment they significantly reduce the amount of log generated by the update operation.

Use operator updateXML to update documents. UpdateXML replaces a node with a node. It operates on fragments and leaf-level nodes. The operator takes a series of path expressions and values. It replaces the node identified by each path expression with the corresponding value. UpdateXML only supports the XPath 1.0 specification; it does not support XPath 2.0 as defined by the XQuery standard.

If the path expression resolves to an attribute or text node then the replacement value must be a simple scalar data type. When updating text nodes the path expression must end with /text(). This allows UpdateXML to differentiate between an update of the parent element and an update of the text node.

If the path expression resolves to an element the replacement value must be supplied as an XMLType. The XMLType must contain a well formed fragment that can legally replace the target element. When updating a complex element, the entire branch is replaced with the content of the fragment.

UpdateXML works with schema-based and non-schema based XML. If the document being updated is schema-based the result of the update must pass the validation rules for the underlying storage model.

Object-relational storage allows certain updateXML operations to be re-written as SQL that operates directly on the underlying tables. Operations on text and attribute nodes will normally be completely re-written. Some operations that involve fragment replacement may be performed functionally, in which case the update operation will involve replacing the entire document. Re-writing updateXML operations into SQL reduces CPU and I/O costs significantly. It also greatly reduces the amount of log generated by the update operation. A re-written updateXML operation will often execute thousands of times faster than a document based update.

When updating an XMLType that is based on the CLOB storage model, UpdateXML operations are implemented using the interfaces defined by the W3C DOM API. When the update is complete the contents of DOM are serialized as text and written back to the underlying CLOB.

The Binary XML storage model uses XML Index and the sliding insert feature of Oracle's secure files to optimize updateXML operations. Sliding inserts enable partial updates of the XML content.

All UpdateXML operations are transactional. They need to be committed before they are visible outside of the session performing the update. They can be rolled back if something goes wrong before the commit is issued.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

The first update replaces the values of the text nodes belonging to elements User and Requestor. The update is performed on any document where element Reference contains the value **EABEL-20030409123336251PDT**.

Command Output

```
set feedback on
set linesize 132
column "Full Name" format A64
set echo on
select p.*
  from PURCHASEORDER,
       xmltable
       (
          '/PurchaseOrder'
          passing object_value
          columns
          EMAIL       path 'User',
          FULL_NAME   path 'Requestor'
       ) p
  where XMLExists('$p/PurchaseOrder[Reference/text()="EABEL-20030409123336251PDT"]' passing object_value as "p")
/
```

EMAIL	FULL_NAME
EABEL	Ellen S. Abel

Result : 1 Rows Selected.

Command Output

```
update PURCHASEORDER p
   set object_value = updateXML
                      (
                        object_value,
                        '/PurchaseOrder/User/text()','SBELL',
                        '/PurchaseOrder/Requestor/text()','Sarah J Bell'
                      )
  where XMLExists('$p/PurchaseOrder[Reference/text()="EABEL-20030409123336251PDT"]' passing object_value as "p")
/
```

1 row Updated.

Command Output

```
select p.*
  from PURCHASEORDER,
       xmltable
       (
          '/PurchaseOrder'
          passing object_value
          columns
          EMAIL       path 'User',
          FULL_NAME   path 'Requestor'
       ) p
  where XMLExists('$p/PurchaseOrder[Reference/text()="EABEL-20030409123336251PDT"]' passing object_value as "p")
/
```

EMAIL	FULL_NAME
SBELL	Sarah J Bell

Result : 1 Rows Selected.

- The path expressions resolve to text nodes so the replacement values are supplied as strings.
- Object-relational storage will allows this operation to be re-written as a direct update of the underlying tables.
- XMLExists identifies which documents to update. The update operation is applied to all the documents identified by the XMLExists operator

The second update shows a common mistake made when using updateXML. The intent of the update is to set the value of element Description to **The Wizard of Oz** where element Description contains the value **The Red Shoes**. The update is performed on any document where element Reference contains the value **EABEL-20030409123336251PDT**.

Command Output

```
--
select 1.*
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
         columns
         DESCRIPTION path 'Description'
       ) l
 where XMLexists('$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]' passing object_value as "p")
/
```

DESCRIPTION
Samurai 2: Duel at Ichijoji Temple
The Red Shoes
A Night to Remember

Command Output

```
update PURCHASEORDER
   set object_value = updateXML
                      (
                        object_value,
                        '/PurchaseOrder/LineItems/LineItem/Description/text()',
                        'The Wizard of Oz'
                      )
 where XMLexists('$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]/LineItems/LineItem[Description="The Red Shoes"]' passing object_value as "p")
/
```

```
1 row Updated.
```

Command Output

```
select 1.*
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
         columns
         DESCRIPTION path 'Description'
       ) l
 where XMLexists('$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]' passing object_value as "p")
/
```

DESCRIPTION
The Wizard of Oz
The Wizard of Oz
The Wizard of Oz

- Executing the update changes the value of all occurrences of element Description to **The Wizard of Oz**. This is the correct behavior for this statement!

- The predicates are supplied using XMLExists. XMLExists identifies which documents to update; it does not identify which nodes within the document are updated.

- The path expression passed to updateXML determines which nodes are updated. When more than one node matches the path expression, all the nodes are updated.

- If the intent is to update only a specific instance of a node that occurs multiple times with the document, the path expression in the UpdateXML operator must contain predicates that explicitly identify which node to update.

The third update uses a fragment update to undo the effect of the previous example. The update replaces the content of element LineItems. The update is performed on any document where element Reference contains the value **EABEL-20030409123336251PDT**.

Command Output

```
--
update PURCHASEORDER
   set object_value = updateXML
                      (
                        object_value,
                        '/PurchaseOrder/LineItems',
                        XMLType
                        (
                         '<LineItems>
        <LineItem ItemNumber="1">
                            <Description>Samurai 2: Duel at Ichijoji Temple</Description>
                            <Part Id="37429125526" UnitPrice="29.95" Quantity="3"/>
                          </LineItem>
                          <LineItem ItemNumber="2">
                            <Description>The Red Shoes</Description>
                            <Part Id="37429128220" UnitPrice="39.95" Quantity="4"/>
                          </LineItem>
                          <LineItem ItemNumber="3">
                            <Description>A Night to Remember</Description>
                            <Part Id="715515009058" UnitPrice="39.95" Quantity="1"/>
                           </LineItem>
                       </LineItems>'
                        )
                      )
 where XMLexists('$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]' passing object_value as "p")
/
```

```
1 row Updated.
```

Command Output

```
select 1.*
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
         columns
         DESCRIPTION path 'Description'
       ) 1
 where XMLexists('$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]' passing object_value as "p")
/
```

DESCRIPTION
Samurai 2: Duel at Ichijoji Temple
The Red Shoes
A Night to Remember

```
Result : 3 Rows Selected.
```

- XMLExists identifies which documents to update. The update operation is applied to all the documents identified by the XMLExists operator

- Element LineItems is replaced with the contents of the XMLType. The XMLType contains a copy of the original LineItems element. The update deletes all the existing children of element LineItems are creates a new branch for element LineItems from the contents of the XMLType.

The last update shows the correct syntax for operation attempted by the second update. The update sets the value of element Description to **The Wizard of Oz** where element Description contains the value **The Red Shoes**. The update is performed on any document where element Reference contains the value **EABEL-20030409123336251PDT**.

```
--
update PURCHASEORDER
set object_value = updateXML
                (
                    object_value,
                    '/PurchaseOrder/LineItems/LineItem/Description[text()="The Red Shoes"]/text()',
                    'The Wizard of Oz'
                )
 where XMLexists('$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]' passing object_value as "p")
/

1 row Updated.

Command Output

select l.*
  from PURCHASEORDER,
       XMLTable
       (
         '$p/PurchaseOrder/LineItems/LineItem'
         passing object_value as "p"
         columns
         DESCRIPTION path 'Description'
       ) l
 where XMLexists('$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]' passing object_value as "p")
/

DESCRIPTION
Samurai 2: Duel at Ichijoji Temple
The Wizard of Oz
A Night to Remember
```

- The predicates are supplied using XMLExists and updateXML. XMLExists identifies which documents to update; it does not identify which nodes within the document are updated.

- The path expression passed to updateXML determines which nodes are updated. The path expression includes a predicate which identifies which instance of element Description to update.

- In most cases it is not necessary to specify the predicate that selects which node to update in the XMLExists operator.

4.3.2 Delete-Insert-Append Operations

Operator UpdateXML can update any node in an XML document. UpdateXML can also add or remove nodes from a document, but only by performing a fragment replacement of the parent node. This step shows how to add and remove nodes directly using operators InsertChildXML, AppendChildXML, InsertXMLBefore and DeleteXML.

The operators only support the XPath 1.0 specification; they do not support XPath 2.0 as defined by the XQuery standard. Operations performed using the operators are transactional; changes made using these operators are not visible from other sessions until the transaction is committed. Until the transaction is committed any changes made using these operators can be undone by issuing a rollback.

Adding or removing nodes is effectively an update to the target document, so these operators are used in the context of a SQL update statement.

Operator DeleteXML deletes the node identified by each path expression. Deleting a node removes all of its children. DeleteXML can remove elements and attributes. DeleteXML works with schema-based and non-schema based XML. If the document being updated is schema-based the result of the update must pass the validation rules for the underlying storage model. When DeleteXML is used to delete a member of a collection the operation is re-written into a direct delete on the nested table when DOM Fidelity is disabled.

Operator InsertXMLBefore inserts a new element into the DOM tree immediately before the element identified by the path expression. The new element becomes the previous sibling of the element identified by the path expression. The path expression must identify an element, and only elements can be inserted. If the path references the parent element's first child, the inserted element becomes the first new first child. The new element can contain simple or complex content. InsertXMLBefore works with schema-based and non-schema based XML. If the document being updated is schema-based the result of the update must pass the validation rules for the underlying storage model.

Operator InsertChildXML inserts a new node into the DOM tree. The node can be an element or attribute. The path expression must always reference an element.

- Attributes are attached to the element referenced by the path expression. To insert an Attribute set the node name parameter to the attribute name, complete with an leading @. Set the new value parameter to the value of the attribute.

- For non-schema based XML, elements are inserted as the last child of the element referenced by the path expression.

- For schema-based XML the element is inserted at the first valid location. If the element being inserted is a member of a collection it is inserted as the last member the collection.

- A new element can contain simple or complex content. When inserting a substitutable element the node name parameter should contain the name of head of the substitution group.

- When InsertChildXML is used to add a new member to a collection the operation is re-written as a direct insert into the nested table when DOM Fidelity is disabled.

InsertXMLBefore works with schema-based and non-schema based XML. If the document being updated is schema-based the result of the update must pass the validation rules for the underlying storage model.

Operator AppendChildXML appends a new element into the DOM tree as the last child of the element identified by the path expression. The path expression must identify an element, and only elements can be inserted. If the path expression references an element that has no children, the inserted element becomes the first child of the referenced element. The new element can contain simple or complex content. AppendChildXML works with schema-based and non-schema based XML. If the document being updated is schema-based the result of the update must pass the validation rules for the underlying storage model.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

The first update uses DeleteXML to delete an instance of element LineItem from the document. The update deletes any instance of element LineItem where attribute ItemNumber contains the value **2**. The insert is performed on any document where element Reference contains the value **EABEL-20030409123336251PDT**.

Command Output

```
set feedback on
set linesize 132
set long 10240
set pages 0
set echo on
column USER format A32
column LINEITEMS format A40
--
-- DeleteXML followed by insertXMLBefore
--
select USER, LINEITEMS
  from PURCHASEORDER,
       xmltable
       (
         '$p/PurchaseOrder'
         passing object_value as "p"
         columns
         USERID              path 'User' ,
         LINEITEMS  xmltype  path 'LineItems/LineItem'
       )
 where xmlExists('$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]' passing object_value as "p")
/
```

USER	LINEITEMS
SCOTT	<LineItem ItemNumber="1"> <Description>Samurai 2: Duel at Ichijoji Temple</Description> <Part Id="37429125526"UnitPrice="29.95"Quantity="3"/> </LineItem> <LineItem ItemNumber="2"> <Description>The Red Shoes</Description> <Part Id="37429128220"UnitPrice="39.95"Quantity="4"/> </LineItem> <LineItem ItemNumber="3"> <Description>A Night to Remember</Description> <Part Id="715515009058"UnitPrice="39.95"Quantity="1"/> </LineItem>

```
Result : 1 Rows Selected.
```

Command Output

```
update PURCHASEORDER
   set object_value = deleteXML
                      (
                        object_value,
                        'PurchaseOrder/LineItems/LineItem[@ItemNumber="2"]'
                      )
 where xmlExists
       (
         '$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]'
         passing object_value as "p"
       )
/
```

```
1 row Updated.
```

Command Output

```
select USER, LINEITEMS
  from PURCHASEORDER,
       xmltable
       (
         '$p/PurchaseOrder'
         passing object_value as "p"
         columns
         USERID              path 'User' ,
         LINEITEMS  xmltype  path 'LineItems/LineItem'
       )
 where xmlExists
       (
         '$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]'
         passing object_value as "p"
       )
/
```

USER	LINEITEMS
SCOTT	<LineItem ItemNumber="1"> <Description>Samurai 2: Duel at Ichijoji Temple</Description> <Part Id="37429125526"UnitPrice="29.95"Quantity="3"/> </LineItem> <LineItem ItemNumber="3"> <Description>A Night to Remember</Description> <Part Id="715515009058"UnitPrice="39.95"Quantity="1"/> </LineItem>

```
Result : 1 Rows Selected.
```

- XMLExists identifies which documents to update. The update operation is applied to all the documents identified by the XMLExists operator

- DeleteXML delete all occurrences of element LineItem which match the supplied path expression from the documents identified by XMLExists. Any children of the deleted elements are also deleted.

The second update uses InsertXMLBefore to insert a new instance of element LineItem into the document. The element is inserted as the previous sibling of any instance of element LineItem which contains an ItemNumber attribute with the value **1**. The insert is performed on any document where element Reference contains the value **EABEL-20030409123336251PDT**.

Command Output

```
update PURCHASEORDER
   set object_value = insertXMLBefore
                      (
                        object_value,
                        '/PurchaseOrder/LineItems/LineItem[@ItemNumber="1"]',
                        XMLType
                        (
                         '<LineItem ItemNumber="0">
                            <Description>Rififi</Description>
                            <Part Id="37429155622" UnitPrice="29.95" Quantity="2"/>
                          </LineItem>'
                        )
                      )
 where xmlExists
       (
         '$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]'
        passing object_value as "p"
       )
/
```

```
1 row Updated.
```

Command Output

```
select USER, LINEITEMS
  from PURCHASEORDER,
       xmltable
       (
         '$p/PurchaseOrder'
        passing object_value as "p"
        columns
        USERID              path 'User' ,
        LINEITEMS  xmltype  path 'LineItems/LineItem'
       )
 where xmlExists
       (
         '$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]'
        passing object_value as "p"
       )
/
```

USER	LINEITEMS
SCOTT	<LineItem ItemNumber="0"> <Description>Rififi</Description> <Part Id="37429155622"UnitPrice="29.95"Quantity="2"/> </LineItem> <LineItem ItemNumber="1"> <Description>Samurai 2: Duel at Ichijoji Temple</Description> <Part Id="37429125526"UnitPrice="29.95"Quantity="3"/> </LineItem> <LineItem ItemNumber="3"> <Description>A Night to Remember</Description> <Part Id="715515009058"UnitPrice="39.95"Quantity="1"/> </LineItem>

```
Result : 1 Rows Selected.
```

- XMLExists identifies which documents to update. The update operation is applied to all the documents identified by the XMLExists operator

- InsertXMLBefore inserts the new node into the DOM Tree immediately before the node referenced by the path expression.

The third update uses InsertChildXML to insert a new instance of element LineItem into the document. Since the document is schema based, the element is inserted as the last member of the LineItem collection. The insert is performed on any document where element Reference contains the value **EABEL-20030409123336251PDT**.

Command Output

```
update PURCHASEORDER
   set object_value = insertChildXML
                      (
                        object_value,
                        '/PurchaseOrder/LineItems',
                        'LineItem',
                        XMLType
                        (
                         '<LineItem ItemNumber="4">
                            <Description>Dreyer Box Set</Description>
                            <Part Id="37429158425" UnitPrice="79.95" Quantity="1"/>
                          </LineItem>'
                        )
                      )
 where xmlExists
       (
         '$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]'
         passing object_value as "p"
       )
/

1 row Updated.
```

Command Output

```
select USER, LINEITEMS
  from PURCHASEORDER,
       xmltable
       (
         '$p/PurchaseOrder'
         passing object_value as "p"
         columns
         USERID              path 'User' ,
         LINEITEMS  xmltype  path 'LineItems/LineItem'
       )
 where xmlExists
       (
         '$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]'
         passing object_value as "p"
       )
/
```

USER	LINEITEMS
SCOTT	<LineItem ItemNumber="0"> <Description>Rififi</Description> <Part Id="37429155622"UnitPrice="29.95"Quantity="2"/> </LineItem> <LineItem ItemNumber="1"> <Description>Samurai 2: Duel at Ichijoji Temple</Description> <Part Id="37429125526"UnitPrice="29.95"Quantity="3"/> </LineItem> <LineItem ItemNumber="3"> <Description>A Night to Remember</Description> <Part Id="715515009058"UnitPrice="39.95"Quantity="1"/> </LineItem> <LineItem ItemNumber="4"> <Description>Dreyer Box Set</Description> <Part Id="37429158425"UnitPrice="79.95"Quantity="1"/> </LineItem>

```
Result : 1 Rows Selected.
```

- XMLExists identifies which documents to update. The update operation is applied to all the documents identified by the XMLExists operator

- InsertChildXML inserts the new node into the DOM Tree at the first valid location. Since the element being inserted is a member of a collection the element is inserted as the last member of the collection.

The last update uses AppendChildXML to insert a new instance of element LineItem into the document. The element is inserted as the last child of element LineItems. The insert is performed on any document where element Reference contains the value **EABEL-20030409123336251PDT**.

Command Output

```
update PURCHASEORDER
   set object_value = appendChildXML
                      (
                        object_value,
                        '/PurchaseOrder/LineItems',
                        XMLType
                        (
                         '<LineItem ItemNumber="5">
                            <Description>Hard Boiled</Description>
                            <Part Id="715515009041" UnitPrice="39.95" Quantity="3"/>
                          </LineItem>'
                        )
                      )
 where xmlExists
       (
         '$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]'
         passing object_value as "p"
       )
/

1 row Updated.
```

Command Output

```
select USER, LINEITEMS
  from PURCHASEORDER,
       xmltable
       (
         '$p/PurchaseOrder'
         passing object_value as "p"
         columns
         USERID              path 'User' ,
         LINEITEMS  xmltype  path 'LineItems/LineItem'
       )
 where xmlExists
       (
         '$p/PurchaseOrder[Reference="EABEL-20030409123336251PDT"]'
         passing object_value as "p"
       )
/
```

USER	LINEITEMS
SCOTT	<LineItem ItemNumber="0"> <Description>Rififi</Description> <Part Id="37429155622"UnitPrice="29.95"Quantity="2"/> </LineItem> <LineItem ItemNumber="1"> <Description>Samurai 2: Duel at Ichijoji Temple</Description> <Part Id="37429125526"UnitPrice="29.95"Quantity="3"/> </LineItem> <LineItem ItemNumber="3"> <Description>A Night to Remember</Description> <Part Id="715515009058"UnitPrice="39.95"Quantity="1"/> </LineItem> <LineItem ItemNumber="4"> <Description>Dreyer Box Set</Description> <Part Id="37429158425"UnitPrice="79.95"Quantity="1"/> </LineItem> <LineItem ItemNumber="5"> <Description>Hard Boiled</Description> <Part Id="715515009041"UnitPrice="39.95"Quantity="3"/> </LineItem>

```
Result : 1 Rows Selected.
```

- XMLExists identifies which documents to update. The update operation is applied to all the documents identified by the XMLExists operator

- AppendChildXML inserts the new node into the DOM Tree as the last child of the element referenced by the path expression

4.4.1 Make Views

This step creates a set of relational views containing data obtained from XML content. The views enable access to XML data using purely relational constructs. This allows tools and developers that understand Oracle Database and the relational model to work with XML content. The views also allow the full power of SQL and advanced features of Oracle Database 11g to be used on XML content.

The views are based operator XMLTable. XMLTable is used to create a mapping between the nodes in the XML document and the columns of the view. The hierarchy of the XML document can be exposed as a series related master-detail views. Relational queries on the view are as efficient as XQuery operations on the XML.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

The first statement creates view PURCHASEORDER_MASTER_VIEW on top of the XMLType table PURCHASEORDER.

Command Output

```
set echo on
create or replace view PURCHASEORDER_MASTER_VIEW
as
  select m.*
    from PURCHASEORDER p,
         xmlTable
         (
            '$p/PurchaseOrder'
            passing p.OBJECT_VALUE as "p"
            columns
            REFERENCE            path  'Reference/text()',
            REQUESTOR            path  'Requestor/text()',
            USERID               path  'User/text()',
            COSTCENTER           path  'CostCenter/text()',
            SHIP_TO_NAME         path  'ShippingInstructions/name/text()',
            SHIP_TO_ADDRESS      path  'ShippingInstructions/address/text()',
            SHIP_TO_PHONE        path  'ShippingInstructions/telephone/text()',
            INSTRUCTIONS         path  'SpecialInstructions/text()'
         ) m
/
```

View created.

Command Output

```
describe PURCHASEORDER_MASTER_VIEW
```

View : PURCHASEORDER_MASTER_VIEW

Column	Type
REFERENCE	VARCHAR2(30 CHAR)
REQUESTOR	VARCHAR2(128 CHAR)
USERID	VARCHAR2(10 CHAR)
COSTCENTER	VARCHAR2(4 CHAR)
SHIP_TO_NAME	VARCHAR2(20 CHAR)
SHIP_TO_ADDRESS	VARCHAR2(256 CHAR)
SHIP_TO_PHONE	VARCHAR2(24 CHAR)
INSTRUCTIONS	VARCHAR2(2048 CHAR)

- There will be one row in the view for each document in table PURCHASEORDER.
- Operator XMLTable defines the columns in the view and which node in the document supplies the columns value.
- With object-relational storage the data type of each column is derived from the underlying storage model. For other storage models the SQL data type needs to be explicitly provided as part of the XMLTable operator.

The second statement creates view PURCHASEORDER_LINEITEM_VIEW on top of the XMLType table PURCHASEORDER.

Command Output

```
--
create or replace view PURCHASEORDER_LINEITEM_VIEW
as
  select m.REFERENCE, l.*
    from PURCHASEORDER p,
         xmlTable
         (
            '$p/PurchaseOrder'
            passing p.OBJECT_VALUE as "p"
            columns
            REFERENCE           path  'Reference/text()',
            LINEITEMS   xmlType path  'LineItems/LineItem'
         ) m,
         xmlTable
         (
           '$l/LineItem'
           passing m.LINEITEMS as "l"
           columns
           ITEMNO         path '@ItemNumber',
           DESCRIPTION    path 'Description',
           PARTNO         path 'Part/@Id',
           QUANTITY       path 'Part/@Quantity',
           UNITPRICE      path 'Part/@UnitPrice'
         ) l
/
```

View created.

Command Output

```
describe PURCHASEORDER_LINEITEM_VIEW
```

View : PURCHASEORDER_LINEITEM_VIEW

Column	Type
REFERENCE	VARCHAR2(30 CHAR)
ITEMNO	NUMBER(38,0)
DESCRIPTION	VARCHAR2(256 CHAR)
PARTNO	VARCHAR2(14 CHAR)
QUANTITY	NUMBER(38,0)
UNITPRICE	NUMBER(14,2)

- The statement uses nested XMLTable operators to expose the contents of the collection of LineItem elements as rows in a view. Each row in the view will be tagged with the value of the Reference element.

- There will be one row in the view for each LineItem element in table PURCHASEORDER.

- The first XMLTable operator generates a virtual table containing columns REFERENCE and LINEITEMS from the contents of table PURCHASEORDER. Column REFERENCE will contain the value of element Reference. Column LINEITMS will contain the set of LineItem elements. This XMLTable operator generates one row for document in table PURCHASORDER.

- The second XMLTable operator generates a virtual table containing columns ITEMNO, DESCRIPTION, PARTNO, QUANTITY and UNITPRICE. The data values for each column come from column LINEITEMS. This XMLTable operator generates one row for each LineItem element in column LINEITEMS.

- The view contains the result of a correlated join between the outputs of the XMLTable operators.

- With object-relational storage the data type of each column is derived from the underlying storage model. For other storage models the SQL data type needs to be explicitly provided as part of the XMLTable operator.

4.4.2 Query Views

Views created using XMLTable look and act just like any other relational view. Any SQL query can be executed against the view. However, the views are not normally updateable.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

```
set autotrace on explain
--
select REFERENCE, COSTCENTER, SHIP_TO_NAME
  from PURCHASEORDER_MASTER_VIEW
 where USERID = 'SBELL'
/
```

REFERENCE	COSTCENTER	SHIP_TO_NAME
SBELL-20030809123337353PDT	S30	Sarah J. Bell
SBELL-20031209123338304PDT	S30	Sarah J. Bell
SBELL-20031209123338505PDT	S30	Sarah J. Bell
SBELL-20030209123335771PDT	S30	Sarah J. Bell
SBELL-20030709123333763PDT	S30	Sarah J. Bell
SBELL-20030109123335280PDT	S30	Sarah J. Bell
SBELL-20030409123336331PDT	S30	Sarah J. Bell
SBELL-20030409123336231PDT	S30	Sarah J. Bell
SBELL-20030509123336362PDT	S30	Sarah J. Bell
SBELL-20030509123336532PDT	S30	Sarah J. Bell
SBELL-20031109123338204PDT	S30	Sarah J. Bell
SBELL-20031009123337673PDT	S30	Sarah J. Bell

Result : 12 Rows Selected.

Command Output Result Plan

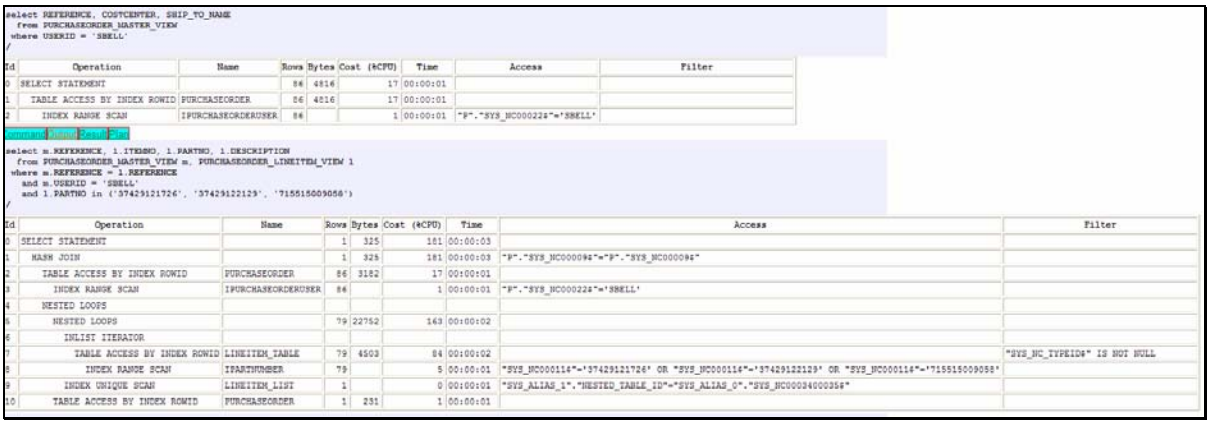
```
select m.REFERENCE, l.ITEMNO, l.PARTNO, l.DESCRIPTION
  from PURCHASEORDER_MASTER_VIEW m, PURCHASEORDER_LINEITEM_VIEW l
 where m.REFERENCE = l.REFERENCE
   and m.USERID = 'SBELL'
   and l.PARTNO in ('37429121726', '37429122129', '715515009058')
/
```

REFERENCE	ITEMNO	PARTNO	DESCRIPTION
SBELL-20031209123338304PDT	9	37429122129	The Lady Vanishes
SBELL-20030709123333763PDT	19	37429122129	The Lady Vanishes
SBELL-20030109123335280PDT	7	715515009058	A Night to Remember
SBELL-20030109123335280PDT	11	37429121726	Seven Samurai
SBELL-20030409123336331PDT	2	715515009058	A Night to Remember
SBELL-20030509123336532PDT	2	37429121726	Seven Samurai
SBELL-20031109123338204PDT	12	715515009058	A Night to Remember

Result : 7 Rows Selected.

- The first query shows a simple query against PURCHASEORDER_MASTER_VIEW
- The second query shows a relational join between PURCHASE_MASTER_VIEW and PURCHASEORDER_LINEITEM_VIEW. The join is based on the value of column REFERENCE.
- Both queries are perfect normal SQL queries using standard SQL Syntax. No knowledge of XML is required to query the views. There is nothing XML specific in the query or the result set.
- XQuery re-write ensures that the performance of SQL queries against the views is identical to the performance of XQuery operations on the base tables.

Click the PLAN tab to view the query plan for each of the queries



```
select REFERENCE, COSTCENTER, SHIP_TO_NAME
  from PURCHASEORDER_MASTER_VIEW
 where USERID = 'SBELL'
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		84	4816	17	00:00:01		
1	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	84	4816	17	00:00:01		
2	INDEX RANGE SCAN	IPURCHASEORDERUSER	84		1	00:00:01	"P"."SYS_NC000228"='SBELL'	

Command Output Result Plan

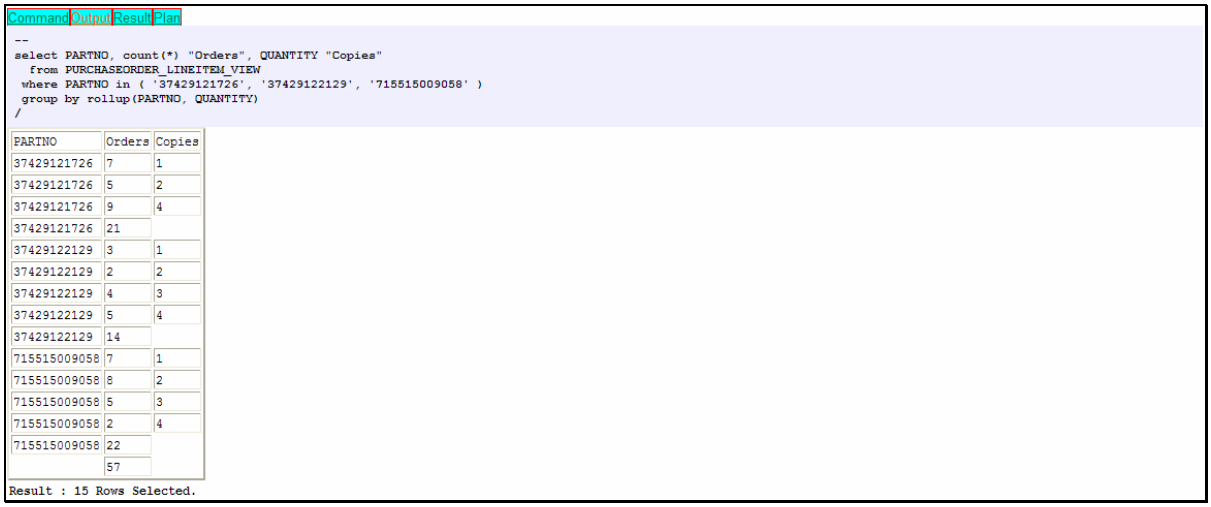
```
select m.REFERENCE, l.ITEMNO, l.PARTNO, l.DESCRIPTION
  from PURCHASEORDER_MASTER_VIEW m, PURCHASEORDER_LINEITEM_VIEW l
 where m.REFERENCE = l.REFERENCE
   and m.USERID = 'SBELL'
   and l.PARTNO in ('37429121726', '37429122129', '715515009058')
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	325	181	00:00:03		
1	HASH JOIN		1	325	181	00:00:03	"P"."SYS_NC000094"="P"."SYS_NC000094"	
2	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	86	3182	17	00:00:01		
3	INDEX RANGE SCAN	IPURCHASEORDERUSER	84		1	00:00:01	"P"."SYS_NC000228"='SBELL'	
4	NESTED LOOPS							
5	NESTED LOOPS		79	22752	163	00:00:02		
6	INLIST ITERATOR							
7	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	79	4503	84	00:00:02		"SYS_NC_TYPEID$" IS NOT NULL
8	INDEX RANGE SCAN	IPARTNUMBER	79		5	00:00:01	"SYS_NC0000114"='37429121726' OR "SYS_NC0000114"='37429122129' OR "SYS_NC0000114"='715515009058'	
9	INDEX UNIQUE SCAN	LINEITEM_LIST	1		0	00:00:01	"SYS_ALIAS_1"."NESTED_TABLE_ID"="SYS_ALIAS_0"."SYS_NC00034000356"	
10	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	231	1	00:00:01		

- The query plans for both queries are very straight-forward.

- Note that ACL-based security has been disabled to simply the explain plan results.

XQuery does not yet support all the advanced features of the SQL. Relational views of XML data address this issue. The next query uses the Business Intelligence and Analytic capabilities of SQL to perform a simple analysis of XML data.



Command Output Result Plan

```
--
select PARTNO, count(*) "Orders", QUANTITY "Copies"
  from PURCHASEORDER_LINEITEM_VIEW
 where PARTNO in ( '37429121726', '37429122129', '715515009058' )
 group by rollup(PARTNO, QUANTITY)
/
```

PARTNO	Orders	Copies
37429121726	7	1
37429121726	5	2
37429121726	9	4
37429121726	21	
37429122129	3	1
37429122129	2	2
37429122129	4	3
37429122129	5	4
37429122129	14	
715515009058	7	1
715515009058	8	2
715515009058	5	3
715515009058	2	4
715515009058	22	
	57	

Result : 15 Rows Selected.

- Relational views of XML data allow SQL features, such as group by and rollup, to work directly with relational content.

- The query uses SQL analytics to analyze the PurchaseOrder documents. The query determines how many copies of certain items are being ordered. For part **37429121726**, there 7 documents where the quantity ordered is 1, 5 documents where the quantity ordered is 2 and 9 documents where the quantity ordered is 4. There are a total of 21 documents that contain orders for that part.

4.4.3 Make XML View

The SQL/XML publishing functions are a set of extensions to the SQL standard that allows a SQL query to return XML documents instead of tabular data. Oracle, IBM and Microsoft were all actively involved in the development of the SQL/XML publishing functions, although Microsoft have not yet provided an implementation.

The operators defined by the standard are extremely powerful and flexible. They allow complex XML documents to be generated using simple SQL statements. The primary operators defined by the standard are

- XMLElement: XMLElement creates an element; the element can contain simple or complex content.
- XMLAttributes: XMLAttributes adds attributes to an element generated by XMLElement.
- XMLForest: XMLForest generates a single fragment from a set of data values. The name comes from the fact that the output of the operator contains one or more trees.
- XMLAgg: XMLAgg is a grouping function that generates a single fragment from the set of elements generated by a sub-query.

Many vendors implement the SQL/XML operators in an inefficient manner, by post-processing the result set of a conventional SQL query using DOM API's and then printing the DOM. In this model, the database is not aware that query will be used to generate XML so it cannot optimize the query for XML generation. Using DOM to process a large SQL result will also consume significant amounts of CPU and memory.

Oracle implements the SQL/XML operators as part of the database kernel. This allows the query plans to be optimized based on the structure of the XML being generated. The advantage of this is performance, since the query-plan for a query that returns data as tabular result set may be different from the query plan for a query that returns the same data as XML. The Oracle implementation also avoids the use of DOM, significantly reducing the amount of CPU and memory required to generate the XML.

This step uses the SQL/XML publishing functions to create an XML view. The view hides the structure of the underlying relational tables and allows direct access to relational data from XQuery and XSL. The documents in the view can be published as resources in the Oracle XML DB repository.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

```
create or replace view DEPARTMENT_XML of xmltype
with object id
(
  substr(extractValue(sys_nc_rowinfo$,'/Department/Name'),1,30)
)
as
select xmlElement
       (
         "Department",
         xmlAttributes( d.DEPARTMENT_ID as "DepartmentId"),
         xmlElement("Name", d.DEPARTMENT_NAME),
         xmlElement
         (
           "Location",
           xmlForest
           (
              STREET_ADDRESS as "Address", CITY as "City", STATE_PROVINCE as "State",
              POSTAL_CODE as "Zip",COUNTRY_NAME as "Country"
           )
         ),
         xmlElement
         (
           "EmployeeList",
           (
             select xmlAgg
                    (
                      xmlElement
                      (
                        "Employee",
                        xmlAttributes ( e.EMPLOYEE_ID as "employeeNumber" ),
                        xmlForest
                        (
                          e.FIRST_NAME as "FirstName", e.LAST_NAME as "LastName", e.EMAIL as "EmailAddress",
                          e.PHONE_NUMBER as "Telephone", e.HIRE_DATE as "StartDate", j.JOB_TITLE as "JobTitle",
                          e.SALARY as "Salary", m.FIRST_NAME || ' ' || m.LAST_NAME as "Manager"
                        ),
                        xmlElement ( "Commission", e.COMMISSION_PCT )
                      )
                    )
               from HR.EMPLOYEES e, HR.EMPLOYEES m, HR.JOBS j
              where e.DEPARTMENT_ID = d.DEPARTMENT_ID
                and j.JOB_ID = e.JOB_ID
                and m.EMPLOYEE_ID = e.MANAGER_ID
           )
         )
       ) as XML
  from HR.DEPARTMENTS d, HR.COUNTRIES c, HR.LOCATIONS l
 where d.LOCATION_ID = l.LOCATION_ID
   and l.COUNTRY_ID  = c.COUNTRY_ID
/
View created.
```

- The statement generates view DEPARTMENT_XML. The view is an XMLType view. It provides a persistent XML representation of the data from tables in the Oracle Sample Schema HR.

- An XMLType view is an object view. There must be a unique id for each row in the view.

- The XMLType view will contain one document for each row generated by the SQL statement. In this case the SQL statement will generate one row for each row in table DEPARTMENT.

- The root element of each document will be named Department (XMLElement)

- Element Department will contain an attribute called DepartmentId, and elements called Name, Location and EmployeeList.

- The value of the DepartmentId attribute will come from the column DEPARTMENT_ID in table DEPARTMENT table.

- The value of element Name will come from column DEPARTMENT_NAME column in table DEPARTMENT.

- Element Location will contain elements Address, City, State, Zip and Country. The content for element Location is found by joining table LOCATION with table DEPARTMENT using column LOCATION_ID.

- The content for element Location is generated using XMLForest. XMLForest will only generate the elements from columns that contain non-null values. The content for elements Address, City, State and Zip comes from columns STREET_ADDRESS, CITY, STATE_PROVINCE and POSTAL_CODE in table LOCATIONS.

- The content of element Country is found by joining table COUNTRIES with table LOCATIONS using column COUNTRY_ID. The content of element Country comes from column COUNTRY_NAME in table COUNTRIES.

- Element EmployeeList will consist of zero or more Employee elements. The set of Employee elements is found by joining table EMPLOYEES with table DEPARTMENTS using column DEPARTMENT_ID.

- Element Employee element contain attribute EmployeeNumber and elements FirstName, LastName, EmailAddress, Telephone, StartDate, JobTitle, Salary, Manager and Commission. Element Commission will always be present, since it is generated using XMLElement. The content of element Commission comes from column COMMISSION. If the value of column COMMISSION is null then element commission will appear as an empty element. The other elements are generated using XMLForest so they will only appear if the relational column contains a non-null value.

- The content for elements FirstName, LastName, EmailAddress, Telephone, StartDate and Salary comes from the columns FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE and SALARY.

- The content of element JobTitle is found by joining table EMPLOYEES with table JOBS using column JOB_ID. The content of element JobTitle comes from column JOB_TITLE in table JOBS.

- The content of element Manager is found by a self join on table EMPLOYEES using column MANAGER_ID and column EMPLOYEE_ID. The content of element Manager comes from columns FIRST_NAME and LAST_NAME in the row where column EMPLOYEE_ID matches the current value of column MANAGER_ID.

The select statement generates one document for each row generated by the SQL statement. To generate a result set that contains a single XML document use the following statement

select XMLElement ("element", XMLAgg ((SQL statement))) from dual

where "element" is name of the root element and "SQL statement" is the SQL that generates the required XML content.

4.4.4 Query XML View

XMLType views allow XQuery expressions to be used to access relational content. Click the icon to launch the XML DB demonstration framework and run the SQL script.

The first query uses a simple path expression that selects data based on the value of an element that occurs once in each document.

```
-- select xmlserialize( DOCUMENT OBJECT_VALUE as CLOB encoding UTF-8 INDENT SIZE=2 )
select *
  from DEPARTMENT_XML d
 where XMLExists('$d/Department[Name="Executive"]' passing object_value as "d")
/
```

```
OBJECT_VALUE
<Department DepartmentId="90">
  <Name>Executive</Name>
  <Location>
    <Address>2004 Charade Rd</Address>
    <City>Seattle</City>
    <State>Washington</State>
    <Zip>98199</Zip>
    <Country>United States of America</Country>
  </Location>
  <EmployeeList>
    <Employee employeeNumber="101">
      <FirstName>Neena</FirstName>
      <LastName>Kochhar</LastName>
      <EmailAddress>NKOCHHAR</EmailAddress>
      <Telephone>515.123.4568</Telephone>
      <StartDate>1989-09-21</StartDate>
      <JobTitle>Administration Vice President</JobTitle>
      <Salary>17000</Salary>
      <Manager>Steven King</Manager>
      <Commission/>
    </Employee>
    <Employee employeeNumber="102">
      <FirstName>Lex</FirstName>
      <LastName>De Haan</LastName>
      <EmailAddress>LDEHAAN</EmailAddress>
      <Telephone>515.123.4569</Telephone>
      <StartDate>1993-01-13</StartDate>
      <JobTitle>Administration Vice President</JobTitle>
      <Salary>17000</Salary>
      <Manager>Steven King</Manager>
      <Commission/>
    </Employee>
  </EmployeeList>
</Department>
Result : 1 Rows Selected.
```

- The view allows XQuery expressions to be used in conjunction with relational data.
- This example finds documents in view DEPARTMENT_XML where element Name contains the value **Executive**. The path expression that identifies which documents to return is specified using XMLExists.
- The query returns an XML document containing data from multiple relational tables. The format of the document is determined the SQL/XML operators that were used in the view definition.

The second query uses a more complex path expression that selects data based on the value of an element that occurs multiple times in each document.

Command Output Result Plan

```
--
select XMLCast( XMLQuery ( '$d/Department/Name' passing object_value as "d" returning content) as VARCHAR2(32) ) DEPARTMENT_NAME
  from DEPARTMENT_XML d
 where XMLExists('$d/Department/EmployeeList/Employee[LastName="Grant"]' passing object_value as "d")
/
```

DEPARTMENT_NAME
Shipping

Result : 1 Rows Selected.

- This example returns the value of element Name for any document that contains element Employee where element LastName = **Grant**. The path expression that identifies which documents to return is specified using XMLExists.

- XMLCast is used to convert the cast the result from an XQuery data type into the SQL data type VARCHAR2.

XQuery operations on XMLType views are re-written in SQL operations on the underlying tables. This allows for very efficient execution of these queries. Click the PLAN tab to view the explain output for these queries.

```
-- select xmlserialize( DOCUMENT OBJECT_VALUE as CLOB encoding UTF-8 INDENT SIZE=2 )
select *
  from DEPARTMENT_XML d
 where XMLExists('$d/Department[Name="Executive"]' passing object_value as "d")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	80	4	00:00:01		
1	SORT AGGREGATE		1	114				
2	HASH JOIN		10	1140	9	00:00:01	"M"."EMPLOYEE_ID"="E"."MANAGER_ID"	
3	MERGE JOIN		10	950	5	00:00:01		
4	TABLE ACCESS BY INDEX ROWID	JOBS	19	513	2	00:00:01		
5	INDEX FULL SCAN	JOB_ID_PK	19		1	00:00:01		
6	SORT JOIN		10	680	3	00:00:01	"J"."JOB_ID"="E"."JOB_ID"	"J"."JOB_ID"="E"."JOB_ID"
7	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	10	680	2	00:00:01		
8	INDEX RANGE SCAN	EMP_DEPARTMENT_IX	10		1	00:00:01	"E"."DEPARTMENT_ID"=:B1	
9	TABLE ACCESS FULL	EMPLOYEES	107	2033	3	00:00:01		
10	NESTED LOOPS		1	80	4	00:00:01		
11	NESTED LOOPS		1	68	4	00:00:01		
12	TABLE ACCESS FULL	DEPARTMENTS	1	19	3	00:00:01		"D"."DEPARTMENT_NAME"='Executive'
13	TABLE ACCESS BY INDEX ROWID	LOCATIONS	1	49	1	00:00:01		
14	INDEX UNIQUE SCAN	LOC_ID_PK	1		0	00:00:01	"D"."LOCATION_ID"="L"."LOCATION_ID"	
15	INDEX UNIQUE SCAN	COUNTRY_C_ID_PK	1	12	0	00:00:01	"L"."COUNTRY_ID"="C"."COUNTRY_ID"	

```
pause
```

Command Output Result Plan

```
--
select XMLCast( XMLQuery ( '$d/Department/Name' passing object_value as "d" returning content) as VARCHAR2(32) ) DEPARTMENT_NAME
  from DEPARTMENT_XML d
 where XMLExists('$d/Department/EmployeeList/Employee[LastName="Grant"]' passing object_value as "d")
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		2	76	7	00:00:01		
1	NESTED LOOPS							
2	NESTED LOOPS		2	76	7	00:00:01		
3	NESTED LOOPS		2	64	5	00:00:01		
4	VIEW	VW_SQ_1	2	26	2	00:00:01		
5	HASH UNIQUE		2	48				
6	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	2	48	2	00:00:01		"E"."MANAGER_ID" IS NOT NULL
7	INDEX RANGE SCAN	EMP_NAME_IX	2		1	00:00:01	"E"."LAST_NAME"='Grant'	
8	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	19	1	00:00:01		
9	INDEX UNIQUE SCAN	DEPT_ID_PK	1		0	00:00:01	"ITEM_1"="D"."DEPARTMENT_ID"	
10	INDEX UNIQUE SCAN	LOC_ID_PK	1		0	00:00:01	"D"."LOCATION_ID"="L"."LOCATION_ID"	
11	TABLE ACCESS BY INDEX ROWID	LOCATIONS	1	6	1	00:00:01		"L"."COUNTRY_ID" IS NOT NULL

- The explain plan output shows that both queries have been re-written in SQL operations on the underling relational tables.

4.4.5 Folder Departments

The rows in an XMLType view can be used to create documents in the Oracle XML DB repository. This allows the content of the documents to be accessed directly by standard desktop applications. When the content of the document is dynamically generated at the point the document is opened.

This step uses published the each row in view DEPARTMENT_XML as document in the Oracle XML DB repository. The documents are published in folder **Departments**. Click the icon to launch the XML DB demonstration framework and run the SQL script.

Command Output

```
set echo on
create or replace trigger DEPARTMENT_DML
instead of INSERT or UPDATE or DELETE
on DEPARTMENT_XML
begin
  null;
end;
/
```

```
Trigger created.
```

Command Output

```
declare
  cursor getDepartments is
    select ref(d) XMLREF,
           XMLCast( XMLQUery ( '$d/Department/Name' passing object_value as "d" returning content) as VARCHAR2(32) ) NAME
      from DEPARTMENT_XML d;
  res boolean;
  targetFolder varchar2(1024) :=  '/home/SCOTT/Departments';
begin
  if dbms_xdb.existsResource(targetFolder) then
     dbms_xdb.deleteResource(targetFolder,dbms_xdb.DELETE_RECURSIVE);
  end if;
  res := dbms_xdb.createFolder(targetFolder);
  for dept in getDepartments loop
    res := DBMS_XDB.createResource(targetFolder || '/' || dept.NAME || '.xml', dept.XMLREF);
  end loop;
end;
/
```

```
PL/SQL procedure successfully completed.
```

Command Output

```
select path
  from path_view
 where under_path(RES,'/home/SCOTT/Departments') = 1
/
```

PATH
/home/SCOTT/Departments/Accounting.xml
/home/SCOTT/Departments/Administration.xml
/home/SCOTT/Departments/Benefits.xml
/home/SCOTT/Departments/Construction.xml
/home/SCOTT/Departments/Contracting.xml
/home/SCOTT/Departments/Control And Credit.xml
/home/SCOTT/Departments/Corporate Tax.xml
/home/SCOTT/Departments/Executive.xml
/home/SCOTT/Departments/Finance.xml
/home/SCOTT/Departments/Government Sales.xml
/home/SCOTT/Departments/Human Resources.xml
/home/SCOTT/Departments/IT Helpdesk.xml
/home/SCOTT/Departments/IT Support.xml
/home/SCOTT/Departments/IT.xml
/home/SCOTT/Departments/Manufacturing.xml
/home/SCOTT/Departments/Marketing.xml
/home/SCOTT/Departments/NOC.xml
/home/SCOTT/Departments/Operations.xml
/home/SCOTT/Departments/Payroll.xml
/home/SCOTT/Departments/Public Relations.xml
/home/SCOTT/Departments/Purchasing.xml
/home/SCOTT/Departments/Recruiting.xml
/home/SCOTT/Departments/Retail Sales.xml
/home/SCOTT/Departments/Sales.xml
/home/SCOTT/Departments/Shareholder Services.xml
/home/SCOTT/Departments/Shipping.xml
/home/SCOTT/Departments/Treasury.xml

Result : 27 Rows Selected.

- The first step creates an instead of trigger that will suppress any DML operations on the contents of the view. This trigger could be replaced with a more complex trigger that would allow updates to the document trigger updates to the underlying relational tables.

- The next step uses package DBMS_XDB to create folder **Departments** and add a document for each row in view DEPARTMENT_XML. The name of each document is derived from the content of element Name.

- First function existsResource checks whether the folder already exists in the repository. If the folder exists it is deleted using procedure deleteResouce. A recursive delete is used to automatically delete any documents in the folder. A new folder is created using function createFolder.

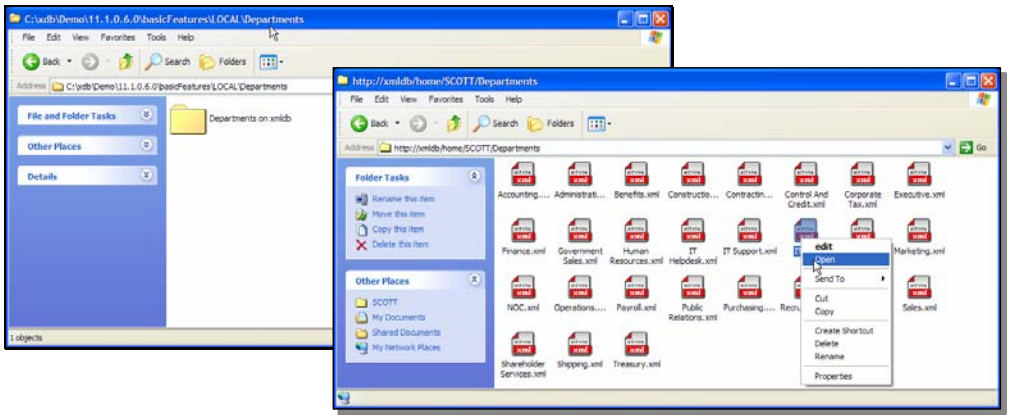
- Next the procedure iterates over the contents of DEPARTMENT_XML using function createResource to create a new document from each row in the view. The resource is created using a REF XMLType that maps the content of the resource to the corresponding row in the view.

- The final step performs a simple query on PATH_VIEW to show the set of documents that were created. PATH_VIEW is a global view that makes the contents of the XML DB repository accessible from SQL. The query uses the UNDER_PATH operator to select the path to documents in folder **Departments**.

4.4.6 View Departments

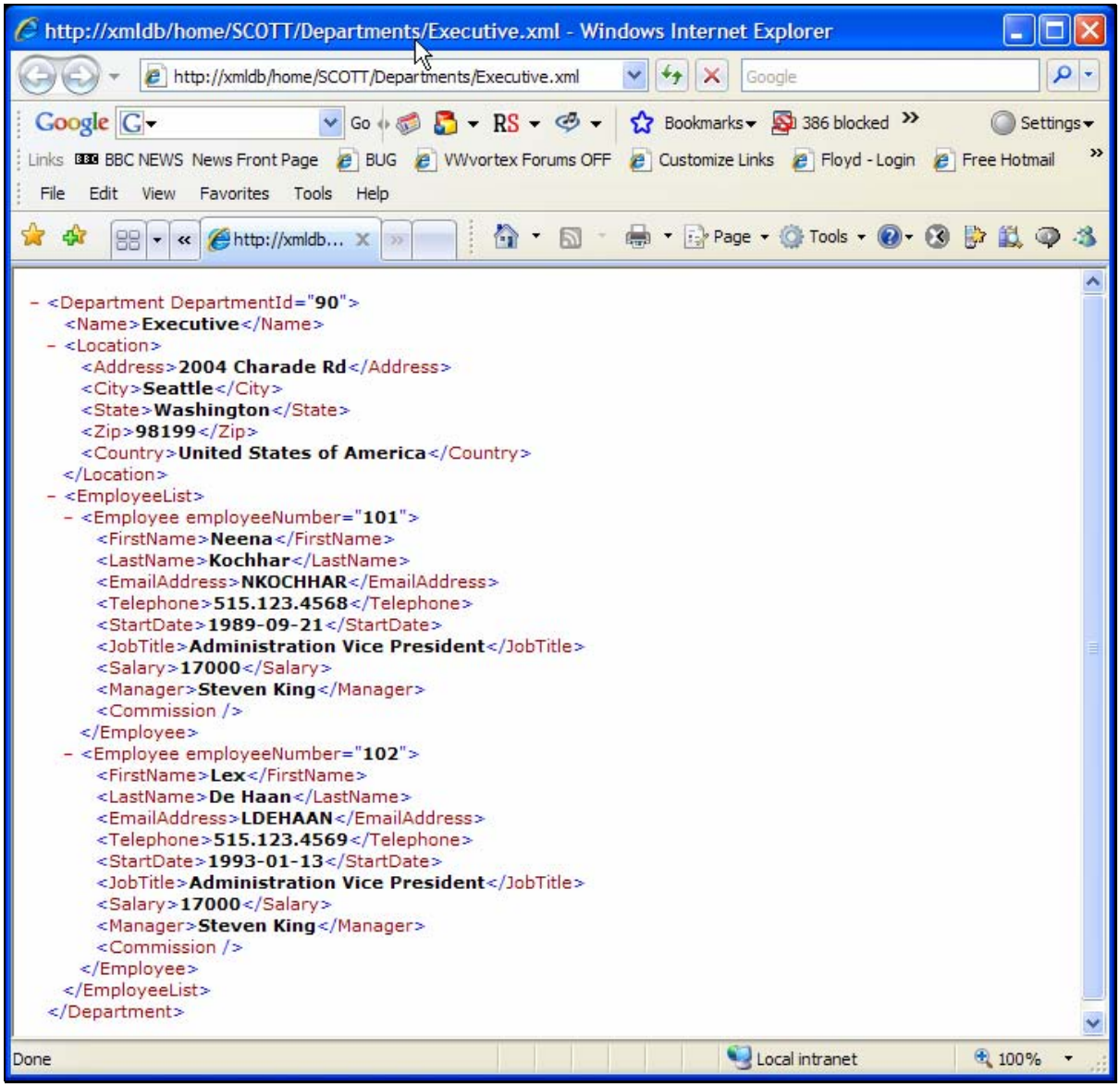
This step uses Microsoft Windows Explorer and Microsoft Internet Explorer to view the content of the resources created in the previous step.

Right click the icon and select explore to open a new window containing the local folder Departments. This folder contains a shortcut called "Departments on ***hostname***". This is a link to folder Department, in the demonstration user's home folder on the Oracle XML DB repository Double click the short cut to view the content of the remote folder.



- The remote folder contains one document for each row in the view DEPARTMENT_XML. The documents appears as XML documents when viewed in Windows Explorer

Right Click on document EXECUTIVE.XML and select Open. This will launch Internet Explorer to display the contents of the document. If the document opens with some other application use the Folder Options feature of Windows Explorer to adjust the file association for 'XML' files.



- Creating resources in the Oracle XML DB repository allows data in the view to be accessed from standard desktop applications. that support the HTTP, WebDAV and FTP protocols

- The content of the document is dynamically generated when the document is opened by fetching the appropriate row from view DEPARTMENT_XML
- The only code needed to expose the content of the relational tables as document in the repository is the CREATE VIEW statement the PL/SQL procedure that created resources from each row in the view. No connectivity software, such as ODBC is required to access the content of the view.

4.5.1 Make XML View (JPN)

The next example shows that SQL/XML provides full support for Asian languages. In this an example the names of the elements are attributes are in Japanese. Click the icon to launch the XML DB demonstration framework and run the SQL script.

Command Output

```
set echo on
--
create or replace view DEPARTMENT_XML_JPN of xmltype
with object id
(
  substr(extractValue(sys_nc_rowinfo$,'/部門/部門ID'),1,30)
)
as
select xmlElement
       (
         "部門",
         xmlAttributes( d.DEPARTMENT_ID as "部門ID"),
         xmlElement("部門名", d.DEPARTMENT_NAME),
         xmlElement
         (
           "住所",
           xmlForest
           (
              STREET_ADDRESS as "番地", CITY as "市町村", STATE_PROVINCE as "都道府県",
              POSTAL_CODE as "郵便番号",COUNTRY_NAME as "国名"
           )
         ),
         xmlElement
         (
           "社員リスト",
           (
             select xmlAgg
                    (
                      xmlElement
                      (
                        "社員",
                        xmlAttributes ( e.EMPLOYEE_ID as "社員番号" ),
                        xmlForest
                        (
                          e.FIRST_NAME as "名", e.LAST_NAME as "姓", e.EMAIL as "Emailアドレス",
                          e.PHONE_NUMBER as "電話番号", e.HIRE_DATE as "入社年月日", j.JOB_TITLE as "役職",
                          e.SALARY as "給料", m.FIRST_NAME || ' ' || m.LAST_NAME as "上司"
                        ),
                        xmlElement ( "COMMISSION", e.COMMISSION_PCT )
                      )
                    )
               from HR.EMPLOYEES e, HR.EMPLOYEES m, HR.JOBS j
              where e.DEPARTMENT_ID = d.DEPARTMENT_ID
                and j.JOB_ID = e.JOB_ID
                and m.EMPLOYEE_ID = e.MANAGER_ID
           )
         )
       ) as XML
  from HR.DEPARTMENTS d, HR.COUNTRIES c, HR.LOCATIONS l
 where d.LOCATION_ID = l.LOCATION_ID
   and l.COUNTRY_ID  = c.COUNTRY_ID
/
View created.
```

4.5.2 Query XML View (JPN)

The following query show that Oracle XML DB supports XQuery operations where the path expressions contain Asian language names. In this example the View created in the previous example is queried using a path expression that contains Japanese names. Click the icon to launch the XML DB demonstration framework and run the SQL script.

```
select object_value
  from DEPARTMENT_XML_JPN d
 where XMLExists('/部門[部門名="Executive"]' passing object_value)
/
```

```
OBJECT_VALUE
<部門 部門ID="90">
  <部門名>Executive</部門名>
  <住所>
    <番地>2004 Charade Rd</番地>
    <市町村>Seattle</市町村>
    <都道府県>Washington</都道府県>
    <郵便番号>98199</郵便番号>
    <国名>United States of America</国名>
  </住所>
  <社員リスト>
    <社員 社員番号="101">
      <名>Neena</名>
      <姓>Kochhar</姓>
      <Emailアドレス>NKOCHHAR</Emailアドレス>
      <電話番号>515.123.4568</電話番号>
      <入社年月日>1989-09-21</入社年月日>
      <役職>Administration Vice President</役職>
      <給料>17000</給料>
      <上司>Steven King</上司>
      <COMMISSION/>
    </社員>
    <社員 社員番号="102">
      <名>Lex</名>
      <姓>De Haan</姓>
      <Emailアドレス>LDEHAAN</Emailアドレス>
      <電話番号>515.123.4569</電話番号>
      <入社年月日>1993-01-13</入社年月日>
      <役職>Administration Vice President</役職>
      <給料>17000</給料>
      <上司>Steven King</上司>
      <COMMISSION/>
    </社員>
  </社員リスト>
</部門>
Result : 1 Rows Selected.
```

- Oracle XML DB is able to evaluate a path expression containing Japanese and return an XML document containing Japanese tag names.

Click the Plan tab to view the query plan for this query.

Command Output Result Plan

```
set echo on
--
set autotrace on explain
--
select object_value
  from DEPARTMENT_XML_JPN d
 where XMLExists('/部門[部門名="Executive"]' passing object_value)
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	80	4	00:00:01		
1	SORT AGGREGATE		1	114				
2	HASH JOIN		10	1140	9	00:00:01	"M"."EMPLOYEE_ID"="E"."MANAGER_ID"	
3	MERGE JOIN		10	950	5	00:00:01		
4	TABLE ACCESS BY INDEX ROWID	JOBS	19	513	2	00:00:01		
5	INDEX FULL SCAN	JOB_ID_PK	19		1	00:00:01		
6	SORT JOIN		10	680	3	00:00:01	"J"."JOB_ID"="E"."JOB_ID"	"J"."JOB_ID"="E"."JOB_ID"
7	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	10	680	2	00:00:01		
8	INDEX RANGE SCAN	EMP_DEPARTMENT_IX	10		1	00:00:01	"E"."DEPARTMENT_ID"=:B1	
9	TABLE ACCESS FULL	EMPLOYEES	107	2033	3	00:00:01		
10	NESTED LOOPS		1	80	4	00:00:01		
11	NESTED LOOPS		1	68	4	00:00:01		
12	TABLE ACCESS FULL	DEPARTMENTS	1	19	3	00:00:01		"D"."DEPARTMENT_NAME"='Executive'
13	TABLE ACCESS BY INDEX ROWID	LOCATIONS	1	49	1	00:00:01		
14	INDEX UNIQUE SCAN	LOC_ID_PK	1		0	00:00:01	"D"."LOCATION_ID"="L"."LOCATION_ID"	
15	INDEX UNIQUE SCAN	COUNTRY_C_ID_PK	1	12	0	00:00:01	"L"."COUNTRY_ID"="C"."COUNTRY_ID"	

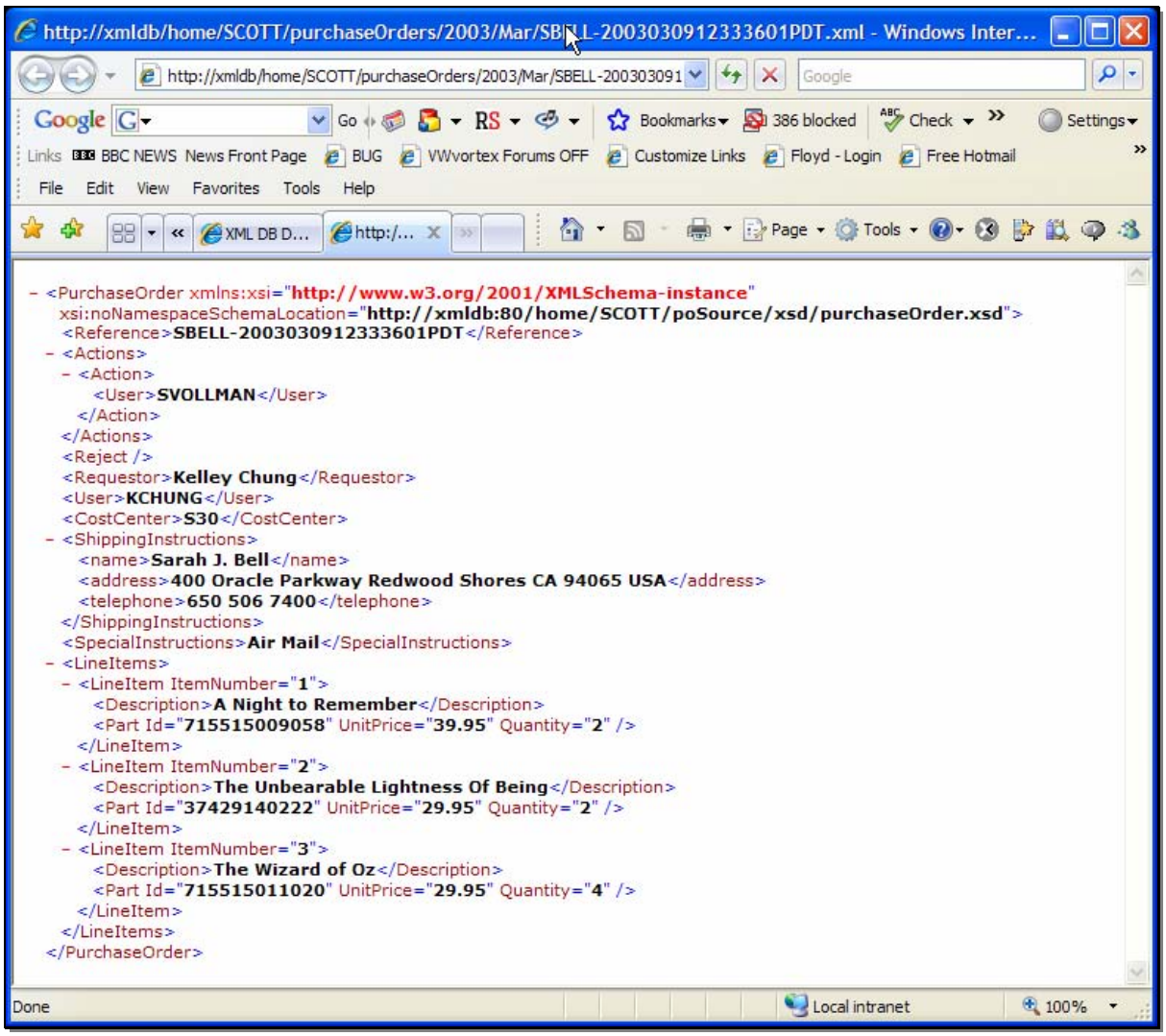
- The query plan for this query is identical to the query plan for the equivalent English language query.

5.1.1 View document (HTTP)

The Oracle XML DB repository allows content to be organized in file/folder hierarchy. The content can be accessed using a combination of the HTTP protocol and a URL. Click the icon to launch Internet Explorer. Internet Explorer will use HTTP to fetch the contents of the document identified by the following URL

/home/*USER*/PurchaseOrders/2003/Mar/SBELL-.2003030912333601PDT.XML

If the document opens with some other application use the Folder Options feature of Windows Explorer to adjust the file association for 'XML' files.



- If the browser does not own an active, authenticated HTTP connection to the repository, Internet Explorer will prompt for a username and password before accessing the document. If this happens enter the demonstration user's username and password and click OK.

- This access method appeals to developers, who are used to using HTTP URLs to access content.

- Supporting URL-based access allows desktop applications, such as Microsoft Word or XMLSPY to use the FTP, HTTP or WebDAV protocols to access and update content managed by the Oracle XML DB repository.

5.1.2 View document (SQL)

The URL access metaphor is also available when working in SQL. Operator xdbUriType and XQuery function fn:doc can be used to access a document based on its URL, or location inside the repository. This step uses the SQL operator xdbUriType to access the content of a document based on its URL. Click the icon to launch the XML DB demonstration framework and run the SQL script.

```
--
select xdbURIType('/home/SCOTT/purchaseOrders/2003/Mar/SBELL-2003030912333601PDT.xml').getXML() XML
  from dual
/
```

XML

```
<PurchaseOrder xsi:noNamespaceSchemaLocation="http://xmldb:80/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
  <Reference>SBELL-2003030912333601PDT</Reference>
  <Actions>
    <Action>
      <User>SVOLLMAN</User>
    </Action>
  </Actions>
  <Reject/>
  <Requestor>Kelley Chung</Requestor>
  <User>KCHUNG</User>
  <CostCenter>S30</CostCenter>
  <ShippingInstructions>
    <name>Sarah J. Bell</name>
    <address>400 Oracle Parkway Redwood Shores CA 94065 USA</address>
    <telephone>650 506 7400</telephone>
  </ShippingInstructions>
  <SpecialInstructions>Air Mail</SpecialInstructions>
  <LineItems>
    <LineItem ItemNumber="1">
      <Description>A Night to Remember</Description>
      <Part Id="715515009058"UnitPrice="39.95"Quantity="2"/>
    </LineItem>
    <LineItem ItemNumber="2">
      <Description>The Unbearable Lightness Of Being</Description>
      <Part Id="37429140222"UnitPrice="29.95"Quantity="2"/>
    </LineItem>
    <LineItem ItemNumber="3">
      <Description>The Wizard of Oz</Description>
      <Part Id="715515011020"UnitPrice="29.95"Quantity="4"/>
    </LineItem>
  </LineItems>
</PurchaseOrder>
```

Result : 1 Rows Selected.

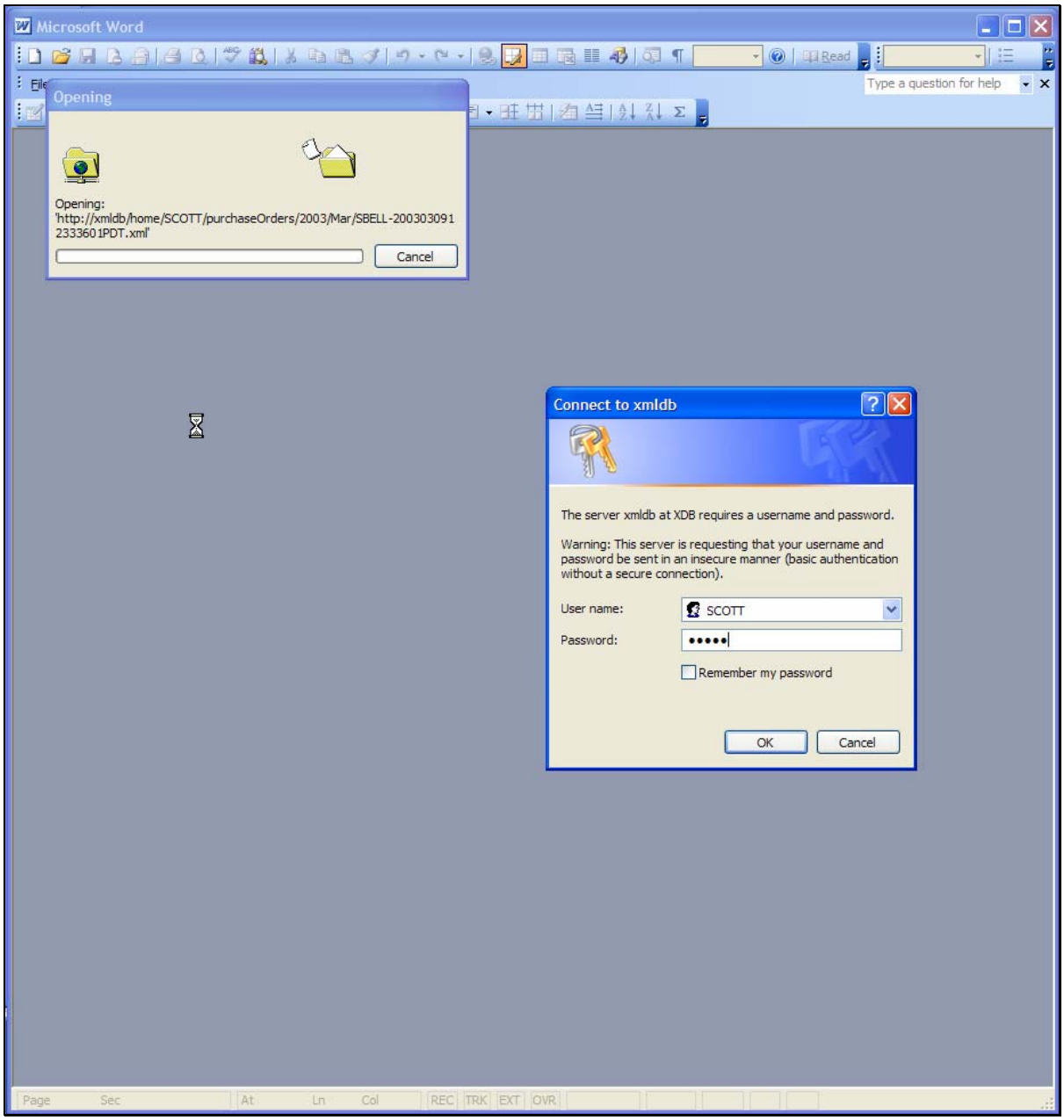
- Operator xdbUriType provides methods that access different types of content, including CLOB, BLOB and XML. It can also be used to access the resource document, rather than the content.

- The URL has to be absolute from the folder of the XML DB repository.

5.1.3 Edit Document

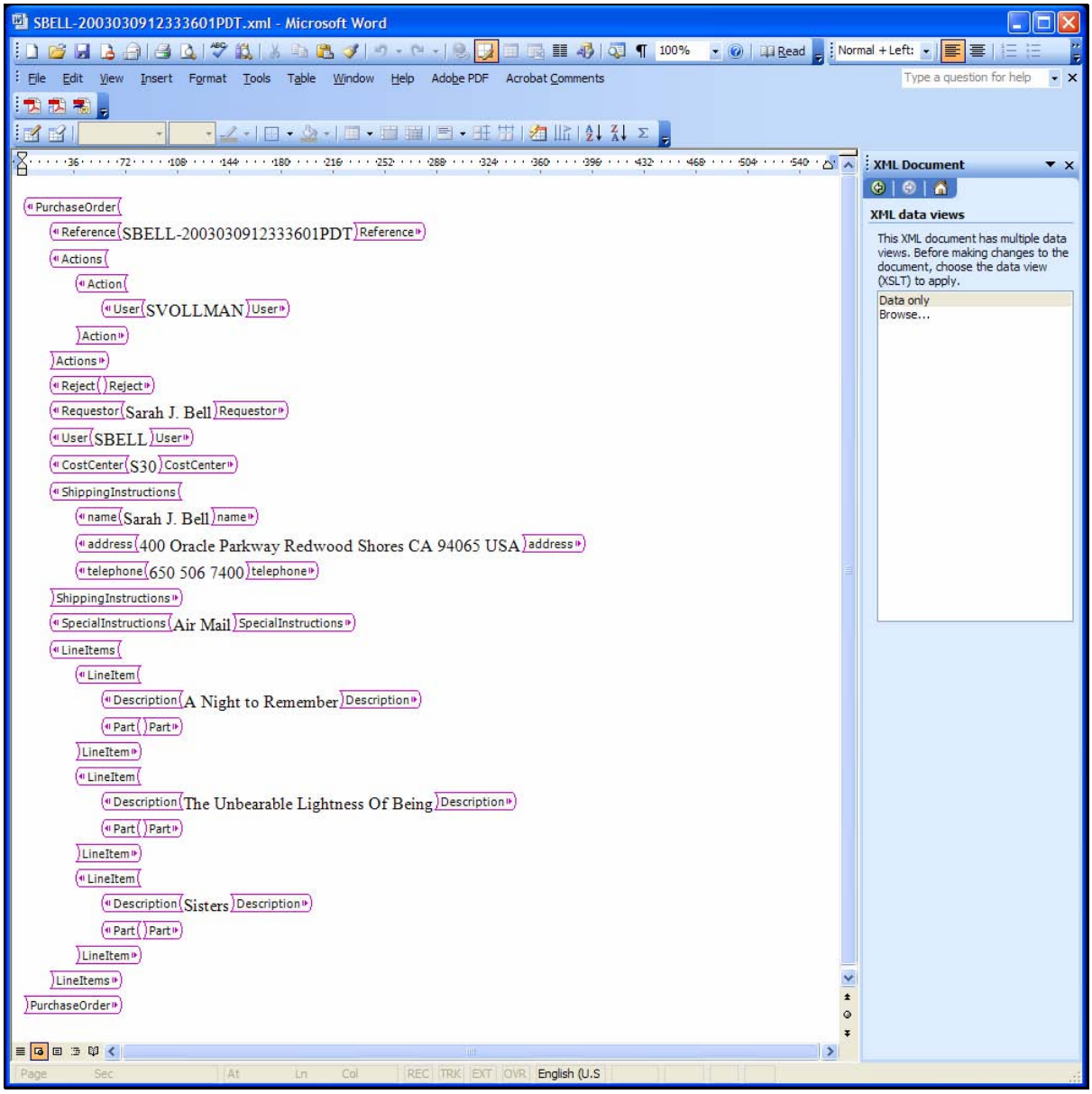
A major advantage of the Oracle XML DB repository is that it allows standard desktop applications such as Microsoft Word to access content stored in Oracle Database 11g. This step demonstrates how to use WebDAV to open and update a document stored in the Oracle XML DB repository using Microsoft Word. When Word opens the document it uses a DAV to prevent conflicting updates. This requires an authenticated connection to the database.

Click the icon to launch Microsoft Word and open the document. When Word prompts for a username and password enter the demonstration user's username and password and click OK.



- Word will use WebDAV to open the document.

By default when Word opens an XML document it choose the Web Layout view for the document. The element names appear as pink boxes. The associated text nodes appear between the pink boxes.



- In this example the document is a PurchaseOrder. PurchaseOrder documents are schema-based XML so the content of the document comes directly from table PURCHASEORDER

- Microsoft Word 2003 has a rudimentary understanding of XML documents.

- WebDAV support makes it possible for tools like Microsoft Word to directly access content stored in Oracle Database 10g. No Oracle or WebDAV specific software is required for Word to access content stored in the Oracle XML DB repository.

Microsoft has published an XML schema, known as WordProcessingML, which specifies an XML format for storing Word documents. This format allows Word documents to be stored as XML, rather than using Microsoft's proprietary DOC format. There is no loss of fidelity when a Word Document is stored using the XML format. In Microsoft Office 2007 the WordProcessingML format becomes the default storage model for Microsoft Word documents.

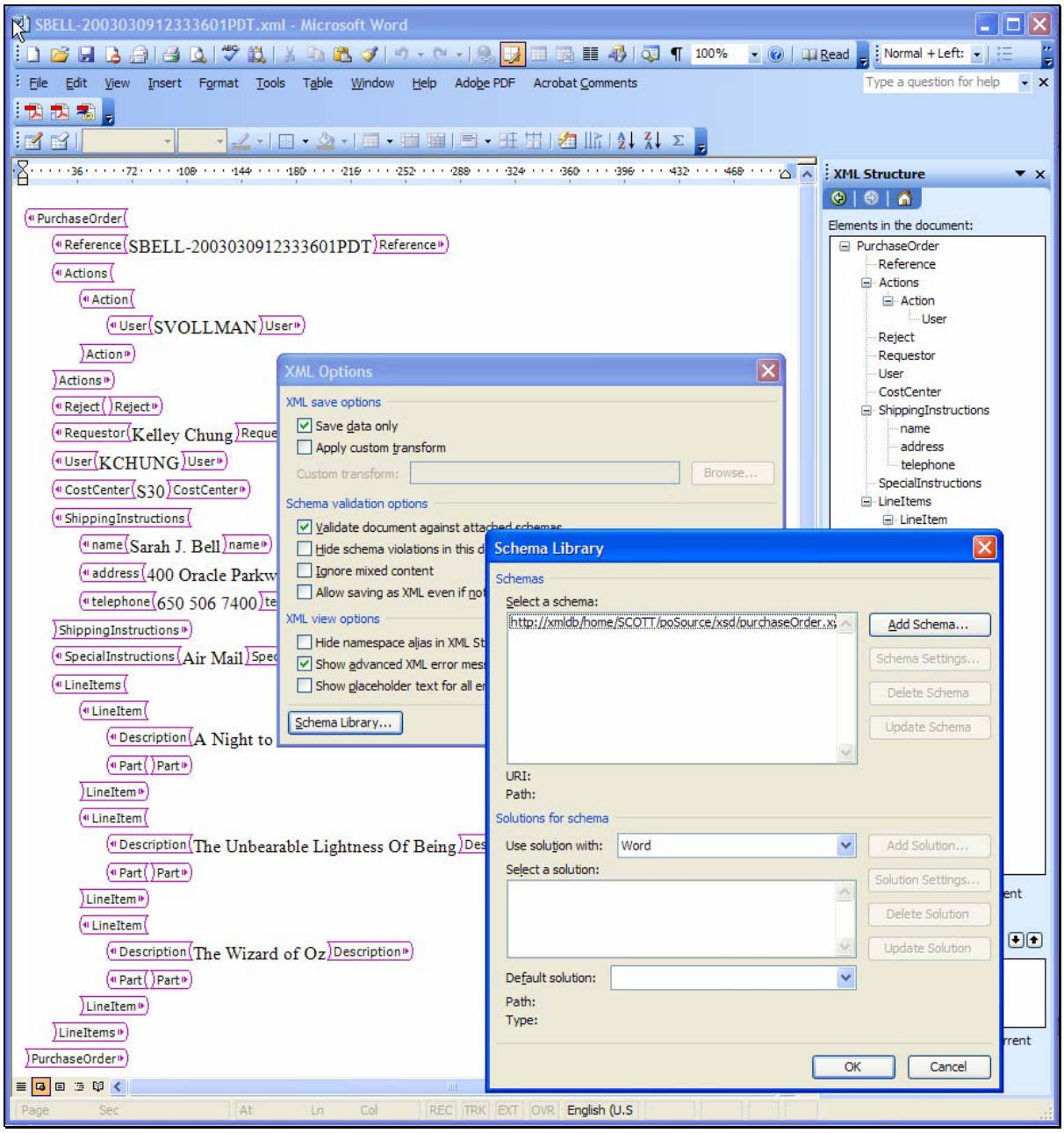
The WordProcessingML format will allow many different kinds of word processing and XML editing tools to be used to share and co-author content. This will accelerate the rate at which XML becomes the primary storage medium for all kinds of content.

Microsoft has proposed that the XML schemas for Word, Excel etc, collectively known as "The Open Office XML File format", be recognized as an open standard. The proposal has been presented and sponsored by ECMA, the European Computer Manufacturer's association.

The Open source community and Open Office.org project has proposed an alternative standard, known as "OpenOffice.org XML**"**, based on OpenDoc. This standard has been developed under the OASIS umbrella. Unfortunately OpenOffice.org XML does not use XML Schema to define its document format; it uses the RelaxNG specification for this purpose.

Word can show the structure of an XML document in the Task pane. If the Task Pane is not visible enable it by clicking the View menu and selecting the Task Pane option. To view the structure of the XML document click the drop down list box at the top of the Task Pane and select option XML Structure. Word will display the structure of the XML document in the Task Pane.

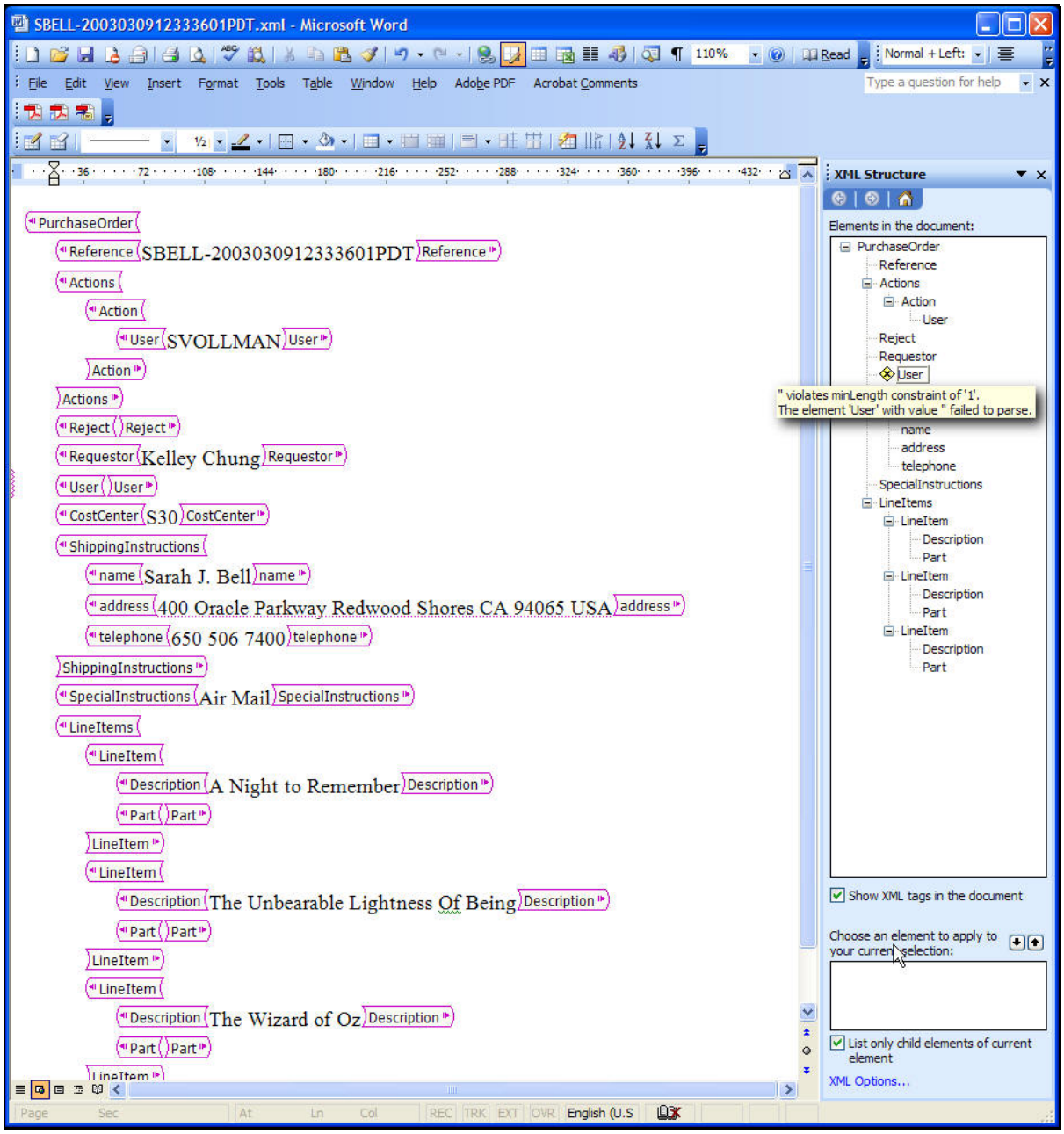
The latest versions of Microsoft Word provide support for XML Schema validation. An XML Schema must be attached (registered), before Microsoft Word can use it for validation. To attach an XML Schema to Microsoft Word, select the XML Structure option in the Task Pane menu and click the XML Options link at the bottom of the Task Pane. Word will open the XML Options dialog. Click the Schema Library button to see the list of available XML schemas. If the required XML Schema is not listed use the Add Schema button to attach the XML schema to Microsoft Word. This feature does not appear to support XML schemas that do not specify a targetNamespace, such as the PurchaseOrder XML schema used in this demonstration.



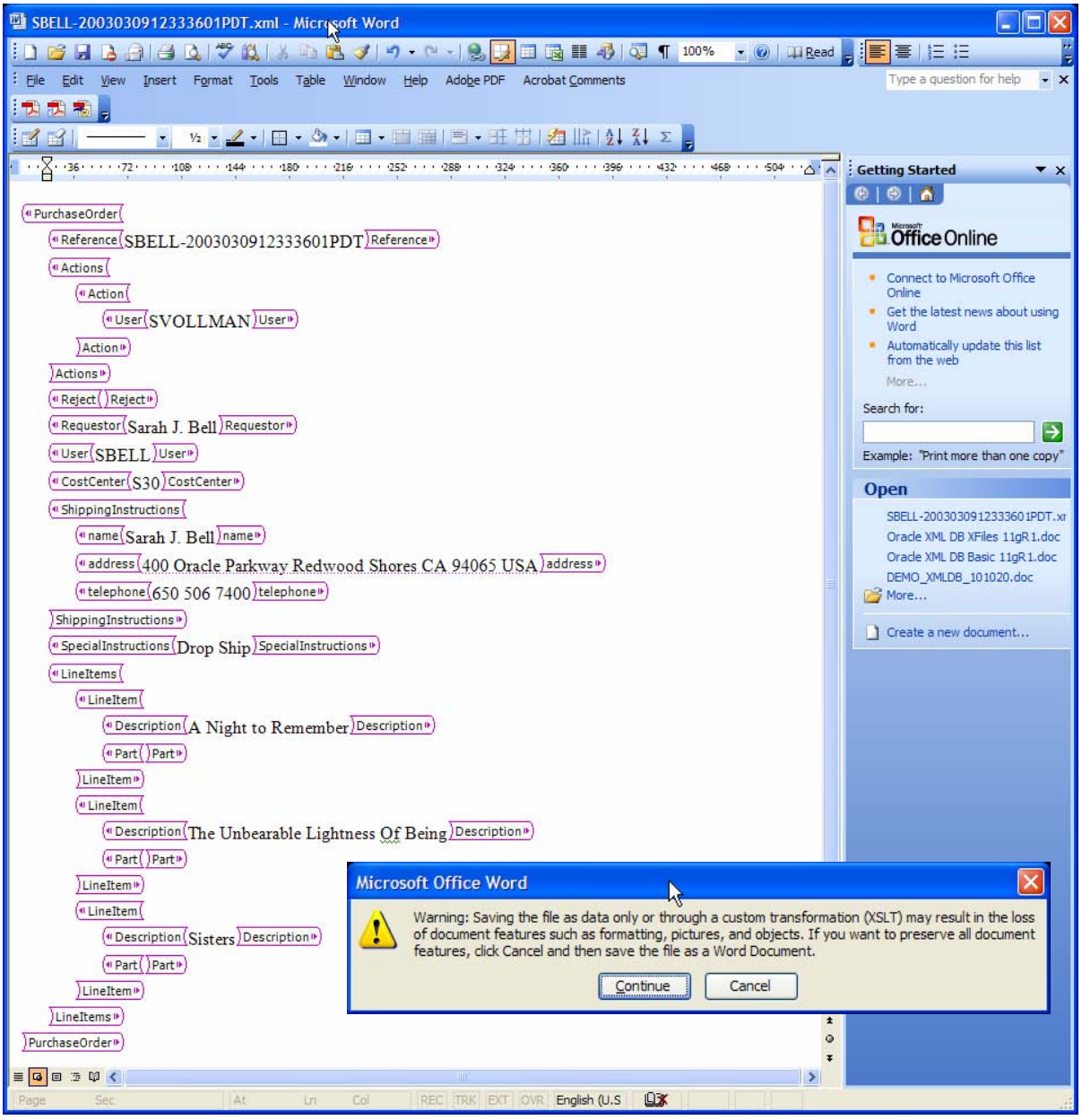
When Word opens an instance document it will automatically attach the XML Schema and display the Schema URL or alias in the window at the bottom of the XML structure dialog. If the XML Schema is not automatically attached, open the Tools menu, and select option Templates and Add-Ins. Select the XML Schema tab on the Template and Add-Ins dialog and manually select the XML Schema. Word should then be able to validate the document against the XML Schema.

Remove the content of element User. If the XML Schema is attached correctly word will immediately place a yellow diamond icon containing a black X next to element User in the task pane. This indicates

that the content of element User is not schema-valid. Place the mouse on the icon to display an error message that explains why the document in not schema-valid.



Undo the change to element User and change the value of element Special Instructions to Drop Ship. Click Save. Word will prompt for confirmation that the document should be saved as data. Click Continue. Do not close the document after saving it.



- Word uses the WebDAV protocol and an HTTP POST operation to write the updated document back to the Oracle XML DB repository. The content of the document is used to update the row in table PURCHASERODER that corresponds to this document.

- Schema aware XML Editors are available from many sources including Microsoft, JustSystem and Altova. These editors make it very easy to create schema-valid XML content.

- WebDAV support makes it easy for standard tools be access and edit content stored in the Oracle XML DB repository.

- Each WebDAV operation (POST, PUT) is treated as a separate atomic transaction. This means that changes made to a document using WebDAV are visible to other users as soon as the operation is complete.

- Only WebDAV enabled tools such as the latest versions of Microsoft Word (2000, XP, 2003 and 2007) can edit content stored in the Oracle XML DB repository. It is not possible to edit content in the Oracle XML DB repository using editors like NOTEPAD or WORDPAD, since these tools do not support the WebDAV or FTP protocol.

- Many other vendors, such as Altova, Adobe and Macromedia now provide WebDAV support in their products. All of these products can work directly with content stored in the Oracle XML DB repository.

5.1.4 View Updated Document (XQuery)

The URL access metaphor is also available through the XQuery function fn:doc. Oracle XML DB treats the URL passed to fn:doc as a path from the root of the Oracle XML DB repository. This step uses fn:doc to access the content the updated document. Click the icon to launch the XML DB demonstration framework and run the SQL script.

Command Output

```
set long 10000
set pagesize 250
set echo on
--
select XMLQuery( 'fn:doc("/home/SCOTT/purchaseOrders/2003/Mar/SBELL-2003030912333601PDT.xml")' returning content) XML
  from dual
/
```

XML

```
<PurchaseOrder NS0:noNamespaceSchemaLocation="http://xmldb:80/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
  <Reference>SBELL-2003030912333601PDT</Reference>
  <Actions>
    <Action>
      <User>SVOLLMAN</User>
    </Action>
  </Actions>
  <Reject/>
  <Requestor>Kelley Chung</Requestor>
  <User>KCHUNG</User>
  <CostCenter>S30</CostCenter>
  <ShippingInstructions>
    <name>Sarah J. Bell</name>
    <address>400 Oracle Parkway Redwood Shores CA 94065 USA</address>
    <telephone>650 506 7400</telephone>
  </ShippingInstructions>
  <SpecialInstructions>Drop Ship</SpecialInstructions>
  <LineItems>
    <LineItem ItemNumber="1">
      <Description>A Night to Remember</Description>
      <Part Id="715515009058"UnitPrice="39.95"Quantity="2"/>
    </LineItem>
    <LineItem ItemNumber="2">
      <Description>The Unbearable Lightness Of Being</Description>
      <Part Id="37429140222"UnitPrice="29.95"Quantity="2"/>
    </LineItem>
    <LineItem ItemNumber="3">
      <Description>The Wizard of Oz</Description>
      <Part Id="715515011020"UnitPrice="29.95"Quantity="4"/>
    </LineItem>
  </LineItems>
</PurchaseOrder>
```

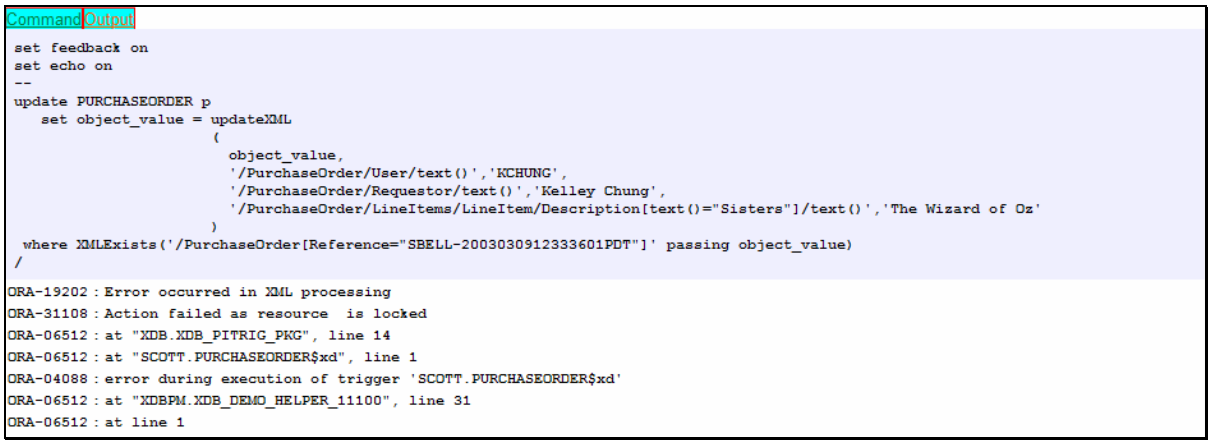
```
Result : 1 Rows Selected.
```

- Changes made using Word are visible from SQL as soon as the document is saved.

5.1.5 Show DAV Locking

The DAV specification defines a locking model that ensures that two sessions cannot save conflicting changes to a document. Oracle XML DB supports the DAV locking model. The DAV locking enforced for updates performed via SQL as well us updates via protocols. This step uses SQL to update the document that is currently opened for editing in Word. When word opens a document for editing it requests a DAV lock on the document to prevent conflicting updates. The DAV lock is released when the document is closed.

Click the icon to launch the XML DB demonstration framework and run the SQL script.



Command Output

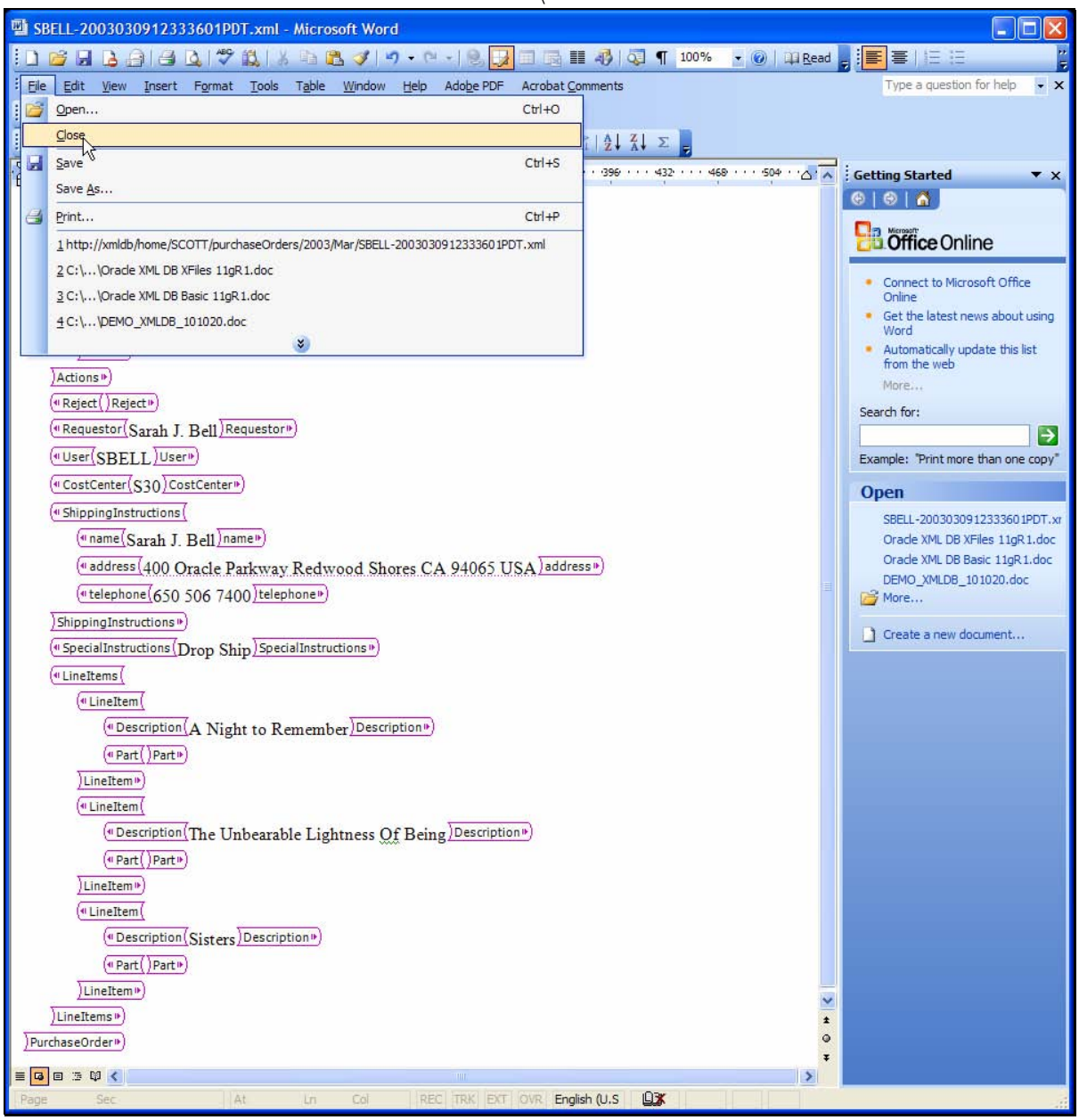
```
set feedback on
set echo on
--
update PURCHASEORDER p
   set object_value = updateXML
                      (
                        object_value,
                        '/PurchaseOrder/User/text()','KCHUNG',
                        '/PurchaseOrder/Requestor/text()','Kelley Chung',
                        '/PurchaseOrder/LineItems/LineItem/Description[text()="Sisters"]/text()','The Wizard of Oz'
                      )
 where XMLExists('/PurchaseOrder[Reference="SBELL-20030309123336O1PDT"]' passing object_value)
/
ORA-19202 : Error occurred in XML processing
ORA-31108 : Action failed as resource  is locked
ORA-06512 : at "XDB.XDB_PITRIG_PKG", line 14
ORA-06512 : at "SCOTT.PURCHASEORDER$xd", line 1
ORA-04088 : error during execution of trigger 'SCOTT.PURCHASEORDER$xd'
ORA-06512 : at "XDBPM.XDB_DEMO_HELPER_11100", line 31
ORA-06512 : at line 1
```

- Since the document is opened for editing in Word, the document is protected by a DAV lock. This means that it cannot be updated from SQL. ORA-31108 is thrown if an attempt is made to update a DAV locked document from SQL.

- Oracle XML DB provides full support for DAV locks
- The Oracle XML DB uses DAV locking to ensure that XML editors and other tools compliant with the DAV specification cannot make conflicting updates to content stored in the Oracle XML DB repository.
- When an XMLType table is created, the XML Schema registration process creates triggers that enforce the DAV Locking model when updates are performed using SQL.

5.1.6 Close Document

This step releases the DAV Lock by closing the document in Microsoft Word. Double-click the icon to return to Microsoft Word. Close the document by opening the File menu and selecting Close. This will end the editing session for this document and Microsoft office will close the document and release the DAV lock.


SBELL-20030309123336O1PDT.xml - Microsoft Word
File Edit View Insert Format Tools Table Window Help Adobe PDF Acrobat Comments
Open... Ctrl+O
Close
Save Ctrl+S
Save As...
Print... Ctrl+P
1 http://xmldb/home/SCOTT/purchaseOrders/2003/Mar/SBELL-20030309123336O1PDT.xml
2 C:\...\Oracle XML DB XFiles 11gR1.doc
3 C:\...\Oracle XML DB Basic 11gR1.doc
4 C:\...\DEMO_XMLDB_101020.doc
Getting Started
Office Online
Connect to Microsoft Office Online
Get the latest news about using Word
Automatically update this list from the web
More...
Search for:
Example: "Print more than one copy"
Open
SBELL-20030309123336O1PDT.xr
Oracle XML DB XFiles 11gR1.doc
Oracle XML DB Basic 11gR1.doc
DEMO_XMLDB_101020.doc
More...
Create a new document...
Actions
Reject Reject
Requestor Sarah J. Bell Requestor
User SBELL User
CostCenter S30 CostCenter
ShippingInstructions
name Sarah J. Bell name
address 400 Oracle Parkway Redwood Shores CA 94065 USA address
telephone 650 506 7400 telephone
ShippingInstructions
SpecialInstructions Drop Ship SpecialInstructions
LineItems
LineItem
Description A Night to Remember Description
Part Part
LineItem
LineItem
Description The Unbearable Lightness Of Being Description
Part Part
LineItem
LineItem
Description Sisters Description
Part Part
LineItem
LineItems
PurchaseOrder

5.1.7 Update Document

The step shows that the SQL update succeeds once the DAV lock is released.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

Command Output

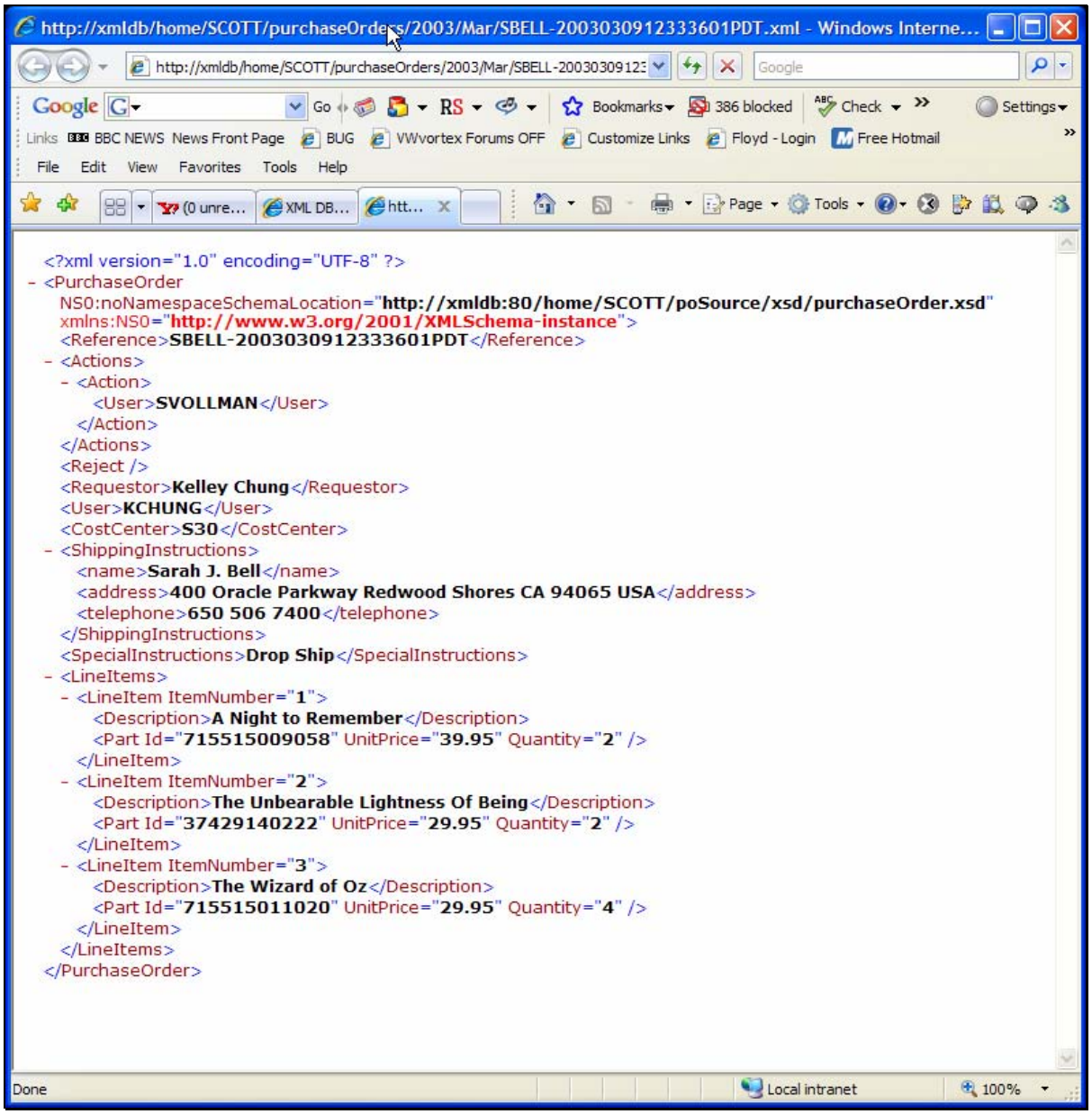
```
set feedback on
set echo on
--
update PURCHASEORDER p
   set object_value = updateXML
                      (
                        object_value,
                        '/PurchaseOrder/User/text()','KCHUNG',
                        '/PurchaseOrder/Requestor/text()','Kelley Chung',
                        '/PurchaseOrder/LineItems/LineItem/Description[text()="Sisters"]/text()','The Wizard of Oz'
                      )
 where XMLExists('/PurchaseOrder[Reference="SBELL-20030309123336O1PDT"]' passing object_value)
/

1 row Updated.
```

- Once the DAV lock is released the update succeeds.
- An update performed using UpdateXML is far more efficient that anupdate performed using Microsoft Word.
- When a document is saved using an editor like Microsoft Word the entire document is sent back to the database. The database does not know which parts of the document have been updated. The database has to blindly replace the existing document with the new document. This means that all existing data must be deleted before the new content is inserted. This is process expensive in terms of parsing, CPU and memory utilization as well as the amount of log generated.
- Using Update XML to update a document allows the database to determine exactly which bits of the document are being updated. With object-relational storage it re-writes the updateXML operator into a direct update of the underlying tables. With Binary XML storage and Oracle Secure Files only the part of the LOB that contain changed data are updated. This reduces the amount of parsing and overhead generated by the update.

5.1.8 View updated document (HTTP)

This step show that the changes made using Microsoft Word and SQL are visible when the document is opened in the browser. Click the icon to launch Internet Explorer. Internet Explorer will use HTTP to fetch the contents of the document. If the document opens with some other application use the Folder Options feature of Windows Explorer to adjust the file association for 'XML' files.



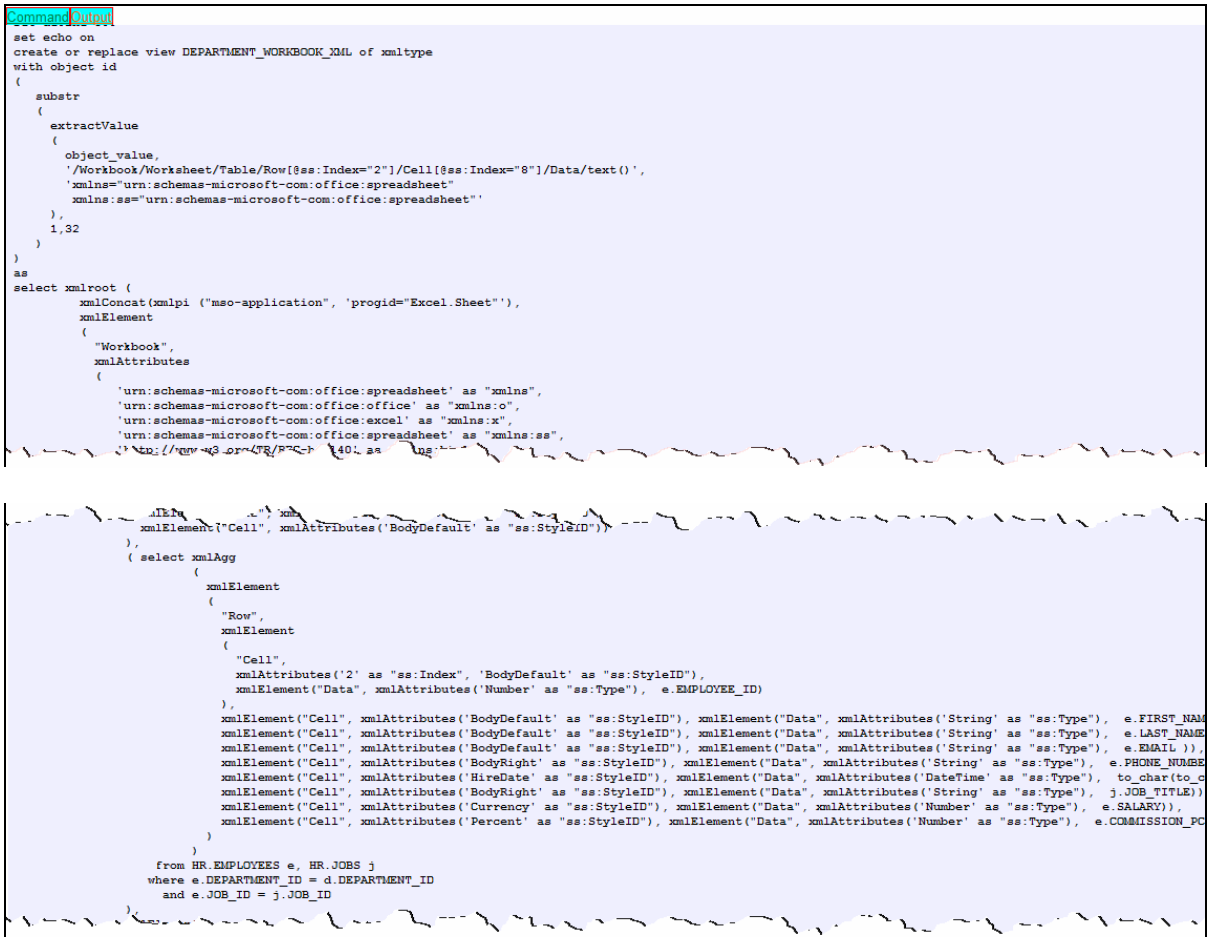
- If prompted for a username and password enter the demonstration user's username and password and click OK.

5.2.1 Create Spreadsheets

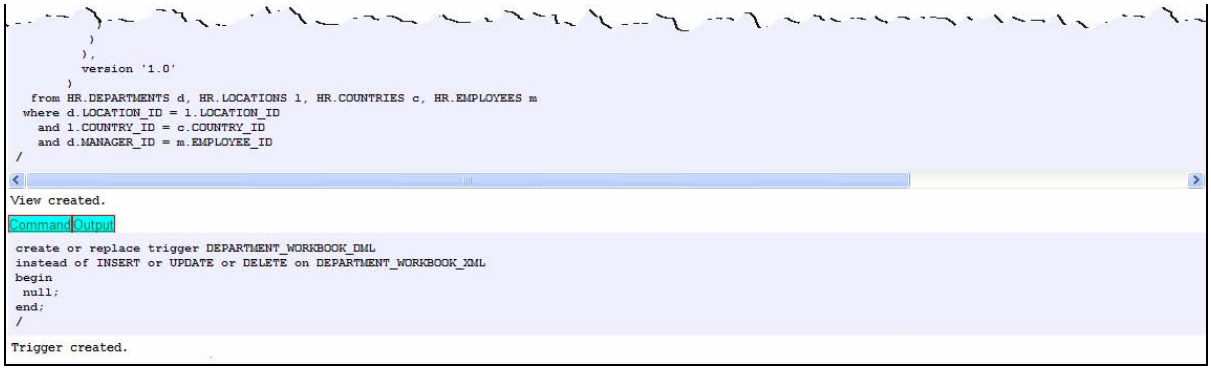
Microsoft has published an XML schema, known as SpreadsheetML, which specifies an XML format for storing Excel spreadsheets. This format allows Excel documents to be stored as XML, rather than using Microsoft's proprietary XLS format. There is no loss of fidelity when am Excel spreadsheet is stored using the XML format. In Microsoft Office 2007 the SpreadsheetML format becomes the default storage model for Microsoft Excel documents.

Applications that can generate XML documents that comply with the SpreadsheetML XML schema will be able to create rich spreadsheets that Excel can understand. Unlike a CSV file, a SpreadsheetML document can contain formulae and formatting information.

This section shows how to use SQL/XML publishing functions to publish relational data directly to Microsoft Excel. It is a very good example of using the SQL/XML operators to generate a complex XML document. Click the icon to launch the XML DB demonstration framework and run the SQL script.



```
Command Output
set echo on
create or replace view DEPARTMENT_WORKBOOK_XML of xmltype
with object id
(
   substr
   (
     extractValue
     (
       object_value,
       '/Workbook/Worksheet/Table/Row[@ss:Index="2"]/Cell[@ss:Index="8"]/Data/text()',
       'xmlns="urn:schemas-microsoft-com:office:spreadsheet"
        xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"'
     ),
     1,32
   )
)
as
select xmlroot (
         xmlConcat(xmlpi ("mso-application", 'progid="Excel.Sheet"'),
         xmlElement
         (
           "Workbook",
           xmlAttributes
           (
              'urn:schemas-microsoft-com:office:spreadsheet' as "xmlns",
              'urn:schemas-microsoft-com:office:office' as "xmlns:o",
              'urn:schemas-microsoft-com:office:excel' as "xmlns:x",
              'urn:schemas-microsoft-com:office:spreadsheet' as "xmlns:ss",
              'http://www.w3.org/TR/REC-html40' as "xmlns:html"
           ),
           xmlElement("Cell", xmlAttributes('BodyDefault' as "ss:StyleID"))
         ),
         ( select xmlAgg
                 (
                   xmlElement
                   (
                     "Row",
                     xmlElement
                     (
                       "Cell",
                       xmlAttributes('2' as "ss:Index", 'BodyDefault' as "ss:StyleID"),
                       xmlElement("Data", xmlAttributes('Number' as "ss:Type"),  e.EMPLOYEE_ID)
                     ),
                     xmlElement("Cell", xmlAttributes('BodyDefault' as "ss:StyleID"), xmlElement("Data", xmlAttributes('String' as "ss:Type"),  e.FIRST_NAM
                     xmlElement("Cell", xmlAttributes('BodyDefault' as "ss:StyleID"), xmlElement("Data", xmlAttributes('String' as "ss:Type"),  e.LAST_NAME
                     xmlElement("Cell", xmlAttributes('BodyDefault' as "ss:StyleID"), xmlElement("Data", xmlAttributes('String' as "ss:Type"),  e.EMAIL )),
                     xmlElement("Cell", xmlAttributes('BodyRight' as "ss:StyleID"), xmlElement("Data", xmlAttributes('String' as "ss:Type"),  e.PHONE_NUMBE
                     xmlElement("Cell", xmlAttributes('HireDate' as "ss:StyleID"), xmlElement("Data", xmlAttributes('DateTime' as "ss:Type"),  to_char(to_c
                     xmlElement("Cell", xmlAttributes('BodyRight' as "ss:StyleID"), xmlElement("Data", xmlAttributes('String' as "ss:Type"),  j.JOB_TITLE))
                     xmlElement("Cell", xmlAttributes('Currency' as "ss:StyleID"), xmlElement("Data", xmlAttributes('Number' as "ss:Type"),  e.SALARY)),
                     xmlElement("Cell", xmlAttributes('Percent' as "ss:StyleID"), xmlElement("Data", xmlAttributes('Number' as "ss:Type"),  e.COMMISSION_PC
                   )
                 )
            from HR.EMPLOYEES e, HR.JOBS j
           where e.DEPARTMENT_ID = d.DEPARTMENT_ID
             and e.JOB_ID = j.JOB_ID
         ),
```



```
        )
       ),
       version '1.0'
      )
  from HR.DEPARTMENTS d, HR.LOCATIONS l, HR.COUNTRIES c, HR.EMPLOYEES m
 where d.LOCATION_ID = l.LOCATION_ID
   and l.COUNTRY_ID = c.COUNTRY_ID
   and d.MANAGER_ID = m.EMPLOYEE_ID
/

View created.

Command Output

create or replace trigger DEPARTMENT_WORKBOOK_DML
instead of INSERT or UPDATE or DELETE on DEPARTMENT_WORKBOOK_XML
begin
 null;
end;
/

Trigger created.
```

- The first step creates XMLType view DEPARTMENT_WORKBOOK_XML.

- View DEPARTMENT_WORKBOOK_XML will contain one row for each row in table DEPARTMENTS.

- The structure of the document generated by the SQL/XML operators is based on Microsoft's SpreadsheetML XML Schema. The documents are valid instances of the SpreadsheetML XML schema.

- The information that controls the look and feel of the spreadsheet in provided by a static library of style definitions. The library is called **Styles.xml**. The document is located in folder **Workbooks**. The content is of the library is included into the view using operator xdbUriType. The look and feel of the spreadsheet can be altered by changing the contents of this document.

- The CREATE_VIEW statement is large since it generates all of the boilerplate XML required by a valid SpreadsheetML document.

- The core of the CREATE VIEW statement generates a collection of Row elements. The collection will contain one Row element for each row in table EMPLOYEE that belongs to the current Department.

- Element Row contains a collection of Cell elements. There will be one Cell for each column supplying data to the spreadsheet.

- Element Cell contains attribute Style and element Data. Attribute Style contains the formatting rules for the cell. Element Data contains attribute Type and the data value for the cell. Attribute Type contains the typing information for the data value.

- The next step creates trigger DEPARTMENT_WORKBOOK_XML. This trigger ensures that insert, update and delete operations on the view are not propagated to the underlying tables. A more complex trigger could be used the ability to update the underlying tables by saving the spreadsheet.

The code that generates the collection of Row elements is shown below:

```
select xmlAgg
       (
        xmlElement
        (
          "Row",
          xmlElement
          (
            "Cell",
            xmlAttributes('2' as "ss:Index", 'BodyDefault' as "ss:StyleID"),
            xmlElement("Data", xmlAttributes('Number' as "ss:Type"), e.EMPLOYEE ID)
          ),
          xmlElement
          (
            "Cell",
             xmlAttributes('BodyDefault' as "ss:StyleID"),
             xmlElement("Data", xmlAttributes('String' as "ss:Type"), e.FIRST NAME)
          ),
          xmlElement
          (
            "Cell",
            xmlAttributes('BodyDefault' as "ss:StyleID"),
            xmlElement("Data", xmlAttributes('String' as "ss:Type"), e.LAST NAME)
          ),
          xmlElement
          (
             "Cell", xmlAttributes('BodyDefault' as "ss:StyleID"),
             xmlElement("Data", xmlAttributes('String' as "ss:Type"), e.EMAIL)
          ),
          xmlElement
          (
            "Cell", xmlAttributes('BodyRight' as "ss:StyleID"),
            xmlElement("Data", xmlAttributes('String' as "ss:Type"), e.PHONE NUMBER)
          ),
          xmlElement
          (
            "Cell", xmlAttributes('HireDate' as "ss:StyleID"),
             xmlElement
             (
               "Data",
               xmlAttributes('DateTime' as "ss:Type"),
               to char(to char(e.HIRE DATE,'YYYY-MM-DD"T00:00:00.000"'))
             )
           ),
           xmlElement
           (
             "Cell",
             xmlAttributes('BodyRight' as "ss:StyleID"),
             xmlElement("Data", xmlAttributes('String' as "ss:Type"), j.JOB TITLE)
           ),
           xmlElement
           (
             "Cell",
             xmlAttributes('Currency' as "ss:StyleID"),
             xmlElement("Data", xmlAttributes('Number' as "ss:Type"), e.SALARY)
           ),
           xmlElement
           (
             "Cell",
             xmlAttributes('Percent' as "ss:StyleID"),
             xmlElement("Data", xmlAttributes('Number' as "ss:Type"), e.COMMISSION PCT)
           )
         )
       )
   from HR.EMPLOYEES e, HR.JOBS j
        where e.DEPARTMENT ID = d.DEPARTMENT ID
        and e.JOB ID = j.JOB ID
```

A resource is created in folder /home/***USER***/Workbooks/Departments for each row in view DEPARTMENT_WORKBOOK_XML.

Command Output

```
--

declare
  cursor getDepartments is
    select ref(d) XMLREF,
           substr
           (
             extractValue
             (
               object_value,
               '/Workbook/Worksheet[1]/@ss:Name',
               'xmlns="urn:schemas-microsoft-com:office:spreadsheet"
               xmlns:ss="urn:schemas-microsoft-com:office:spreadsheet"'
             ),
             1,32
           ) NAME
      from DEPARTMENT_WORKBOOK_XML d;
  res boolean;
  targetFolder varchar2(1024) :=  '/home/SCOTT/Workbooks/Departments';
begin
  if dbms_xdb.existsResource(targetFolder) then
     dbms_xdb.deleteResource(targetFolder,dbms_xdb.DELETE_RECURSIVE);
  end if;
  res := dbms_xdb.createFolder(targetFolder);
  for dept in getDepartments loop
    res := DBMS_XDB.createResource(targetFolder || '/' || dept.NAME || '.xml', dept.XMLREF);
  end loop;
end;
/
```

```
Trigger created.
```

Command Output

```
--
select path
  from path_view
 where under_path(res,'/home/SCOTT/Workbooks/Departments') = 1
/
```

PATH
/home/SCOTT/Workbooks/Departments/Accounting.xml
/home/SCOTT/Workbooks/Departments/Administration.xml
/home/SCOTT/Workbooks/Departments/Executive.xml
/home/SCOTT/Workbooks/Departments/Finance.xml
/home/SCOTT/Workbooks/Departments/Human Resources.xml
/home/SCOTT/Workbooks/Departments/IT.xml
/home/SCOTT/Workbooks/Departments/Marketing.xml
/home/SCOTT/Workbooks/Departments/Public Relations.xml
/home/SCOTT/Workbooks/Departments/Purchasing.xml
/home/SCOTT/Workbooks/Departments/Sales.xml
/home/SCOTT/Workbooks/Departments/Shipping.xml

```
Result : 11 Rows Selected.
```

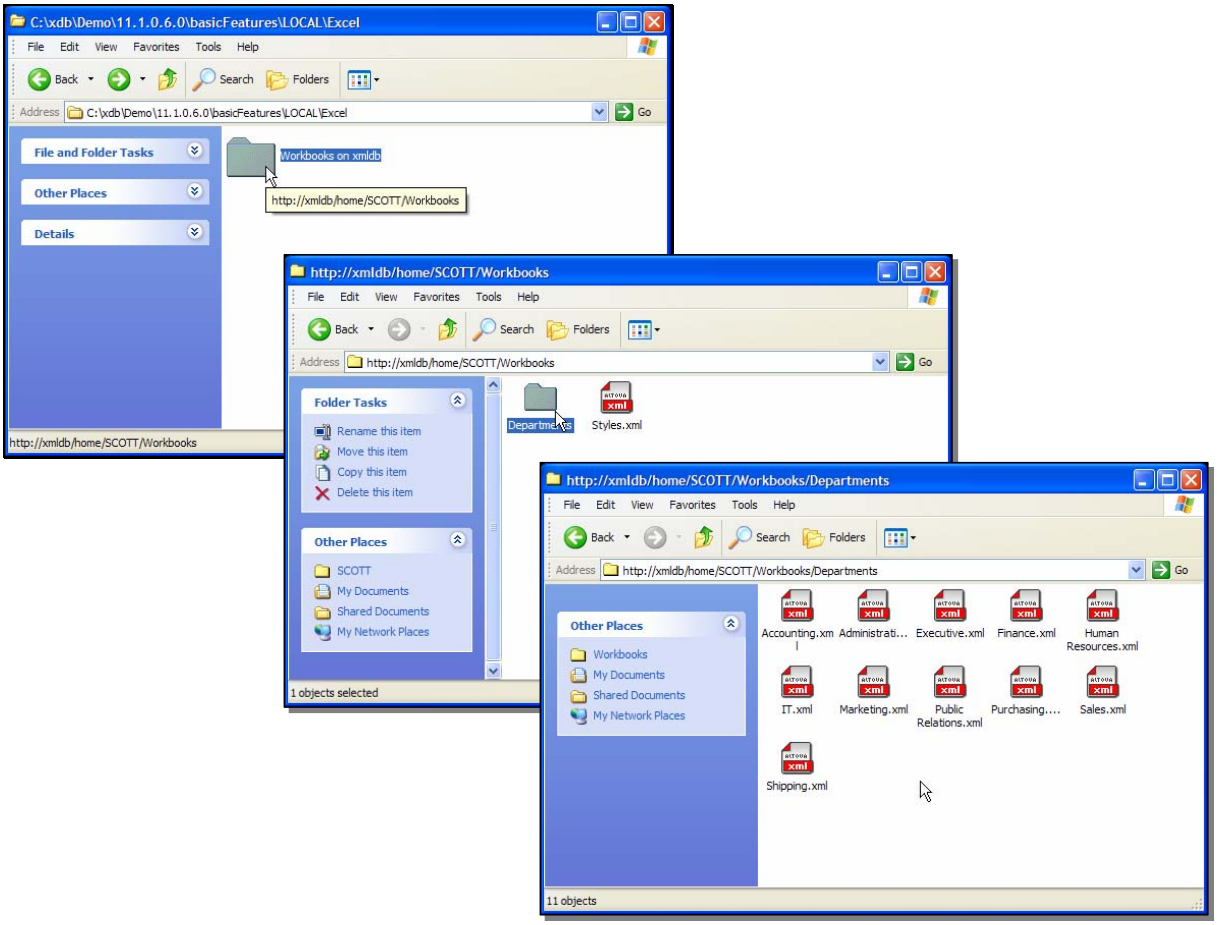
5.2.2 View Spreadsheets

This step uses Microsoft Windows Explorer to view the content of the spreadsheets created in the previous step.

Right click the icon and select explore to open a new window containing the local folder Excel. This folder contains a shortcut called "Workbooks on ***hostname***". This is a link to folder Workbooks, in the demonstration user's home folder on the Oracle XML DB repository Double click the short cut to view the content of the remote folder. If prompted for a username and password enter the demonstration user's username and password and click OK.

The folder contains a folder called Departments and a document called Styles.xml. Double click the Departments folder to view its contents.

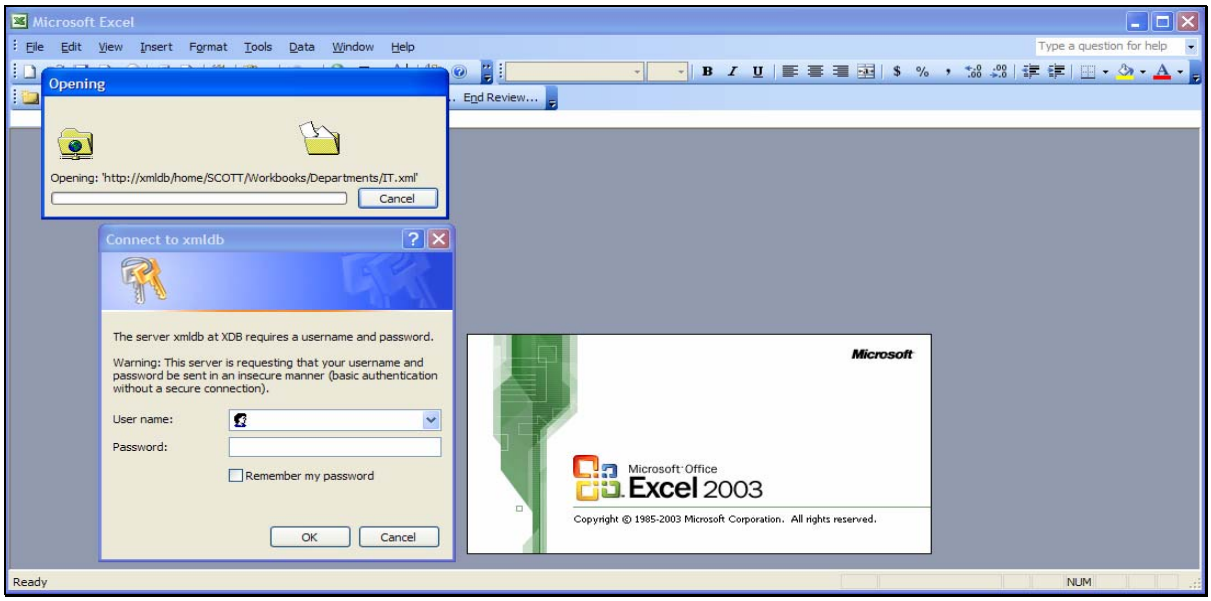
This folder contains the set of XML documents generated by the previous step. There is one document for each row in the view DEPARTMENT_WORKBOOK_XML. The documents appears as XML documents when viewed in Windows Explorer



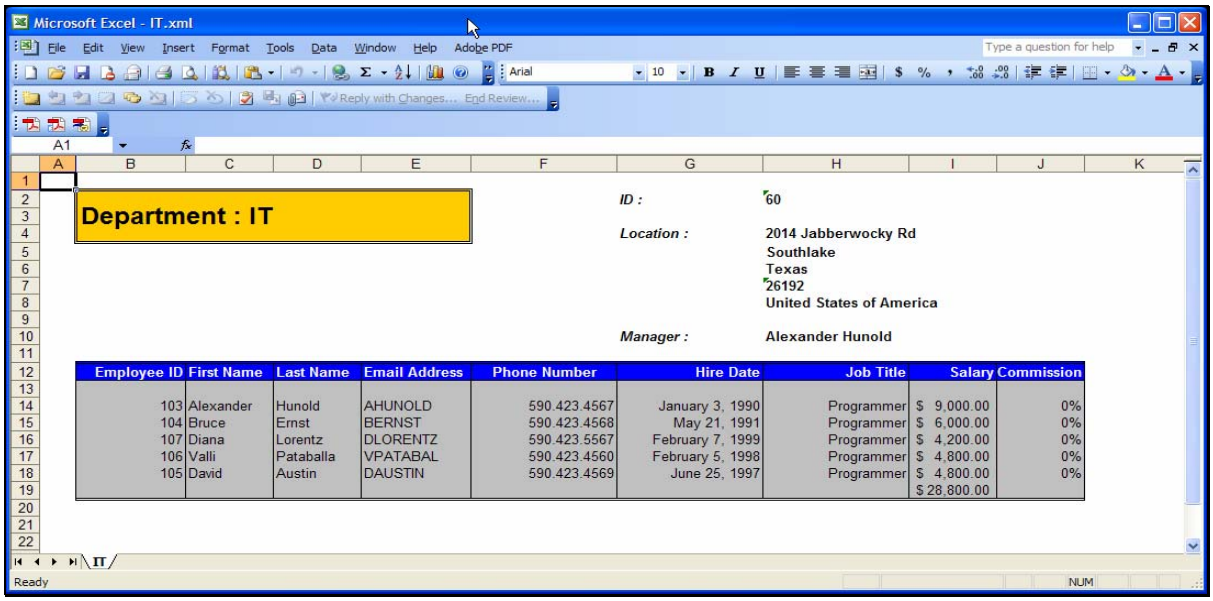
5.2.3 Open Spreadsheet (IT)

This step shows the view contains valid SpreadsheetML documents that can be opened using Microsoft Excel. When Excel opens the document it uses a DAV to prevent conflicting updates. This requires an authenticated connection to the database.

Click the icon to launch Microsoft Excel and open the document. When Excel prompts for a username and password enter the demonstration user's username and password and click OK.



- Excel will use WebDAV to open the document.

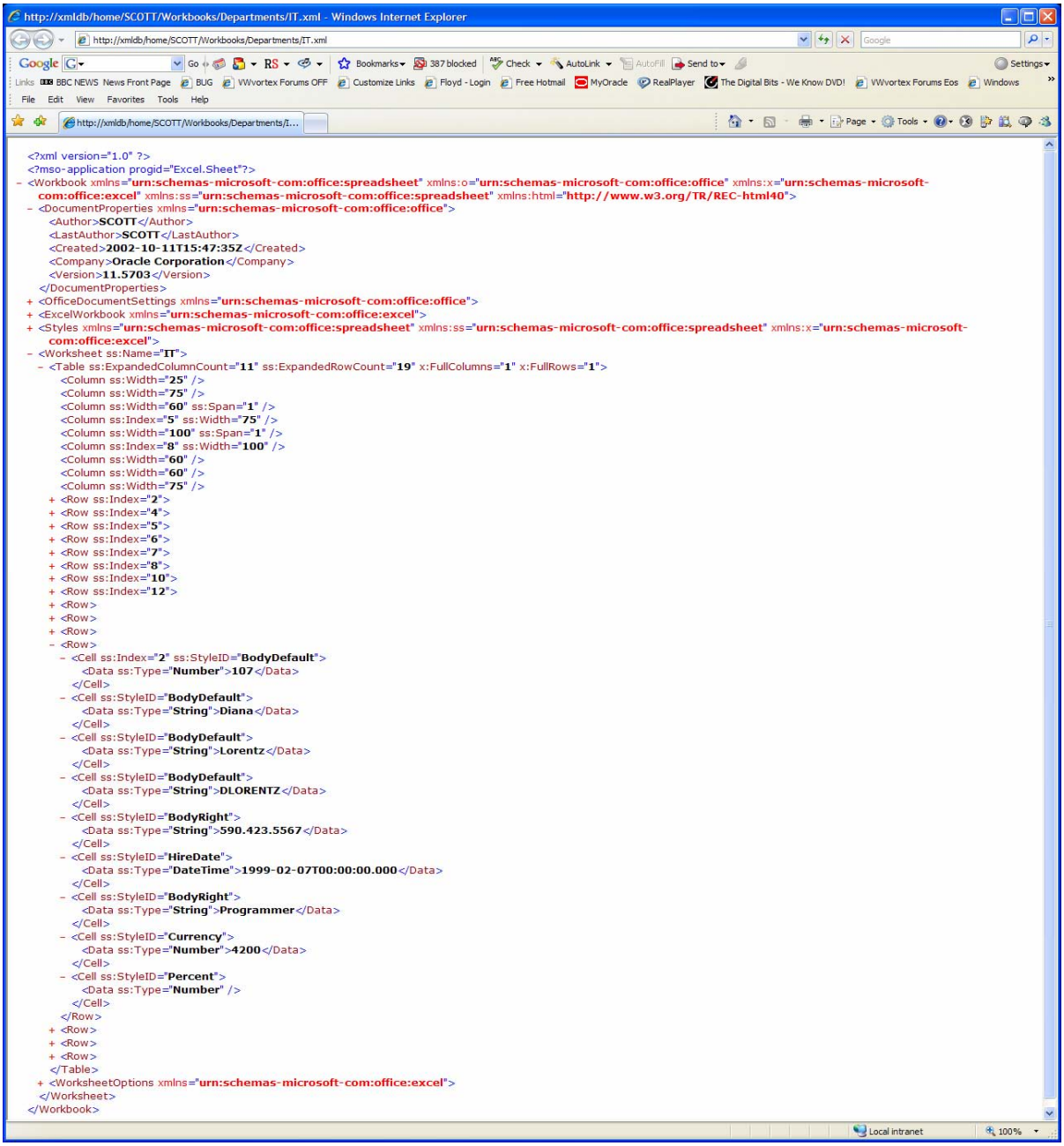


- Excel recognizes the content of the document as document as SpreadsheetML and treats it just like any other spreadsheet document.

Close Excel.

5.2.4 View Spreadsheet XML

This step shows uses Internet Explorer to view the SpreadsheetML document as XML Click the icon to launch Internet Explorer. Internet Explorer will use HTTP to fetch the contents of the document



- The content of the document is XML compliant with Microsoft's SpreadsheetML XML schema.

5.2.5 Update Employees

This step uses SQL to update the base tables for DEPARTMENT_WORKBOOK_XML. The update moves all employees with the last name of **GRANT** to the IT department.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

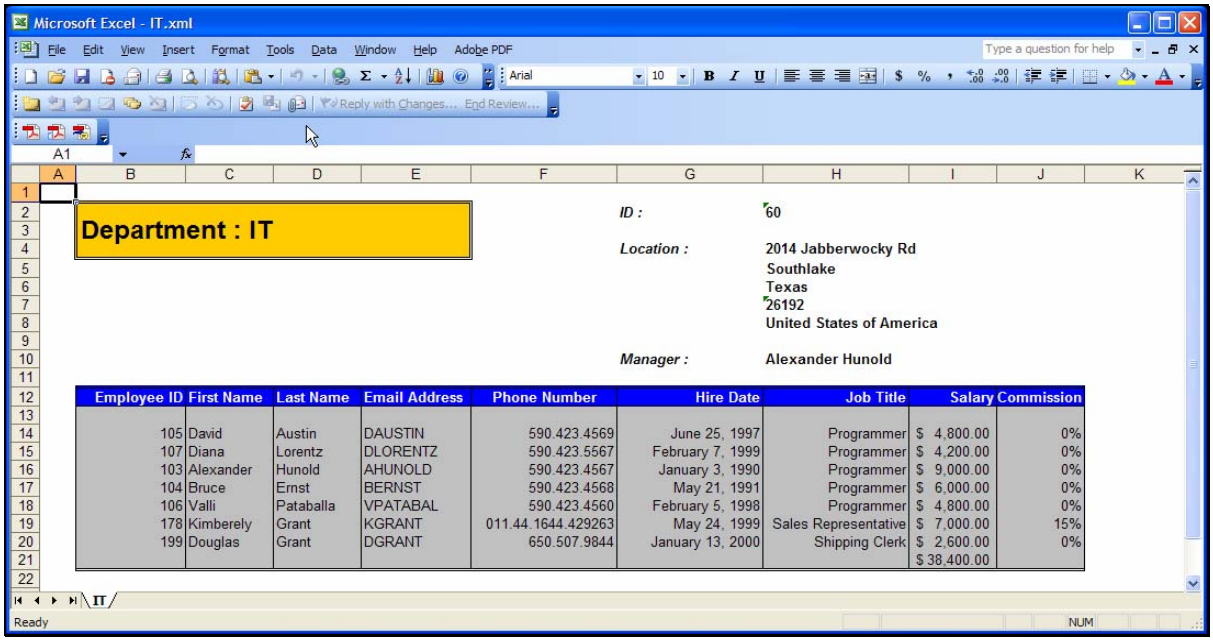
Command Output

```
set echo on
--
update hr.employees
   set department_id = (select department_id from hr.departments where department_name = 'IT')
 where last_name = 'Grant'
/
1 row Updated.
```

5.2.6 Re-Open Spreadsheet (IT)

This step shows the content of the spreadsheet represents the state of the view at the point the document is opened. When the spreadsheet is re-opened the content will reflect the update to the base tables. No intermediate steps are required to re-generate the spreadsheet when updates occur to the base tables.

Click the icon to launch Microsoft Excel and re-open the document. When Excel prompts for a username and password enter the demonstration user's username and password and click OK.



- The employees with last name **GRANT** are now included in the spreadsheet for the IT department.

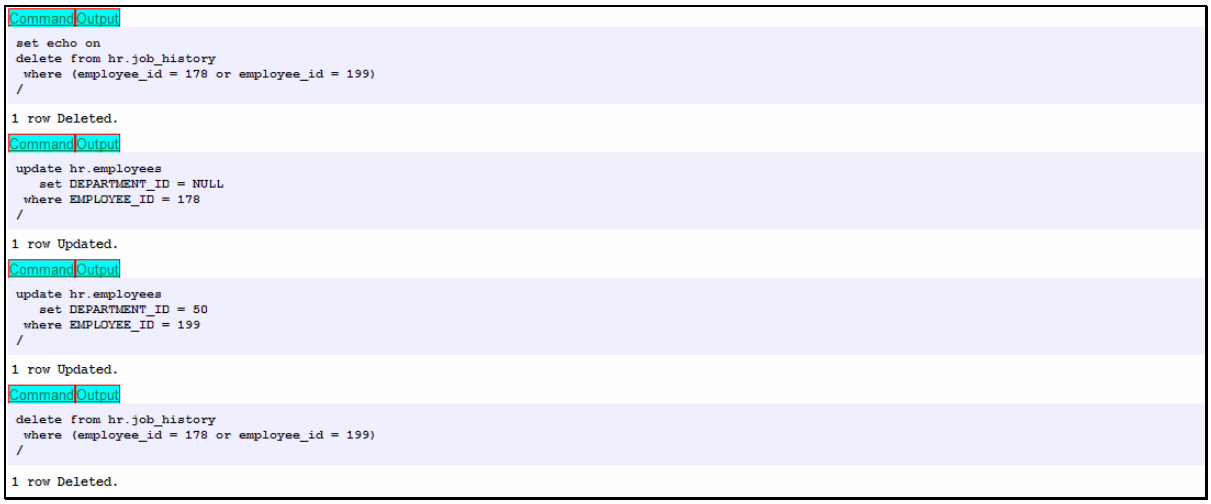
- The changes made using SQL will only be visible to Excel once they have been committed. In this case, the update was made using a SOAP request to Database Native Web Services so the transaction was committed when the request completed.

- This approach allows Excel to be used as real-time viewer for relational data.

- The deployment of these applications is very simple. Once the view has been created no additional software is required on the client or application server to enable this functionality.

5.2.7 Reset Employees

This step runs a SQL script that undoes the changes resulting from the update performed in step 5.2.5. Click the icon to launch the XML DB demonstration framework and run the SQL script.



6.1.1 Show Schema Changes

As application requirements change XML Schemas evolve over time. Oracle XML DB provides functionality that helps manage this process.

Procedure inPlaceEvolve in package DBMS_XMLSCHEMA allows simple changes to be made to an XML Schemas without unloading and reloading the data in any tables or columns that are bound to the XML Schema. This means that the time taken to perform the inPlaceEvolve is constant, it is not related to the amount of data in the tables or columns that impacted by the operation. In general, procedure inPlaceEvolve is used to make changes that do not invalidate the existing instance documents. Complete examples for inPlaceEvolve can be found in the Oracle Database 11g XML DB advanced features demonstration.

Procedure copyEvolve in package DBMS_XMLSCHEMA allows any kind of change to be made to an XML Schema. However, unlike procedure inPlaceEvolve, procedure copyEvolve unloads and reloads the data in tables or columns that are bound to the XML Schema. This means that the time taken to perform the copyEvolve processing is directly proportional to the amount of data in the tables and columns that are impacted by the operation. If the changes to the XML Schema will invalidate the existing instance documents the copyEvolve process must be provided with an XSL stylesheet that will transform the existing documents into a format that is compliant with the new XML Schema.

Operator XMLDIFF provides an XML representation of the differences between two XML documents. It can be used to analyze the differences between two XML Schemas and determine whether the changes can be made using procedure inPlaceEvolve or whether procedure copyEvolve will be necessary.

This step uses XMLDIFF to analyze the differences between the current version of the PurchaseOrder XML Schema, /home/***USER***/poSource/xsd/PurchaseOrder.xsd and the new version of the XML Schema, /home/***USER***/poSource/evolution/revisedPurchaseOrder.xsd. to update the document that is currently opened for editing in Word. When word opens a document for editing it requests a DAV lock on the document to prevent conflicting updates. The DAV lock is released when the document is closed.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

```
Command Output
set echo on
select XMLDiff
       (
         xdburitype('/home/SCOTT/poSource/xsd/purchaseOrder.xsd').getXML(),
         xdburitype('/home/SCOTT/poSource/evolution/revisedPurchaseOrder.xsd').getXML()
       )  SCHEMA_DIFFERENCES
 from dual
/

SCHEMA_DIFFERENCES
-<xd:xdiff xsi:schemaLocation="http://xmlns.oracle.com/xdb/xdiff.xsd http://xmlns.oracle.com/xdb/xdiff.xsd">
   <?oracle-xmldiffoperations-in-docorder="true" output-model="snapshot" diff-algorithm="global"?>
  -<xd:insert-node-before xd:node-type="comment"xd:xpath="/comment()[1]">
    -<xd:content>
       <!-- edited with XMLSPY v2004 rel. 2 U (http://www.xmlspy.com) by Mark D. Drake (Oracle XML DB) -->
     </xd:content>
   </xd:insert-node-before>
   <xd:delete-node xd:node-type="comment"xd:xpath="/comment()[1]"/>
   <xd:delete-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]"xd:attr-local="storeVarrayAsTable"xd:attr-nsuri="http://xmlns.oracle.com/xdb"/>
  -<xd:append-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:element[1]"xd:attr-local="tableProps"xd:attr-nsuri="http://xmlns.oracle.com/xdb">
     <xd:content> varray XMLDATA.ACTIONS.ACTION store as table ACTION_TABLE ( (constraint ACTION_PKEY primary key (NESTED_TABLE_ID, ARRAY_INDEX)) ) varray XMLDATA.LINEITEMS.LINEITEM store as table LI
   </xd:append-node>
  -<xd:append-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:element[1]"xd:attr-local="columnProps"xd:attr-nsuri="http://xmlns.oracle.com/xdb">
     <xd:content> constraint PURCHASEORDER_PKEY primary key (XMLDATA.REFERENCE), constraint VALID_EMAIL_ADDRESS foreign key (XMLDATA.USERID) references HR.EMPLOYEES (EMAIL)</xd:content>
   </xd:append-node>
   <xd:delete-node xd:node-type="element"xd:xpath="/xs:schema[1]/xs:complexType[1]/xs:sequence[1]/xs:element[1]"/>
  -<xd:insert-node-before xd:node-type="element"xd:xpath="/xs:schema[1]/xs:complexType[1]/xs:sequence[1]/xs:element[7]">
    -<xd:content>
       <xs:element name="BillingAddress"type="AddressType"minOccurs="0"xdb:SQLName="BILLING_ADDRESS"/>
     </xd:content>
   </xd:insert-node-before>
  -<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]/xs:complexType[1]/xs:sequence[1]">
    -<xd:content>
       <xs:element name="Notes"type="NotesType"minOccurs="0"xdb:SQLType="CLOB"xdb:SQLName="NOTES"/>
     </xd:content>
   </xd:append-node>
  -<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]/xs:complexType[1]">
    -<xd:content>
       <xs:attribute name="Reference"type="ReferenceType"use="required"xdb:SQLName="REFERENCE"/>
     </xd:content>
   </xd:append-node>
  -<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]/xs:complexType[1]">
    -<xd:content>
       <xs:attribute name="DateCreated"type="xs:dateTime"use="required"xdb:SQLType="TIMESTAMP WITH TIME ZONE"/>
     </xd:content>
   </xd:append-node>
  -<xd:update-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:complexType[3]/xs:sequence[1]/xs:element[1]"xd:attr-local="name">
     <xd:content>Part</xd:content>
   </xd:update-node>
  -<xd:update-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:complexType[3]/xs:sequence[1]/xs:element[1]"xd:attr-local="type">
     <xd:content>PartType</xd:content>
   </xd:update-node>
  -<xd:update-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:complexType[3]/xs:sequence[1]/xs:element[1]"xd:attr-local="SQLName"xd:attr-nsuri="http://xmlns.oracle.com/xdb">
     <xd:content>PART</xd:content>
   </xd:update-node>
  -<xd:update-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:complexType[3]/xs:sequence[1]/xs:element[2]"xd:attr-local="name">
     <xd:content>Quantity</xd:content>
   </xd:update-node>
  -<xd:update-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:complexType[3]/xs:sequence[1]/xs:element[2]"xd:attr-local="type">
     <xd:content>quantityType</xd:content>
   </xd:update-node>
   <xd:delete-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:complexType[3]/xs:sequence[1]/xs:element[2]"xd:attr-local="SQLName"xd:attr-nsuri="http://xmlns.oracle.com/xdb"/>
  -<xd:insert-node-before xd:node-type="element"xd:xpath="/xs:schema[1]/xs:complexType[4]/xs:attribute[1]">
    -<xd:content>
      -<xs:simpleContent>
        -<xs:extension base="UPCCodeType">
           <xs:attribute name="Description"type="DescriptionType"use="required"xdb:SQLName="DESCRIPTION"/>
           <xs:attribute name="UnitCost"type="moneyType"use="required"/>
         </xs:extension>
       </xs:simpleContent>
     </xd:content>
   </xd:insert-node-before>
   <xd:delete-node xd:node-type="element"xd:xpath="/xs:schema[1]/xs:complexType[4]/xs:attribute[1]"/>
   <xd:delete-node xd:node-type="element"xd:xpath="/xs:schema[1]/xs:complexType[4]/xs:attribute[2]"/>
   <xd:delete-node xd:node-type="element"xd:xpath="/xs:schema[1]/xs:complexType[4]/xs:attribute[3]"/>
  -<xd:insert-node-before xd:node-type="element"xd:xpath="/xs:schema[1]/xs:complexType[7]/xs:sequence[1]/xs:element[2]">
    -<xd:content>
      -<xs:choice>
         <xs:element name="address"type="AddressType"minOccurs="0"/>
         <xs:element name="fullAddress"type="FullAddressType"minOccurs="0"xdb:SQLName="SHIP_TO_ADDRESS"/>
       </xs:choice>
     </xd:content>
   </xd:insert-node-before>
   <xd:delete-node xd:node-type="element"xd:xpath="/xs:schema[1]/xs:complexType[7]/xs:sequence[1]/xs:element[2]"/>
  -<xd:update-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:simpleType[3]/xs:restriction[1]"xd:attr-local="base">
     <xd:content>xs:decimal</xd:content>
   </xd:update-node>
  -<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]/xs:simpleType[3]/xs:restriction[1]">
    +<xd:content>
   </xd:append-node>
  -<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]/xs:simpleType[3]/xs:restriction[1]">
    +<xd:content>
   </xd:append-node>
  -<xd:update-node xd:node-type="attribute"xd:parent-xpath="/xs:schema[1]/xs:simpleType[11]"xd:attr-local="name">
     <xd:content>FullAddressType</xd:content>
   </xd:update-node>
  -<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
    -<xd:content>
      -<xs:complexType name="AddressType"xdb:SQLType="ADDRESS_T">
        +<xs:sequence>
       </xs:complexType>
     </xd:content>
   </xd:append-node>
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
  +<xd:append-node xd:node-type="element"xd:parent-xpath="/xs:schema[1]">
 </xd:xdiff>
Result : 1 Rows Selected.
```

- There are significant differences between the two XML Schemas. The changes will invalidate the existing instance documents so procedure copyEvolve must be used to evolve the XML Schema.

- A tableProps annotation has been added This annotation provides explicit names for the nested tables used to manage the collection of Action element and the collection of LineItem elements.

- A columnProps annotation has been added. This annotation recreates the primary key constraint on node reference and the foreign-key constraint on element user.

- The first element in the first sequence of the first global complexType has been deleted. This is element Reference in the complexType PurchaseOrderType.

- Element BillingAddress has been inserted into complexType PurchaseOrderType. The element was inserted before the seventh element.

- Element Notes has been appended to complexType PurchaseOrderType.

- Attribute Reference has been added to complexType PurchaseOrderType.

- Attribute DateCreated has been added to complexType PurchaseOrderType.

- The third complexType has been modified. This is the complexType LineItemType. LineItemType will now contain elements Part and Quantity. Element Part is an instance of complexType PartType. Element Quantity is an instance of simpleType quantityType.

- The fourth global complexType has been modified. This is the complexType PartType. The new model for PartType is complexType / simpleContent. It is an extension of the simpleType UPCCodeType and contains attributes Description and UnitPrice. The existing Attributes Id, UnitPrice and Quantity are deleted.

- The seventh global complexType has been modified. This is the complexType ShippingInstuctionsType. A choice of element Address or element FullAddress has been substituted for the element Address.

- The third global simpleType has been modified. This is the simpleType quantityType. The type is now a restriction of xs:decimal. A maximum of 4 digits are permitted to the right of the decimal points. A total of 8 digits are permitted.

- The eleventh global simpleType has been modified. This is the simpleType addressType. Its name is now FullAddressType.

- ComplexType AddressType has been added to the XML schema. It contains elements StreetLine1, StreetLine2, City, a of choice either element State and ZipCode, Province and PostCode or County and PostCode, and Country.

- SimpleTypes StreetType, CityType, StateType, ZipCodeType, ProvinceType, PostCodeType and CountyType and CountryType, NotesType and UPCCodeType have been added to the XML schema.

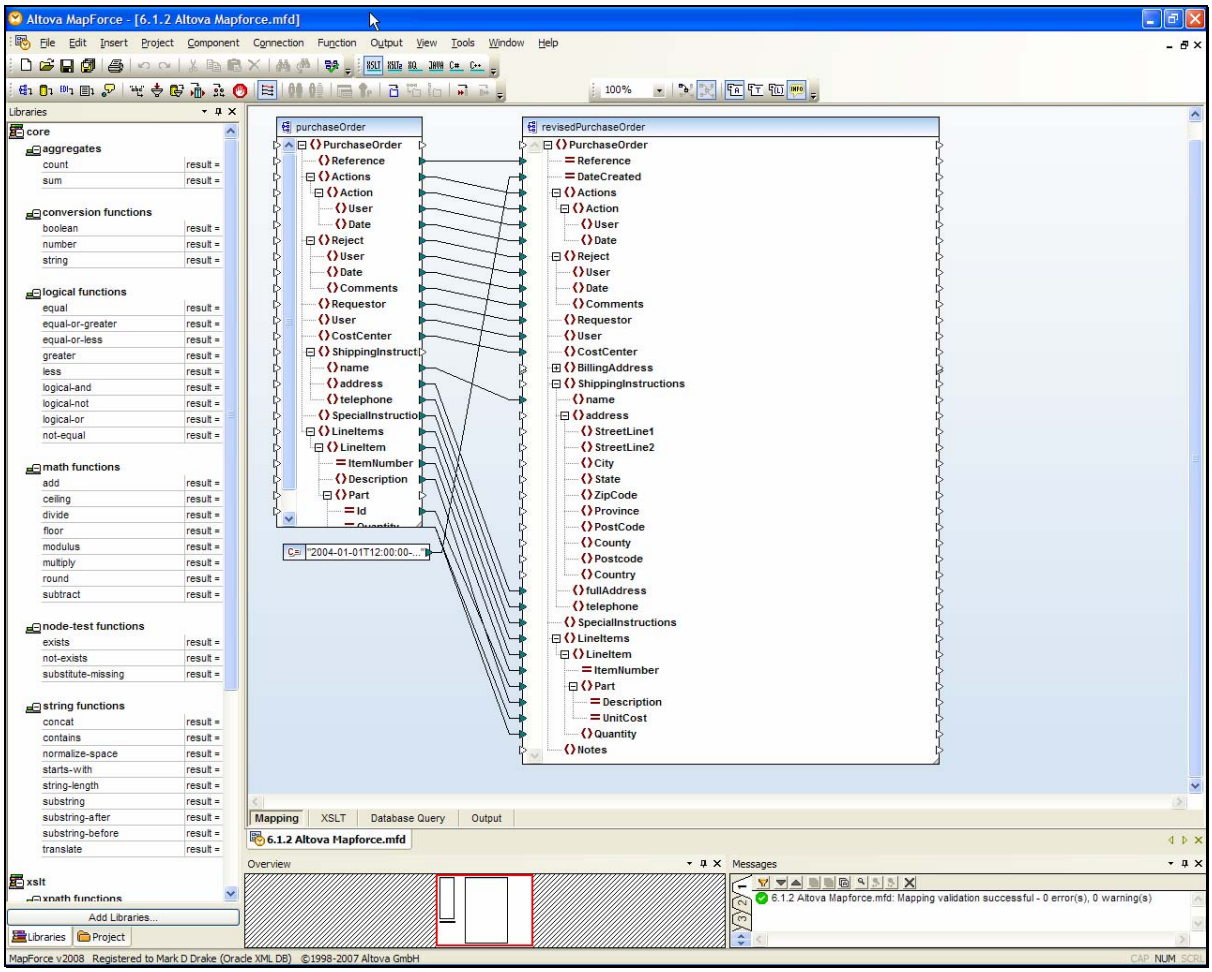
- Since the changes to the XML schema invalidate the existing instance documents an XSL stylesheet will be required to transform the existing PurchaseOrder documents.

6.1.2 Generate Stylesheet.mfd

There are a number of tools that can generate an XSL that will transform XML from one format to another. The latest release of Oracle's JDeveloper includes this functionality, as does Altova's MapForce.

This step shows to use Mapforce to quickly generate the XSL stylesheet required to complete the copyEvolve process. Mapforce provides a graphical interface that can be used to define the mapping between the nodes in the old XML Schema and the nodes in the new XML Schema. Once the mapping is complete it automatically generates the required stylesheet.

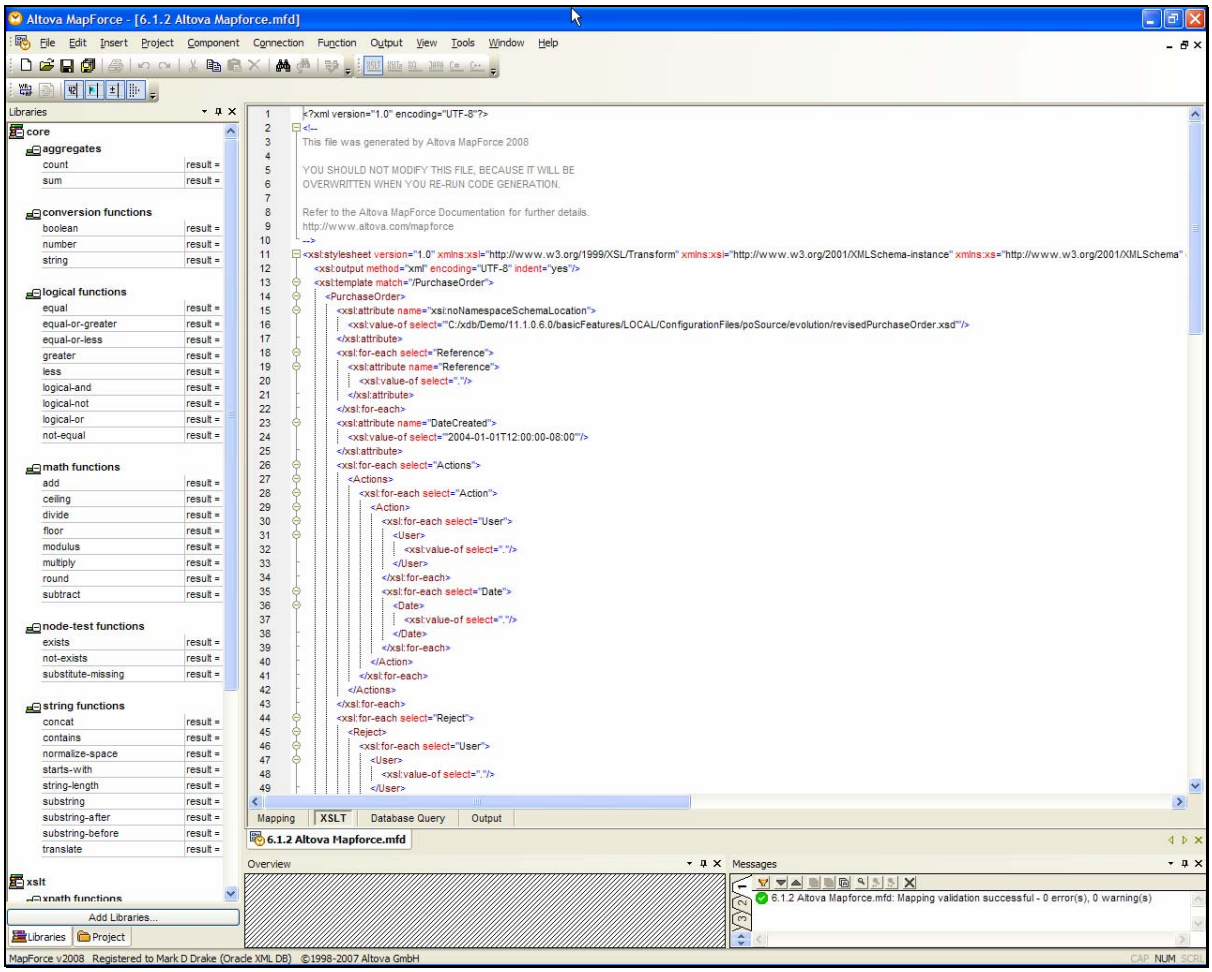
Click the icon to open a MapForce project that defines the mapping between the nodes in the old version and new versions of the PurchaseOrder XML Schema.



There is a one-to-one mapping between the nodes in the original XML schema and the nodes in the new XML Schema with the following exceptions:

- Element Reference is mapped to attribute Reference.
- Element Description in element LineItem is mapped to attribute Description in element Part.
- Attribute Quantity in element Part is mapped to element Quantity in element LineItem
- Attribute Id in element Part becomes the content of element Part.
- Element Address in element ShippingInstructions is mapped to element FullAddress in element ShippingInstructions.

Click the XSLT tab to set the generated XSL Stylesheet.



- The XSLT generated by Mapforce is a standard XSL style sheet. It contains nothing XML DB specific.

6.2.1 Evolve Schema

This next step uses the stylesheet generated in the previous step and procedure copyEvolve to migrate to the new XML Schema. Click the icon to launch the XML DB demonstration framework and run the SQL script.

Command Output

```
set echo on
begin
  dbms_xmlschema.CopyEvolve
  (
     xdb$string_list_t('http://xmldb:80/home/SCOTT/poSource/xsd/purchaseOrder.xsd'),
     XMLSequenceType(xdburitype('/home/SCOTT/poSource/evolution/revisedPurchaseOrder.xsd').getXML()),
     XMLSequenceType(xdburitype('/home/SCOTT/poSource/evolution/evolvePurchaseOrder.xsl').getXML())
  );
end;
/

PL/SQL procedure successfully completed.
```

- The CopyEvolve process copies the existing documents into a temporary table, drops the any tables or columns that are bound to the XML schema, registers the new XML Schema, recreates the bound tables and columns and then copies the data back in.

- The time taken will be directly proportional to the amount of data currently in the bound tables and columns

- The XSL stylesheet will be applied to the old documents before they are inserted into the new tables and columns

Indexes, constraints and triggers on the old table will not be preserved by copyEvolve unless they are defined in the new XML Schema. Index and trigger definitions will need to be changed to comply with the new storage model. The following statements recreate the indexes and triggers defined on table PURCHASEORDER.

Command Output

```
--
create index iPurchaseOrderUser on PurchaseOrder
(extractValue(object_value,'/PurchaseOrder/User'))
/

Index created.
```

Command Output

```
create index iLineItemPartNumber on LINEITEM_TABLE
(ITEMNUMBER, PART.SYS_XDBBODY$, NESTED_TABLE_ID)
/

Index created.
```

Command Output

```
create index iPartNumber on LINEITEM_TABLE
(PART.SYS_XDBBODY$, NESTED_TABLE_ID)
/

Index created.
```

Command Output

```
create or replace trigger VALIDATE_PURCHASEORDER
before insert on PURCHASEORDER
for each row
begin
   if (:new.object_value is not null) then
     :new.object_value.schemavalidate();
   end if;
end;
/

Trigger created.
```

Command Output

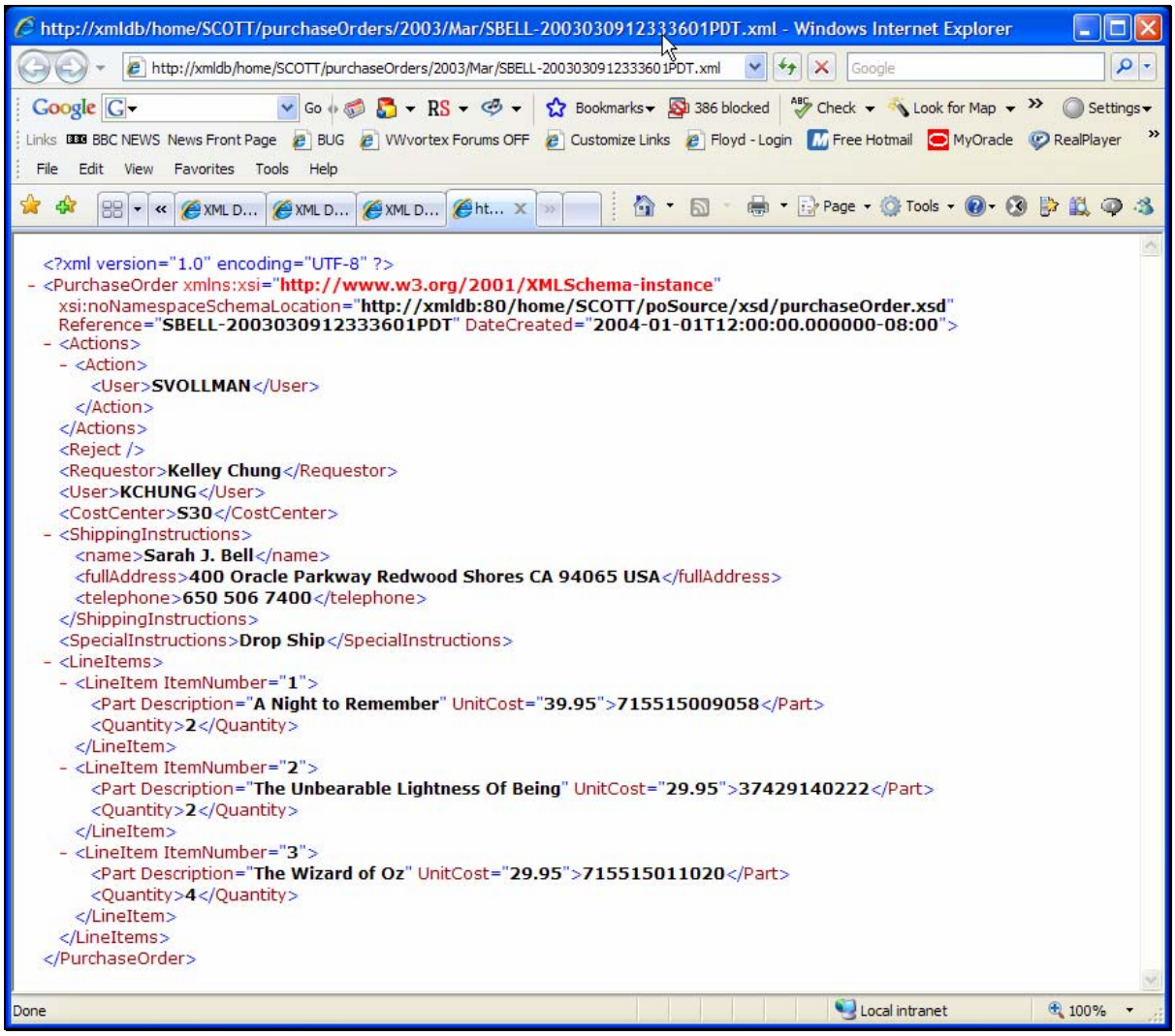
```
call dbms_stats.gather_schema_stats(USER)
/

PL/SQL procedure successfully completed.
```

6.2.2 Show Transformed Document

This step show that while the XML documents have transformed to be compliant with the new XML Schema none of the metadata associated with the documents has been lost and the documents can still be accessed as resources in the Oracle XML DB repository.

Click the icon to launch Internet Explorer. Internet explorer will use HTTP to fetch the contents of the document. If prompted for a username and password enter the demonstration user's username and password and click OK.



- The PurchaseOrder document has been transformed and is compliant with new version of XML Schema PurchaseOrder.xsd.

7.1.1 Folder Restricted Queries (1)

One of the major advantages of the Oracle XML DB repository is that is can be accessed and updated from SQL and PL/SQL. Most other content management platforms do not provide SQL access to their content repositories. Making the repository SQL accessible has a number of advantages including

- It can be accessed and updated from any product that can establish a SQL connection to the Oracle Database.

- SQL is a familiar and powerful language for performing ad-hoc queries. The where clause in a repository query can filter documents based on content, metadata and location, or any combination of these factors.

The Oracle XML DB Repository is stored in the database schema owned by the user XDB. The XDB account is a locked account. Developers should never attempt to directly access or update the tables in the XDB schema. Table XDB$RESOURCE contains the metadata for every resource (file or folder) in the repository. The content of schema-based XML document is stored outside of the XDB schema in the default table specified by the XML schema. The content of all of other types of document is stored in table XDB$RESOURCE.

The repository is exposed via two system views, RESOURCE_VIEW and PATH_VIEW. Public synonyms make these views available to all database users. The views provide access to identification and location information as well as metadata and content. The views can be used to update Metadata and content.

Columns PATH (PATH_VIEW) and ANY_PATH (RESOURCE_VIEW) provide location information about each document. The information is provided in the form of an absolute URL, based on root of the Oracle XML DB repository. This URL is effective the local part of the URL that can be used to access the document via the Oracle XML DB HTTP Server. The URL-based access metaphor is very familiar to the majority of web-developers. URL based significantly reduces the complexity and cost of developing web-based applications that access content stored in the Oracle XML DB repository.

Column RES in PATH_VIEW and RESOURCE_VIEW contains metadata and content. Packages DBMS_XDB and DBMS_XDBRESOURCE provide procedures and functions that can access and update metadata and content.

The IETF WebDAV standard defines the minimum set of metadata that DAV a server is expected to maintain. Oracle XML DB maintains the meta-data for each resource as an XML document in column RES. The XML is managed as schema-based XMLType. The XML Schema is identified by the URL http://xmlns.oracle.com/xdb/XDBResource.xsd. Oracle XML DB resource documents can contain user-defined metadata in addition to the metadata defined by XML Schema. The metadata can be free-form or schema-based. Resource documents can be accessed, queried and updated just like any other XML document stored in Oracle XML DB.

Column RESID in PATH_VIEW and RESOURCE_VIEW provides an absolute identifier for each document in the repository.

Column LINK in PATH_VIEW provides information about the link that places the resource in the containing folder.

The operators EQUALS_PATH and UNDER_PATH allow folder restricted queries to be performed on RESOURCE_VIEW and PATH_VIEW. A folder restricted query selects a document based on it's location in the repository. Note that path names are case sensitive in SQL. The names in the URL are derived from element ChildName in column LINK and not element DisplayName in column RES. There is an unfortunate bug in the Windows implementation of DAV that incorrectly derives the name of the resources in a folder from element DisplayName. This leads to a situation where a folder appears to contain more than one document with a particular name.

Operator EQUALS_PATH retrieves a document from RESOURCE_VIEW or PATH_VIEW based on a specific location. A document is returned only if its location is an exact match for the URL supplied to operator EQUALS_PATH.

Operator UNDER_PATH retrieves records from RESOURCE_VIEW or PATH_VIEW based on containership. Documents are returned if they exist in the folder identified by the URL or one it's subfolders. Operator UNDER_PATH takes an optional depth parameter that controls how many levels of the directory hierarchy to be traverse when searching for results.

The XML DB repository includes a new hierarchical index that allows folder restricted queries to be resolved very efficiently. The index works with operators UNDER_PATH and EQUALS_PATH It does not work with the computed columns PATH and ANY_PATH. Folder restricted queries should always be performed using operators EQUALS_PATH and UNDER_PATH. The computed columns PATH and ANY_PATH should never be referenced in the where clause of a SQL Query.

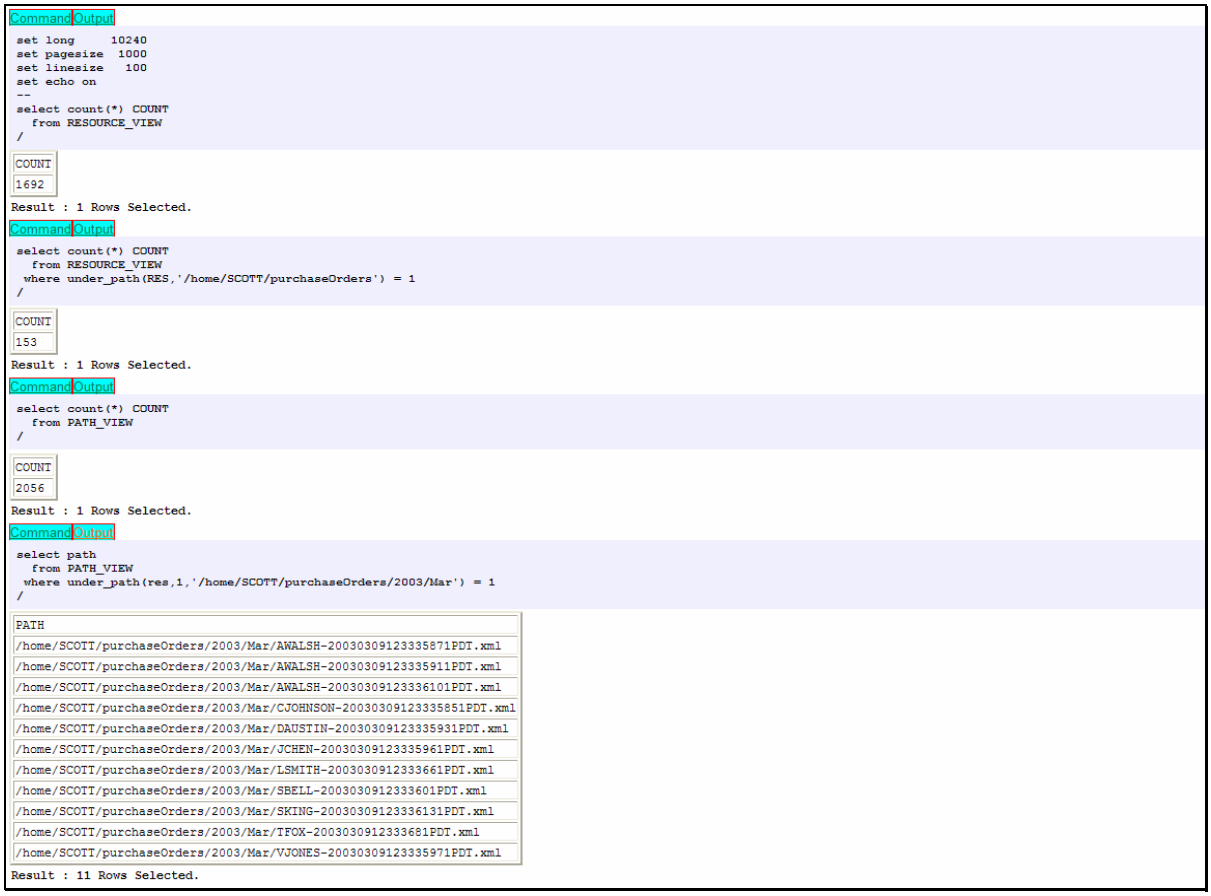
RESOURCE_VIEW contains one row for each document stored in the Oracle XML DB repository. PATH_VIEW contains one row for path to a document stored in the repository. The XML DB repository allows a resource to be identified by more than one path, so it's possible for PATH_VIEW to contain more rows than RESOURCE_VIEW.

Every document and folder in the repository is protected by an Access Control List (ACL). Queries on RESOURCE_VIEW and PATH_VIEW are always ACL restricted. This means that the result of a query can only include documents that the user has permission to access. Users can only access documents if the ACL on the document grants them at least read-properties permission and they can resolve at least one path to folder that contains the document. The path can only be resolved if the user has read-contents permission on a folder containing the document and all of its parents. In Oracle Database 11g the Oracle XML DB ACL implementation is compliant with the DAV-ACL specification. ACL based security is enforced through Row-Level Security when repository content is accessed from SQL.

The repository also supports a simple linear versioning model that it managed using package DBMS_XDB_VERSION.

Oracle Database 11g includes support for Repository Events. Repository events allow developers to attach PL/SQL or Java code to events on resources. Events are associated with the creation, modification and deletion of resources, adding or removing resources from a folder, or reading a resource. Events enable a programmable, intelligent repository in the same way that database triggers enabled a programmable, intelligent database.

The following examples show some basic RESOURCE_VIEW and PATH_VIEW queries. Click the icon to launch the XML DB demonstration framework and run the SQL script.



Command Output

```
set long     10240
set pagesize  1000
set linesize   100
set echo on
--
select count(*) COUNT
  from RESOURCE_VIEW
/
```

COUNT
1692

Result : 1 Rows Selected.

Command Output

```
select count(*) COUNT
  from RESOURCE_VIEW
 where under_path(RES,'/home/SCOTT/purchaseOrders') = 1
/
```

COUNT
153

Result : 1 Rows Selected.

Command Output

```
select count(*) COUNT
  from PATH_VIEW
/
```

COUNT
2056

Result : 1 Rows Selected.

Command Output

```
select path
  from PATH_VIEW
 where under_path(res,1,'/home/SCOTT/purchaseOrders/2003/Mar') = 1
/
```

PATH
/home/SCOTT/purchaseOrders/2003/Mar/AWALSH-20030309123335871PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/AWALSH-20030309123335911PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/AWALSH-20030309123336101PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/CJOHNSON-20030309123335851PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/DAUSTIN-20030309123335931PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/JCHEN-20030309123335961PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/LSMITH-20030309123336661PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/SBELL-20030309123336601PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/SKING-20030309123336131PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/TFOX-20030309123336681PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/VJONES-20030309123335971PDT.xml

Result : 11 Rows Selected.

- The first query counts the number of resources the current user has access to.

- The second query counts the number of resources the current user has access to and which located under the folder /home/SCOTT/purchaseOrders. Operator UNDER_PATH is used to specify the folder restriction for the query.

- The third query counts the number of district paths that lead to the documents the current user has access to. While the user only has access to 1692 documents, there are 2056 distinct paths to these documents.

- The fourth query selects the PATH to all documents located under folder /home/SCOTT /purchaseOrders/2003/Mar. The value of column PATH is a URL that can be used with operator xdburitype to access the content of the document. The value can also be used directly from a browser or WebDAV enabled application to access the content of the document.

  Note that the computed column PATH is used to fetch the PATH, but the folder restriction is specified using operator UNDER_PATH.

Command Output

```
--
select count(*) from RESOURCE_VIEW COUNT
 where under_path(RES,'/home/SCOTT/purchaseOrders') = 1
/
```

COUNT(*)
153

```
Result : 1 Rows Selected.
```

Command Output

```
select count(*) from PATH_VIEW COUNT
 where under_path(RES,'/home/SCOTT/purchaseOrders') = 1
/
```

COUNT(*)
153

```
Result : 1 Rows Selected.
```

- These queries count the number of resources and distinct access paths to resources under the folder /home/SCOTT/purchaseOrders. There are 153 resources and 153 access paths, one for each resource.

Command Output

```
begin
  dbms_xdb.link
  (
    '/home/SCOTT/purchaseOrders/2003/Mar',
    '/home/SCOTT/purchaseOrders',
    'CurrentMonth',
    DBMS_XDB.LINK_TYPE_WEAK
  );
end;
/

PL/SQL procedure successfully completed.
```

Command Output

```
select path from PATH_VIEW
 where under_path(res,1,'/home/SCOTT/purchaseOrders/CurrentMonth') = 1
/
```

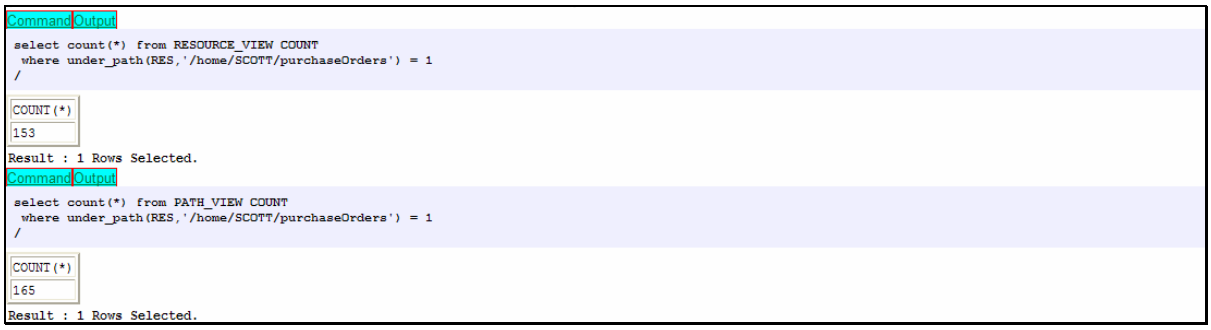
PATH
/home/SCOTT/purchaseOrders/CurrentMonth/AWALSH-20030309123335871PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/AWALSH-20030309123335911PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/AWALSH-20030309123336101PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/CJOHNSON-20030309123335851PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/DAUSTIN-20030309123335931PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/JCHEN-20030309123335961PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/LSMITH-20030309123336611PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/SBELL-20030309123336011PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/SKING-20030309123336131PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/TFOX-20030309123336811PDT.xml
/home/SCOTT/purchaseOrders/CurrentMonth/VJONES-20030309123335971PDT.xml

```
Result : 11 Rows Selected.
```

- The PL/SQL block creates a link in folder /home/SCOTT/purchaseOrders called CurrentMonth The target of the link is folder /home/SCOTT/purchaseOrders/2003/Mar. The operation did not create a new resource; it simply created a new access path for the existing resource.

- This allows the path /home/SCOTT/purchaseOrders/CurrentMonth to be used as an alias for /home/SCOTT/purchaseOrders/2003/Mar.

- A query of the content of /home/SCOTT/purchaseOrders/CurrentMonth returns the same result as a query on the content of /home/SCOTT/purchaseOrders/2003/Mar. However the computed paths returned reflect the path used to identify the folder.



- These queries show the effect of creating the link on RESOURCE_VIEW and PATH_VIEW. The link did not create a new resource so the number of rows in RESOURCE_VIEW is unchanged. However the link did create a new access path for each of the eleven documents in the target folder and one for the folder itself. This can be seen by the fact there now twelve additional entries in PATH_VIEW.

- The last query shows the basic metadata this is maintain for each document or folder in the Oracle XML DB repository.

- Resource documents contain standard metadata such as Display Name, Creator, Owner, Creation Date and Last Modification Date. They can be queried and updated just like any other XML document.

7.1.2 Folder Restricted Queries (2)

The following examples show some more complex RESOURCE_VIEW and PATH_VIEW queries. Click the icon to launch the XML DB demonstration framework and run the SQL script.

Command Output

```
set pagesize    50
set linesize   132
set long     10240
column COMPLEX_TYPE format A32
column SQL_TYPE     format A32
set echo on
select any_path
  from resource_view r
 where under_path(res,'/home/SCOTT') = 1
   and XMLExists
       (
         'declare default element namespace "http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :)
         $r/Resource[ContentType="text/xml"  and ends-with(DisplayName,"xsl")]'
         passing RES as "r"
       )
/
```

ANY_PATH
/home/SCOTT/poSource/evolution/evolvePurchaseOrder.xsl
/home/SCOTT/poSource/xsl/empdept.xsl
/home/SCOTT/poSource/xsl/purchaseOrder.xsl

Result : 3 Rows Selected.

- The first query performs a search on resource metadata to find XSL stylesheets. If finds the path to all documents under folder /home/SCOTT where the element ContentType contains the value **text/xml** and the value of element display name ends with **xsl**.

Command Output

```
--
select any_path
  from resource_view r
 where under_path(res,'/home/SCOTT/purchaseOrders') = 1
   and XMLExists
       (
         'declare namespace r = "http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :)
          $r/r:Resource[starts-with(r:DisplayName,"SBELL")]/r:Contents/PurchaseOrder'
          passing RES as "r"
       )
/
```

ANY_PATH
/home/SCOTT/purchaseOrders/2003/Apr/SBELL-20030409120030431PDT.xml
/home/SCOTT/purchaseOrders/2003/Apr/SBELL-20030409123336200304.xml
/home/SCOTT/purchaseOrders/2003/Aug/SBELL-20030809123337353PDT.xml
/home/SCOTT/purchaseOrders/2003/Dec/SBELL-20031209123338304PDT.xml
/home/SCOTT/purchaseOrders/2003/Dec/SBELL-20031209123338505PDT.xml
/home/SCOTT/purchaseOrders/2003/Feb/SBELL-20030209123335771PDT.xml
/home/SCOTT/purchaseOrders/2003/Jan/SBELL-20030109123335280PDT.xml
/home/SCOTT/purchaseOrders/2003/Jul/SBELL-20030709123333763PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/SBELL-20030309123333601PDT.xml
/home/SCOTT/purchaseOrders/2003/May/SBELL-20030509123336362PDT.xml
/home/SCOTT/purchaseOrders/2003/May/SBELL-20030509123336532PDT.xml
/home/SCOTT/purchaseOrders/2003/Nov/SBELL-20031109123338204PDT.xml
/home/SCOTT/purchaseOrders/2003/Oct/SBELL-20031009123337673PDT.xml

Result : 13 Rows Selected.

- The second query performs search on metadata and content. It finds all the path to documents under folder /home/SCOTT/purchaseOrders where the value of element DisplayName starts with **SBELL** and the root element of the content is **PurchaseOrder**.

- Since element PurchaseOrder is in the noNamespace namespace it not possible to use the default namespace for the XDB Resource namespace. An explicit namespace prefix mapping has to be supplied in order to reference nodes in the XDBResource namespace. All references to nodes in the XDB Resource namespace must be explicitly qualified with the prefix.

- This technique can be used to query both schema-based and non schema-based content.

```
select path
  from path_view, purchaseorder p
 where ref(p) = extractValue(res,'/Resource/XMLRef')
   and XMLExists('$p/PurchaseOrder/LineItems/LineItem[Part="715515009058"]' passing object_value as "p")
   and under_path(res,'/home/SCOTT/purchaseOrders/2003/Mar') = 1
/
```

PATH
/home/SCOTT/purchaseOrders/2003/Mar/CJOHNSON-20030309123335851PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/LSMITH-20030309123336661PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/SBELL-20030309123336601PDT.xml

Result : 3 Rows Selected.

- The third query shows a more efficient way to query resources that contain schema-based XML. An explicit join is performed between RESOURCE_VIEW and the default table that manages the content. This allows location and metadata information to be combined with content very efficiently.

  The join condition compares the value of element XMLRef with a REF XMLType value generated from the rows in the default table. The XMLRef is used by the Oracle XDB repository to track which row in the default table contains the content of a given schema-based resource. Currently, the legacy operator extractValue must be used to fetch the content of element XMLRef.

- The query finds the path to documents under folder /home/SCOTT/purchaseOrders/2003/Mar which contain an order for part **715515009058**.

- This technique of identifying rows in the default table using a REF can also be used to perform update and delete operations. The REF XMLType value of the row(s) to update or delete is obtained by using a sub-select that returns the value of the element XMLRef for the required rows(s).

Command Output

```
--
select COMPLEX_TYPE, SQL_TYPE
  from resource_view,
       xmlTable
       (
         xmlNamespaces
         (
           'http://www.w3.org/2001/XMLSchema' as "xsd",
           'http://xmlns.oracle.com/xdb' as "xdb",
           'http://xmlns.oracle.com/xdb/XDBResource.xsd' as "res"
         ),
         '/res:Resource/res:Contents/xsd:schema/xsd:complexType'
         passing RES
         columns
         COMPLEX_TYPE varchar2(64) path '@name',
         SQL_TYPE varchar2(64) path '@xdb:SQLType'
       )
 where equals_path(RES,'/home/SCOTT/poSource/xsd/purchaseOrder.xsd') = 1
/
```

COMPLEX_TYPE	SQL_TYPE
PurchaseOrderType	PURCHASEORDER_T
LineItemsType	LINEITEMS_T
LineItemType	LINEITEM_T
PartType	PART_T
ActionsType	ACTIONS_T
RejectionType	REJECTION_T
ShippingInstructionsType	SHIPPING_INSTRUCTIONS_T

Result : 7 Rows Selected.

- This query shows how to query the content of a resource that contains non-schema-based XML when there are multiple namespaces in play.

- The query operates on the content of the original version of XML Schema PurchaseOrder.xsd. This document is stored in the Oracle XML DB repository as a non-schema-based XML document.

- The query returns the name of each global complexType defined by the XML Schema and the value of the corresponding SQLType annotation.

- The document is located using operator EQUALS_PATH.

- The path expression /res:Resource/r:esContents/xsd:Schema/xsd:complexType returns the set of global complexType defined by the XMLSchema when the content is accessed via PATH_VIEW or RESOURCE_VIEW.

- Explicit namespace prefix mapping have to be supplied for all the namespaces that will be referenced by the query. In this example operator XMLNamespaces is used to pass the required namespace prefix mapping to operator XMLTable.

7.1.3 Folder Restricted Query Plan

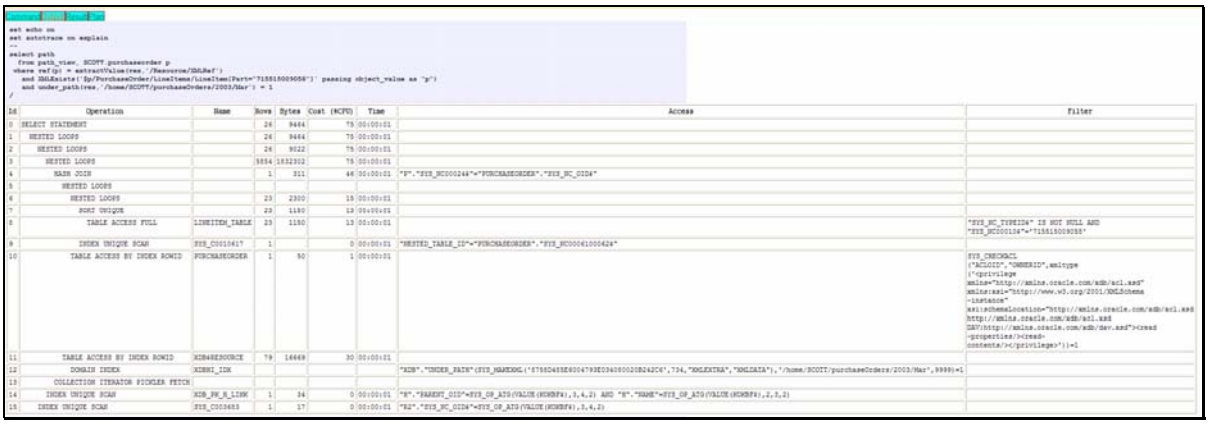
This step shows how the Oracle's patented Hierarchical Indexing technology optimizes folder restricted queries. I

This step shows the query plan for a query on RESOURCE_VIEW. Normal users will encounter error ORA-01039 when generating a query plan for a query that includes RESOURCE_VIEW. To avoid this error the query must be run by user who has access to objects owned by XDB and the objects owned by the demonstration user. Consequently this step is run as the DBA used to install the demonstration. The DBA must have permission to use Database Native Web Services (DBNWS).

The permissions required to use DBNWS can be granted by calling procedure enableWebServicesDemo in the package XDBPM.XDB_DEMO_HELPER_11100. The procedure expects a single argument which is the name of the user to grant the permissions to. . Since a DBA cannot grant themselves privileges this procedure must be invoked by another DBA.

Click the icon to launch the XML DB demonstration framework and run the SQL script. The username field will be pre-filled with the name of the DBA and the password field will be empty. Enter the password and hit or click out of the password field. If the password is entered correctly the form should refresh and execute the script. If an HTTP authentication dialog appears Database Native Web Services have not been correctly configured for the DBA user.

The script will generate execute the third query from the previous step. It will also generate the query plan. Click the plan table to view the query plan for the query. Click the Output tab to expand the output area.



```
set echo on
set autotrace on explain
--
select path
  from path_view, SCOTT.purchaseorder p
 where ref(p) = extractValue(res,'/Resource/XMLRef')
   and XMLExists('$p/PurchaseOrder/LineItems/LineItem[Part="715515009058"]' passing object_value as "p")
   and under_path(res,'/home/SCOTT/purchaseOrders/2003/Mar') = 1
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		26	9464	75	00:00:01		
1	NESTED LOOPS		26	9464	75	00:00:01		
2	NESTED LOOPS		26	9022	75	00:00:01		
3	NESTED LOOPS		5854	1832302	75	00:00:01		
4	HASH JOIN		1	311	46	00:00:01	"P"."SYS_NC000246"="PURCHASEORDER"."SYS_NC_OID#"	
5	NESTED LOOPS							
6	NESTED LOOPS		23	2300	15	00:00:01		
7	SORT UNIQUE		23	1150	13	00:00:01		
8	TABLE ACCESS FULL	LINEITEM_TABLE	23	1150	13	00:00:01		"SYS_NC_TYPEID#" IS NOT NULL AND "SYS_NC000104"='715515009058'
9	INDEX UNIQUE SCAN	SYS_C0010417	1		0	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC00061000424"	
10	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	90	1	00:00:01		SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read-contents/></privilege>'))=1
11	TABLE ACCESS BY INDEX ROWID	XDB$RESOURCE	79	14669	30	00:00:01		
12	DOMAIN INDEX	XDBHI_IDX					"XDB"."UNDER_PATH"(SYS_MAKEXML('67560D48E400479ED3408002B242C6',734,"XMLEXTRA","XMLDATA"),'/home/SCOTT/purchaseOrders/2003/Mar',9999)=1	
13	COLLECTION ITERATOR PICKLER FETCH							
14	INDEX UNIQUE SCAN	XDB_PK_H_LINK	1	24	0	00:00:01	"H"."PARENT_OID"=SYS_OP_ATG(VALUE(KOKBF$),3,4,2) AND "H"."NAME"=SYS_OP_ATG(VALUE(KOKBF$),2,3,2)	
15	INDEX UNIQUE SCAN	SYS_C003683	1	17	0	00:00:01	"H2"."SYS_NC_OID#"=SYS_OP_ATG(VALUE(KOKBF$),3,4,2)	

- The hierarchical index, XDBHI_IDX, is used to resolve the folder restricted query. The hierarchical index allows path based queries to be resolved without the use of connect by operations.

- The hierarchical index is implemented as an Oracle domain index, using Oracle's extensible indexing technology.

8.1.1 Path-based Full-Text Search (Un-Indexed)

The combination of XML and Full-Text based searching is very attractive. The XQuery Full-Text extension is still a work-in-progress. In the mean time the full power of the Full-Text features of Oracle Database 11g are enabled via an Oracle supplied XQuery extension function, known as ora:contains. This function provides most of the functionality of Oracle's SQL contains function in a form that can be used in an XQuery path and flower expressions. The ora:contains function allows full-text search to be performed on any node in an XML document.

The following example shows a full text search on the value of Element description. It uses ora:contains and the **$** operator, to find LineItem elements where the element Description contains words like **seven**. Operator XMLTable is used to return the results as virtual table.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

Command Output Result Plan

```
set linesize 132
set pagesize 50
column DESCRIPTION format A40
set echo on
set timing on
--
set autotrace on explain
--
select REFERENCE, LINENUMBER, PARTNO, DESCRIPTION
  from PURCHASEORDER,
       XMLTable
       (
         '/PurchaseOrder'
         passing object_value
         columns
         REFERENCE         path '@Reference',
         LINEITEMS xmltype path 'LineItems'
       ),
       XMLTable
       (
         '/LineItems/LineItem[ora:contains(Part/@Description,"$(Seven)") > 0 and Quantity = 2]'
         passing LINEITEMS
         columns
         LINENUMBER        path '@ItemNumber',
         PARTNO            path 'Part/text()',
         DESCRIPTION       path 'Part/@Description'
       )
/
```

REFERENCE	LINENUMBER	PARTNO	DESCRIPTION
TFOX-20031209123338354PDT	8	37429124529	The Seventh Seal
VJONES-20031209123338374PDT	8	37429121726	Seven Samurai
AWALSH-20031109123338448PDT	8	37429124529	The Seventh Seal
SBELL-20030509123336532PDT	2	37429121726	Seven Samurai
SKING-20030509123336392PDT	6	37429124529	The Seventh Seal
TFOX-20030609123336782PDT	12	37429124529	The Seventh Seal
SMCCAIN-20030809123337173PDT	5	37429121726	Seven Samurai
VJONES-20030909123337363PDT	2	37429121726	Seven Samurai
SMCCAIN-20030409120030451PDT	23	37429121726	Seven Samurai

- The result includes orders for the **The Seventh Seal** and **Seven Samurai**. Both of these descriptions contain words that are like **seven**.

Click the plan tab to see the query plan for this query.

Command Output Result Plan

```
set linesize 132
set pagesize 50
column DESCRIPTION format A40
set echo on
set timing on
--
set autotrace on explain
--
select REFERENCE, LINENUMBER, PARTNO, DESCRIPTION
  from PURCHASEORDER,
       XMLTable
       (
         '/PurchaseOrder'
         passing object_value
         columns
         REFERENCE         path '@Reference',
         LINEITEMS xmltype path 'LineItems'
       ),
       XMLTable
       (
         '/LineItems/LineItem[ora:contains(Part/@Description,"$(Seven)") > 0 and Quantity = 2]'
         passing LINEITEMS
         columns
         LINENUMBER        path '@ItemNumber',
         PARTNO            path 'Part/text()',
         DESCRIPTION       path 'Part/@Description'
       )
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		28	19376	19	00:00:01		
1	HASH JOIN		28	19376	19	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC00061000626$"	
2	TABLE ACCESS FULL	PURCHASEORDER	139	14178	5	00:00:01		SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read-contents/></privilege>'))=1
3	TABLE ACCESS FULL	LINEITEM_TABLE	28	16520	13	00:00:01		"SYS_NC_TYPEID$" IS NOT NULL AND "Quantity"=2 AND SYS_XMLCONTAINS ("SYS_NC00011$",'$(Seven)')>0

- The plan shows that the function ora:contains was executed as a filter. This is acceptable when there are only a small number of documents to process, but would not scale to handle large numbers of documents.

- The namespace for the contains function is http://xmlns.oracle.com/xdb. This namespace prefix ora is pre-defined for this namespace.

8.1.2 Create Full Text Indexes

This section show how to create an Oracle Text index to optimize full text search with the ora:contains function. Creating a text index will allow queries based on ora:contains to scale. Click the icon to launch the XML DB demonstration framework and run the SQL script.

Command Output

```
set linesize 132
set pagesize 50
column DESCRIPTION format A40
set echo on
set timing on
create index iDESCRIPTION_FULL_TEXT
    on LINEITEM_TABLE (PART.DESCRIPTION)
       indexType is CTXSYS.CONTEXT
       parameters ('transactional')
/
Index created.
```

- Full text indexes are of type CTXSYS.CONTEXT.

- Since element Description is a member of the LineItem collection the full-text index must be created on the nested table that manages the LineItem collection.

- Functon ora:contains will only use transactional indexes. This means that **transactional** must be specified in the parameters clause used to create the index. This avoids the non-transaction semantics of the standard text index. When a search is performed a functional evaluation of all non-indexed documents takes place to ensure that all matching rows are found.

- Text indexes cannot be created on Index Organized Tables (IOT). This means that in Oracle Database 10g Release 2 and earlier the nested tables generated by schema registration cannot be indexed using a text index. In order to create a text index in this situation do the following.

    - Use package DBMS_METADATA package to get a copy of the DDL for the default table after schema registration is complete.

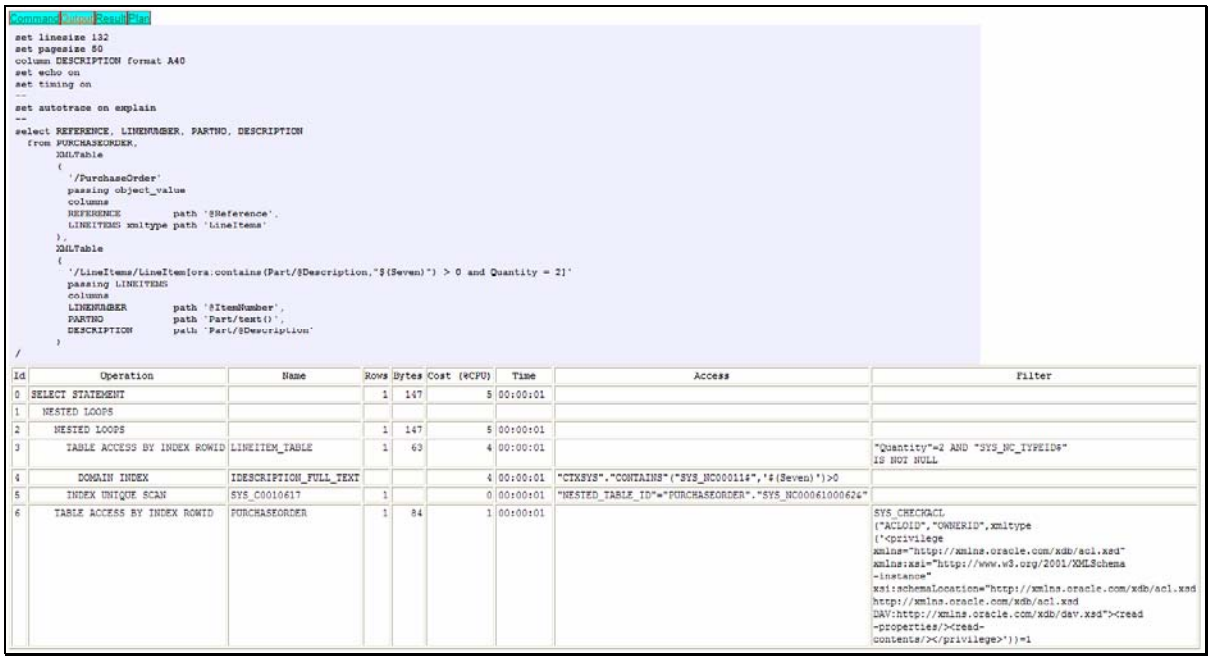
    - Drop the table

    - Edit the DDL to remove the organization index overflow causes from the nested table definitions.

    - Use the edited DLL to recreate the default table. The nested tables will now be heap-organized and cam be indexed with text indexes

8.1.3 Path-based Full-Text Path Search (Indexed)

This section show that creating a text index will improve the performance of path and flower expressions that include the ora:contains function. Click the icon to launch the XML DB demonstration framework and run the SQL script. The query will execute. Click the plan tab to see the new query plan for the query.



```
set linesize 132
set pagesize 50
column DESCRIPTION format A40
set echo on
set timing on
--
set autotrace on explain
--
select REFERENCE, LINENUMBER, PARTNO, DESCRIPTION
  from PURCHASEORDER,
       XMLTable
       (
         '/PurchaseOrder'
         passing object_value
         columns
         REFERENCE         path '$Reference',
         LINEITEMS xmltype path 'LineItems'
       ),
       XMLTable
       (
         '/LineItems/LineItem[ora:contains(Part/$Description,"${Seven}") > 0 and Quantity = 2]'
         passing LINEITEMS
         columns
         LINENUMBER        path '@ItemNumber',
         PARTNO            path 'Part/text()',
         DESCRIPTION       path 'Part/$Description'
       )
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access	Filter
0	SELECT STATEMENT		1	147	5	00:00:01		
1	NESTED LOOPS							
2	NESTED LOOPS		1	147	5	00:00:01		
3	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	1	69	4	00:00:01		"Quantity"=2 AND "SYS_NC_TYPEID$" IS NOT NULL
4	DOMAIN INDEX	IDESCRIPTION_FULL_TEXT			4	00:00:01	"CTXSYS"."CONTAINS"("SYS_NC00011$",'${Seven}')>0	
5	INDEX UNIQUE SCAN	SYS_C0010617	1		0	00:00:01	"NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC00061000624$"	
6	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	84	1	00:00:01		SYS_CHECKACL ("ACLOID","OWNERID",xmltype ('<privilege xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema -instance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read -properties/><read-contents/></privilege>'))=1

- The query plan is now driven off full text index IDESCRIPTION_FULL_TEXT.
- This is a much more scaleable and performant query plan.
- The query did not have to change in order for the index to be used.
- The combination of Oracle Text's full text indexing and operator ora:contains adds powerful, scaleable and performant node level full-text search capabilities to Oracle XML DB.

8.2.0 Document-level Full-Text Search

As well as support full text search on a node with an XML document Oracle XML DB and Oracle Text also support creating a full-text index on the entire document. Document level full text searches are performed using the SQL function contains with is part of Oracle Text.

Oracle Text uses standard SQL to index, search, and analyze text and documents stored in the Oracle database, in files, and on the web. Oracle Text can perform linguistic analysis on documents, as well as search text using a variety of strategies including keyword searching, context queries, Boolean operations, pattern matching, mixed thematic queries, HTML/XML section searching, and so on. It can render search results in various formats including unformatted text, HTML with term highlighting, and original document format. Oracle Text supports multiple languages and uses advanced relevance-ranking technology to improve search quality. Oracle Text also offers advanced features like classification, clustering, and support for information visualization metaphors.

This step show how to create a document level full-text index using Oracle Text and then use the SQL contains function to perform full text searches of XML content.

Click the icon to launch the XML DB demonstration framework and run the SQL script.

The first step creates a full-text index on the content of table PURCHASEORDER.

Command Output Result Plan

```
set long     10240
set pagesize   100
set linesize   132
column REFERENCE format A28
column NODE format A54
set echo on
--
set autotrace on explain
--
create index iPURCHASEORDER_FULL_TEXT
    on purchaseorder p (object_value)
       indextype is ctxsys.context
       parameters ( 'section group ctxsys.NULL_SECTION_GROUP' )
/
Index created.
```

- Full text indexes can be created on all the XML storage models, including object-relational and binary XML storage. The index is created on the XMLType column, in the case of an XMLType table this is the virtual column object_value.

- Index IPURCHASEORDER_FULL_TEXT is oracle full-text index. The indextype for a full-text index in CTXSYS.CONTEXT.

The next step shows some simple full-text queries on the contents of table PURCASHEORDER.

```
Command Output Result Plan
--
select count(*)
  from purchaseorder p, hr.employees e, hr.departments d
 where contains (object_value,'$Seven and $Night') > 0
   and xmlExists('$p/PurchaseOrder[User=$USER]' passing object_value as "p", e.EMAIL as "USER")
   and e.department_id = d.department_id
   and d.department_name = 'Shipping'
/

COUNT(*)
4

Result : 1 Rows Selected.

Command Output
set autotrace off
--
select any_path
  from resource_view, purchaseorder p, hr.employees e, hr.departments d
 where ref(p) = extractValue(res,'/Resource/XMLRef')
   and contains (object_value,'$Seven and $Night') > 0
   and xmlExists('$p/PurchaseOrder[User=$USER]' passing object_value as "p", e.EMAIL as "USER")
   and e.department_id = d.department_id
   and d.department_name = 'Shipping'
/

ANY_PATH
/home/SCOTT/purchaseOrders/2003/Mar/AWALSH-20030309123335871PDT.xml
/home/SCOTT/purchaseOrders/2003/Jan/SBELL-20030109123335280PDT.xml
/home/SCOTT/purchaseOrders/2003/Aug/AWALSH-20030809123337203PDT.xml
/home/SCOTT/purchaseOrders/2003/Sep/VJONES-20030909123337363PDT.xml

Result : 4 Rows Selected.
```

- The first query counts the number of documents in table PURCHASEORDER that were created by a user in the **shipping** department and which contain a word like **seven** or a word like **Night**.

- The second query joins the result of the first query with RESOURCE_VIEW to find a path for each document.

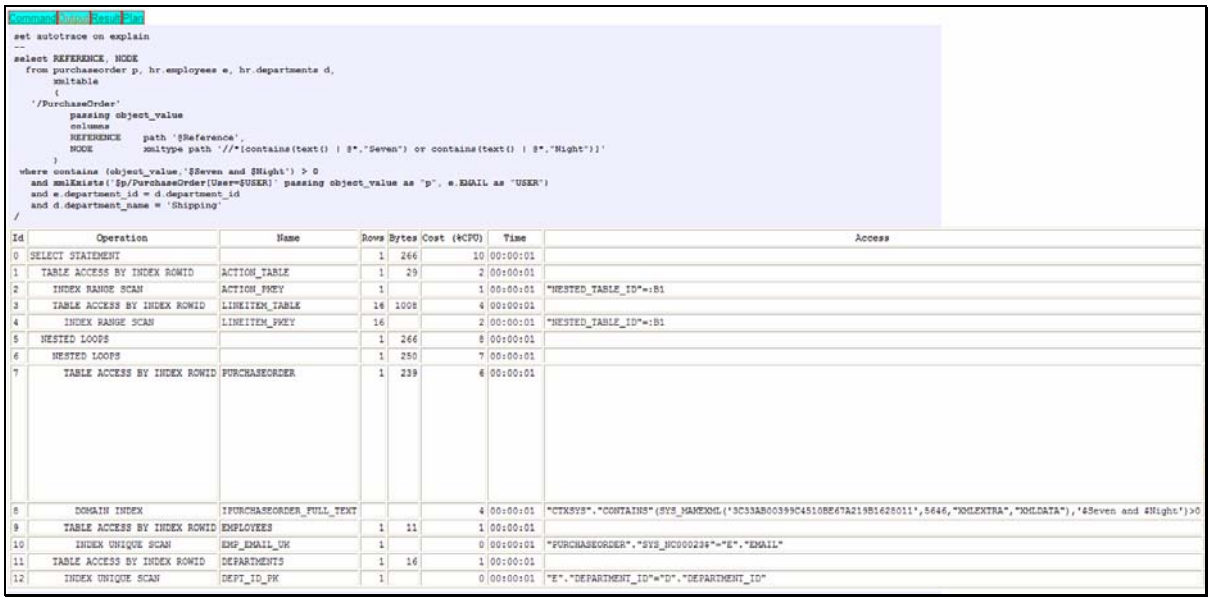
```
Command Output Result Plan
set autotrace on explain
--
select REFERENCE, NODE
  from purchaseorder p, hr.employees e, hr.departments d,
       xmltable
       (
   '/PurchaseOrder'
         passing object_value
         columns
         REFERENCE    path '@Reference',
         NODE         xmltype path '//*[contains(text() | @*,"Seven") or contains(text() | @*,"Night")]'
       )
 where contains (object_value,'$Seven and $Night') > 0
   and xmlExists('$p/PurchaseOrder[User=$USER]' passing object_value as "p", e.EMAIL as "USER")
   and e.department_id = d.department_id
   and d.department_name = 'Shipping'
/

REFERENCE                    NODE
AWALSH-20030309123335871PDT  <fullAddress>1200 East Forty Seventh Avenue New York NY 10024 USA</fullAddress>
                             <Part Description="Nights of Cabiria"UnitCost="39.95">37429138427</Part>
SBELL-20030109123335280PDT   <Part Description="A Night to Remember"UnitCost="39.95">715515009058</Part>
                             <Part Description="Seven Samurai"UnitCost="39.95">37429121726</Part>
AWALSH-20030809123337203PDT  <fullAddress>1200 East Forty Seventh Avenue New York NY 10024 USA</fullAddress>
                             <Part Description="Nights of Cabiria"UnitCost="39.95">37429138427</Part>
VJONES-20030909123337363PDT  <Part Description="Seven Samurai"UnitCost="39.95">37429121726</Part>
                             <Part Description="Nights of Cabiria"UnitCost="39.95">37429138427</Part>

Result : 4 Rows Selected.
```

- The last query uses function ora:contains to locate the nodes in each document that matched the terms passed to the SQL contains operator. Column NODE will contain the elements where the value of the element or one its attributes contains the text that matched the supplied terms.

Click the Plan tab to see the query plan for the last query



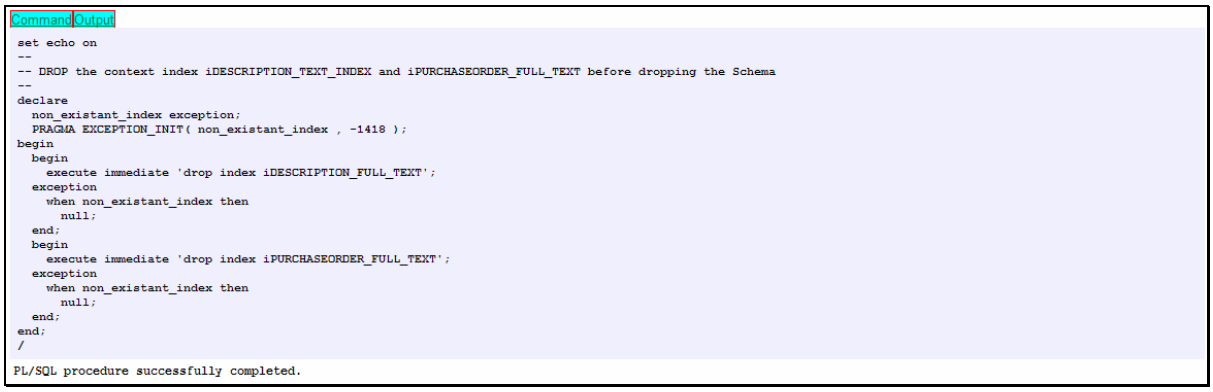
```
set autotrace on explain
--
select REFERENCE, NODE
  from purchaseorder p, hr.employees e, hr.departments d,
       xmltable
       (
    '/PurchaseOrder'
         passing object_value
         columns
         REFERENCE    path '$Reference',
         NODE         xmltype path '//*[contains(text() | $*,"Seven") or contains(text() | $*,"Night")]'
       )
 where contains (object_value,'$Seven and $Night') > 0
   and xmlExists('$p/PurchaseOrder[User=$USER]' passing object_value as "p", e.EMAIL as "USER")
   and e.department_id = d.department_id
   and d.department_name = 'Shipping'
/
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Access
0	SELECT STATEMENT		1	266	10	00:00:01	
1	TABLE ACCESS BY INDEX ROWID	ACTION_TABLE	1	29	2	00:00:01	
2	INDEX RANGE SCAN	ACTION_PKEY	1		1	00:00:01	"NESTED_TABLE_ID"=:B1
3	TABLE ACCESS BY INDEX ROWID	LINEITEM_TABLE	16	1008	4	00:00:01	
4	INDEX RANGE SCAN	LINEITEM_PKEY	16		2	00:00:01	"NESTED_TABLE_ID"=:B1
5	NESTED LOOPS		1	266	8	00:00:01	
6	NESTED LOOPS		1	250	7	00:00:01	
7	TABLE ACCESS BY INDEX ROWID	PURCHASEORDER	1	239	6	00:00:01	
8	DOMAIN INDEX	IPURCHASEORDER_FULL_TEXT			4	00:00:01	"CTXSYS"."CONTAINS"(SYS_NAMEXML('3C33AB00399C4510BE67A219B1628011',5646,"XMLEXTRA","XMLDATA"),'$Seven and $Night')>0
9	TABLE ACCESS BY INDEX ROWID	EMPLOYEES	1	11	1	00:00:01	
10	INDEX UNIQUE SCAN	EMP_EMAIL_UK	1		0	00:00:01	"PURCHASEORDER"."SYS_NC000238"="E"."EMAIL"
11	TABLE ACCESS BY INDEX ROWID	DEPARTMENTS	1	16	1	00:00:01	
12	INDEX UNIQUE SCAN	DEPT_ID_PK	1		0	00:00:01	"E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID"

- The query plan shows the query is driven off index IPURCHASEORDER_FULL_TEXT. This is the full-text index. This means that query should scale to large numbers of documents.

8.3.0 Drop Text Indexes

This step drops the index that were created in steps 8.1.2 and 8.2.0 Running this step allows the previous step to be executed again without errors. Click the icon to launch the XML DB demonstration framework and run the SQL script.



```
set echo on
--
-- DROP the context index iDESCRIPTION_TEXT_INDEX and iPURCHASEORDER_FULL_TEXT before dropping the Schema
--
declare
  non_existant_index exception;
  PRAGMA EXCEPTION_INIT( non_existant_index , -1418 );
begin
  begin
    execute immediate 'drop index iDESCRIPTION_FULL_TEXT';
  exception
    when non_existant_index then
      null;
  end;
  begin
    execute immediate 'drop index iPURCHASEORDER_FULL_TEXT';
  exception
    when non_existant_index then
      null;
  end;
end;
/

PL/SQL procedure successfully completed.
```

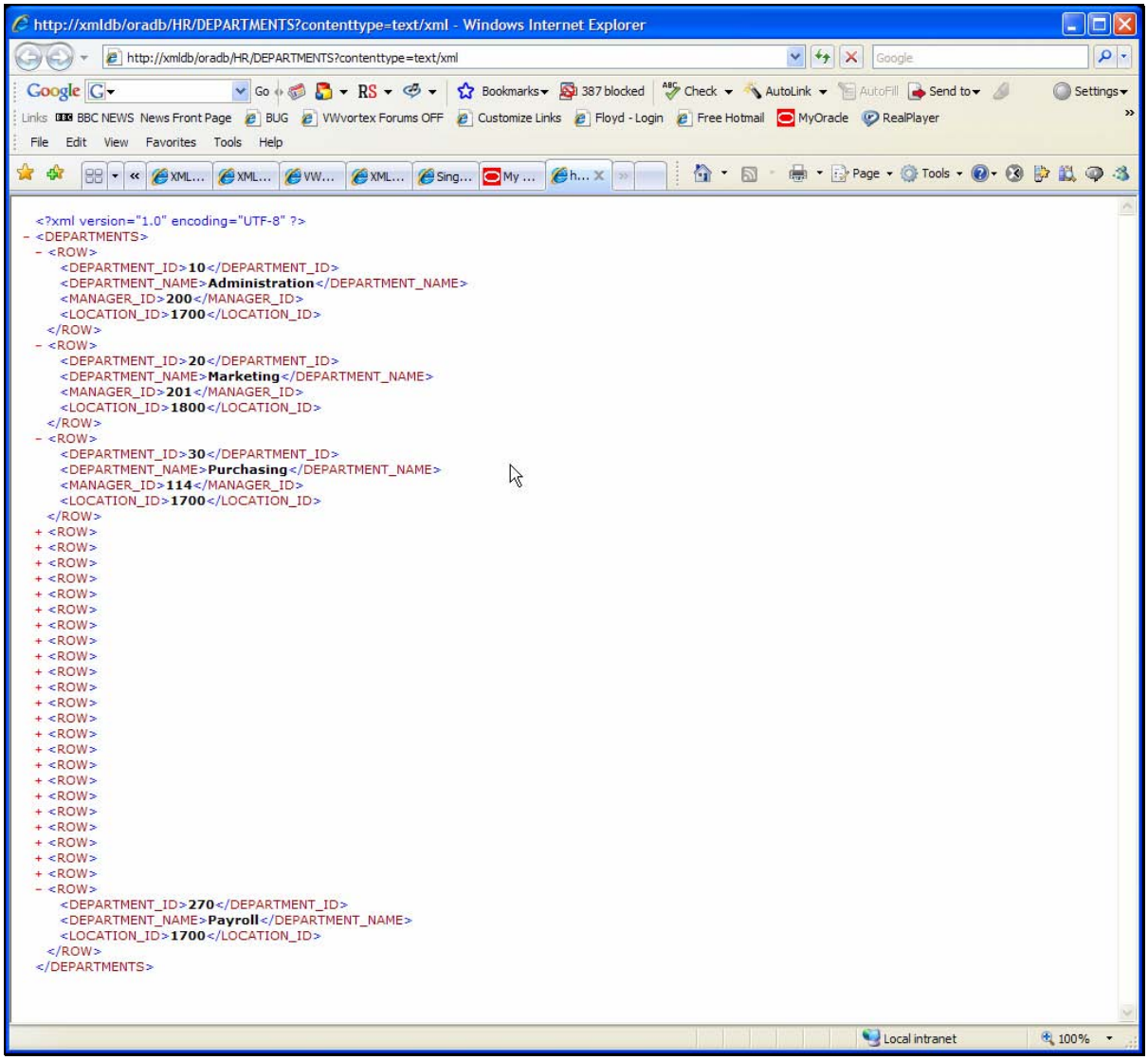
9.1.1 Content of DEPARTMENTS Table

Oracle Database 11g incorporates a number of built in servlets. One of these is the ORAWSV servlet that enables Database Native Web Services. Another is the DBURI servlet. The DBURI servlet provides direct access to the content of relational tables and views using a path-based metaphor. The servlet runs under the Oracle XML DB HTTP server. This allows a simple URL to be used to access content directly from a web browser.

The DBURI servlet is installed under the virtual directory /oradb. The path below the virtual direct consists of the database schema name and the object name. Database schema name is the value of any database schema, public may be used here. Object name is the value of a table or view in the database schema.

The DBURI servlet returns the contents of the table or view as a single XML document. The size of the document will be directly proportional to the number of rows in the table. Care should be taken when using the DBURI servlet to access tables containing large numbers of rows.

This step demonstrates using the DBURI servlet to view the content of a relational table. Click the icon to launch Internet Explorer. Internet explorer will use HTTP to fetch the contents of the document. If prompted for a username and password enter the demonstration user's username and password and click OK.



- The URL is http://xmldb/oradb/HR/DEPARTMENTS?contenttype=text/xml
- This URL displays the complete content of table DEPARTMENTS in the HR schema.
- Parameter contenttype is used to specify the mime-type of the document that is returned to the browser

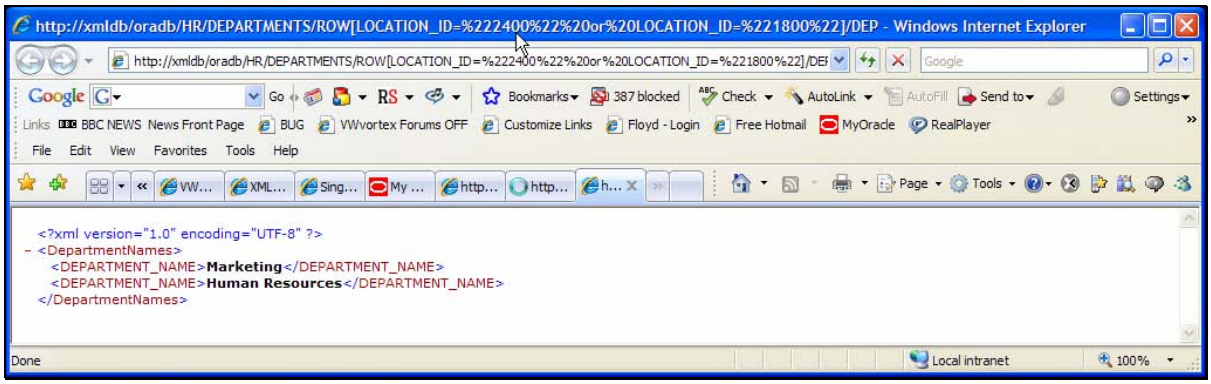
- The name of the root element comes from the object being accessed.
- Each row in the table is enclosed in by a instance of element ROW.
- The columns in each row become the children of the ROW element.

9.1.2 DEPARTMENTS Table with Predicates

Accessing the entire content of a table as single document has its limitations. With a large table the size of the document generated is often too large for the web-browser can handle. A subset of the documents in the table can be returned by adding predicates to the URL passed to the DBURI servlet. The URL can also be used to specify which column to return in the generated XML.

Predicates are specified by treating the content of the generated document as an extension of the path.

This step demonstrates the use predicates to restrict the rows returned by the DBURI servlet. Click the icon to launch Internet Explorer. Internet explorer will use HTTP to fetch the contents of the document. If prompted for a username and password enter the demonstration user's username and password and click OK.

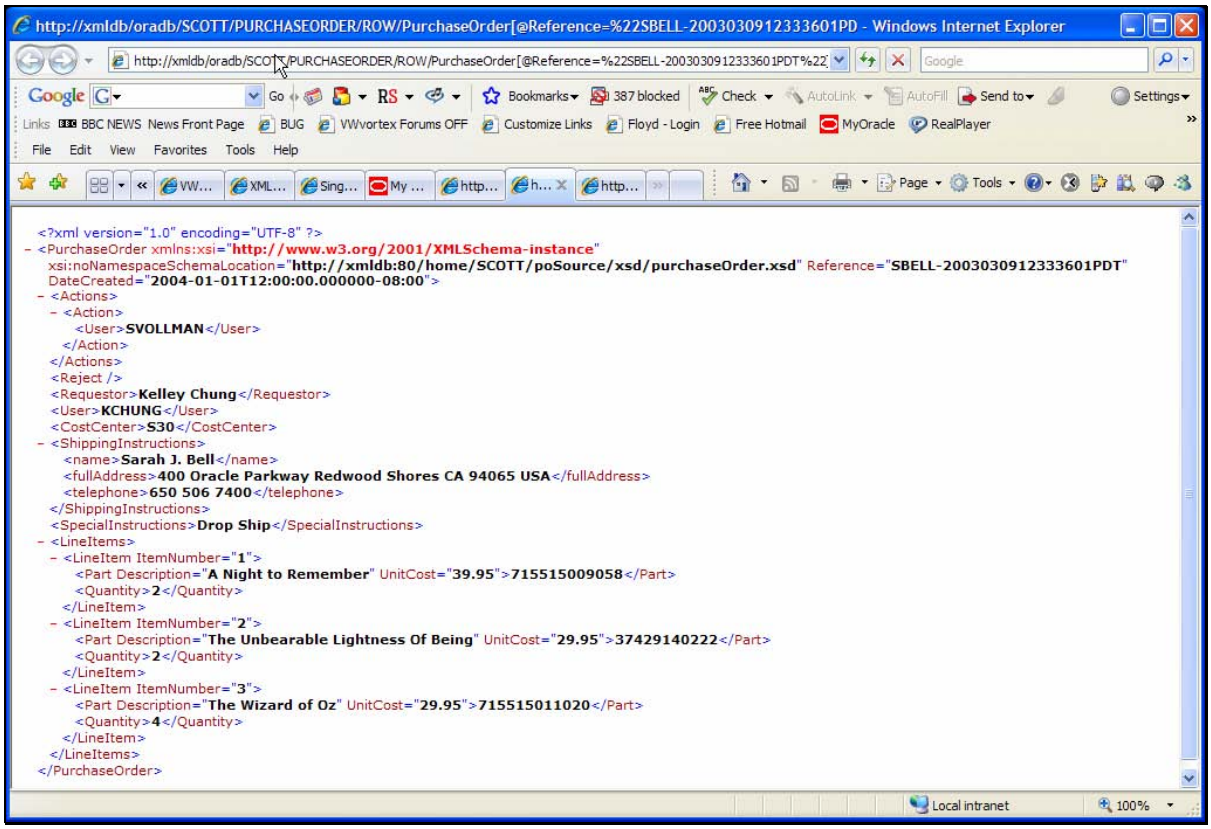


- The URL is http://xmldb/oradb/HR/DEPARTMENTS/ROW[LOCATION_ID="2400" or LOCATION_ID="1800"]/DEPARTMENT_NAME?contenttype=text/xml&rowsettag=DepartmentNames
- This URL displays the value of column DEPARTMENT_NAME for rows in table DEPARTMENTS where column LOCATION_ID contains the value 2400 or column LOCATION_ID contains the value 1800.
- Parameter rowsettag is used to specify the name of the root element of the generated document.

9.2.1 PURCHASEORDER Table XML

The DBURI servlet can be used to access documents in an XMLType table. When working with XMLType tables the path based to the DBURI servlet can contain predicates based on the structure of the XML document. This is very similar in concept to the functionality provided by the W3C XPOINTER standard.

This step demonstrates the use of a predicates to return a single row from an XMLType table using the DBURI servlet. Click the icon to launch Internet Explorer. Internet explorer will use the DBURI servlet to fetch the required document from table PURCHASEORDER. If prompted for a username and password enter the demonstration user's username and password and click OK.



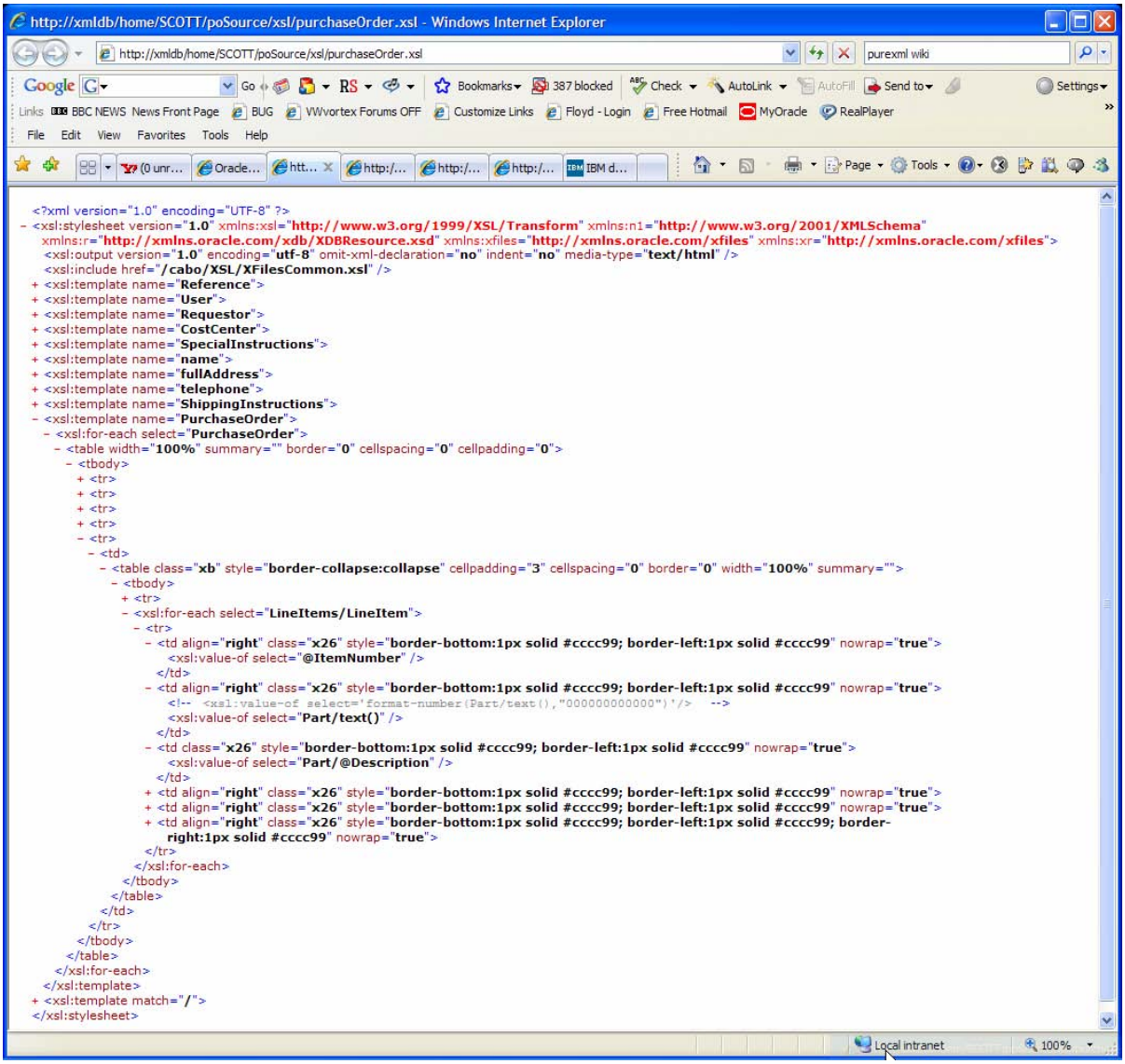
- The URL is http://xmldb/oradb/SCOTT/PURCHASEORDER/ROW/PurchaseOrder[@Reference=”SBELL-2003030912333601PDT”?contenttype=text/xml

- This URL returns element PurchsaseOrder from table PURCHASEORDER in database schema SCOTT where attribute REFERENCE contains the value **SBELL-2003030912333601PDT**.

9.2.2 PURCHASEORDER Stylesheet

Oracle XML DB includes an XSLT engine, based on a Virtual-Machine architecture. The XSLT VM provides Oracle XML DB with a performant and scaleable XSTL processing capability. The XSLT VM operates with standard XSL stylesheet that can be created using products like Oracle JDeveloper and Altova's XMLSPY and Mapforce. It can be invoked in SQL using operator XMLTRANSFORM, or the TRANSFORM method of XMLType. It can also be invoked using package DBMS_XSLPROCESSOR. The stylesheets can be stored as a column in a table or a document in the Oracle XML DB repository.

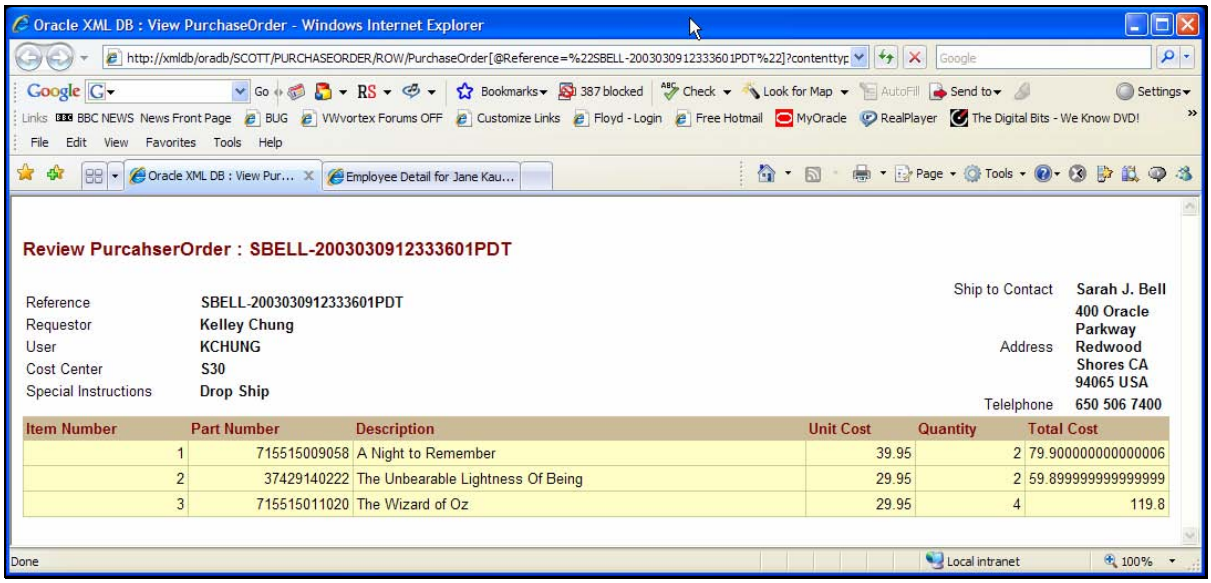
This step shows the content of an XSL stylesheet stored in as a resource in the Oracle XML DB Repository. Click the icon to display the stylesheet using Internet Explorer. If prompted for a username and password enter the demonstration user's username and password and click OK.



9.2.3 PURCHASEORDER with XSL Transformation

The DBURI servlet can apply an XSL stylesheet to the generated document before returning content to the browser. The stylesheet can be used to transform the generated XML into some other XML format, or to generate an HTML page from the XML generated by the DBURI servlet.

This step shows the result of applying the XSL stylesheet from step 9.2.2, to the document generated in step 9.2.1. Click the icon to launch Internet Explorer and display the result of the transformation. If prompted for a username and password enter the demonstration user's username and password and click OK.



- The URL is http://xmldb/oradb/SCOTT/PURCHASEORDER/ROW/PurchaseOrder[@Reference="SBELL-20030309123336O1PDT"]?contenttype=text/html&transform=/home/SCOTT/poSource/xsl/purchaseOrder.xsl

- The result of the transformation is an HTML page containing data from an XML document stored in Oracle Database 11g.

- Parameter contenttype is set to text/html since the output of the transformation is HTML, rather than XML.

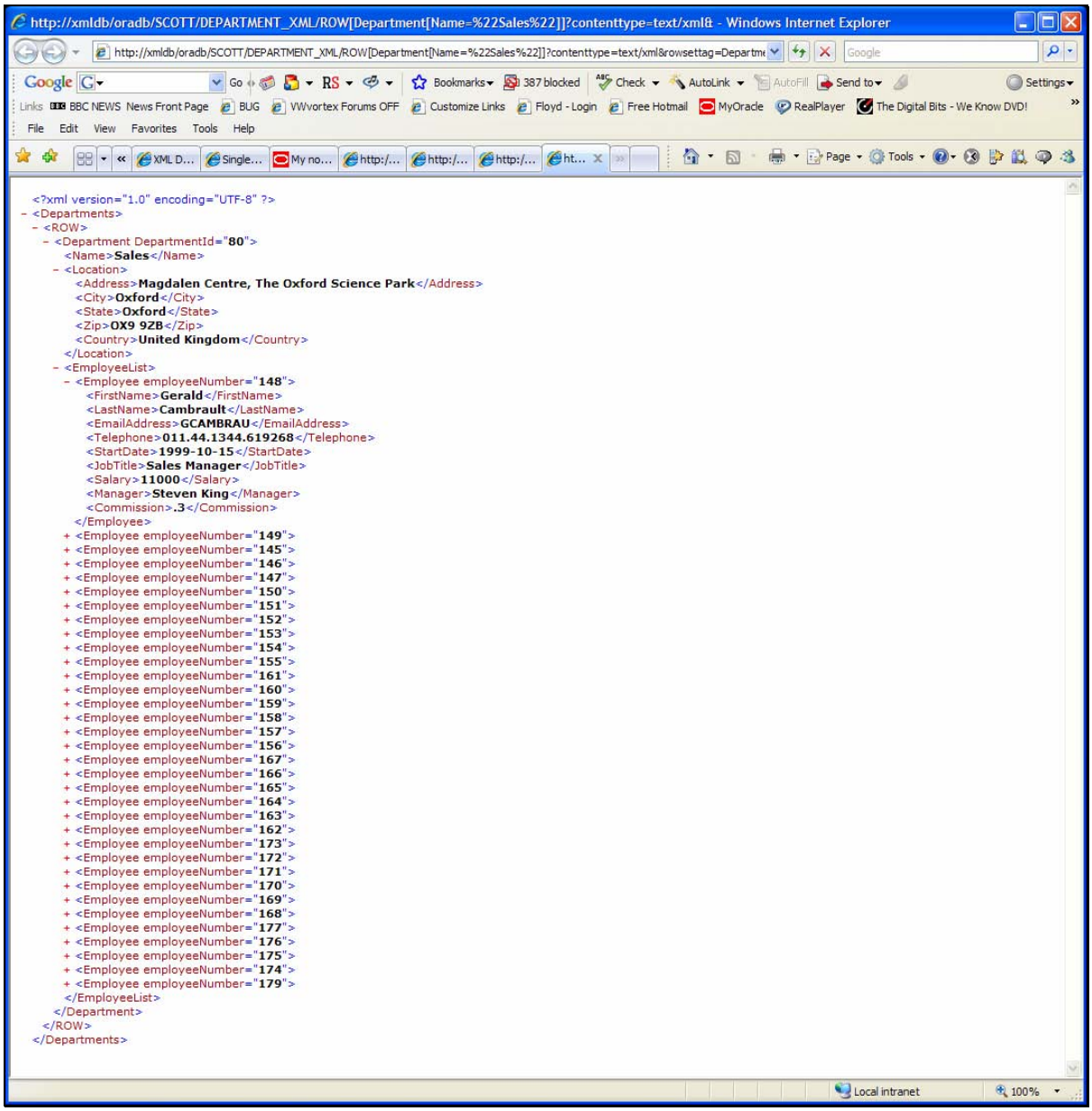
- Parameter transform is used to specify the stylesheet to use. The value of transform is the absolute location of the stylesheet within the Oracle XML DB repository.

- This simple technique allows a URL to be used to display a document managed using Oracle XML DB as an HTML page in a web browser.

9.3.1 DEPARTMENT_XML View XML

The DBURI servlet can also be used to access content from an XMLType view.

This step shows using the DBURI servlet to view content from view DEPARTMENT_XML. This view was created in step 4.4.1 Click the icon to launch Internet Explorer and display the content of the **SALES** department. If prompted for a username and password enter the demonstration user's username and password and click OK.



- The URL is http://xmldb/oradb/SCOTT/DEPARTMENT_XML/ROW[Department[Name=%22Sales%22]]?contenttype=text/xml&rowsettag=Departments

- This URL returns a document containing rows where the value of the element Name in element Department is **SALES.**

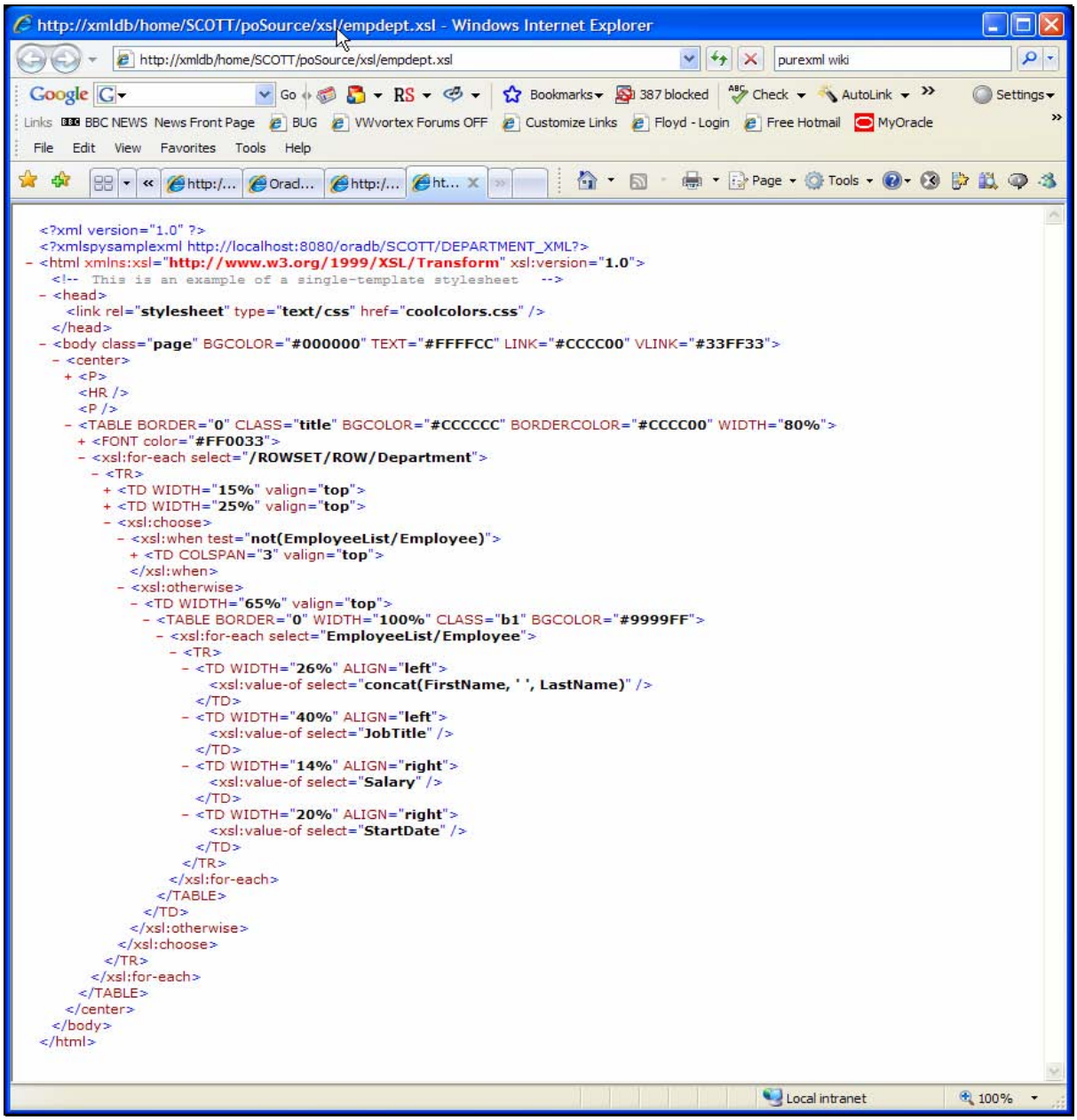
- The rowsettag is used to set the name of root element for the generated document to Departments.

- Element Departments will contain an instance of element ROW for each document in view DEPARTMENTS_XML that contains an element Name with the value **SALES**.

- The mimetype of the generated document will be set to text/xml when the response is returned to the browser.

9.3.2 DEPARTMENT_XML Stylesheet

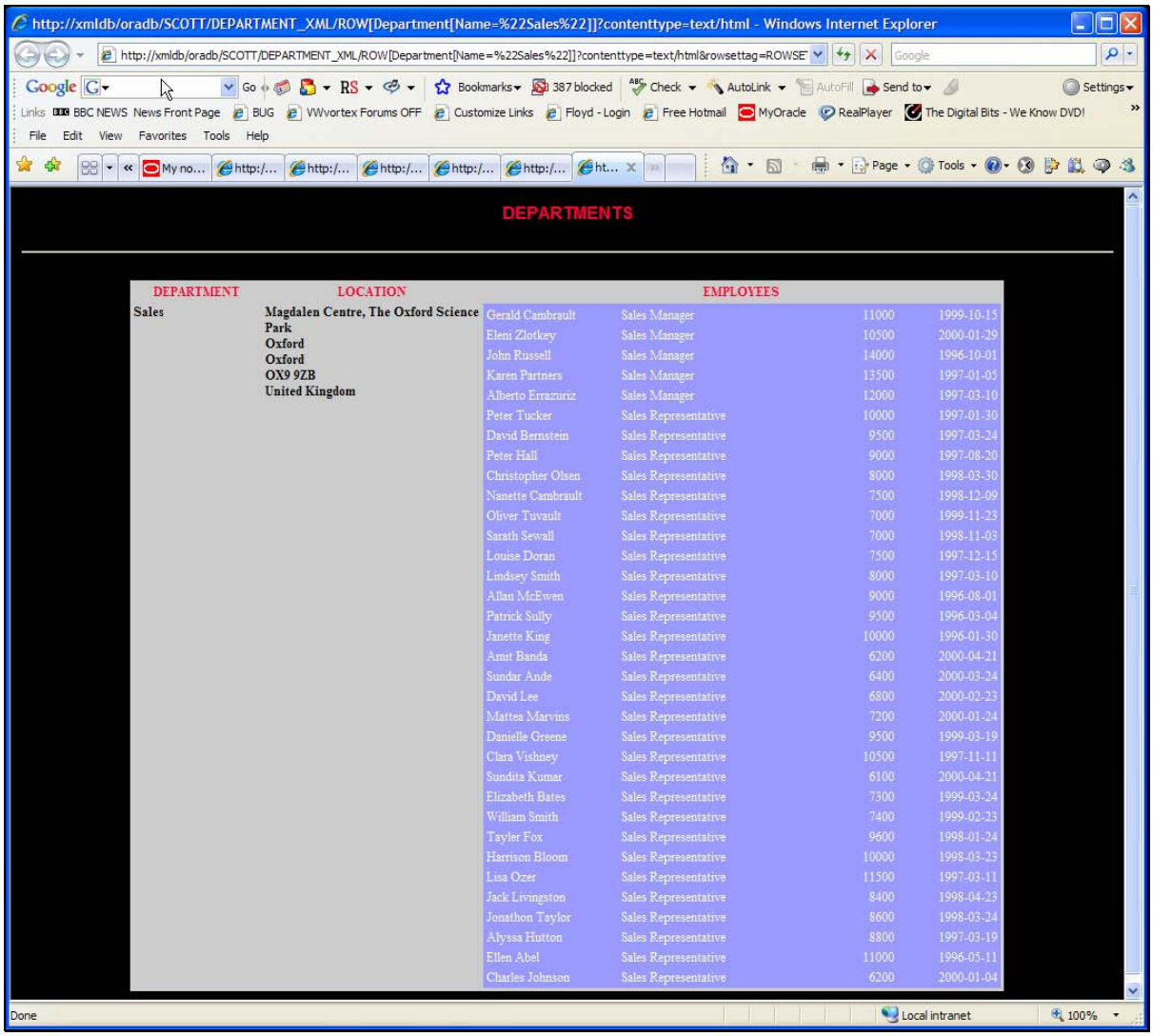
This step shows the a simple XSL stylesheet stored that can be used to transform the content of view DEPARTMENT_XML. Click the icon to display the stylesheet using Internet Explorer. If prompted for a username and password enter the demonstration user's username and password and click OK.



9.3.3 DEPARTMENT_XML with XSL Transformation

This step shows how the content of relational tables in the HR schema can be published as an HTML page using an XMLType view and an XSL stylesheet.

This step shows the result of applying the XSL stylesheet from step 9.3.2, to the document generated in step 9.3.1. Click the icon to launch Internet Explorer and display the result of the transformation. If prompted for a username and password enter the demonstration user's username and password and click OK.



- The URL is http://xmldb/oradb/SCOTT/DEPARTMENT_XML/ROW[Department[Name=%22Sales%22]]?contenttype=text/html&rowsettag=ROWSET&transform=/home/SCOTT/poSource/xsl/empdept.xsl

- The result of the transformation is an HTML page containing data from the relational tables in the HR schema.

- When the URL is accessed the following occurs:

    - The DBURI servlet receives a request to access content in view DEPARTMENT_XML.

    - The DBURI servlet queries view DEPARTMENT_XML to get the required document.

    - The data in the relational tables is transformed into XML by the SQL/XML operators used in the create view statement that defined view DEPARTMENT_XML.

    - The DBURI servlet applies an XSL stylesheet which transforms the XML into HTML

    - The HTML is returned to the browser.

- This simple technique allows a URL to be used to display data in relational tables as an HTML page in a web browser.

Oralce Database 11g Oracle XML DB
October 2007
Author: Mark D Drake
Contributing Authors: Oracle XML DB Development Team

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com