

ORACLE DATABASE 10G RELEASE 2 XML DB

*An Oracle Technical White Paper
May 2005*

What is the Oracle XML Database?	4
Oracle XML DB Major Features.....	5
XMLType	6
XML Schema.....	9
Namespaces	9
XML Schema and Namespaces	9
Registering an XML Schema.....	10
DOM Fidelity	11
Annotating an XML Schema	11
Identifying and Processing Instance Documents	11
Structured Vs Unstructured Storage	12
XML Schema Evolution	13
XML / SQL Duality	14
SQL/XML	15
XPath Re-write	16
Oracle Database-Native XQuery	17
FLWOR Expressions in XQuery	18
XMLQuery SQL Function.....	18
XMLTable SQL Construct.....	19
Rewrite for XQuery.....	19
XML DB Repository	20
The Oracle XML DB Protocol Architecture	25
Programmatic Access.....	26
Oracle XML DB performance.....	27
XML Storage Requirements	27
XML Memory Management	27
XML Parsing Optimizations.....	28
Node Searching Optimizations	28
XML Schema Optimizations	29
Load Balancing.....	29
Non-Native Code.....	29
Type Conversions.....	30

XML Manageability	31
Managing Oracle XML DB Applications with Oracle Enterprise Manager.....	31
Sample Application: Using Oracle XML DB to manage Purchase Orders.....	33
Loading Configuration Documents into the Oracle XML DB Repository	33
Oracle XML DB and the W3C XML Schema Standard.....	34
Creating an XML Schema	34
Annotating an XML Schema	35
Registering an XML Schema	38
Storing XML in Oracle XML DB.....	42
Adding Database Integrity to XML data	46
Querying and Indexing XML with Oracle XML DB	52
Querying XML	52
Query Plan Analysis.....	56
Indexing XML Content	61
Path-based Access and Update of XML Content.....	66
Relational Access to XML Content.....	81
XML Access to relational Content	86
SQL/XML Operators	86
The DBUri Servlet	90
XSL Transformation.....	93
XSL Transformation with the DBUri Servlet.....	95
Summary.....	97

WHAT IS THE ORACLE XML DATABASE?

Oracle XML DB is the term used to describe technology in the Oracle Database 10g that delivers high-performance storage and retrieval of XML. This technology extends the popular Oracle relational database, delivering all of the functionality associated with a native XML database, in addition to all of the functionality provided by the most sophisticated and complete relational database currently available.

Oracle XML DB provides a storage-independent, content-independent and programming-language-independent infrastructure to store and manage XML data. It delivers new methods for navigating and querying XML content stored inside the database and introduces an XML Repository for managing XML document hierarchies. With Oracle XML DB, you get all the advantages of relational database technology and XML technology at the same time.

Oracle XML DB delivers the following functionality:

- A native XML data-type that is used to store and manage XML documents.
- A set of methods and SQL operators which allow XML operations to be performed on XML content.
- The ability to absorb a standard W3C XML Schema data model into the Oracle database.
- XML/SQL duality, allowing XML operations on SQL data and SQL operations on XML content.
- Industry-standard methods for accessing and updating XML, including XPath and SQL/XML.
- A simple, light-weight XML Repository that allows XML content to be organized and managed using a file / folder / URL metaphor.
- Native database support for industry-standard, content-oriented, protocols including FTP, HTTP and WebDAV making it possible to move XML content into and out of the Oracle database.
- Multiple, industry-standard APIs that allow programmatic access and update of XML content from Java, 'C' and PL/SQL.
- XML-specific memory management and optimizations.
- The ability to bring the enterprise class management capabilities of the Oracle database, -- reliability, availability, scalability and unbreakable security -- to bear on XML content.

**Oracle XML DB
extends the Oracle
relational database,
adding all of the
functionally associated
with native XML
databases**

ORACLE XML DB MAJOR FEATURES

Any database that is going to be used manage XML must be capable of doing more than simply providing persistent storage for XML documents. It must provide standard database features like transaction control, data integrity, replication, reliability, availability, security and scalability. It must also provide the features required to efficiently index, query, update and search XML content in an XML centric manner.

One of the major challenges with using a traditional relational database to manage XML content is the hierarchical nature of the XML world. Examples of this include

Uniquely identifying an XML document: The standard way of identifying an XML document is via a URL.

Defining relationships between XML documents: The standard way of defining a relationship between two documents is via URL based standards like XLink.

Accessing and updating the content of an XML document. The standard ways of addressing and updating content is via WC3 standards like XPath.

URLs and XPath expressions are both intrinsically hierarchical in nature. A URL uses a path through a folder hierarchy to uniquely identify a document. An XPath expression uses a path through the XML document's node hierarchy to identify a particular piece of an XML document.

At first glance, the hierarchical metaphors used with XML do not map easily into the relational model. A relational database uses a table-row metaphor to organize content. Rows are identified using Unique Keys. Primary-Key Foreign-Key relationships are used to define the relationships between content. Content is accessed and updated using table-row-column based operations.

Oracle XML DB addresses these challenges by introducing new SQL operators and methods that provide direct support for the hierarchical nature of XML. These operators allow XML centric metaphors, such as XPath expressions and URLs, to be used to query and update XML documents that are stored in an Oracle database.

The major features of Oracle XML DB are as follows:

XMLType

XMLType is a native server data-type that allows the database to understand that a column or table contains XML; in the same way that the DATE data-type allows the database to understand that a column contains a date. XMLType also provide methods that allow common operations such as schema validation and XSL Transformations to be performed on XML content.

The XMLType data-type can be used just like any other data-type. It can be used when creating a column in a relational table. It can be used when declaring PL/SQL variables, and when defining and calling PL/SQL procedures and functions. Since XMLType is an object type, it is also possible to create a table of XMLType.

The XMLType data type can also be used when defining views. Creating an XMLType view, or a relational view that includes an XMLType column, allows Oracle XML DB to be used to expose content stored relational tables and external data sources as XML documents.

By default, an XMLType table or column can contain any well formed XML document. The content of the document is stored as XML text using the CLOB data type. This allows for maximum flexibility in terms of the shape of the XML structures that can be stored in a single table or column and the highest rates of ingestion and retrieval.

An XMLType table or column can be constrained to an XML Schema. Constraining a column or table to an XML Schema has a number of advantages associated with it:

- The database will ensure that only XML documents that validate against the XML Schema can be stored in the column or table.
- Since the contents of the table or column are conformant with a known XML structure, Oracle XML DB can use the information contained in the XML Schema to provide more intelligent query and update processing of the XML.
- Constraining the XMLType to an XML Schema provides the option of storing the content of the document using structured-storage techniques. Structured-storage decomposes or ‘shreds’ the content of the XML document and stores it as a set of SQL objects rather than simply storing the document as text in CLOB. The object-model used to store the document is automatically derived from the contents of the XML Schema.

XMLType is a native data-type that is used to store and manage XML documents in columns or tables

The following diagram shows how XML can be stored and retrieved using Oracle XML DB and the XMLType data type:

XML can be stored one of two ways:

An XMLType column in a relational table.

An XML object in an XMLType table.

Non Schema based XML is always stored as CLOB.

Schema based XML can be stored as a CLOB or as a set of objects.

Relational and external data can be exposed as XML using views.

The view can be a relational view containing a column of XMLType or can be a XMLType View.

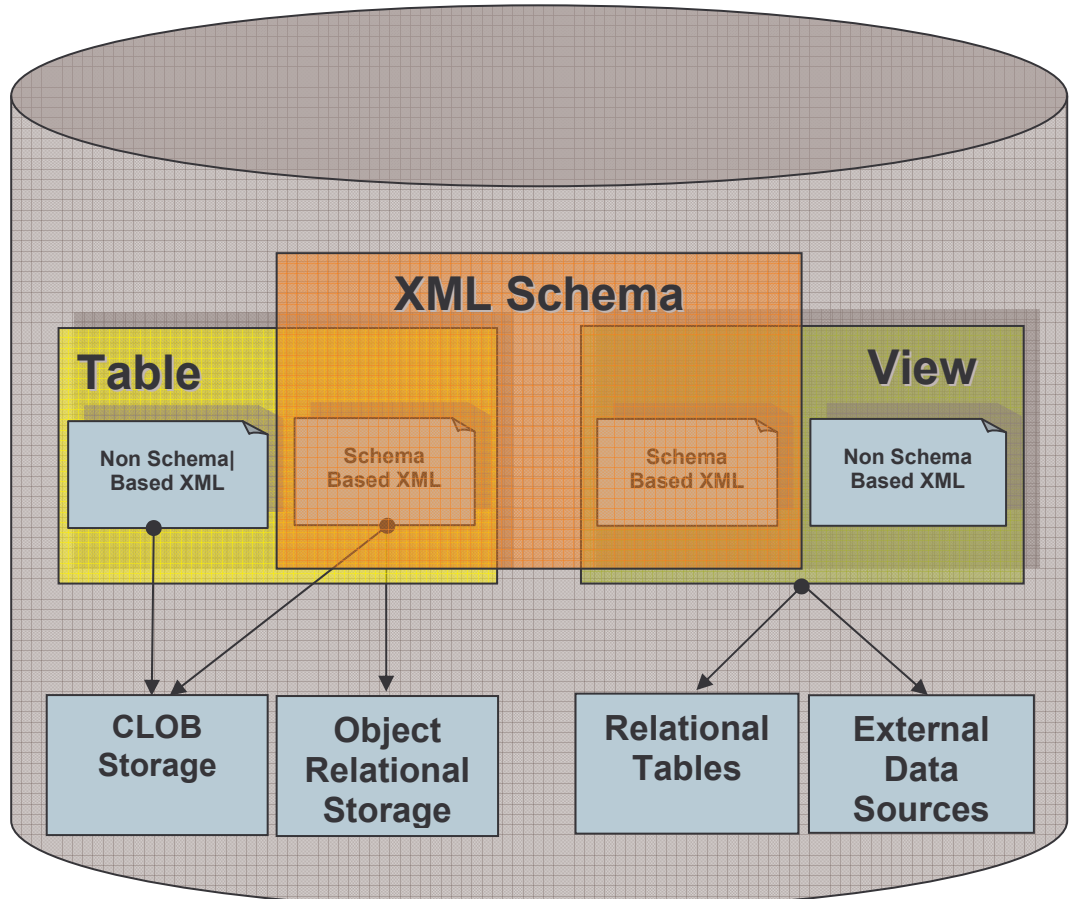


Figure I. XMLType storage options

The XMLType data-type provides constructors that allow an XMLType to be created from a VARCHAR, CLOB or BFILE data-type. It also provides a number of XML specific methods which can operate on XMLType objects. The methods provided by XMLType provide support for common operations like extracting a subset of the nodes contained in the XMLType (**extract()**, **extractValue()**), checking whether or not a particular node exists in the XMLType (**existsNode()**), validating the contents of the XMLType against an XML Schema (**schemaValidate()**), and performing XSL Transformation (**transform()**).

The following screen shot shows creating a simple table with an XMLType column and then performing insert and query options against the table.

The XMLType data type can be used just like any other data type.

The constructors of the XMLType allow XMLType to be created from VARCHAR and CLOB.

Inserting into a table with an XMLType column is like inserting into any other table.

New SQL operators make it possible to perform SQL queries over XML content.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Jan 6 10:57:13 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> create table XML_DOCUMENT_TABLE
2 (
3     FILENAME          varchar2(64),
4     XML_DOCUMENT       XMLType
5 )
6 /

Table created.

SQL> declare
2     XML_TEXT CLOB := '<PurchaseOrder>
3                       <Reference>AMCEWEN-20030409123336271PDT</Reference>
4                       <Actions/>
5                       <Reject/>
6                       <Requestor>Allan D. McEwen</Requestor>
7                       <User>AMCEWEN</User>
8                       <CostCenter>S30</CostCenter>
9                       <ShippingInstructions/>
10                      <SpecialInstructions>Expedite</SpecialInstructions>
11                      <LineItems>
12                        <LineItem ItemNumber="1">
13                          <Description>Traffic</Description>
14                          <Part Id="696306038924" UnitPrice="39.95" Quantity="2"/>
15                        </LineItem>
16                      </LineItems>
17                      </PurchaseOrder>';
18 begin
19     insert into XML_DOCUMENT_TABLE values ('AMCEWEN-20030409123336200304.xml',XMLTYPE(XML_TEXT));
20 end;
21 /

PL/SQL procedure successfully completed.

SQL> commit
2 /

Commit complete.

SQL> select extractValue(XML_DOCUMENT,'/PurchaseOrder/Reference') REFERENCE,
2         extractValue(XML_DOCUMENT,'/PurchaseOrder/CostCenter') COSTCENTER,
3         extractValue(XML_DOCUMENT,'/PurchaseOrder/Requestor') REQUESTOR
4   from XML_DOCUMENT_TABLE
5 /

REFERENCE                                COST REQUESTOR
-----
AMCEWEN-20030409123336271PDT             S30 Allan D. McEwen

SQL> |

```

Figure II. creating a relational table with a column of XMLType

XML Schema is a W3C standard for specifying the structure, content, and certain semantics of a set of XML documents

XML SCHEMA

Comprehensive support for the W3C XML Schema standard is one of the key features of Oracle XML DB. XML Schema is a W3C standard for specifying the structure, content, and certain semantics of a set of XML documents. The XML Schema standard is described by the W3C in <http://www.w3.org/TR/xmlschema-0/>. Since an XML Schema is used to define a class of XML documents, the term “*instance document*” is often used to describe an XML document that conforms to a particular XML Schema.

The W3C Schema Working Group publishes an XML Schema, often referred to as the “*Schema for Schemas*”. This XML Schema provides the definition, or vocabulary, of the XML Schema language. An XML Schema is an XML document, which is compliant with the vocabulary defined by the “*Schema for Schemas*”. An XML Schema document uses the vocabulary defined by W3C XML Schema Working Group to create a collection of type definitions and element declarations which declare a shared vocabulary that describe the contents and structure of a new class of XML documents.

The XML Schema language defines 47 scalar data types. This provides for strong typing of the elements and attributes. The XML Schema standard also supports the use of object-oriented techniques like inheritance and extension, making it possible for the schema designer to create complex objects from the base data types defined by the XML Schema language. The W3C XML Schema vocabulary also includes constructs that can be used to define ordering, default values, mandatory content, nesting, repeated sets, etc. Oracle XML DB supports all of constructs defined by the XML Schema standard, except for *redefines*.

The most common usage of an XML Schema is as a mechanism for validating that instance documents are conformant with a given XML Schema. The XMLType data type provides the methods **isSchemaValid()** and **schemaValidate()** that allow Oracle XML to use an XML Schema in this manner.

NAMESPACES

It is possible for two different XML Schemas to use the same name when defining an object (element, attribute, complex type, simple type, etc). Since the two objects are in different XML Schemas they cannot be treated as being the same item. This means that an instance document must identify which XML Schema a particular node is based on. The XML Namespace specification defines a mechanism which accomplishes this. An *XML namespace* is a collection of names, identified by a URI reference which are used in XML documents as element types and attribute names.

XML SCHEMA AND NAMESPACES

The *targetNamespace* attribute is used to define the namespace associated with a given XML Schema. The attribute is included in the definition of the ‘*schema*’ element. If an XML Schema specifies a *targetNamespace*, all elements and types defined by the schema are associated with this namespace. This implies that any XML document that contains these elements and types must identify which namespace the elements and types are associated with. If the XML Schema does not specify a *targetNamespace*, all elements and types defined by the schema are associated with the *NULL* namespace.

The XML Schema standard also defines a mechanism that allows an instance document to identify which XML Schemas are required in order to process or validate the document. The XML Schemas in question are identified on a namespace by namespace basis using attributes defined in the WC3 XMLSchema-Instance namespace.

In order to use this mechanism the instance document must first declare the XMLSchema-instance namespace. The XMLSchema-Instance namespace is declared by including a variant of the following namespace declaration in the root element of the instance document:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Once the XMLSchema-instance namespace has been declared the set of required XML Schemas is defined by adding *schemaLocation* and *noNamespaceSchemaLocation* attributes to the root element of the document.

The *noNamespaceSchemaLocation* attribute is used to identify any XML Schemas that did not provide an explicit namespace declaration for the objects they define. The content of the *noNamespaceSchemaLocation* attribute is a hint, typically in the form of a URL, which describe where to find the required XML Schema. This hint is often referred to as the “*Schema Location Hint*”

The *schemaLocation* attribute is used to identify all of the XML Schemas that declared an explicit namespace using the *targetNamespace* attribute. The content of the *schemaLocation* attribute is a set of entries, once for each XML Schema. Each entry consists of a pair of values. The left hand value is the namespace declared by the XML Schema. The right hand value is the “*Schema Location Hint*”.

REGISTERING AN XML SCHEMA

Before Oracle XML DB can make use of the information contained in an XML Schema, the XML Schema must be registered with the database. An XML Schema is registered by calling the PL/SQL procedure named **dbms_xmlschema.register_schema()**. The XML Schema is registered under a URL. The URL is used internally as a unique key used to identify the XML Schema. At no point does Oracle XML DB require direct access to the URL specified when registering the XML Schema.

When an XML Schema is registered with the database, a default table is created for each global element defined by the XML Schema. When an instance document is loaded in the Oracle XML DB repository, the content of the document will be stored in the default table. The default tables created by registering an XML Schema are XMLType tables, i.e. they are Object Tables, where each row in the table is represented as an instance of the XMLType data type.

Oracle XML DB can use the information contained in an XML Schema to derive an object model that allows XML content that is compliant with the XML Schema to be decomposed and stored in the database as a set of objects. The constructs defined by the XML Schema are mapped directly into SQL Types generated using the SQL 1999 Type Framework that is part of the Oracle database.

Using the SQL 1999 Type Framework to manage XML provides a number of significant benefits:

- It allows Oracle XML DB to leverage the full power of the Oracle database when managing XML.
- It can lead to significant reductions in the amount of space required to store the document.
- It can reduce the amount of memory required to query and update XML content.
- Capturing the XML Schema objects as SQL Types helps share the abstractions across schemas, and also across their SQL storage counterparts.
- It allows Oracle XML to support constructs defined by the XML Schema standard that do not easily map directly into the conventional relational model.

DOM FIDELITY

Using SQL 1999 objects to persist XML allows Oracle XML DB to guarantee DOM fidelity. DOM, or the Document Object Model, is a W3C standard that defines a set of platform- and language-neutral interfaces that allow a program to dynamically access and update the content, structure and style of a document. In order to provide DOM fidelity Oracle XML DB must ensure that a DOM generated from a document that has been shredded and stored in Oracle XML DB will be identical to a DOM generated from the original document.

Providing DOM Fidelity requires Oracle XML DB to preserve all of the information contained in an XML document. This includes maintaining the order in which elements appear within a collection and within a document as well as storing and retrieving out-of-band data like comments, processing instructions and mixed text. By guaranteeing DOM fidelity, Oracle XML DB is able to ensure that there is no loss of information when the database is used to store and manage XML documents.

ANNOTATING AN XML SCHEMA

Oracle XML DB provides the application developer or database administrator with control over how much decomposition, or ‘shredding’, takes place when an XML document is stored in the database. The XML Schema standard allows vendors to define schema annotations that add directives for specific schema processors. The Oracle XML DB schema processor recognizes a set of annotations that make it possible to customize the mapping between the XML Schema data types and the SQL data types, control how collections are stored in the database, and specify how much of a document should be shredded.

If you do not specify any annotations to your XML Schema to customize the mapping, Oracle XML DB will make certain default choices that may or may not be optimal for your application.

IDENTIFYING AND PROCESSING INSTANCE DOCUMENTS

Oracle XML DB assumes that instance documents will use the W3C approved mechanism for identifying which XML Schemas they are associated with. It assumes that the Document Location Hint contained in an instance document will map directly to the URL that was specified when registering the associated XML Schema with the database.

For example, assume an XML Schema defines a global element PurchaseOrder. The XML Schema does not include a targetNamespace declaration. The XML Schema has been registered under the URL <http://xmlns.oracle.com/demo/purchaseOrder.xsd>. In order for an XML document to be recognized as an instance of the XML Schema the root element of an instance document would need to look like

```
<PurchaseOrder xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:noNamespaceSchemaLocation="http://xmlns.oracle.com/demo/purchaseOrder.xsd">
```

STRUCTURED Vs UNSTRUCTURED STORAGE

One of the key decisions to be made when using Oracle XML DB is persist XML documents is when to use structured-storage and when to use unstructured storage.

Unstructured-storage provides for the highest possible throughput when inserting and retrieving entire XML documents. It also provides the greatest degree of flexibility in terms of the structure of the XML that can be stored in a XMLType table or column. These throughput and flexibility benefits come at the expense of certain aspects of intelligent processing. There is little that the database can do to optimize queries or updates on XML that has been stored using a CLOB data type.

Structured-storage provides a number of advantages for managing XML, including optimized memory management, reduced storage requirements, b-tree indexing and in-place updates. These advantages come at a cost of somewhat increased processing overhead during ingestion and retrieval and reduced flexibility in terms of the structure of the XML that can be managed by a given XMLType table or column.

The relevant merits of Structured and Unstructured storage are outlined in the following table

	Unstructured Storage	Structured Storage
Throughput	Highest possible throughput when ingesting and retrieving the entire content of an XML document.	The decomposition process results in slightly reduced throughput when ingesting or retrieving the entire content of an XML document.
Flexibility	Provides the maximum amount of flexibility in terms of the structure of the XML documents that can be stored in an XMLType column or table.	Limited Flexibility. Only documents that conform to the XML Schema can be stored in the XMLType table or column. Changes to the XML Schema will require evolution of the registered Schema.
XML Fidelity	Delivers Document Fidelity: Maintains the original XML byte for byte, which may be important to some applications	DOM Fidelity: A DOM created from an XML document that has been stored in the database will be identical to a DOM created from the original document. However trailing new lines, white space characters between tags and some data formatting may be lost

Optimized Update Operations	Optimized update operations are not possible. When any part of the document is updated the entire document must be written back to disk.	The majority of update operations can be optimized using Query re-write. This allows in-place, piece-wise update, leading to significantly reduced response times and greater throughput.
XPath based queries.	XPath operations are evaluated by constructing DOM from CLOB and using functional evaluations. This can be very expensive when performing operations on large collections of documents.	Where possible, XPath operations are evaluated using XPath-Rewrite, leading to significantly improved performance, particularly with large collections of documents.
SQL Constraint Support	SQL constraints are not currently available.	SQL constraints are supported.
Indexing Support	Text and Functional indexes.	B-Tree, Text and Functional Indexes.
Optimized Memory Management	XML operations on the document require creating a DOM from the document	XML operations can be optimized to reduce memory requirements.

Table 1: Pros and cons of XML storage options

XML SCHEMA EVOLUTION

XML Schema evolution is the term used to describe the process that takes place when the structure of an XML Schema changes. Oracle Database 10g Release 1 introduces support for XML Schema evolution, allowing a developer to register a new version of a registered XML Schema with Oracle Database 10g Release 1.

The current implementation of XML Schema evolution requires that all instance documents be compliant with the current version of the registered schema. Oracle XML DB allows an XSL style sheet to be used to transform any existing documents into documents that are compliant with the new version of the XML Schema as part of the XML Schema evolution process.

In Oracle Database 10g Release 1, XML Schema evolution makes a complete copy of all of the instance documents as part of the migration process. This means that, like most application upgrades, an upgrade involving XML Schema evolution requires careful planning to ensure that adequate resources are available and application downtime is minimized.

XML / SQL DUALITY

Much of the valuable information within an organization is in the form of semi-structured and unstructured data. Typically this data is contained in files stored on a file server or in a CLOB column inside a database. The information in these files is in proprietary or application specific formats. It can only be accessed via specialist tools, such as word processors or spreadsheets, or programmatically using complex, proprietary APIs. Searching across this information is limited to the facilities provided by a crawler or full text indexing.

One of the major drivers behind the rapid adoption of XML is that it allows for stronger management and more open access to semi-structured and unstructured content. Replacing proprietary file formats with XML allows organizations to achieve much higher levels of reuse of their semi-structured and unstructured data. The content can be accurately described using XML Schema. The content can be easily accessed and updated using standard APIs based on DOM and XPath.

For instance, information contained in an Excel spreadsheet is only accessible to the Excel program, or to a program that uses Microsoft's COM APIs. The same information, stored in an XML document is accessible to any tool that can leverage the XML programming model.

Structured data on the other hand does not suffer from these limitations. Structured data is typically stored as rows in tables within a relational database. These tables are accessed and searched using the relational model and the power and openness of SQL from a variety of tools and processing engines.

By delivering on the promise of XML / SQL duality, Oracle XML DB erases the traditional boundary between applications that work structured data and those that work with semi-structured and unstructured content. With Oracle XML DB the relational and XML metaphors become interchangeable. XML/SQL duality means that the same data can be exposed as rows in a table and manipulated using SQL or exposed as nodes in an XML document and manipulated using techniques like DOM or XSL transformation. The access metaphor and processing techniques used are totally independent of the underlying storage format.

This means that the XML programmer can leverage the power of the relational model when working with XML content and the SQL programmer can leverage the flexibility of XML when working with relational content. This provides application developers with maximum flexibility, allowing them to use the most appropriate tools to solving a particular business problem.

This allows Oracle XML DB to be used to provide new, simple solutions to a number of common business problems.

- Relational data can quickly and easily be converted into HTML pages. Oracle XML DB provides new SQL operators that make it possible to generate XML directly from a SQL query. The XML can be transformed into other formats, such as HTML using the database-resident XSLT processor.
- Organizations can easily leverage all of the information contained in their XML documents without having to incur the overhead of converting back and forth between different formats. Oracle XML DB makes it possible to access XML content using SQL queries, On-line Analytical Processing (OLAP) and Business-Intelligence/Data Warehousing operations.
- Text, Spatial Data, and Multimedia operations can be performed on XML content.

**Oracle XML DB
erases the traditional
boundaries between
structured, semi-
structured and
unstructured content.**

**SQL operations can
be performed on
XML content.**

**XML operations can
be performed on
relational data**

SQL/XML

Oracle XML DB also provides an implementation of the majority of the operators that will be incorporated into the forthcoming SQL/XML standard. SQL/XML is defined by specifications prepared by the International Committee for Information Technology Standards (Technical Committee H2), which is the main standards body for developing standards for the syntax and semantics of database languages, including SQL. See <http://sqlx.org> and http://www.ncits.org/tc_home/h2.htm for more information.)

The SQL/XML operators fall into two categories. The first category consists of a set of operators that make it possible to query and access XML content as part of normal SQL operations. The second category consists of a set of operators that provide an industry standard method for generating XML from the result of a SQL Select statement.

The SQL/XML operators make it possible to address XML content in any part of a SQL Statement. The SQL/XML operators use XPath notation is used to traverse the XML structure and identify the node or nodes on which to operate. XPath is a popular syntax (see <http://www.w3.org/TR/xpath>) familiar to both programmer and content-creators, and the ability to embed XPath expressions within SQL statements greatly simplifies XML access.

The **existsnode()** operator is used in the where clause of a SQL statement to restrict the set of documents returned by a query. The **existsnode()** operator takes an XPath expression and expression and applies it an XML document. The operator and returns true (1) or false (0) depending on whether or not the document contains a node which matches the XPath expression.

The **extract()** operator takes an XPath expression and returns the node or nodes that matches the expression as an XML document or fragment. If a single node matches the XPath expression the result will be a well-formed XML document. If multiple nodes match the XPath expression the result will be a document fragment.

The **extractvalue()** operator takes an XPath expression and returns the corresponding leaf level node. The XPath expression passed to **extractvalue()** should identify a single attribute, or an element which has precisely one text node child. The result is returned in the appropriate SQL data type.

The **updatexml()** operator allows partial updates to be made to an XML document, based on a set of XPath expressions. Each XPath expression identifies a target node in the document, and a new value for that node. The **updatexml()** operator allows multiple updates to be specified for a single XML document.

The **xmlsequence()** operator converts a document fragment into a set of well formed XML documents.

Detailed examples of the way in which these functions are used are provided in the PurchaseOrder example.

XPATH RE-WRITE

The SQL/XML operators, and the corresponding XMLType methods, allow XPath expressions to be used to search collections of XML documents and to access a subset of the nodes contained within an XML document. Oracle XML DB has two methods of evaluating XPath expressions that operate on XMLType columns and tables.

For XML that has been stored using structured storage techniques, Oracle XML DB will attempt to translate the XPath expression in a SQL/XML operator into an equivalent SQL query. The SQL query references the Object-Relational data structures that underpin a schema-based XMLType. While this process is referred to as XPath-Rewrite, it can also occur when performing update operations.

For XML that has been stored using unstructured storage, Oracle XML DB will evaluate the XPath using functional evaluation. Functional evaluation builds a DOM tree for each XML document and then resolves the XPath programmatically using the methods provided by the DOM API. If the operation involves updating the DOM tree, the entire XML document has to be written back to disc when the operation is completed.

**Query Re-Write
makes it possible to
execute select and
update operations
that include XPath
expressions at near-
relational speeds**

XPath-Rewrite makes it possible for the database to efficiently process SQL statements containing one or more XPath expressions using conventional relational SQL. By translating the XPath expression into a conventional SQL statement, Oracle XML DB insulates the database optimizer from having to understand XPath notation and the XML data model. The database optimizer simply processes the re-written SQL statement in the same manner as it would any other SQL statement.

This means that the database optimizer is able to derive an execution plan based on conventional relational algebra. This allows Oracle XML DB to leverage all of the features of the database and ensure that SQL Statements containing XPath expressions are executed in a highly performant and efficient manner. There is very little overhead associated with the process of XPath-Rewrite, and this means that XPath-Rewrite allows Oracle XML DB to execute XPath based queries at near-relational speed while still preserving the XML abstraction.

XPath-Rewrite is possible in the following circumstances.

- The SQL statement contains SQL/XML operators or XMLType methods that use XPath expressions to refer to one or more nodes within a set of XML documents.
- The XMLType column or table containing the XML documents is associated with a registered XML Schema.
- The XMLType column or table uses structured storage techniques to provide the underlying storage model.
- The nodes referenced by the XPath expression can be mapped, via the XML Schema, to attributes of the underlying SQL object model.

The XPath-Rewrite process is as follows:

- Identify the set of XPath expressions included in the SQL statement.
- Translate each XPath expression into an Object Relational SQL expression that references the tables, types and attributes of the underlying SQL: 1999 object model.
- Re-write the original SQL statement into an equivalent Object Relational SQL statement
- Pass the new SQL statement to the database optimizer for plan generation and query execution.

In certain cases XPath-Rewrite is not possible. This can occur when there is no SQL equivalent for a particular XPath expression. In this situation Oracle XML DB will create a DOM and then use the DOM to perform a functional evaluation of the XPath expressions.

In general, functional evaluation of a SQL statement will be much more expensive than XPath-Rewrite, particularly if the number of documents that needs to be processed is large. However the advantage of functional evaluation is that it can be used to evaluate any XPath expression, regardless of whether or not the XMLType is stored using structured storage and regardless of the expression's complexity. When documents are stored using unstructured storage (in a CLOB), functional evaluation will be necessary any time the **extract()**, **extractvalue()**, **updatexml()** operators are used. The **existsNode()** operator will also result in functional evaluation unless a *CTXPath* index or functional index can be used to resolve the query.

Understanding the concept of Query-re-write, and the conditions under which query re-write can take place, is one of the key steps in developing Oracle XML DB applications that will deliver the required levels of scalability and performance

ORACLE DATABASE-NATIVE XQUERY

XQuery 1.0 is an XML Query Language developed by W3C that will become the recommended query language to query XML from a variety of data sources. Various companies are adopting XQuery as the way to query XML stored in database rows or from WebServices and to construct new XML values.

On the SQL side, the XML datatype was introduced in SQL 2003 as a way to encapsulate XML in SQL. The SQL committee is now working to integrate the querying of XML using XQuery. This is being accomplished by introducing a new SQL function: *XMLQuery*, and a new construct: *XMLTable* both of which operate on XML and SQL values using XQuery. The former

is known as **XQuery-centric** approach as it allows querying and constructing XML using XQuery. The latter is known as **SQL-centric** approach as it allows breaking apart the XQuery values into relational values.

Oracle Database 10g Release2 enables XQuery support in the database server through these SQL standard functions.

FLWOR EXPRESSIONS IN XQUERY

At the heart of XQuery is the FLWOR expression that supports iteration and binding of variables to intermediate results. This kind of expression is often useful for computing joins between two or more documents and for restructuring data. The name FLWOR, pronounced "flower", reflects the keywords `for`, `let`, `where`, `order by`, and `return`.

Similar to the `FROM` clause in SQL, the `for` and `let` clauses in a FLWOR expression generate a sequence of tuples of bound variables called the **tuple stream**. Performing the same function as the `WHERE` clause in SQL, the `where` clause serves to filter the tuple stream, retaining some tuples and discarding others. The `order by` clause mimics the `ORDER BY` clause in SQL to impose an ordering on the tuple stream. Finally, the `return` clause works like the `SELECT` clause in SQL to construct the result of the FLWOR expression. The `return` clause is evaluated once for every tuple in the tuple stream, after filtering by the `where` clause, using the variable bindings in the respective tuples. The result of the FLWOR expression is an ordered sequence containing the concatenated results of these evaluations.

XMLQUERY SQL FUNCTION

The XMLQuery function 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.

Below is the syntax that will be supported in Oracle Database 10g Release 2:

```
XMLQUERY (<XQuery-string-literal>
          [PASSING [BY VALUE] <expression-returning-XMLType>]
          RETURNING CONTENT)
```

The XQuery string literal is a complete XQuery expression including the prolog etc. The `PASSING` clause must be followed by an expression returning an XMLType that is used as the context for evaluating the XQuery expression.

To run XQuery on XMLType columns, tables, views, or expressions generated by SQL/XML functions, it is recommended that users pass the value as an argument to the XMLQuery function. However, to query any relational table or view as XML without having to first create SQL/XML views on top of them, users can use Oracle provided XQuery function: `ora:view()` within an XQuery expression.

XMLTABLE SQL CONSTRUCT

The XMLTable construct is used to map the result of an XQuery evaluation into relational rows and columns so that the user can query the XQuery result as a virtual relational table using SQL. The XMLTable construct can only be used in the from clause of SQL queries.

Below is the syntax that will be supported in Oracle Database 10g Release 2:

```
<XML table> ::=
  "XMLTable" "(" (<XQuery-string-literal>
    ["PASSING" ["BY" "VALUE"] <xml-value-expression> ]
    ["COLUMNS" <XML-table-columns>]
    ")"
```

```
<XML-table-columns> ::= <XML-table-column>
                        [", " <XML-table-column>]...
```

```
<XML-table-column> ::= <column-name> [<data-type>]
                        [PATH <string-literal>] [ "DEFAULT" <value-expression> ]
```

REWRITE FOR XQUERY

Similar to XPath-rewrite for XPath-based SQL/XML functions, Oracle's database-native XQuery implementation excels with extensive XQuery rewrites. XQuery rewrites take full advantage of Oracle's high performance relational query engine. With XML documents stored using the structured storage approach, XQuery can be rewritten into pure relational queries to completely avoid building DOM trees of XML documents in memory. Query performance can be orders of magnitude faster with rewrites applied.

In the example below, the XQuery can be rewritten into an equivalent relational query to attain the same performance level as pure relational queries.

```
SELECT XMLQuery(
  'for $b in ora:view("SITE_TAB")/site/people/person
   where $b/@id = "person0"
   return $b/name' returning content)
FROM dual;
```

```
SELECT ( SELECT XMLAgg(XMLElement("name", p.name))
FROM SITE_TAB s, PERSON_TAB p
WHERE p.id = 'person0' AND
p.NESTED_TABLE_ID=s."SYS_NC0004700048$"
)
FROM dual;
```

XML DB REPOSITORY

The relational model, with its powerful table-row-column metaphor, is widely accepted as the most effective mechanism for managing structured data. The relational metaphor is not so effective when it comes to managing semi-structured and unstructured data, such as document- or content-oriented XML. A book does not easily lend itself to being represented as a set of rows in a table. It is more natural to represent a book as a hierarchy, book – chapter – section – paragraph, and to represent the hierarchy as a set of folders and subfolders.

Relational databases are traditionally considered to be poor at managing hierarchical structures and performing the kind of hierarchical traversal operations that are required to resolve a path. In order to resolve this problem, Oracle XML DB introduces the concept of a query-able, hierarchically organized XML Repository. The Oracle XML DB repository allows the hierarchical metaphor to be used to manage document-centric XML content. Using the repository, it is possible to represent XML content as documents in a folder hierarchy, and use hierarchical metaphors, such as paths and URLs, to access documents and represent the relationships between documents.

Oracle XML DB includes a new, patented hierarchical index which speeds up folder- and path-traversals within the Oracle XML DB Repository. The hierarchical index is totally transparent to the end-user, and allows Oracle XML DB to perform folder and path traversals at speeds that are comparable to or faster than conventional file-systems.

One of the major advantages of the Oracle XML DB repository is that it allows content authors and editors to work directly with XML content stored in the database. The majority of the tools used to author XML are now able to access content using internet protocols like HTTP, FTP and WebDAV. Oracle XML DB adds native support for these popular protocols to the database, allowing authors and editors direct access to content managed by the Oracle XML DB repository. Oracle XML DB also provides the application programmer with direct access to the repository from both SQL and PL/SQL. This makes it possible to develop applications which access and update content managed by the Oracle XML DB repository.

The WebDAV protocol is an IETF standard that defines a set of extensions to HTTP that allow an HTTP Server to act as a file server for a DAV enabled client. 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. This terminology is adopted by the Oracle XML DB repository.

Since the HTTP, FTP and WebDAV protocols were designed with document-centric operations in mind, they are typically more efficient than Oracle NET for manipulating large volumes of content. By providing support for these protocols at the database level Oracle XML DB allows Windows Explorer™, Microsoft Office™ and products from vendors like AltaVista™, Macromedia™ and Adobe™ to work directly with XML content stored in the Oracle XML DB repository.

Oracle Database 10g Release 1 adds NLS support for the Protocols supported by Oracle XML DB. This allows documents to be loaded into the repository from clients using different character sets. While content is loaded into the Oracle XML DB repository, it is converted into the database character set. When documents are retrieved from the repository they are converted back into the client's character set. With HTTP and WebDAV this process is automatic, based on the mechanisms defined by the HTTP protocol. With FTP, **quote** commands are provided that allow the client to identify the client character set.

The Oracle XML DB Repository makes it possible to use a familiar file/folder metaphor to store, organize and access XML content stored in the database

Support for standard Internet Protocols like WebDAV and FTP allow standard tools like Windows Explorer to access content stored in the Oracle XML DB repository

The following screen shot shows the root level directory of the Oracle XML DB repository as seen from Microsoft's Windows Explorer™.

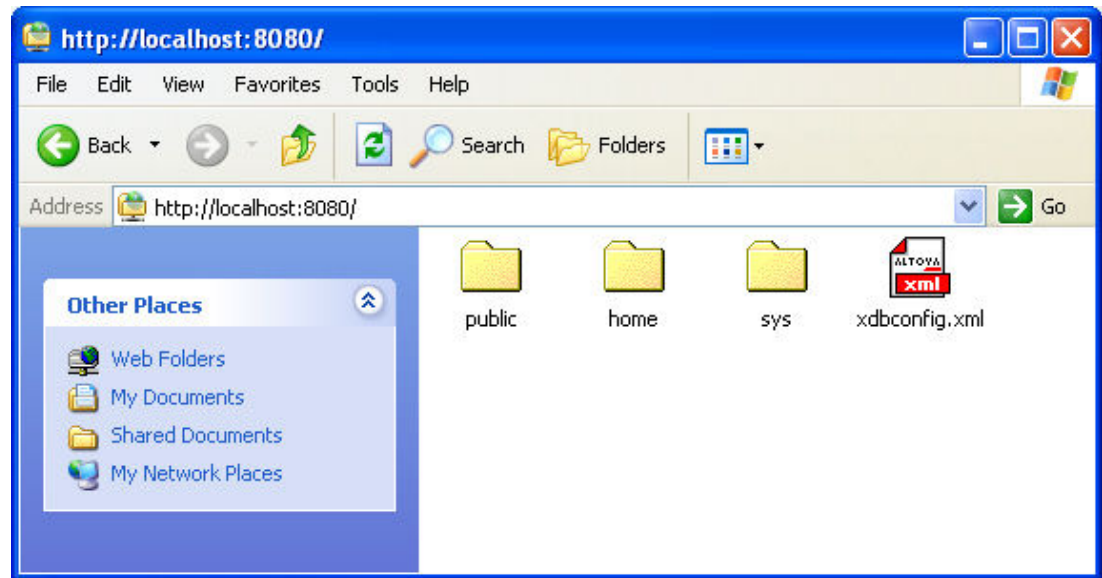


Figure III. Microsoft Web Folder™'s view of Oracle XML DB repository.

As can be seen, WebDAV clients, such as Microsoft's Windows Explorer™, can connect directly to the XML DB repository. No additional Oracle or Microsoft specific software or other complex middleware needs to be installed in order to enable this functionality. This means that end users can work directly with the Oracle XML DB Repository using the tools and interfaces that they are already familiar with.

Each document in the Oracle XML DB is secured with an Access Control List. An Access Control List (ACL) is an XML document that contains a set of Access Control Entries (ACE). Each ACE grants or revokes a set of permissions to a particular user or group (database role). This access control mechanism is based on the WebDAV specification. The repository also provides support for basic versioning based on the WebDAV standard.

Another benefit of the XML DB Repository is that it is query-able from SQL. Content stored in the Oracle XML DB repository can be accessed and updated from SQL and PL/SQL. It is possible to interrogate the structure of the repository in complex ways ("how many documents with a .xsl extension are under a location other than /home/mystylesheetdir"?). It is also possible to mix path-based repository access with content-based access to the documents ("how many documents not under /home/purchaseOrders have a node named "/PurchaseOrder/User/text() with a value of "SBELL"?)

All the meta-data required to manage the Oracle XML DB repository is stored in a database schema owned by the database user XDB. This user is created as part of the Oracle XML DB installation procedure. The primary table in this schema is an XMLType table called **XDB\$RESOURCE**. The table contains one row for each file or folder managed by the Oracle XML DB repository. The documents in this table are referred to as *resource* documents.

**A new type of Index,
the Hierarchical
Index, makes path-
based access as fast
as Primary Key
based access**

The **XDB\$RESOURCE** table is not directly exposed to the SQL programmer. Instead the contents of the repository are exposed via two public views, the **RESOURCE_VIEW** and the **PATH_VIEW**. The **RESOURCE_VIEW** and **PATH_VIEW** provide the SQL programmer the ability to access and update both the meta-data and content of a document stored in the Oracle XML DB repository.

Both the **RESOURCE_VIEW** and the **PATH_VIEW** contain a column called **RES**. This is an XMLType column that can be used to access and update the *resource* associated with a document stored in the Oracle XML DB repository. These views also make it possible to create SQL statements which access and update *resource* documents based on a path notation. Operations on these views translate into operations on the underlying tables in the Oracle XML DB repository.

Oracle XML DB includes two new repository specific SQL operators: **exists_path()** and **under_path()**. These operators make it possible to include path based predicates in the *where* clause of a SQL statement. This means that SQL operations can select repository content based on the location of the content within the folder hierarchy. The Hierarchical Index ensures that path-based queries are executed very efficiently.

When Schema Based XML documents are stored in the Oracle XML DB repository the document content is stored as an object in the default table identified by the XML Schema. The repository contains the meta-data about the document and a pointer (**REF of XMLTYPE**) that identifies the row in the default table that contains the content.

It is also possible to store other kinds of documents in the repository. When non-xml and non-schema based XML documents are stored in the repository, the content of the document is stored in a LOB alongside the meta-data about the document

Since the Oracle XML DB repository can be accessed and updated using SQL, any application capable of calling a PL/SQL procedure can work with the Oracle XML DB repository. All SQL and PL/SQL repository operations are transactional, and access to the repository and its contents is subject to database security as well as XML DB Repository Access Control Lists (ACLs). When repository content is accessed via SQL ACL based security is enforced via Row Level security.

The supplied PL/SQL packages **DBMS_XDB**, **DBMS_XDBZ** and **DBMS_XDB_VERSION** provide the SQL programmer the ability to perform common tasks on the repository itself. The methods provided by these packages make it possible to create, delete and rename documents and folders, to move a file or folder within the folder hierarchy, to set and change the access permissions on a file or folder and the ability to initiate and manage versioning.

The following diagram show the overall architecture of the Oracle XML DB repository

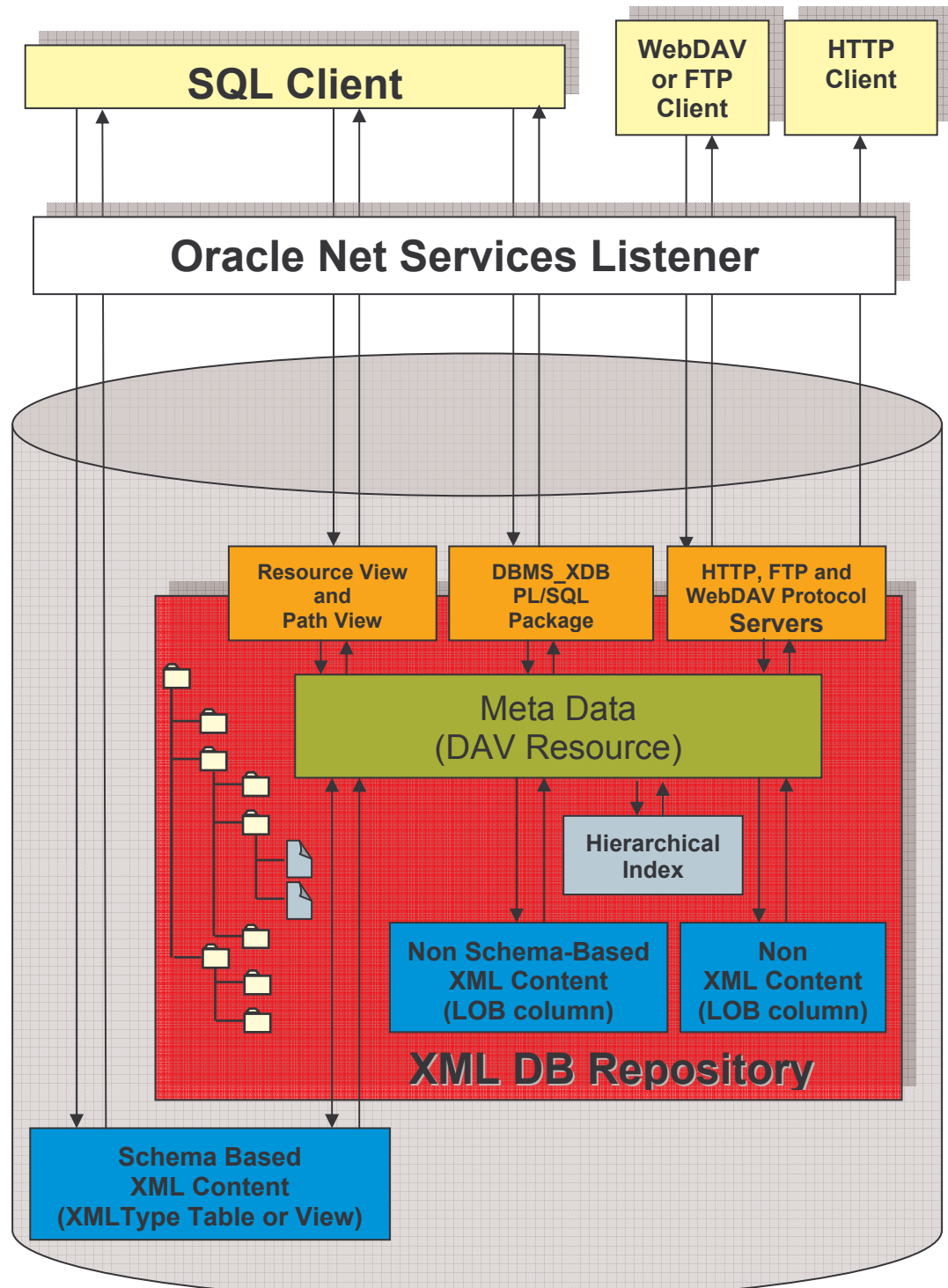


Figure IV. Oracle XML DB repository architecture overview

SQL and non SQL clients access the repository using the Oracle Net Services Listener

SQL clients access the repository via the `PATH_VIEW` and `RESOURCE_VIEW` or the `DBMS_XDB` PL/SQL package

Non-SQL clients access content via the Oracle XML DB protocol servers.

The protocol servers are Oracle MTS processes

All meta-data is managed by the Oracle XML DB repository.

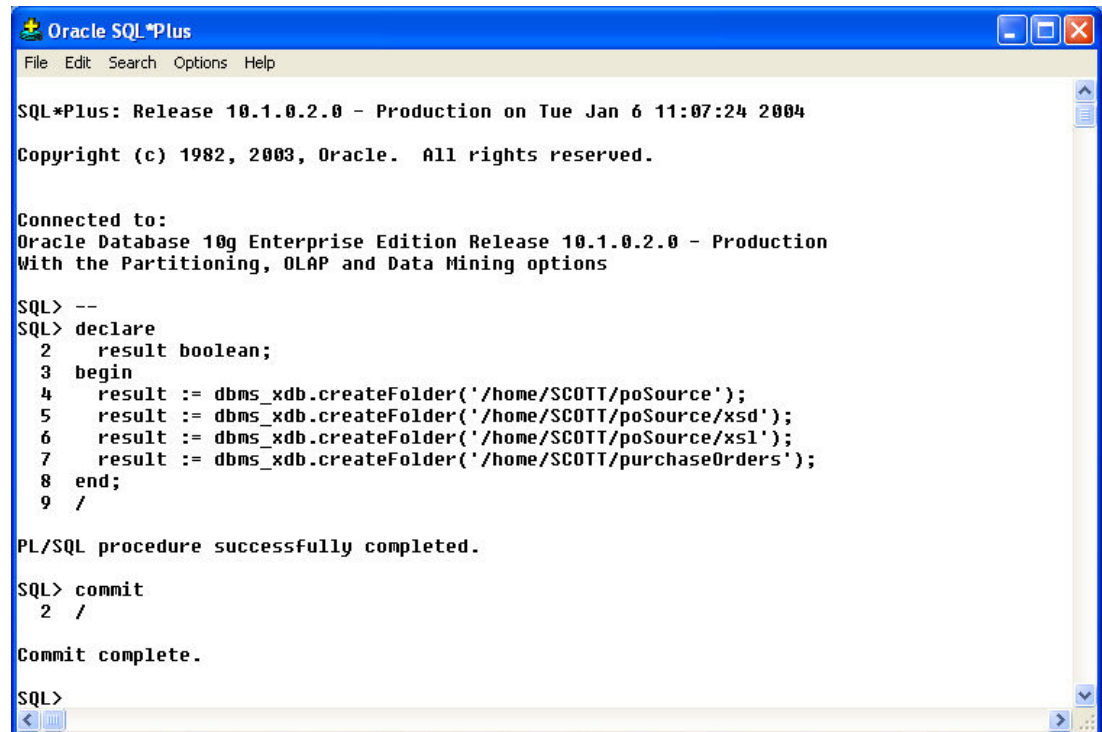
All content, other than schema-based XML is stored in the repository.

The content of Schema-based XML documents is stored in the default table defined by the Schema.

The Hierarchical Index enables high speed lookup based on a path

SQL clients can access schema-based XML directly or via the repository

The following screen shot shows the PL/SQL package DBMS_XDB being used to create a set of subfolders beneath the folder `/home/SCOTT`.



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Jan 6 11:07:24 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> --
SQL> declare
2   result boolean;
3   begin
4     result := dbms_xdb.createFolder('/home/SCOTT/poSource');
5     result := dbms_xdb.createFolder('/home/SCOTT/poSource/xsd');
6     result := dbms_xdb.createFolder('/home/SCOTT/poSource/xsl');
7     result := dbms_xdb.createFolder('/home/SCOTT/purchaseOrders');
8   end;
9   /

PL/SQL procedure successfully completed.

SQL> commit
2   /

Commit complete.

SQL>

```

Figure V. Using PL/SQL to update Oracle XML DB Repository

Note that as a consequence of the transactional semantics enforced by the database, folders created using SQL statements will not be visible to other database users until the transaction is committed. Concurrent access to the Oracle XML DB repository is controlled by the same mechanism as is used to control concurrency in the database. The integration of the repository with the database brings the benefits of strong management to XML content.

One common problem encountered when using a relational database to maintain hierarchical folder structures is the problem of ensuring a high degree of concurrency when adding and removing items in a folder. In conventional file system there is no concept of a transaction. Each operation (add a file, create a subfolder, rename a file, delete a file, etc.) is treated as an atomic transaction. Once the operation has completed the change is immediately available to all other users of the file system.

As the above example shows, one of the key advantages of the Oracle XML DB Repository is the ability to use SQL to perform repository operations in the context of a logical transaction. This means that applications can create long-running transactions that include updates to one or more folders. In this situation a conventional locking strategy that takes an exclusive lock on each updated folder or directory tree would quickly result in significant concurrency problems and performance degradation.

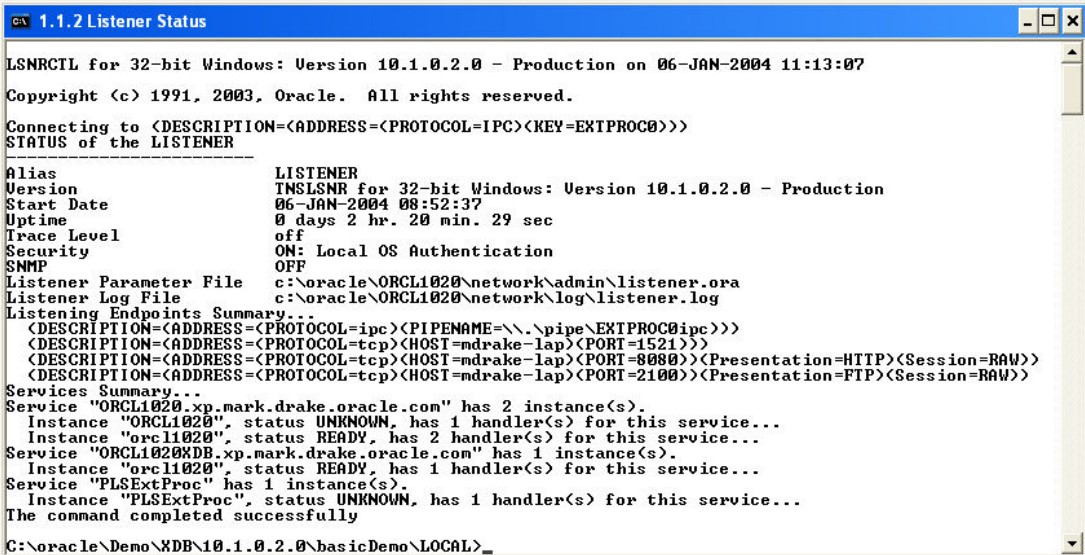
XML DB solves this problem by providing for *name-level locking* rather than *folder-level locking*. Repository operations such as creating, renaming, moving or deleting a sub-folder or file do not require that the user performing the operation be granted an exclusive write lock on the target folder. The Repository manages concurrent folder operations by locking the name within the folder rather than the folder itself. The name and the modification type are put on a queue. Only when the transaction is committed is the folder locked and the contents of the folder modified. This model allows Oracle XML DB to allow multiple applications to perform concurrent updates on the contents of a folder. The queue is also used to manage concurrency with the folder by preventing two applications from creating objects with the same name. Queuing folder modifications until commit time also has the side benefit of minimizing I/O when a number of changes are made to a single folder in the same transaction.

This concurrency problem is seen most commonly in situations where a number of applications are generating files quickly in the same directory, such as when generating trace or log files, or when maintaining a spool directory for printing or email delivery.

THE ORACLE XML DB PROTOCOL ARCHITECTURE

One of the key features of the Oracle XML DB architecture is that the HTTP, WebDAV and FTP protocols are supported using the same architecture that is used to support Oracle Net Services in a Shared Server configuration. The Listener listens for HTTP and FTP requests in the same way that it listens for Oracle Net Services requests. When the listener receives an HTTP or FTP request it hands it off to an Oracle Shared Server process which services it and sends the appropriate response back to the client.

As can be seen from the following screen short, the TNS Listener command **lsnrctl status** can be used to verify that HTTP and FTP support has been enabled.



```

1.1.2 Listener Status
LSNRCTL for 32-bit Windows: Version 10.1.0.2.0 - Production on 06-JAN-2004 11:13:07
Copyright (c) 1991, 2003, Oracle. All rights reserved.
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC)))
STATUS of the LISTENER
Alias LISTENER
Version TNSLSMR for 32-bit Windows: Version 10.1.0.2.0 - Production
Start Date 06-JAN-2004 08:52:37
Uptime 0 days 2 hr. 20 min. 29 sec
Trace Level off
Security ON: Local OS Authentication
SNMP OFF
Listener Parameter File c:\oracle\ORCL1020\network\admin\listener.ora
Listener Log File c:\oracle\ORCL1020\network\log\listener.log
Listening Endpoints Summary...
(DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(PIPENAME=\\.\pipe\EXTPROC0ipc)))
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=mdrake-lap)(PORT=1521)))
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=mdrake-lap)(PORT=8080))(Presentation=HTTP)(Session=RAW))
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=mdrake-lap)(PORT=2100))(Presentation=FTP)(Session=RAW))
Services Summary...
Service "ORCL1020.xp.mark.drake.oracle.com" has 2 instance(s).
Instance "ORCL1020", status UNKNOWN, has 1 handler(s) for this service...
Instance "orcl1020", status READY, has 2 handler(s) for this service...
Service "ORCL1020XDB.xp.mark.drake.oracle.com" has 1 instance(s).
Instance "orcl1020", status READY, has 1 handler(s) for this service...
Service "PLSExtProc" has 1 instance(s).
Instance "PLSExtProc", status UNKNOWN, has 1 handler(s) for this service...
The command completed successfully
C:\oracle\Demo\XDB\10.1.0.2.0\basicDemo\LOCAL>

```

Figure VI. Listener Status, with FTP and HTTP protocol support enabled

PROGRAMMATIC ACCESS

All of the Oracle XML DB functionality is accessible from 'C', PL/SQL and Java. Today, the most popular methods for building web-based applications are servlets/JSPs (Java Server Pages) and XSL/XSPs (XML Style sheets / XML Server Pages). Typically, servlets and JSPs access data via JDBC, while XSL/XSPs expect data in the form of XML documents, which are processed via a DOM (Document Object Model) API implementation. Oracle XML DB supports both styles of application development. 'C', Java and PL/SQL implementations of the DOM API are provided.

Applications that make use of JDBC, such as those based on servlets, need to have some advance knowledge of the structure of the data they are going to process. The Oracle JDBC drivers allow application programmers to access and update XMLType tables and columns, and to call the PL/SQL procedures that access the Oracle XML DB repository.

Applications that make use of DOM, such as those based on XSLT transformations; typically require less knowledge of the data structure. A DOM based application uses string names to identify pieces of content, and must dynamically walk through the DOM tree to find the required information. Oracle XML DB allows application developers to use the DOM API to access and update XMLType columns and tables. Programming to a DOM API is more flexible than programming via JDBC, but it may require more resources at run time.

ORACLE XML DB PERFORMANCE

One common objection to using XML to represent data is that it generates higher overhead than other representations. Oracle XML DB incorporates a number of features that are specifically designed to address this issue by significantly improving the performance of XML processing.

XML STORAGE REQUIREMENTS

Surveys have shown that data represented in XML and stored in a text file is three times the size of the same data in a Java object or in relational tables. There are two reasons for this. First, tag names (metadata describing the data) and white space (formatting characters) take up a significant amount of space in the document, particularly for highly structured, data-centric XML. Secondly, all of the data in an XML file is represented in human readable (string) format. In the case of numeric data the string representation of a numeric value needs about twice as many bytes as the native (binary) representation

When XML documents are stored in Oracle XML DB using the structured storage option, the 'shredding' process discards all of the tags and white space contained in the document. The amount of space saved by this optimization depends on the ratio of tag names to data, and the number of collections in the document. For highly-structured, data-centric XML the savings can be significant. When a document is printed, or when node-based operations such as XPath evaluations take place, Oracle XML DB uses the information contained in the associated XML Schema to dynamically reconstruct any necessary tag information.

XML MEMORY MANAGEMENT

The Document Object Model (DOM) is the dominant programming model for XML documents. The DOM APIs are very easy to use but the DOM Tree that underpins them is expensive to generate, in terms of memory. A typical DOM implementation maintains approx 80 – 120 bytes of system overhead for each node in the DOM tree. This means that for highly structured data the DOM tree can require 10-20 times more memory than the document on which it is based.

A conventional DOM implementation requires that the entire contents of an XML document be loaded into the DOM tree before any operations can take place. If an application only needs to process a small percentage of the nodes in the document this is extremely inefficient in terms of both memory and processing overhead. The alternative SAX approach reduces the amount of memory required to process an XML document, but has the major disadvantage that it only allows linear processing of the nodes contained in the XML Document.

Oracle XML DB reduces the memory overhead associated with the DOM programming model by managing schema-based XML documents using an internal in-memory structure called a XML Object (XOB). A XOB is much smaller than the equivalent DOM since it does not duplicate information like tag names and node types, which can easily be obtained from the associated XML Schema. Oracle XML DB will automatically use a XOB whenever an application is working with the contents of a schema-based XMLType. The use of the XOB is transparent to the application developer; it is hidden behind the XMLType data-type and the 'C', PL/SQL and Java APIs.

The XOB is also able to reduce the amount of memory required to work with an XML document via a feature called the Lazily-Loaded Virtual DOM. This feature allows Oracle XML DB to defer loading the in-memory representation of nodes that are part of sub-elements or collection until some methods attempt to operate on a node within that object. Consequently, if an application only operates on few nodes in a document, only those nodes and their immediate siblings will be loaded into memory.

The XOB can only be used in conjunction with a XML document that is based on an XML Schema. If the contents of the XML document are not based on an XML Schema, a traditional DOM will be used instead of the XOB.

XML PARSING OPTIMIZATIONS

In order to populate a DOM tree the application must parse the XML document. The process of creating a DOM tree from an XML file is very CPU- intensive. In a typical DOM based application, where the XML documents are stored as text, every document has to be parsed and loaded into the DOM tree before the application can work with it. If the contents of the DOM tree are updated the whole tree has to be serialized back into a text format and written out to disk.

Oracle XML DB eliminates the need to keep re-parsing documents. Once an XML document has been stored using structured storage techniques no further parsing is required when the document is loaded from disk into memory. Oracle XML DB is able to map directly between the on-disk format and in-memory format using information derived from the associated XML Schema. When changes are made to the contents of a schema-based XMLType, Oracle XML DB is able to write just the updated data back to disk.

Once again, when the contents of the XMLType are not based on an XML Schema a traditional DOM will be used instead.

NODE SEARCHING OPTIMIZATIONS

Most DOM implementations suffer from the fact that they use string comparisons when searching for a particular node within the DOM tree. This means that performing even a simple search of a DOM tree can require hundreds or thousands of instruction cycles.

Searching for a node in a XOB is much more efficient than searching for a node in a DOM. A XOB is based on a computed offset model (similar to a C/C++ object) and it uses dynamic hash-tables rather than string comparisons to perform node searches.

XML SCHEMA OPTIMIZATIONS

Making use of the powerful features associated with XML Schema in a conventional XML application typically generates significant amounts of additional overhead. For instance, before an XML document can be validated against an XML Schema, the schema itself must be located, parsed and validated.

Oracle XML DB minimizes the overhead associated with using XML Schema. When an XML Schema is registered with the database it is loaded in the Oracle XML DB Schema cache, along with all of the meta-data required to map between the XML, XOB and on-disk representations of the data. This means that once the XML Schema has been registered with the database no additional parsing or validation of the XML Schema is required before it can be used. The schema cache is shared by all users of the database. Whenever an Oracle XML DB operation requires information contained in the XML Schema it is able access the required information directly from the cache.

LOAD BALANCING

Some operations, such as performing a full Schema validation, or serializing an XML document back into text form can still require significant memory and CPU resources.

Oracle XML DB allows these operations to be off-loaded to the client or mid tier processor. The OCI interface and thick JDBC driver both allow the XOB to be managed by the client. The cached representation of the XML Schema can also be downloaded to the client. This allows operations like XML printing, and XML Schema validation to be performed using client or mid-tier resources, rather than server resources.

NON-NATIVE CODE

Another bottleneck for XML-based Java applications is the cost associated with parsing an XML file. Even natively compiled or JIT compiled Java performs XML parsing operations twice as slowly as native 'C' implementations. One of the major performance bottlenecks in implementing XML applications is the cost of transforming the data contained in an XML document between text, Java and the native server representations. The cost of performing these transformations is proportional to the size and complexity of the XML file and becomes quite severe with even moderately sized files.

Oracle XML DB addresses these issues by implementing all of the Java and PL/SQL interfaces as very thin facades over a native 'C' implementation. This provides for language-neutral XML support (Java, 'C', PL/SQL and SQL are all using the same underlying implementation), as well as for high performance XMP parsing and DOM processing.

TYPE CONVERSIONS

One of the biggest bottlenecks in using Java in conjunction with XML is the cost of type conversions. Internally Java uses UCS-2 to represent all character data. Most XML files and databases do not contain UCS-2 encoded data. This means that all the data contained in an XML file has to be converted from 8 Bit or UTF8 encoding into UCS-2 encoding before it can be manipulated inside a Java program.

XML DB addresses these problems with lazy type conversions. Lazy type conversions mean that the contents of a given node will be not be converted into the format required by Java until the application attempts to access the contents of the node. Data remains in the internal representation until the last possible moment. Avoiding unnecessary type conversions can result in significant performance improvements in the cases where an application only needs to access a few of the nodes contained in an XML document.

Consider the case of a JSP that wants to load a name from the database and print it out in the generated HTML output. Typical JSP implementations would read the name from the database (which probably contains data in the ASCII or ISO8859 character sets) convert the data to UCS-2, and return it to Java as a String. The JSP wouldn't look at the contents of the string, but merely print it out after printing the enclosing HTML, probably converting back to the same ASCII or ISO8859 for the client browser. XML DB provides a **write** interface on XMLType so that any element can write itself directly to a stream (such as a ServletOutputStream) without conversion through Java character sets.

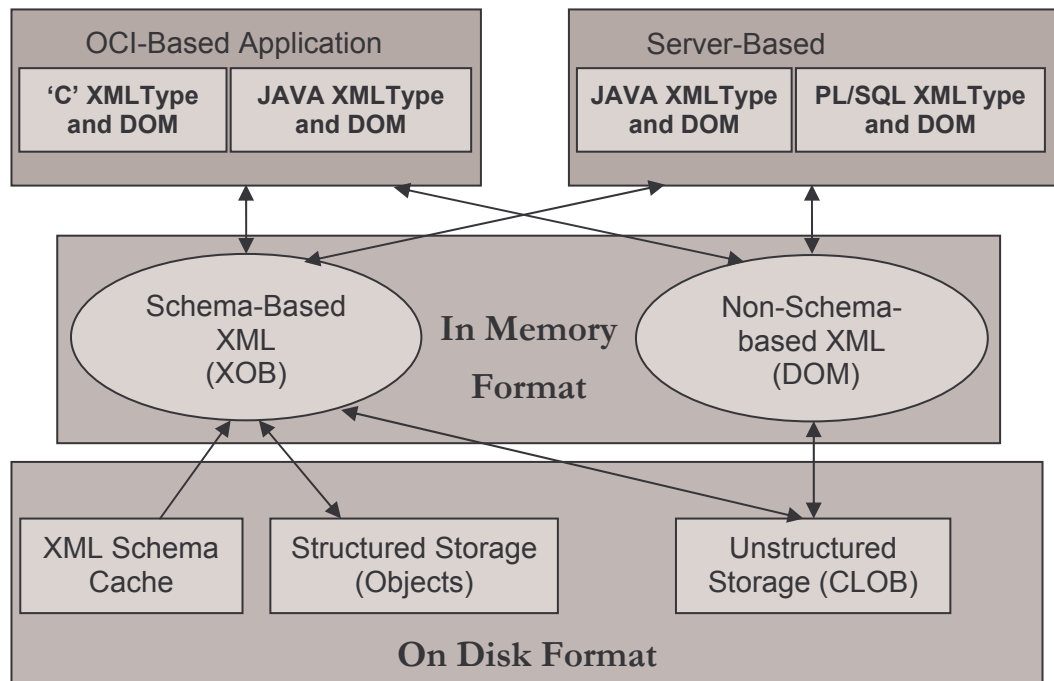


Figure VII. Oracle XML DB Application Programming Stack

XML MANAGEABILITY

Unlike niche ‘native XML’ database, with Oracle XML DB there is no need to compromise on the enterprise-class database features in order to get native XML support. Oracle XML DB is not a separate server; it is an integral part of the Oracle database, providing all of the reliability, high availability, scalability and unbreakable security features needed to run mission-critical applications.

MANAGING ORACLE XML DB APPLICATIONS WITH ORACLE ENTERPRISE MANAGER

You can use Oracle Enterprise Manager (Enterprise Manager) to manage and administer your Oracle XML DB application. Enterprise Manager's graphical user interface facilitates your performing the following tasks:

- Configuration
 - Configuring Oracle XML DB, including protocol server configuration
 - Viewing and editing Oracle XML DB configuration parameters
 - Registering XML schema
- Create resources
 - Managing resource security, such as editing resource ACL definitions
 - Granting and revoking resource privileges
 - Creating and editing resource indexes
 - Viewing and navigating Oracle XML DB Repository
- Create XML schema-based tables and views
 - Creating your storage infrastructure based on XML schemas
 - Editing an XML schema
 - Creating an XMLType table and a table with XMLType columns
 - Creating a view-based XML schema
 - Creating a function-based index based on XPath expressions

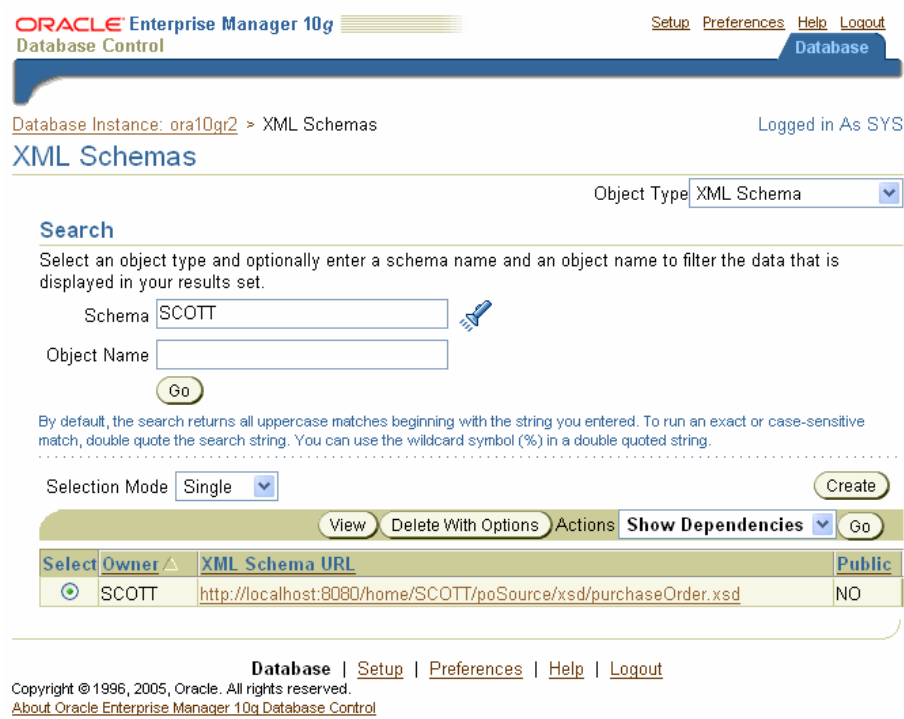


Figure VIII. Oracle Enterprise Manager XML Schema management

The rest of this white paper will take a detailed look at the above features in terms of a simple example of using Oracle XML DB to manage XML documents.

SAMPLE APPLICATION: **USING ORACLE XML DB TO MANAGE PURCHASE ORDERS**

This sample application attempts to show how some of the concepts outlined in the first section of the white paper can be used. It based around the idea of a XML representation of a Purchase Order. In this example, A Purchase Order is represented by a data-centric, highly structured XML document. Each Purchase Order is compliant with a XML Schema.

LOADING CONFIGURATION DOCUMENTS INTO THE ORACLE XML DB REPOSITORY

Many of the operations associated with configuring and using Oracle XML DB are based on processing one or more XML documents. Examples of this include registering an XML Schema and performing an XSL transformation. The easiest way to make the necessary documents available to Oracle is to load them into the Oracle XML DB repository.

WebDAV support makes it possible to load these documents from a local file system into the Oracle XML DB repository using Windows Explorer™. The following screen shot shows a simple drag and drop operation being used to copy the contents of the poSource folder from the local hard drive into the Oracle XML DB Repository.

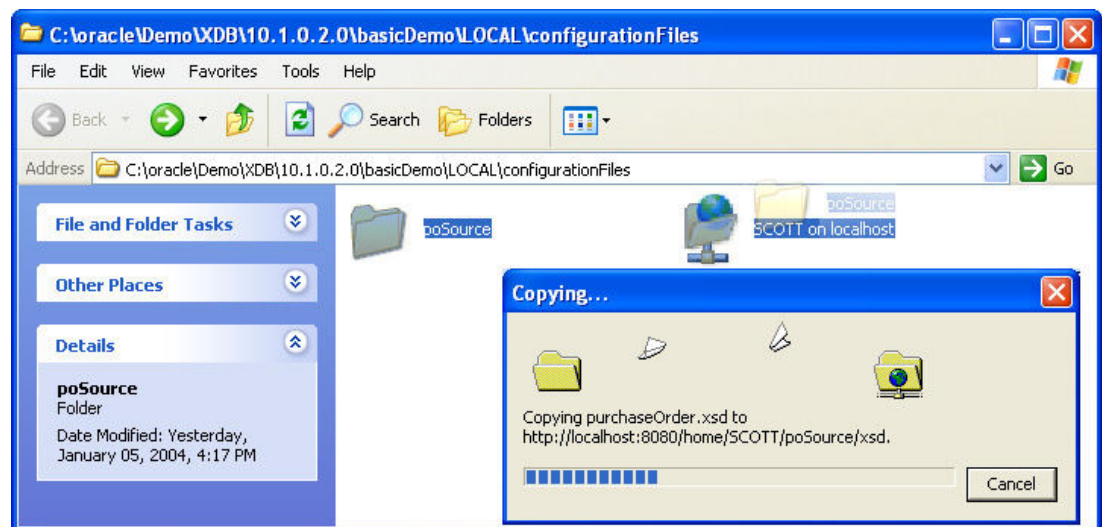


Figure IX. Using Windows Explorer™ to load content into the Oracle XML DB Repository

KEY POINTS:

- Windows Explorer™ and similar tools that support the WebDAV protocol can be used to upload documents into the XML DB repository.
- This procedure uploaded a directory tree containing an XML Schema document, an HTML page and a couple of XSLT style sheets.
- The Oracle XML DB repository can be used to store non XML content, such as HTML files, JPEG images, word documents etc, as well as Schema based and non-Schema based XML content.

ORACLE XML DB AND THE W3C XML SCHEMA STANDARD

CREATING AN XML SCHEMA

The following screen shots show a simple XML Schema being displayed using XMLSpy. XMLSpy is a XML Schema and document editing tool created by Altova. See <http://www.altova.com> for more details about the tool and its features

One of the major features of the XMLSpy IDE is that it provides a developer with a graphical, easy to use, interface for creating and editing an XML Schema. XMLSpy also supports both WebDAV and FTP protocols. This allows XMLSpy to directly access content stored in Oracle XML DB Repository.

In this example XMLSpy is displaying a graphical representation of a simple XML Schema that defines a PurchaseOrder XML document.

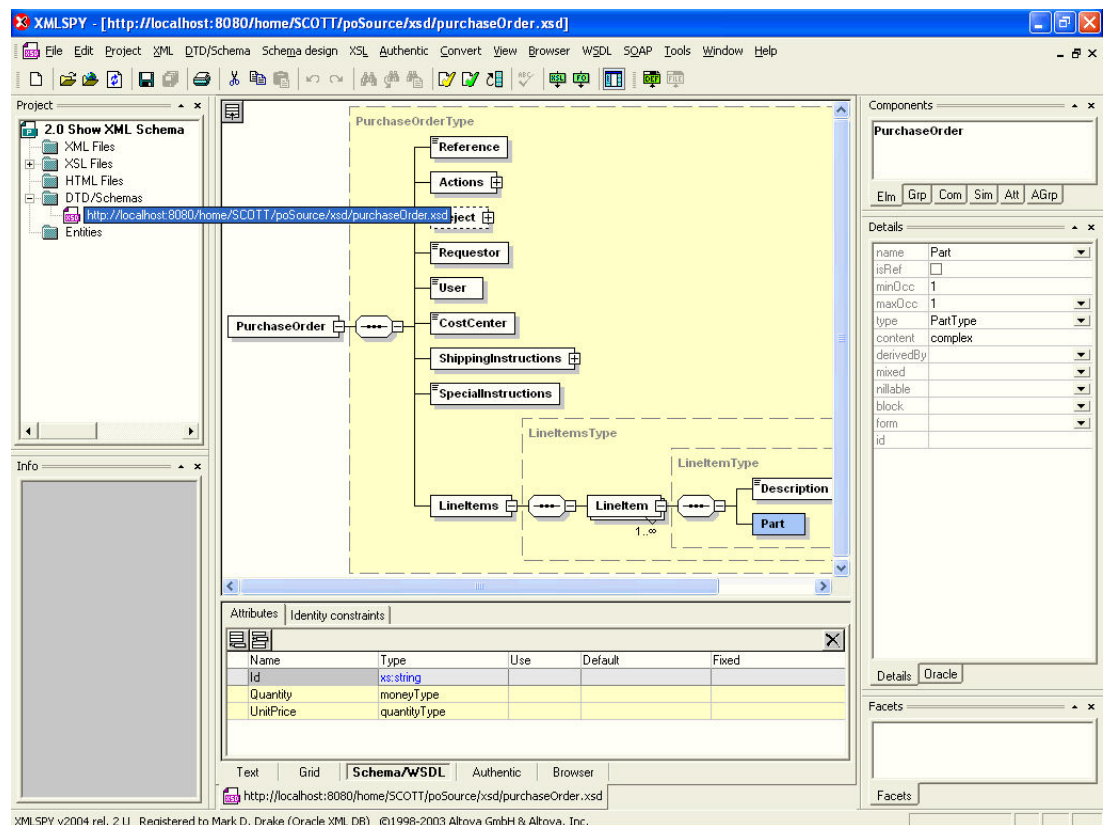


Figure X. XMLSpy showing a Graphical Representation of an XML Schema

KEY POINTS:

- 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 that 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*.

ANNOTATING AN XML SCHEMA

Database administrators and Application developers can annotate the XML Schema to control the naming of the Tables, SQL Objects and SQL Attributes derived from the XML Schema. Annotations can also override the default mapping between the XML Schema data types and SQL data types.

Annotations also make it possible to control how collections with the XML Schema are stored in the database. The available options include storing the collection as CLOB; storing the entire collection as a VARRAY of objects stored in a LOB column; storing the collection as a set of rows in an Index Organized Nested Table, or storing the collection as a set of rows in a separate XMLType table. These options provide significant opportunities to tune the performance of applications that make use of the XMLType data-type to store XML in the database.

XMLSpy also allows the XML Schema editor to work directly with the XML Schema in its native form as can be seen from the following screen shot. This screen shot shows how annotations are added to the XML Schema.

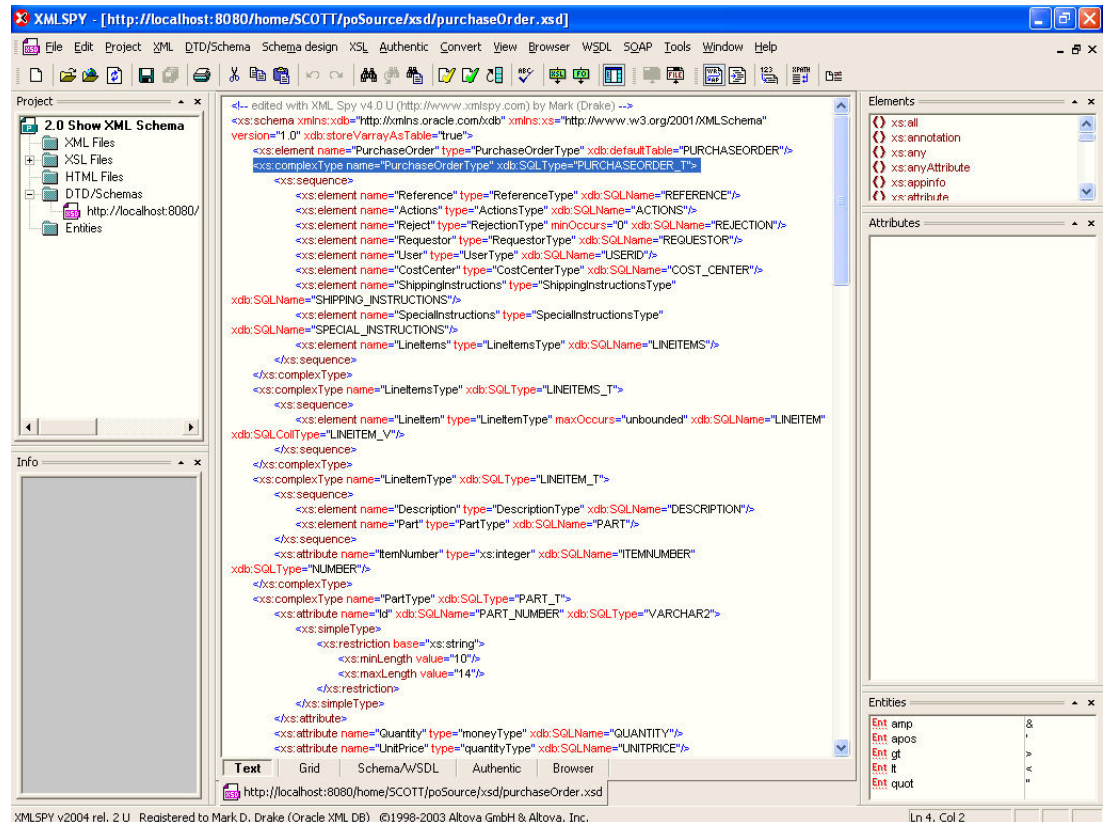


Figure XI. XMLSpy showing the native Representation of an XML Schema

KEY POINTS:

- This schema defines two namespaces.

The first, <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.

The second, <http://xmlns.oracle.com/xdb> is the namespace reserved by Oracle for the Oracle XML DB schema annotations. This namespace is used to add annotations to the 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.
- Oracle XML DB can register an XML Schema that contains no annotations. It makes use of a set of default assumptions to register the XML Schema. The annotations provide the application developer or database administrator with the ability to override these assumptions.

XMLSpy provides an Oracle tab that allows Oracle XML DB Schema Annotations to be entered while working in graphical editing mode.

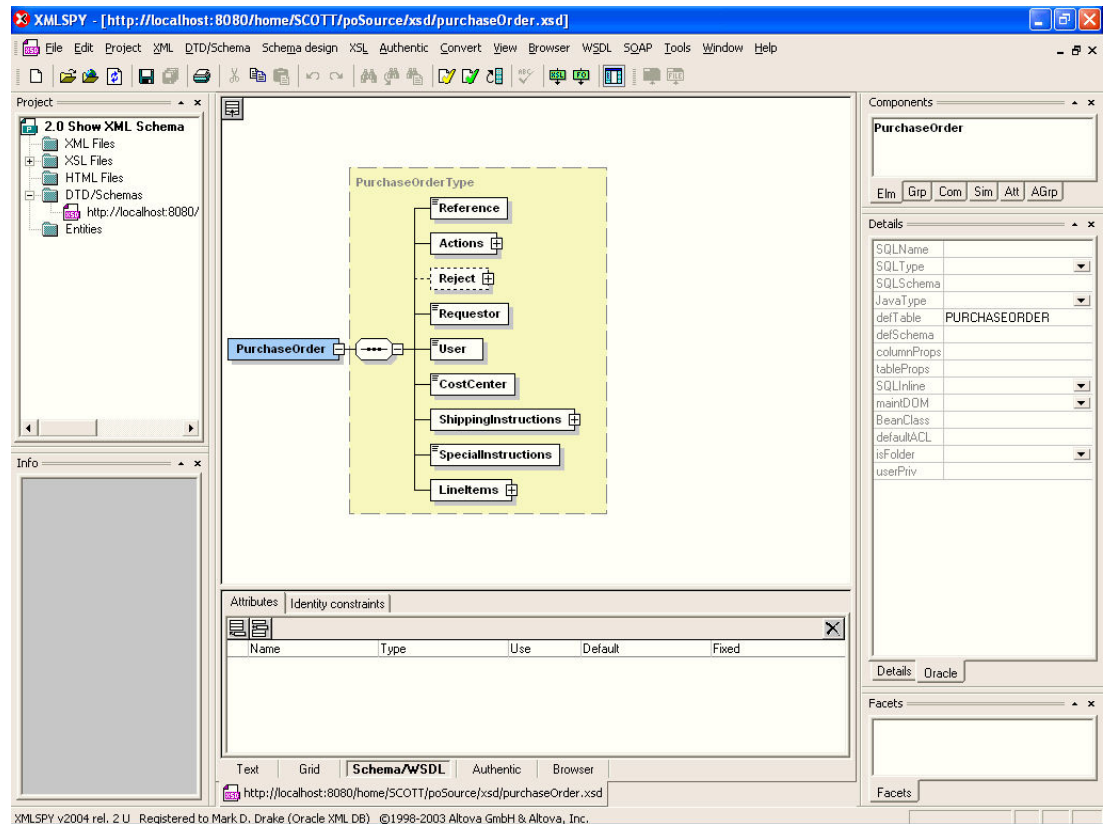


Figure XII. XMLSpy showing support for Oracle XML DB Schema Annotations

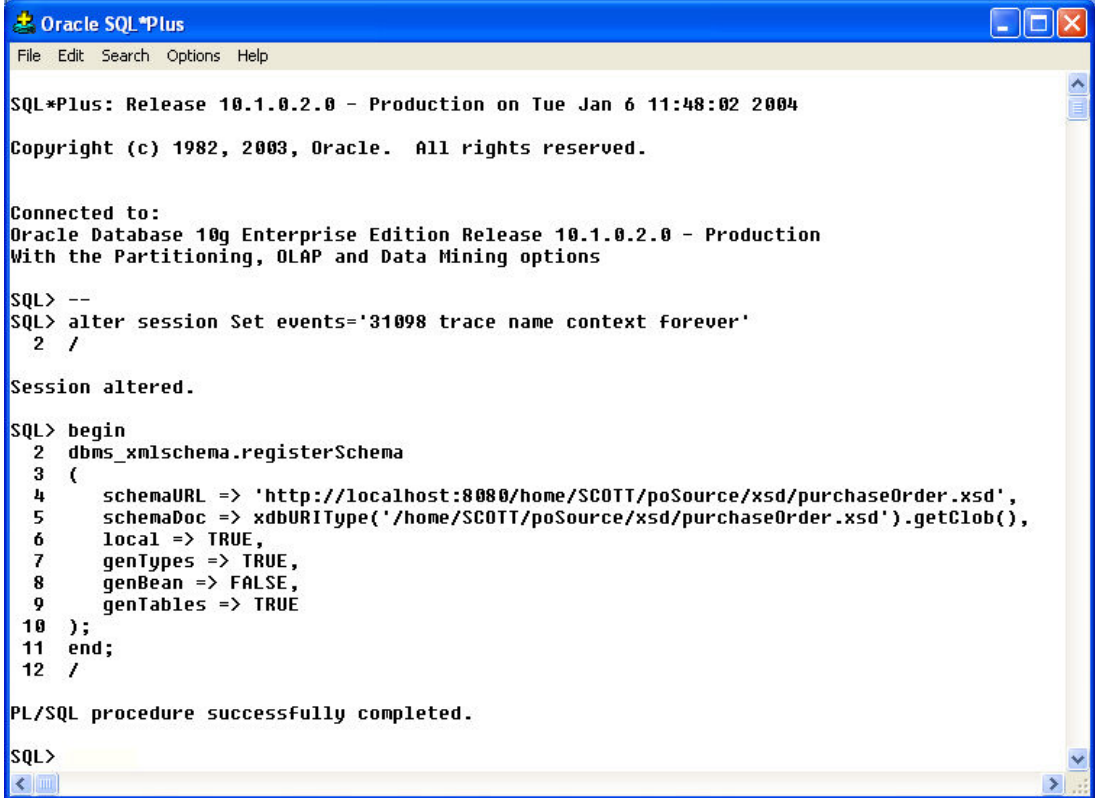
KEY POINTS:

- In this schema the following annotations are being used:
- The storeVarrayAsTable annotation is used to force collections to be stored as Nested Tables.
- The defaultTable annotation is used in the PurchaseOrder element to define that XML documents, compliant with this schema will be stored in a table called PURCHASEORDER
- The SQLType annotation is used to provide an explicit name for the SQL Type that will be generated from the complexType PurchaseOrderType.

REGISTERING AN XML SCHEMA

Before Oracle XML DB can manage the instance documents associated with a given XML Schema the XML Schema must be registered with the database. XML Schema registration is performed using a simple PL/SQL procedure: called **dbms_xmlschema.registerSchema()**. By default, when the XML Schema is registered, Oracle XML DB will automatically generate all of the SQL Object Types and Object tables required to manage the instance documents.

The following screen shot show the process of registering an XML Schema stored in the Oracle XML DB repository.



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Jan 6 11:48:02 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> --
SQL> alter session set events='31098 trace name context forever'
2 /

Session altered.

SQL> begin
2 dbms_xmlschema.registerSchema
3 (
4   schemaURL => 'http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd',
5   schemaDoc => xdbURIType('/home/SCOTT/poSource/xsd/purchaseOrder.xsd').getClob(),
6   local => TRUE,
7   genTypes => TRUE,
8   genBean => FALSE,
9   genTables => TRUE
10 );
11 end;
12 /

PL/SQL procedure successfully completed.

SQL>
  
```

Figure XIII. Using the DBMS_XMLSCHEMA package to register an XML Schema

By default the XML Schema registration process generates the SQL Objects and XMLType tables required to manage the instance documents.

The following screen shot shows the XMLType Table, and some of the SQL Objects that were created as a result of registering the PurchaseOrder XML Schema.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Jan 6 16:49:42 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL>
SQL> describe purchaseorder
      Name                                         Null?     Type
-----
TABLE of SYS.XMLTYPE(XMLSchema "http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd"
      Element "PurchaseOrder") STORAGE Object-relational TYPE "PURCHASEORDER_T"

SQL> --
SQL> describe PURCHASEORDER_T
PURCHASEORDER_T is NOT FINAL
      Name                                         Null?     Type
-----
SYS_XDBPD$REFERENCE                             XDB.XDB$RAW_LIST_T
ACTIONS                                           VARCHAR2(30 CHAR)
REJECTION                                         REJECTION_T
REQUESTOR                                         VARCHAR2(128 CHAR)
USERID                                            VARCHAR2(10 CHAR)
COST_CENTER                                       VARCHAR2(4 CHAR)
SHIPPING_INSTRUCTIONS_T                         SHIPPING_INSTRUCTIONS_T
SPECIAL_INSTRUCTIONS                             VARCHAR2(2048 CHAR)
LINEITEMS_T                                     LINEITEMS_T

SQL>
SQL> desc LINEITEMS_T
LINEITEMS_T is NOT FINAL
      Name                                         Null?     Type
-----
SYS_XDBPD$LINEITEM                             XDB.XDB$RAW_LIST_T
LINEITEM_U                                       LINEITEM_U

SQL>
SQL> desc LINEITEM_U
LINEITEM_U VARRAY(2147483647) OF LINEITEM_T
LINEITEM_T is NOT FINAL
      Name                                         Null?     Type
-----
SYS_XDBPD$ITEMNUMBER                           XDB.XDB$RAW_LIST_T
NUMBER(38)
DESCRIPTION                                     VARCHAR2(256 CHAR)
PART_T                                           PART_T

SQL>

```

Figure XIV. Tables and Objects created as a result of XML Schema registration

KEY POINTS:

- The following objects were created as a result of registering the Schema.

A table called *PURCHASEORDER*.

The *PURCHASEORDER* table is an XMLType table. Each row in the table consists of a single XMLType object.

The table is constrained so that it can only contain documents which conform to the definition of the global element *PurchaseOrder* contained in the XML Schema that was registered with Oracle XML DB under the URL

<http://localhost:8080/home/SCOTT/posource/xsd/purchaseOrder.xsd>

The XMLType is persisted using structured or object-relational storage.

A SQLType called *PURCHASEORDER_T*.

This SQLType provides the definition of the underlying storage model for *PurchaseOrder* documents. The definition of *PURCHASEORDER_T* is derived from the complexType *PurchaseOrderType*. When a *PurchaseOrder* is inserted into the *PURCHASEORDER* table it is shredded and stored as an instance of *PURCHASEORDER_T*.

SQLTypes called *LINEITEMS_T*, *LINEITEMS_V* and *LINEITEM_T*.

During the schema registration process a SQLType is generated for each complexType defined by the XML Schema.

The *PurchaseOrder* XML Schema defines a complexType *LineItemsType* that can contain one or more *LineItem* elements. The definition of a *LineItem* element is provided by the complexType *LineItemType*. When an XML Schema defines that an element can occur more than once, Oracle XML DB uses a VARRAY to manage the members of the collection.

The SQLType *LINEITEM_T* is derived from the complexType *LineItemType*. Each *LineItem* element in the *PurchaseOrder* will be persisted as an instance of the SQLType *LINEITEM_T*.

The SQLType *LINEITEM_V* is used to manage the set of *LINEITEM_T* objects generated from a particular *PurchaseOrder* document. *LINEITEM_V* is defined as a VARRAY of *LINEITEM_T* objects.

The SQLType *LINEITEMS_T* is derived from the complexType *LineItemsType*. It contains a single attribute called *LINEITEM* of type *LINEITEMS_V*.

Since the Oracle XML DB Schema annotation *storeVarrayAsTable*="true" was specified in the root element of the PurchaseOrder XML Schema the default tables generated by schema registration use nested tables to persist the contents of a VARRAY. There will be one nested table for each collection defined by the XML Schema. Each member of the VARRAY will be stored as a separate row in the nested table. The nested tables created by the schema registration process are given system-generated names

Storing collections as nested tables makes it possible to create conventional b-tree indexes over the members of a collection making queries over the collection much more efficient. However in order to create an index on a particular collection it is necessary to know the name of the nested table that manages that collection.

The following screen shot shows the process of renaming the Nested Tables generated by the schema registration process in order to provide more meaningful names.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Tue Jan 6 16:57:16 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> call xdb_utilities.renameCollectionTable ('PURCHASEORDER','XMLDATA"."LINEITEMS"."LINEITEM','LINEITEM_TABLE')
2 /

Call completed.

SQL> call xdb_utilities.renameCollectionTable ('PURCHASEORDER','XMLDATA"."ACTIONS"."ACTION','ACTION_TABLE')
2 /

Call completed.

SQL> select PARENT_TABLE_NAME, PARENT_TABLE_COLUMN, TABLE_NAME
2 from USER_NESTED_TABLES
3 /

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

SQL>

```

Figure XV. Renaming Nested Tables

STORING XML IN ORACLE XML DB

There are several ways of storing XML in an XMLType.

- From SQL or PL/SQL, a simple insert statement can be used to load data. The data can come from a variety of sources. Before the data can be stored as an XMLType it must first be converted from the source form into an XMLType instance using one of the XMLType constructors.
- There are a number of variants of the XMLType constructor that allow an XMLType to be created from a number of different sources including VARCHAR and CLOB data types. The constructors also provide options for reducing the amount of processing associated with creating the XMLType. For instance, if the source XML Document is known to be both Well formed and Valid, the constructor allows flags to be passed that disable the default checking that is typically performed when instantiating the XMLType.
- It is also possible to use the Oracle XML DB repository to store XML in an XMLType table. The XML in question must be schema-based, and it will be stored in the Default Table associated with the XML Schema. For the SQL programmer the PL/SQL package DBMS_XDB provides methods that can be used to load an XML Document into the Oracle XML DB repository. XML documents can also be loaded into the repository using the FTP, HTTP and WebDAV protocols.
- When a schema-based XML document is loaded into the Oracle XML DB repository Oracle XML DB will automatically recognize the document, shred it and store it in the Default Table defined by the XML Schema.

The following example shows the `noNamespaceSchemaLocation` attribute being used to identify that a PurchaseOrder XML document is associated with the PurchaseOrder XML Schema

```
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
  <Reference>EABEL-20030409123336251PDT</Reference>
  <Actions>
    <Action>
      <User>EZLOTKEY</User>
    </Action>
  </Actions>
  <Reject/>
  <Requestor>Ellen S. Abel</Requestor>
  <User>EABEL</User>
  <CostCenter>R20</CostCenter>
  <ShippingInstructions>
    <name>Ellen S. Abel</name>
    <address>300 Oracle Parkway
Redwood Shores
CA
94065
USA</address>
    <telephone>650 506 7300</telephone>
  </ShippingInstructions>
  <SpecialInstructions>Counter to Counter</SpecialInstructions>
  <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>
</PurchaseOrder>
```

Figure XVI. A XML Document with `noNamespaceSchemaLocation` tag

The next two screen shots show Windows Explorer™ being used to load content into an XMLType table. In this example the documents are loaded by copying an entire directory tree from the local hard drive into the Oracle XML DB repository. Each of the documents in the folder hierarchy includes the noNamespaceSchemaLocation attribute that allows Oracle XML DB to determine which table will be used to store the contents of the document.

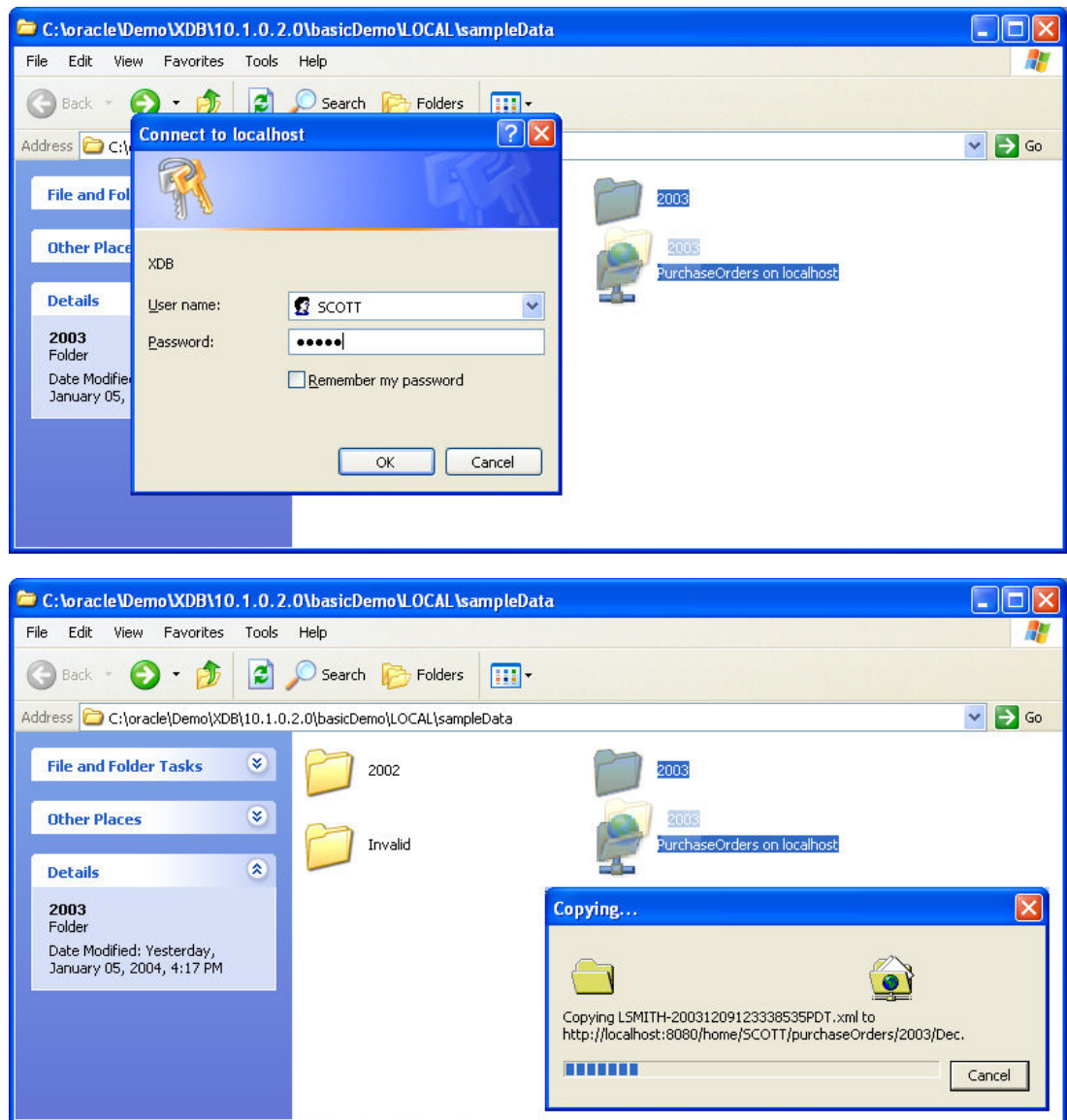


Figure XVII. File Copy Operation in progress

The sequence of events that takes place here is as follows:

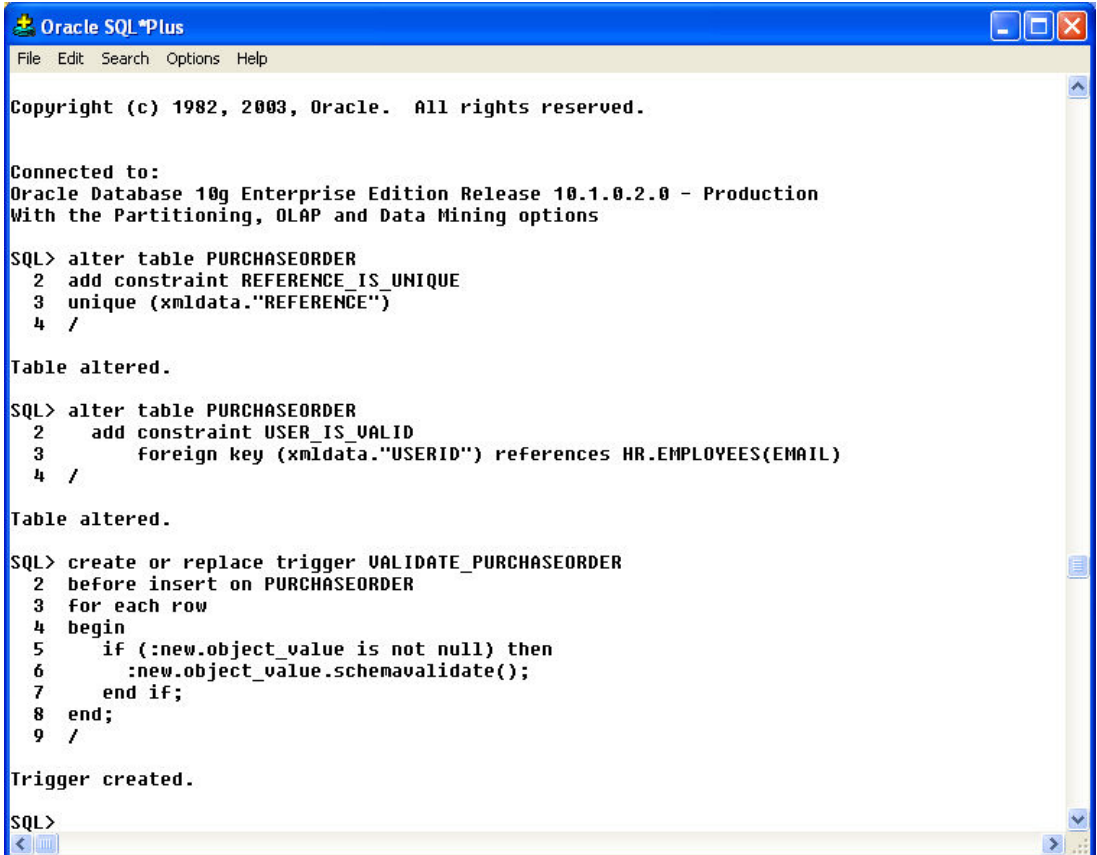
- The user selects a directory tree and drops it into the Oracle XML DB repository.
- The WebDAV client, in this case Windows explorer, generates a series of MKDIR and PUT commands that will copy the contents of the selected directory from the local hard drive into the Oracle XML DB repository.
- The Client attempts to create the first directory or upload the first file.
- The Oracle XML DB repository requests authentication from the client. Since the client has does not have access the required token, the user is prompted for the appropriate username and password.
- The user and password are authenticated using Database LDAP based authentication.
- Oracle XML DB recognizes that the files being transferred are XML files. It then looks at the root element of each document to see if it is associated with a known (registered) XML Schema.
- Since the file is based on a known XML Schema the meta data for the XML Schema is loaded from the XML Schema cache
- The document is parsed and decomposed into a set the SQL Objects that were derived from the XML Schema.
- The SQL Objects created from the XML file are stored in the default table that was generated when the XML Schema was registered with the database.
- A resource document is created for each document stored in the default table. This allows the content of the document to be accessed using the Oracle XML DB repository.
- Each file operation is considered to be an atomic operation.

ADDING DATABASE INTEGRITY TO XML DATA

XML Schema is a very powerful language. However there are some simple data management concepts that are not addressed in the current version of the XML Schema standard. These include the ability to define that the value of an element or attribute has to be unique across of a set of XML documents (a *UNIQUE* constraint), or that the value of an element or attribute must exist in some data source outside of the current document (a *FOREIGN KEY* constraint).

Oracle XML DB allows database-enforced integrity to be applied to XML content. The mechanisms used to enforce integrity on XML are the same as the mechanisms that are used to enforce integrity on conventional relational data. Simple rules, like uniqueness and foreign-key relationships, are enforced by specifying constraints. More complex rules are enforced by specifying database triggers.

The following screen shot shows how constraints and triggers can be used to enforce a set of business rules associated with the PurchaseOrder documents.



```
Oracle SQL*Plus
File Edit Search Options Help

Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> alter table PURCHASEORDER
  2  add constraint REFERENCE_IS_UNIQUE
  3  unique (xmldata."REFERENCE")
  4  /

Table altered.

SQL> alter table PURCHASEORDER
  2  add constraint USER_IS_VALID
  3  foreign key (xmldata."USERID") references HR.EMPLOYEES(EMAIL)
  4  /

Table altered.

SQL> create or replace trigger VALIDATE_PURCHASEORDER
  2  before insert on PURCHASEORDER
  3  for each row
  4  begin
  5      if (:new.object_value is not null) then
  6          :new.object_value.schemavalidate();
  7      end if;
  8  end;
  9  /

Trigger created.

SQL>
```

Figure XVIII. Applying Database Integrity Constraints and Triggers to an XMLType table.

KEY POINTS:

- The unique constraint *REFERENCE_IS_UNIQUE* will enforce the rule that the value of the node */PurchaseOrder/Reference/text()* is unique across all documents stored in the *PURCHASEORDER* table
- The foreign key constraint *USER_IS_VALID* will enforce the rule that the value of the node */PurchaseOrder/User/text()* corresponds to one of the values in the *EMAIL* column in the *EMPLOYEES* table in the *HR* schema.
- The trigger *VALIDATE_PURCHASEORDER* will enforce the rule that the PurchaseOrder document must be in full compliance with the XML Schema associated with the XML document. By default performs a simple 'lax' validation of the incoming XML Document which ensures that mandatory information is present and that there are no unexpected elements or attributes in the document. In order to enforce a full schema validation the **schemaValidate()** method has to be called on the XMLType.

Performing a full XML Schema validation is a fairly expensive process. This design allows the developer to determine if a full validation is required. If the developer is sure that the XML is always valid, they can avoid overhead of performing a full validation on the incoming XML document. If the developer is not sure about the validity of the XML, a simple trigger is all that is required to enforce a full validation.

- In the current release of Oracle XML DB it is necessary to define a constraint in terms of attributes of the associated SQL type. The SQL keyword **object_value** is used to refer to the content of row in an XMLType table from within a trigger.

The following screen shots show what happens when an attempt is made to load an XML document that does not conform to the constraints that have been defined for the *PURCHASEORDER* table. In this example case, the XML document is a valid XML document according to the XML Schema. However, the value of the node */PurchaseOrder/Reference/text()* is a duplicate of one of the documents that has already been loaded into Oracle XML DB.

```

1.3.1 Duplicate Reference
ftp> open localhost 2100
Connected to mdrake-lap.
220 mdrake-lap FTP Server (Oracle XML DB/Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 -
Production) ready.
ftp> user SCOTT TIGER
331 pass required for SCOTT
230 SCOTT logged in
ftp> cd /home/SCOTT/purchaseOrders
250 CWD Command successful
ftp> !type DuplicateReference.xml
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
  <Reference>SBELL-2003030912333601PDT</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>

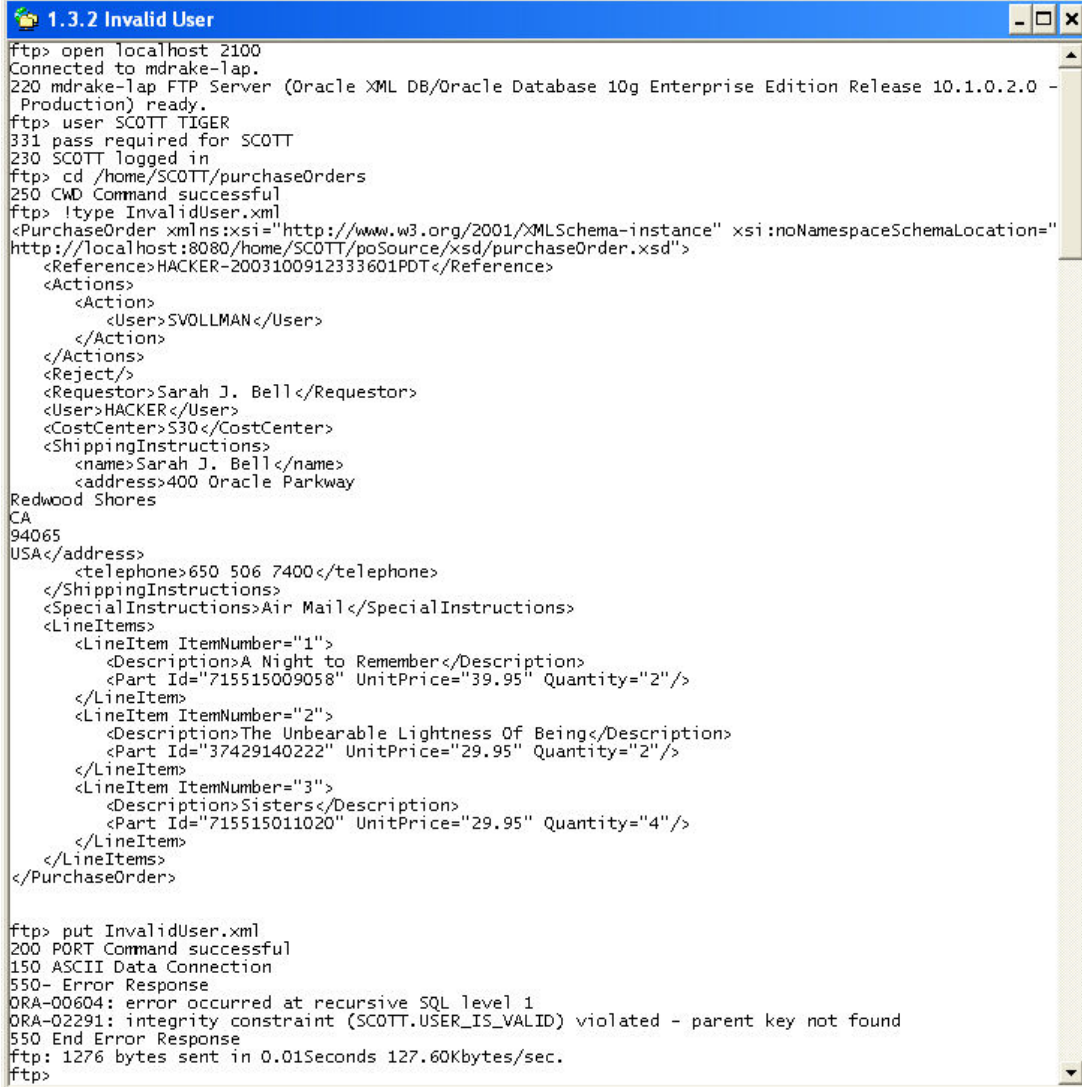
ftp> put DuplicateReference.xml
200 PORT Command successful
150 ASCII Data Connection
550- Error Response
ORA-00604: error occurred at recursive SQL level 1
ORA-00001: unique constraint (SCOTT.REFERENCE_IS_UNIQUE) violated
550 End Error Response
ftp> 1274 bytes sent in 0.00Seconds 1274000.00Kbytes/sec.
ftp>
  
```

Figure XIX. Violating a Unique Constraint via FTP

KEY POINTS:

- Since there already is a document in the database that contains a reference element with the same value, the constraint *REFERENCE_IS_UNIQUE* is violated. Since the unique key constraint is violated the ftp 'put' operation fails.

In the second case, the XML document is a valid XML document according to the XML Schema. However the node `/PurchaseOrder/User/text()` contains the value 'HACKER'.



```

1.3.2 Invalid User
ftp> open localhost 2100
Connected to mdrake-lap.
220 mdrake-lap FTP Server (Oracle XML DB/Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 -
Production) ready.
ftp> user SCOTT TIGER
331 pass required for SCOTT
230 SCOTT logged in
ftp> cd /home/SCOTT/purchaseOrders
250 CWD Command successful
ftp> !type InvalidUser.xml
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
  <Reference>HACKER-2003100912333601PDT</Reference>
  <Actions>
    <Action>
      <User>SVOLLMAN</User>
    </Action>
  </Actions>
  <Reject/>
  <Requestor>Sarah J. Bell</Requestor>
  <User>HACKER</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>

ftp> put InvalidUser.xml
200 PORT Command successful
150 ASCII Data Connection
550- Error Response
ORA-00604: error occurred at recursive SQL level 1
ORA-02291: integrity constraint (SCOTT.USER_IS_VALID) violated - parent key not found
550 End Error Response
ftp> 1276 bytes sent in 0.015seconds 127.60Kbytes/sec.
ftp>

```

Figure XX. Violating a Foreign Key Constraint via FTP

KEY POINTS:

- Since the value "HACKER" does not appear in the `EMAIL` column of the `EMPLOYEES` table, the constraint `USER_IS_VALID` is violated. Since the foreign key constraint is violated the ftp 'put' operation fails.

In the final case, the XML document is not a valid XML document according to the XML Schema. The XML Schema defines a minimum length of 18 characters for the node `/PurchaseOrder/Reference/text()`. In this document the node contains the value “`SBELL-20031009`”, which is only 15 characters long.



```

1.3.3 Invalid Document
ftp> open localhost 2100
Connected to mdrake-lap.
220 mdrake-lap FTP Server (Oracle XML DB/Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 -
Production) ready.
ftp> user SCOTT TIGER
331 pass required for SCOTT
230 SCOTT logged in
ftp> cd /home/SCOTT/purchaseOrders
250 CWD Command successful
ftp> !type InvalidReference.xml
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
  <Reference>SBELL-20031009</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>

ftp> put InvalidReference.xml
200 PORT Command successful
150 ASCII Data Connection
550- Error Response
ORA-00604: error occurred at recursive SQL level 1
ORA-31154: invalid XML document
ORA-19202: Error occurred in XML processing
LSX-00221: "SBELL-20031009" is too short (minimum length is 18)
ORA-06512: at "SYS.XMLTYPE", line 333
ORA-06512: at "SCOTT.VALIDATE_PURCHASEORDER", line 3
ORA-04088: error during execution of trigger 'SCOTT.VALIDATE_PURCHASEORDER'
550 End Error Response
ftp> 1263 bytes sent in 0.00Seconds 1263000.00Kbytes/sec.
ftp>

```

Figure XXI. Violating a `SchemaValidate` constraint via FTP

KEY POINTS:

- The full XML Schema validation performed by the *VALIDATE_PURCHASEORDER* trigger catches the fact that the document is not valid according to the XML Schema. Since the trigger returns an error, the ftp 'put' operation fails and the document is not uploaded. Without the trigger the document would pass the default 'lax' validation performed when a document is loaded into an XMLType table.
- When a document is being uploaded via a protocol, Oracle XML DB always provides the client with the full SQL Error Stack. How the error is interpreted and reported to the user is determined by the error handling built into the client application. Some clients will report the error returned by Oracle XML DB, while others will simply report a generic error message.

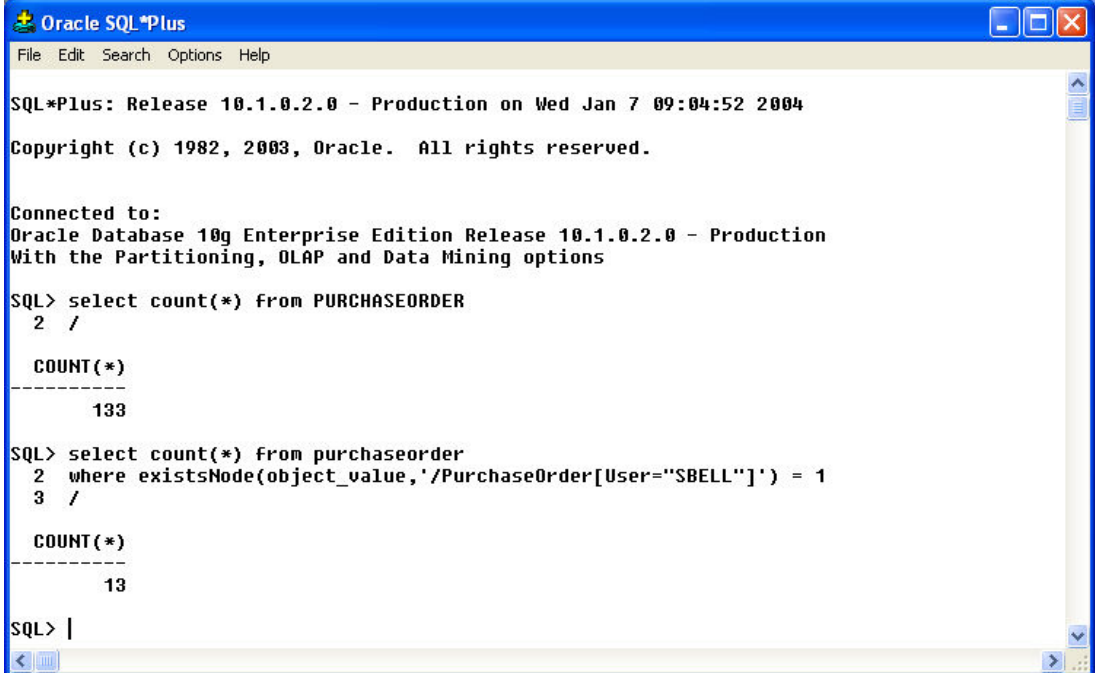
The above examples demonstrate two key features of Oracle XML DB. First, Oracle XML DB makes it possible to implement database enforced business rules on XML Content, in addition to those rules that can be specified using the constructs of XML Schema. Second, the database will enforce these business rules regardless of whether XML is inserted directly into a table, or uploaded using a protocol.

QUERYING AND INDEXING XML WITH ORACLE XML DB

QUERYING XML

Oracle XML DB defines a set of operations that make it possible to query XML content in a very efficient manner. These operations are implemented as a set of methods on the XMLType data-type and as a set of functions that are defined in the SQL/XML standard.

The following screen shot shows some simple, XPath-based queries:



```
Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 09:04:52 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select count(*) from PURCHASEORDER
2 /

COUNT(*)
-----
133

SQL> select count(*) from purchaseorder
2 where existsNode(object_value,'/PurchaseOrder[User="SBELL"]') = 1
3 /

COUNT(*)
-----
13

SQL> |
```

Figure XXII. Simple SQL Queries against XML content

KEY POINTS:

- The first query finds the number of PurchaseOrder documents stored into the Oracle XML DB repository. Since each PurchaseOrder document is stored as a row in the default table defined by the PurchaseOrder XML Schema, the number of documents can be found by counting the number of rows in the *PURCHASEORDER* table.
- The second query uses a simple XPath expression and the **existsnode()** operator to find the number of PurchaseOrder documents where the value of the node */PurchaseOrder/User/text()* contains the value “SBELL”.

The following screen shot shows a query that returns an entire document based on the value of the node `/PurchaseOrder/Reference/text()`.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 09:10:55 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select object_value from purchaseorder
      2 where existsNode(object_value,'/PurchaseOrder[Reference="SBELL-2003030912333601PDT"]') = 1
      3 /

OBJECT_VALUE
-----
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
"http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
  <Reference>SBELL-2003030912333601PDT</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>

SQL>

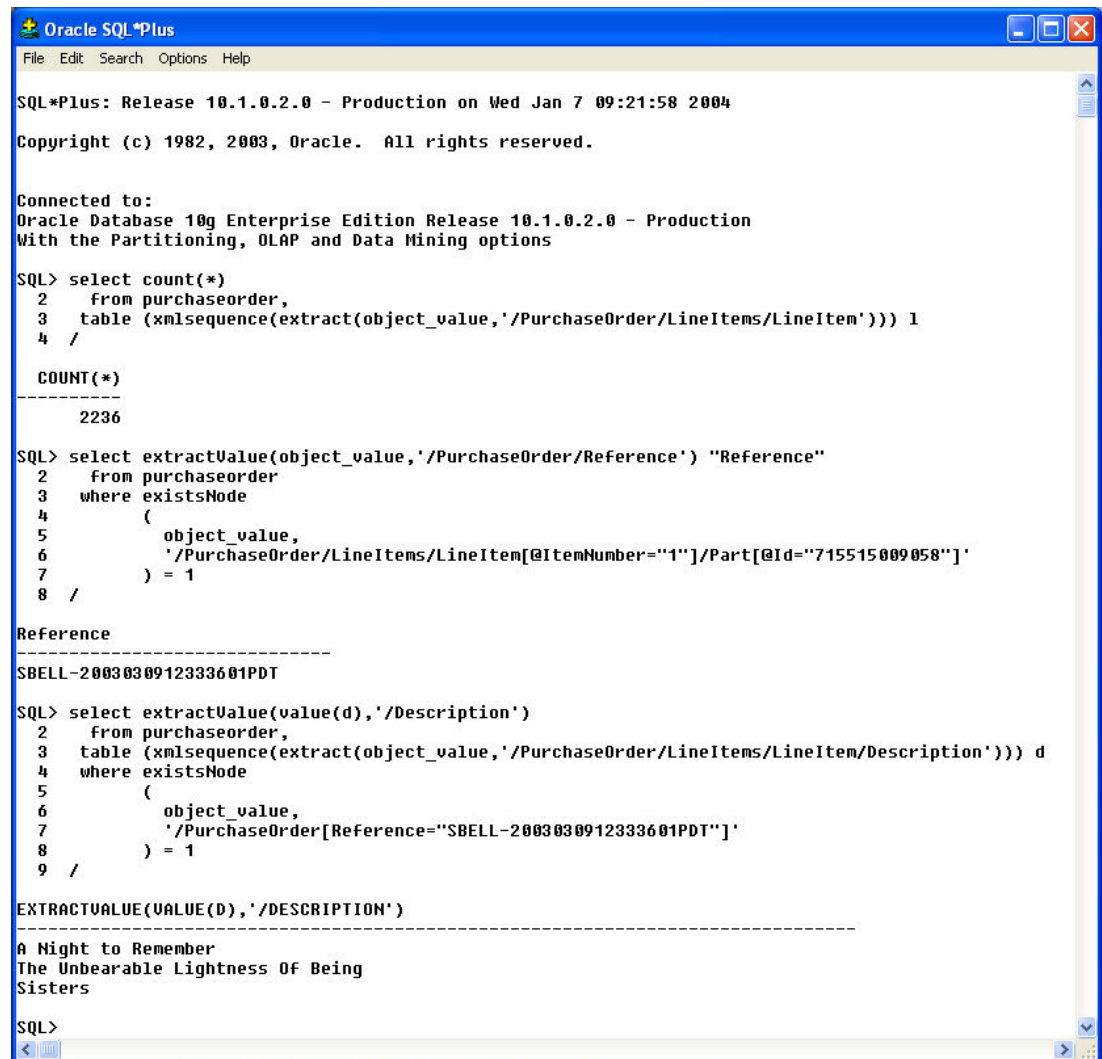
```

Figure XXIII. retrieving the entire XML content

KEY POINTS:

- The **object_value** operator returns the entire contents of the XML Document. It is used when accessing the content of a row in an XMLType table.

The previous queries operated on the values of the text nodes associated with elements that are direct decedents of the root node of the XML document. The following query shows how Oracle XML DB can perform queries against collections, and nodes that are inside collections.



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 09:21:58 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select count(*)
2   from purchaseorder,
3   table (xmlsequence(extract(object_value, '/PurchaseOrder/LineItems/LineItem'))) 1
4   /

COUNT(*)
-----
2236

SQL> select extractValue(object_value, '/PurchaseOrder/Reference') "Reference"
2   from purchaseorder
3   where existsNode
4   (
5       object_value,
6       '/PurchaseOrder/LineItems/LineItem[@ItemNumber="1"]/Part[@Id="715515009058"]'
7   ) = 1
8   /

Reference
-----
SBELL-2003030912333601PDT

SQL> select extractValue(value(d), '/Description')
2   from purchaseorder,
3   table (xmlsequence(extract(object_value, '/PurchaseOrder/LineItems/LineItem/Description'))) d
4   where existsNode
5   (
6       object_value,
7       '/PurchaseOrder[Reference="SBELL-2003030912333601PDT"]'
8   ) = 1
9   /

EXTRACTVALUE(VALUE(D), '/DESCRIPTION')
-----
A Night to Remember
The Unbearable Lightness Of Being
Sisters

SQL>

```

Figure XXIV. querying the contents of Collections

KEY POINTS:

- The first query counts the number of *LineItem* elements in all of the *PurchaseOrder* documents.

The first step is to use the **extract()** operator to get the set of *LineItem* elements from each *PurchaseOrder* document. The result of using the **extract()** operator on a *PurchaseOrder* document will be an XML Fragment containing the set of *LineItem* elements contained in that document

The next step is to use the **xmlsequence()** operator to generate a VARRAY of XMLType objects from the XMLType objects containing the fragments returned by the **extract()** operator. Each XMLType will contain a single LineItem element. There will be one member of the VARRAY for each LineItem element contained in the fragment.

The next step is to use the SQL **table()** operator to create a virtual table from the set of XMLType object contained in the VARRAYs generated by the **xmlsequence()** operator. This means that the virtual table will contain one row for each LineItem element in the set of XML documents processed by the **extract()** operator.

The last step is to obtain the required result by counting the number of rows in the virtual table.

- There is a correlated join between the set of documents in the *PURCHASEORDER* table and the result set generated by the **extract()** operator. This means that the set of LineItem elements from each document will only be processed once by the **xmlsequence()** operator.
- The second query demonstrates that Oracle XML DB can evaluate complex XPath expressions that reference nodes inside a collection. In this example the **extractvalue()** operator is used to find the value of the node */PurchaseOrder/Reference/text()* in documents that contain a */PurchaseOrder/LineItems/LineItem* element where the value of *ItemNumber* attribute is '1' and the value of the *Id* attribute for the associated *Part* element is '75515009058'.
- The third query demonstrates that Oracle XML DB allows normal SQL operations to be performed on the contents of a collection. In this case a virtual table is created from the elements identified by the XPath expression */PurchaseOrder/LineItems/LineItem/Description*. Operations can then be performed directly on each element, as if they were rows in an XMLType table.

QUERY PLAN ANALYSIS

Oracle XML DB delivers high performance execution of queries over XML content. This is possible only if the query plans chosen are able to make use of appropriate indexes. This means that it is extremely important for application developers to analyze their queries and ensure that the query plan chosen is an optimal one. Fortunately, XPath-Rewrite makes it possible to generate conventional query plans for queries that use XPath expressions to query XML content.

The following screen shot shows the output generated by the Explain Plan statement for the query “select PurchaseOrder documents where the value of the node /PurchaseOrder/Reference/text() is ‘SBELL-2003030912333601PDT’”.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 09:55:29 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> --
SQL> explain plan for
  2  select object_value
  3    from purchaseorder
  4   where existsNode(object_value, '/PurchaseOrder[Reference="SBELL-2003030912333601PDT"]') = 1
  5  /

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
-----
Plan hash value: 877147341

-----
| Id | Operation                      | Name                      | Rows | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT                |                           |      1 | 22227 |      1 (0)| 00:00:01 |
|*  1 |  TABLE ACCESS BY INDEX ROWID   | PURCHASEORDER            |      1 | 22227 |      1 (0)| 00:00:01 |
|*  2 |    INDEX UNIQUE SCAN            | REFERENCE_IS_UNIQUE      |      1 |          |      0 (0)| 00:00:01 |
-----

Predicate Information (identified by operation id):
PLAN_TABLE_OUTPUT
-----
 1 - filter(SYS_CHECKACL("ACL0ID","OWNERID",xmltype('<privilege
      xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
      stance" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
      http://xmlns.oracle.com/xdb/acl.xsd DAU:http://xmlns.oracle.com/xdb/dau.xsd"><read-properti
      es/><read-contents/></privilege>'))=1)
 2 - access("PURCHASEORDER"."SYS_NC00009$"='SBELL-2003030912333601PDT')

19 rows selected.

SQL> |

```

Figure XXV. Simple Explain plan for existsnode() query

KEY POINTS:

- The query will be resolved using the unique index *REFERENCE_IS_UNIQUE*. The index was created as result of adding the unique constraint on */PurchaseOrder/Reference/text()* to the *PURCHASEORDER* table. Using this index will ensure the fastest possible response time for this kind of query.
- Query-Re-write converts the XPath expression into a SQL expression. This allows the database optimizer to determine that the index used to enforce uniqueness can also be used to resolve the XPath expression.
- An implicit filter is added to the query. This filter uses the **sys_checkacl()** operator to enforce ACL based security using Row Level Security Policies that are generated from the ACL.

The next screen shot shows the results generated by the Explain Plan statement for the query “count the number of PurchaseOrder documents where the value of the node /PurchaseOrder/User/text() is ‘SBELL’”.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 09:55:36 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> explain plan for
  2   select count(*) from purchaseorder
  3   where existsNode(object_value,'/PurchaseOrder[User="SBELL"]') = 1
  4   /

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
-----
Plan hash value: 426205490

-----
| Id | Operation          | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT   |               |      1 | 22227 |      4  (0)| 00:00:01 |
|  1 |   SORT AGGREGATE   |               |      1 | 22227 |      4  (0)| 00:00:01 |
|*  2 |    TABLE ACCESS FULL| PURCHASEORDER |      1 | 22227 |      4  (0)| 00:00:01 |
-----

Predicate Information (identified by operation id):
-----
  2 - filter("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 DAU:http://xmlns.oracle.com/xdb/dau.xsd"
        ><read-properties><read-contents></privilege>''))=1)

20 rows selected.

SQL> |

```

Figure XXVI. Simple Explain plan for existsnode() query

KEY POINTS:

- The query plan shows that the query will be resolved by performing a full table scan of the *PURCHASEORDER* table
- This plan may be acceptable when there are only a few hundred documents in the table, but would be unacceptable if there 1000's or 1,000,000's of documents in the table.

The final screen shot shows the results generated by the Explain Plan statement for the query “find the value of the node `/PurchaseOrder/Reference/text()` in documents that contain a `/PurchaseOrder/LineItems/LineItem` element where the value of `ItemNumber` attribute is ‘1’ and the value of the `Id` attribute for the associated `Part` element is ‘75515009058’”.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 09:55:41 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> explain plan for
2  select extractValue(object_value,'/PurchaseOrder/Reference')
3  from purchaseorder
4  where existsNode
5  (
6      object_value,
7      '/PurchaseOrder/LineItems/LineItem[@ItemNumber="1"]/Part[@Id="75515009058"]'
8  ) = 1
9  /

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
-----
Plan hash value: 1182755974

-----
| Id | Operation                      | Name                      | Rows | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT                |                           |      1 | 26858 |    819 (1)| 00:00:10 |
|  1 |   NESTED LOOPS                  |                           |      1 | 26858 |    819 (1)| 00:00:10 |
|  2 |    SORT UNIQUE                  |                           |      1 | 4621 |     817 (0)| 00:00:10 |
|* 3 |     INDEX FAST FULL SCAN        | LINEITEM_TABLE_DATA      |      1 | 4621 |     817 (0)| 00:00:10 |
|* 4 |      TABLE ACCESS BY INDEX ROWID | PURCHASEORDER            |      1 | 22237 |        1 (0)| 00:00:01 |
|* 5 |       INDEX UNIQUE SCAN         | LINEITEM_TABLE_MEMBERS   |      1 |      |         0 (0)| 00:00:01 |
-----

Predicate Information (identified by operation id):
-----
3 - filter("ITEMNUMBER"=1 AND "SYS_NC00011$"='75515009058')
4 - filter(SYS_CHECKACL("ACLOID","OWNERID",xmltype('<privilege
      xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
e" xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xsd
DAU:http://xmlns.oracle.com/xdb/dau.xsd"><read-properties/><read-contents/></privilege>')))=1)
5 - access("NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC0003400035$")

Note
-----
- dynamic sampling used for this statement

26 rows selected.

SQL>

```

Figure XXVII. Simple Explain plan for existsnode() operation on collection

KEY POINTS:

- In this example the LineItem elements are stored in an Index Organized Table (IOT). There is an automatic foreign key relationship between the IOT containing the set of LineItem elements and the *PURCHASEORDER* Table.
- The query plan shows that the query will be resolved by performing a full table scan of the IOT containing the LineItem elements. It will then use the foreign key value to join back to the *PURCHASEORDER* table to get the value of the node */PurchaseOrder/Reference/text()*.
- This plan may be acceptable when there are only a few hundred documents in the table, but would be unacceptable if there 1000's or 1,000,000's of documents in the table.

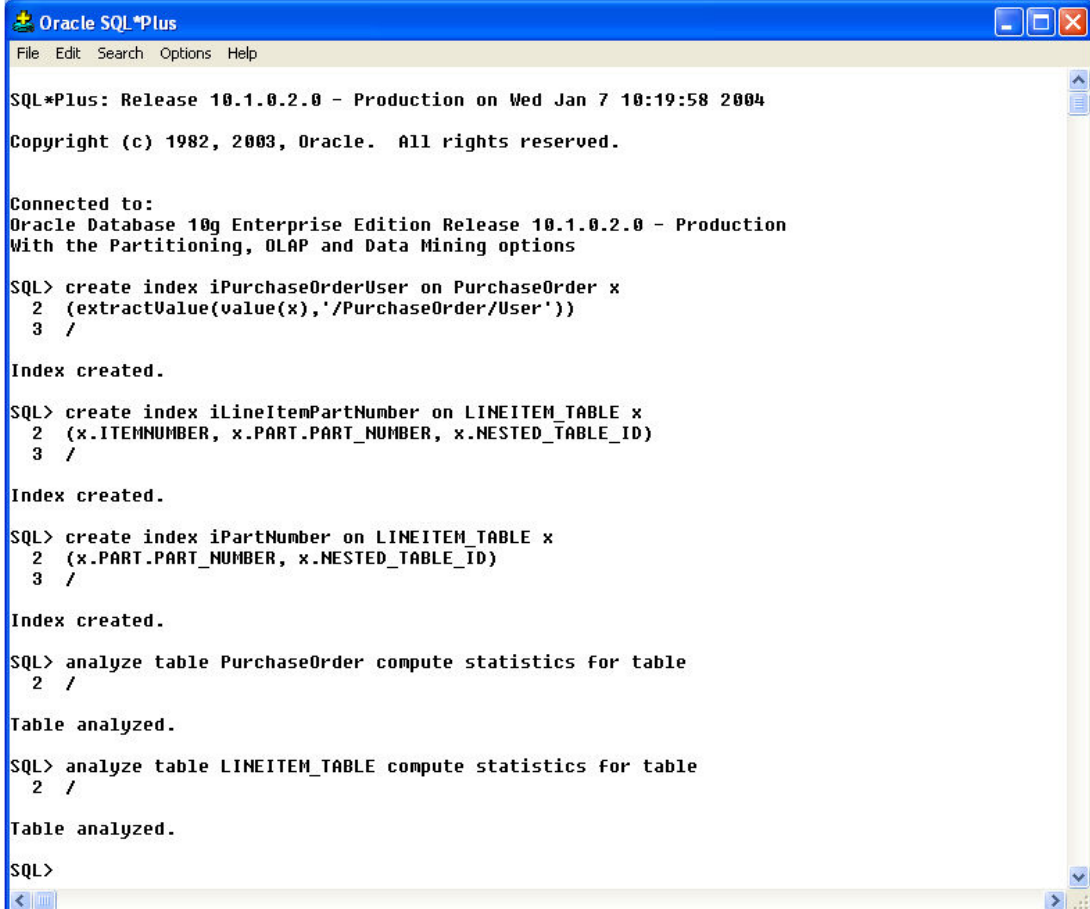
INDEXING XML CONTENT

In a conventional relational database the standard way to ensure query response time is to create indexes that allow the query to be resolved in an efficient manner. Oracle XML DB allows the same techniques to be used to ensure optimal response times for queries over XML content.

Three kinds of indexes can be created on XML Content. Conventional B-Tree indexes can be created on XML content that is XML Schema based, and which is stored using structured storage techniques. Functional and Text based indexes can be created on any kind of XML document, regardless of whether or not the content is stored using structured storage.

Oracle XML DB allows XPath expressions to be used when creating indexes on XML content. During the index creation process Oracle XML DB uses XPath-Rewrite to determine whether or not the XPath expressions included in the Create Index statement can be re-written into equivalent Object Relational SQL expressions. If the XPath expression can be restated using Object Relational SQL, then the index is created as a conventional B-Tree index on the underlying SQL objects. If the XPath expression cannot be restated using Object Relational SQL then a functional index is created.

The following screen shot shows the creation of two B-Tree indexes on the PurchaseOrder documents



```
Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 10:19:58 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> create index iPurchaseOrderUser on PurchaseOrder x
2 (extractValue(value(x), '/PurchaseOrder/User'))
3 /

Index created.

SQL> create index iLineItemPartNumber on LINEITEM_TABLE x
2 (x.ITEMNUMBER, x.PART.PART_NUMBER, x.NESTED_TABLE_ID)
3 /

Index created.

SQL> create index iPartNumber on LINEITEM_TABLE x
2 (x.PART.PART_NUMBER, x.NESTED_TABLE_ID)
3 /

Index created.

SQL> analyze table PurchaseOrder compute statistics for table
2 /

Table analyzed.

SQL> analyze table LINEITEM_TABLE compute statistics for table
2 /

Table analyzed.

SQL>
```

Figure XXVIII. Indexing XML Content

KEY POINTS:

- The first index created is on the node */PurchaseOrder/User/text()*. The XPath-Rewrite process will use the XML Schema to determine the SQL Object and Attribute that corresponds to this node. The create index will result in a B-Tree index being created on the *PURCHASEORDER* table.
- The second index is a compound index on */PurchaseOrder/LineItems/LineItem/@ItemNumber* and */PurchaseOrder/LineItems/LineItem/Part/@Id*. In the PurchaseOrder XML Schema the *LineItem* element is allowed to occur multiple times within a single document. Each *LineItem* element is stored as a separate row in a Nested Table called *LINEITEM_TABLE*. Since a Nested Table is being used to manage the node being indexed the *Create Index* statement has to be specified in terms of an attribute of the Nested Table.

In a relational database application developers are not required to change their application because the indexes on the data change. This is also true for Oracle XML DB.

The following screen shot shows how indexing the node `/PurchaseOrder/User/text()` alters the query plan for the query “count the number of PurchaseOrder documents where the value of the node `/PurchaseOrder/User/text()` is ‘SBELL’”.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 10:03:48 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> explain plan for
2 select count(*) from purchaseorder
3   where existsNode(object_value,'/PurchaseOrder[User="SBELL"]') = 1
4 /

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
-----
Plan hash value: 3299943326

-----
| Id | Operation                      | Name                | Rows | Bytes | Cost (%CPU)| Time     |
-----
|  0 | SELECT STATEMENT                |                     |      1 | 526 |      3 (0)| 00:00:01 |
|  1 |   SORT AGGREGATE                |                     |      1 | 526 |      1 (0)| 00:00:01 |
|* 2 |    TABLE ACCESS BY INDEX ROWID | PURCHASEORDER       |      1 | 526 |      3 (0)| 00:00:01 |
|* 3 |      INDEX RANGE SCAN           | IPURCHASEORDERUSER  |      1 |      |      1 (0)| 00:00:01 |
-----

Predicate Information (identified by operation id):
-----
2 - filter(SYS_CHECKACL('ACLOID','OWNERID',xmltype('<privilege
      xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
      tance" 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-properti
      es/><read-contents/></privilege>')))=1)
3 - access("PURCHASEORDER"."SYS_NC00022$"='SBELL')

20 rows selected.

SQL> |

```

Figure XXIX. simple XPath Query Plan showing correct use of XPath Index

KEY POINTS:

- The query plan shows that the query will be resolved using the `IPURCHASEORDERUSER` index.

The following screen shot shows how creating a compound index on the attributes *ItemNumber* and *Part.PARTNO* of the *LINEITEM_TABLE* alters the query plan for a query that references those nodes. This example shows the query plan generated for the query “find the value of the node */PurchaseOrder/Reference/text()* in documents that contain a */PurchaseOrder/LineItems/LineItem* element where the value of *ItemNumber* attribute is ‘1’ and the value of the *Id* attribute for the associated *Part* element is ‘75515009058’”.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 10:04:31 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> explain plan for
2  select extractValue(object_value, '/PurchaseOrder/Reference')
3  from purchaseorder
4  where existsNode
5  (
6      object_value,
7      '/PurchaseOrder/LineItems/LineItem[@ItemNumber="1"]/Part[@Id="75515009058"]'
8      ) = 1
9  /

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
-----
Plan hash value: 728299625

-----
| Id | Operation                                | Name                                | Rows | Bytes | Cost (%CPU)| Time |
-----
|  0 | SELECT STATEMENT                        |                                     |      1 |    646 |    4 (25)| 00:00:01 |
|  1 |   NESTED LOOPS                          |                                     |      1 |    646 |    4 (25)| 00:00:01 |
|  2 |    SORT UNIQUE                          |                                     |      1 |    120 |    2 (0)| 00:00:01 |
|* 3 |     INDEX UNIQUE SCAN                    | LINEITEM_TABLE_DATA                |      1 |    120 |    2 (0)| 00:00:01 |
|* 4 |     INDEX RANGE SCAN                    | ILINEITEMPARTNUMBER                |      1 |        |    2 (0)| 00:00:01 |
|* 5 |     TABLE ACCESS BY INDEX ROWID        | PURCHASEORDER                      |      1 |    526 |    1 (0)| 00:00:01 |
|* 6 |     INDEX UNIQUE SCAN                    | LINEITEM_TABLE_MEMBERS              |      1 |        |    0 (0)| 00:00:01 |
-----

Predicate Information (identified by operation id):
-----
 3 - access("ITEMNUMBER"=1 AND "SYS_NC00011$"='75515009058')
 4 - access("ITEMNUMBER"=1 AND "SYS_NC00011$"='75515009058')
 5 - filter(SYS_CHECKACL("ACLOID", "OWNERID", xmltype('<privilege
      xmlns="http://xmlns.oracle.com/xdm/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
      e" xsi:schemaLocation="http://xmlns.oracle.com/xdm/acl.xsd http://xmlns.oracle.com/xdm/acl.xsd
      DAU:http://xmlns.oracle.com/xdm/dau.xsd"><read-properties/><read-contents/></privilege>'))=1)
 6 - access("NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC0003400035$")

24 rows selected.

SQL>

```

Figure XXX. query plan for complex XPath based query, showing correct use of Indexes

KEY POINTS:

- The query will be resolved using the compound index *ILINEITEMPARTNUMBER*.
- In both cases the syntax used to define the query has not changed. XPath-Rewrite has allowed the optimizer to analyze the query and determine that the new indexes provide a more efficient way of resolving the queries.
- The XMLType and XPath abstractions make it possible for an application programmer to develop applications independently of the underlying storage technology. Just as with conventional relational applications, creating and dropping indexes makes it possible to tune the performance of an application without having to re-write it.

PATH-BASED ACCESS AND UPDATE OF XML CONTENT

Oracle XML DB provides developers with multiple ways of accessing XML Content. From SQL, a conventional Table / Row metaphor is used to access and update XML content. This approach will be very familiar to SQL programmers who have experience of working with relational database technology.

Content managed by Oracle XML DB can be accessed and updated using a path-based metaphor. This allows a URL to be used when accessing or updating content stored in Oracle XML DB. This approach will appeal to XML developers, who are used to using constructs like URLs and XPath expressions to identify content. Supporting path-based access allows client applications, such as Microsoft Word or XMLSpy to use the FTP, HTTP or WebDAV protocols to access and update content managed by Oracle XML DB.

The following screen shot shows how a standard Web browser, such as Internet Explorer™, can use a URL to access content stored in the Oracle XML DB repository.

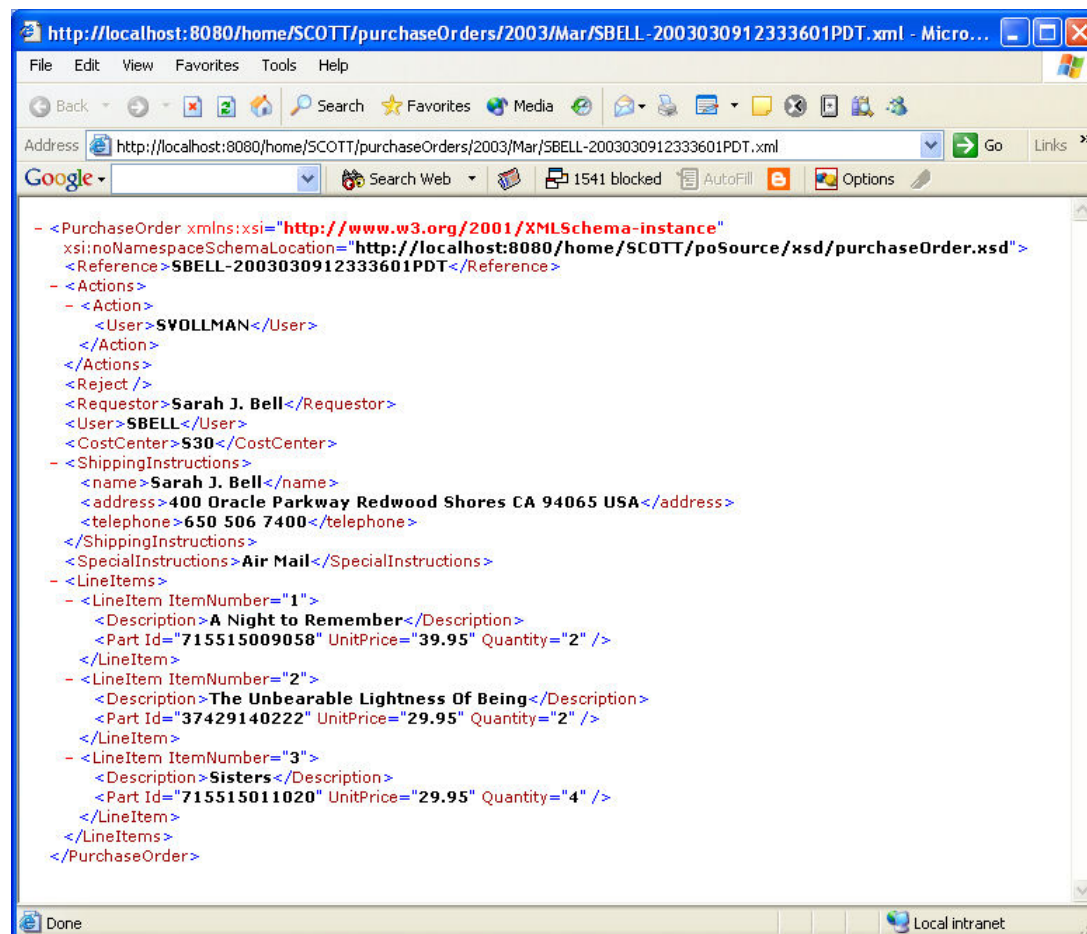


Figure XXXI. path-based access via HTTP and a URL

KEY POINTS:

- No additional software is required in order to use a browser to access content stored in Oracle XML DB. The combination of the HTTP protocol server and path-based access mean that, given the appropriate access permissions, a simple HTTP URL is all that is required to access content.

The path-based mechanism for accessing content is also available to the SQL programmer. The XDBUriType data-type makes it easy to use a path to access the content of a document stored on the Oracle XML DB repository. Oracle XML DB also includes new SQL Operators such as **equals_path()**, which make it possible to perform path-based queries against both the **PATH_VIEW** and the **RESOURCE_VIEW**. The DBUriType data-type and related *DBUri Servlet* allow a path to be used to identify any row in any table via a path.

The following screen shot shows the XDBUriType data-type being used to access content:



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 10:17:26 2004

Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select xdbUriType('/home/SCOTT/purchaseOrders/2003/Mar/SBELL-2003030912333601PDT.xml').getXML()
2 from dual
3 /

XDBURITYPE('/HOME/SCOTT/PURCHASEORDERS/2003/MAR/SBELL-2003030912333601PDT.XML').GETXML()
-----
<PurchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=
"http://localhost:8080/home/SCOTT/poSource/xsd/purchaseOrder.xsd">
  <Reference>SBELL-2003030912333601PDT</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>

SQL>

```

Figure XXXII. Using XDBUriType to access content

Applications can also leverage the path-based metaphor when updating content managed by Oracle XML DB. The following screen shot shows Microsoft Word 2003 using WebDAV to update a row in the *PURCHASEORDER* table by updating a document stored in the Oracle XML DB repository.

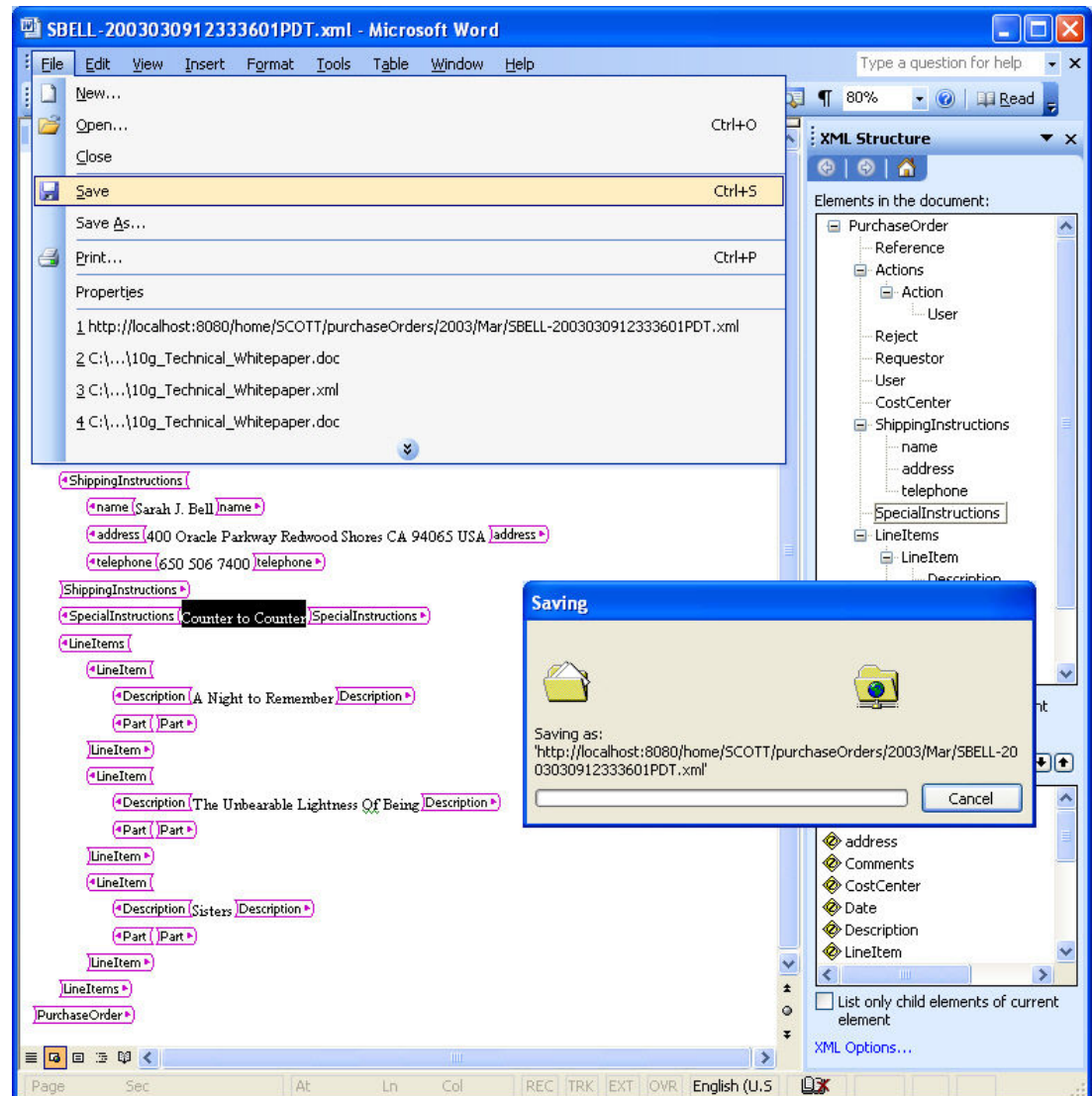


Figure XXXIII. updating content with Microsoft Word and WebDAV

KEY POINTS:

- The client sends a URL via the appropriate *POST* or *PUT* command and then submits the updated document to Oracle XML DB.
- When Oracle XML DB receives the request it identifies which database objects manage the content associated with the URL, and uses the revised document to update these objects accordingly.
- When working via a protocol, each *PUT* or *POST* request is treated as a separate atomic transaction. This means that changes made to a document via a protocol are visible to other users as soon as the request has been processed.

The **RESOURCE_VIEW** and **PATH_VIEW** allow the path-based metaphor to be used when updating content from SQL or PL/SQL. The **updatexml()** operator makes it possible to update the content of an XML document. Two techniques can be used to perform path-based updates of content managed by Oracle XML DB from SQL.

- The first is to use **updatexml()** to update or replace the content of a document by updating the resource associated with the document. The resource is obtained from the **RES** column in the **PATH_VIEW** or **RESOURCE_VIEW**. Once the resource has been located, the body or content of the document can be accessed or updated via the XPath expression */Resource/Contents*. This technique has the advantage that it can be used to access or update any kind of content stored in the XML DB repository.
- The second is to use **updatexml()** to directly update the default table which contains the content of the target document. This technique can only be used when updating an XML document that is based on a registered XML Schema. The rows to be updated are identified by joining the **PATH_VIEW** or **RESOURCE_VIEW** and the default table. The join is based on the value of the node */Resource/XMLRef*.

Since the **updatexml()** operator is now referencing the content of the default table, XPath-Rewrite can be used to map the **updatexml()** operation into a set of updates to the underlying SQL object model. This means that the update will execute many thousands of times faster.

The following screen shot shows the **updatexml()** operator being used to update the content of a row in the default table associated with the PurchaseOrder XML Schema.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 12:44:33 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> update PURCHASEORDER p
2 set object_value = updateXML(object_value, '/PurchaseOrder/User/text()', 'KCHUNG')
3 where ref(p) = (select extractValue(res, '/Resource/XMLRef')
4 from RESOURCE_VIEW
5 where equals_path
6 (
7 res,
8 '/home/' || USER || '/purchaseOrders/2003/Mar/SBELL-2003030912333601PDT.xml'
9 ) = 1)
10 /

1 row updated.

SQL> commit
2 /

Commit complete.

SQL>

```

Figure XXXIV. updating content using *updatexml()* and *EQUALS_PATH()*

KEY POINTS:

- The target row is located by using the **equals_path()** operator to search the **RESOURCE_VIEW** for the resource associated with the specified path.
- The resource contains a node */Resource/XMLRef* which contains a REF to the XMLType object in the default table that contains the content associated with the resource.
- The **updatexml()** operator is used to update the row.

When working from SQL normal transactional behavior is enforced. Multiple **updatexml()** statements can be used within a single logical unit of work. Changes made via **updatexml()** are not visible to other database users until the transaction is committed. At any point rollback can be used to back the set of changes made since the last commit.

When an editor like Microsoft Word updates an XML document stored in Oracle XML DB the database receives an input stream containing the new contents of the document. There is no indication of what has changed in the document. Consequently it is necessary to re-parse the entire document, and update all of the objects that were derived from the original document with the new content.

When **updatexml()** is used to update a schema-based XML document. Oracle XML DB is able to perform a partial-update. XPath-Rewrite is used to translate the **updatexml()** operation into an equivalent SQL statement. The update is performed by directly updating the attributes of underlying objects. This means that an **updatexml()** operation will often execute thousands of times faster than a document based update.

When **updatexml()** is used to update a non-schema-based XML document, Oracle XML DB performs the update by using DOM methods to update the specified nodes and writing the updated DOM back to the underlying CLOB. This means that the **updatexml()** operator is still more efficient than using a XML editor, even with non-schema-based XML.

The following screen shot shows that a simple refresh of the browser is all that is required to see the updated content.

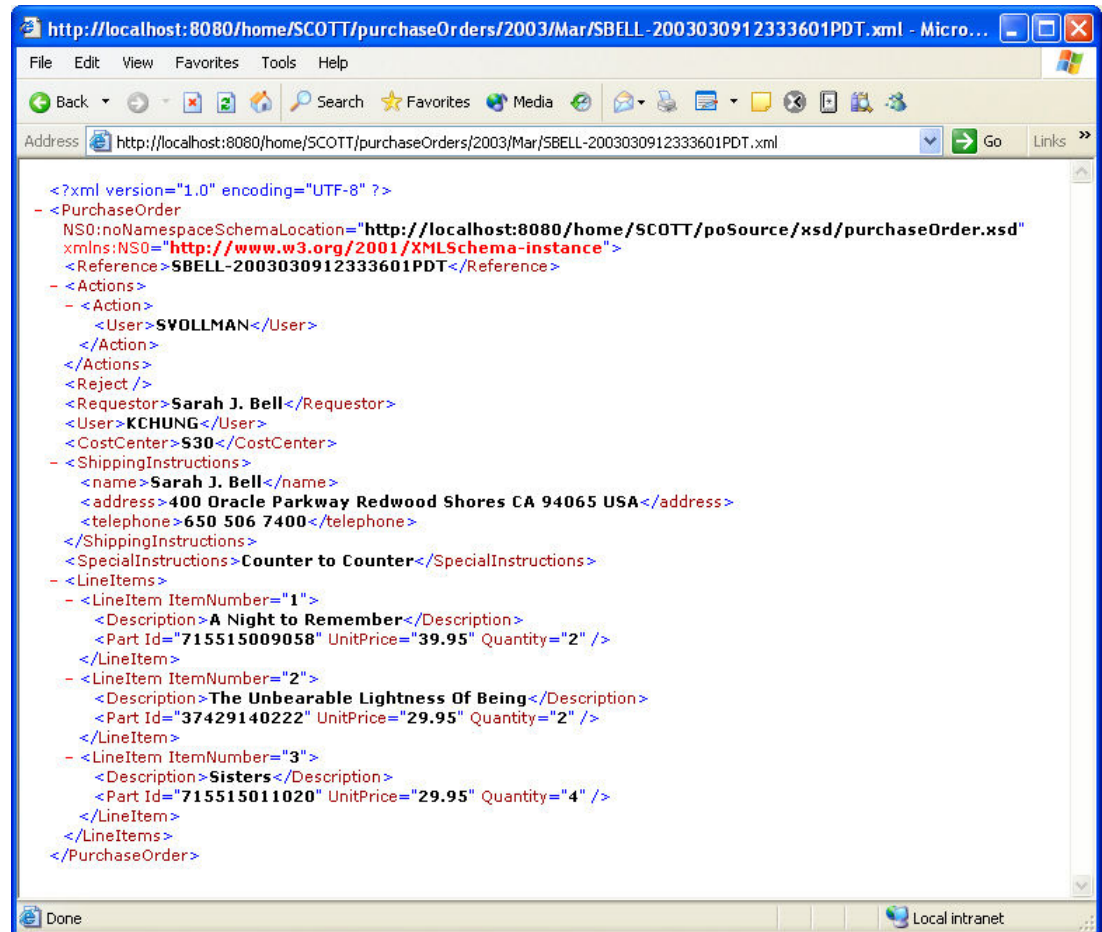
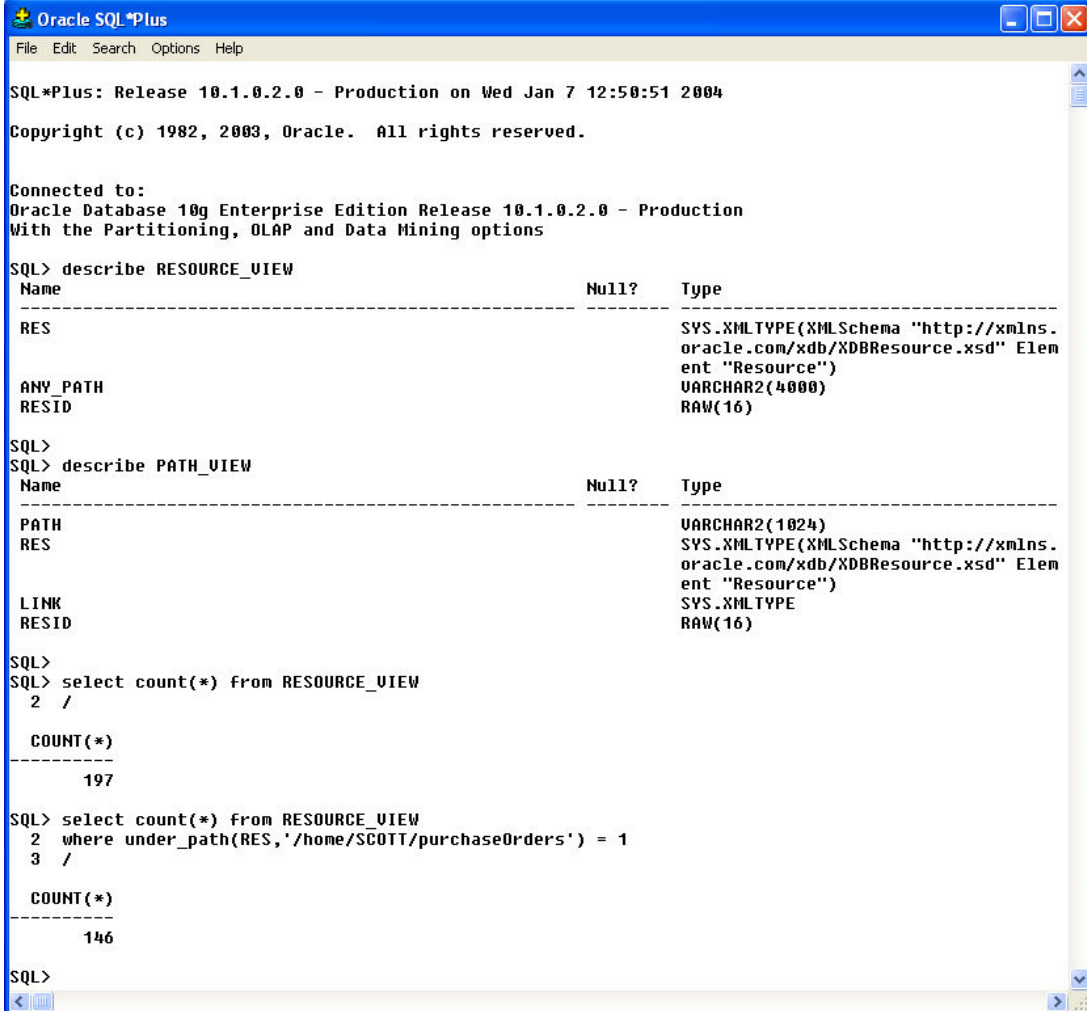


Figure XXXV. Using a Web browser to view updated content

The **RESOURCE_VIEW** and **PATH_VIEW** provide the SQL programmer the ability to access and update the content of a document stored in the Oracle XML DB repository. A SQL programmer can query and update the meta-data contained in the **RESOURCE_VIEW** and **PATH_VIEW**.

The following screen shot shows some simple queries against the **RESOURCE_VIEW**



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 12:50:51 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> describe RESOURCE_VIEW
Name                                     Null?    Type
-----
RES                                     SYS.XMLTYPE(XMLSchema "http://xmlns.oracle.com/xdm/XDBResource.xsd" Element "Resource")
ANY_PATH                                VARCHAR2(4000)
RESID                                   RAW(16)

SQL>
SQL> describe PATH_VIEW
Name                                     Null?    Type
-----
PATH                                    VARCHAR2(1024)
RES                                    SYS.XMLTYPE(XMLSchema "http://xmlns.oracle.com/xdm/XDBResource.xsd" Element "Resource")
LINK                                    SYS.XMLTYPE
RESID                                   RAW(16)

SQL>
SQL> select count(*) from RESOURCE_VIEW
2 /

COUNT(*)
-----
197

SQL> select count(*) from RESOURCE_VIEW
2 where under_path(RES, '/home/SCOTT/purchaseOrders') = 1
3 /

COUNT(*)
-----
146

SQL>

```

Figure XXXVI. Simple Queries against the **RESOURCE_VIEW**

KEY POINTS:

- The **RESOURCE_VIEW** contains one entry for each file or folder stored in the XML DB repository. It consists of three columns. **RES** is the XML document that contains the meta-data properties associated with the document. **ANY_PATH** is a valid path that can be used by the current user to access the document. **RESID** is the OID of the underlying row in the underlying table **XDB\$RESOURCE**
- The **PATH_VIEW** contains one entry for each path that can be used to access a file or folder stored in the XML DB repository. It consists of three columns. **RES** is the XML document that contains the meta-data properties associated with the document. **PATH** is the path that can be used by the current user to access the document. **LINK** is the an XML document that contain the link properties for this path to the document. **RESID** is the OID of the underlying row in the underlying table **XDB\$RESOURCE**
- The first query uses the **count()** operator to count the number of objects currently stored in the Oracle XML DB repository. This query will not return the total number of documents stored in the repository, it will return the number of documents that the user issuing the query has read access to.
- The second query uses the **under_path()** operator to restrict a query to a particular tree within the repository. In this example the count(*) operation is being restricted to the number of objects that are in the tree whose root is identified by the path */home/SCOTT/purchaseOrders*.

The WebDAV Specifications defines the set of meta-data properties that a WebDAV server is expected to maintain for each resource. It also defines that a WebDAV server and a DAV enabled client will exchange meta-data as an XML document. Oracle XML DB maintains the meta-data for each resource as an XML document. This document is compliant with the XML Schema **XDBResource.xsd**.

The resource documents can be accessed, queried and updated just like any other XML stored in Oracle XML DB.

The following screen shot shows the XML representation of the standard set of meta-data properties for the folder `/home/SCOTT/purchaseOrders`.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 12:55:00 2004

Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select r.RES.getClobVal()
2   from RESOURCE_VIEW r
3   where equals_path(RES,'/home/SCOTT/purchaseOrders' ) = 1
4   /

R.RES.GETCLOBVAL()
-----
<Resource xmlns="http://xmlns.oracle.com/xdb/XDBResource.xsd" Hidden="false" Inva
lid="false" Container="true" CustomRslt="false" VersionHistory="false" StickyRe
f="true">
  <CreationDate>2004-01-06T23:07:50.850000</CreationDate>
  <ModificationDate>2004-01-06T23:09:05.518000</ModificationDate>
  <DisplayName>purchaseOrders</DisplayName>
  <Language>en-US</Language>
  <CharacterSet>UTF-8</CharacterSet>
  <ContentType>application/octet-stream</ContentType>
  <RefCount>1</RefCount>
  <ACL>
    <acl description="Private:All privileges to OWNER only and not accessible to
others" xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:dav="DAV:" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.ora
cle.com/xdb/acl.xsd http://xmlns.oracle.com/xdb/acl.xs
d">
      <ace>
        <principal>dav:owner</principal>
        <grant>true</grant>
        <privilege>
          <all/>
        </privilege>
      </ace>
    </acl>
  </ACL>
  <Owner>SCOTT</Owner>
  <Creator>SCOTT</Creator>
  <LastModifier>SCOTT</LastModifier>
</Resource>

SQL>

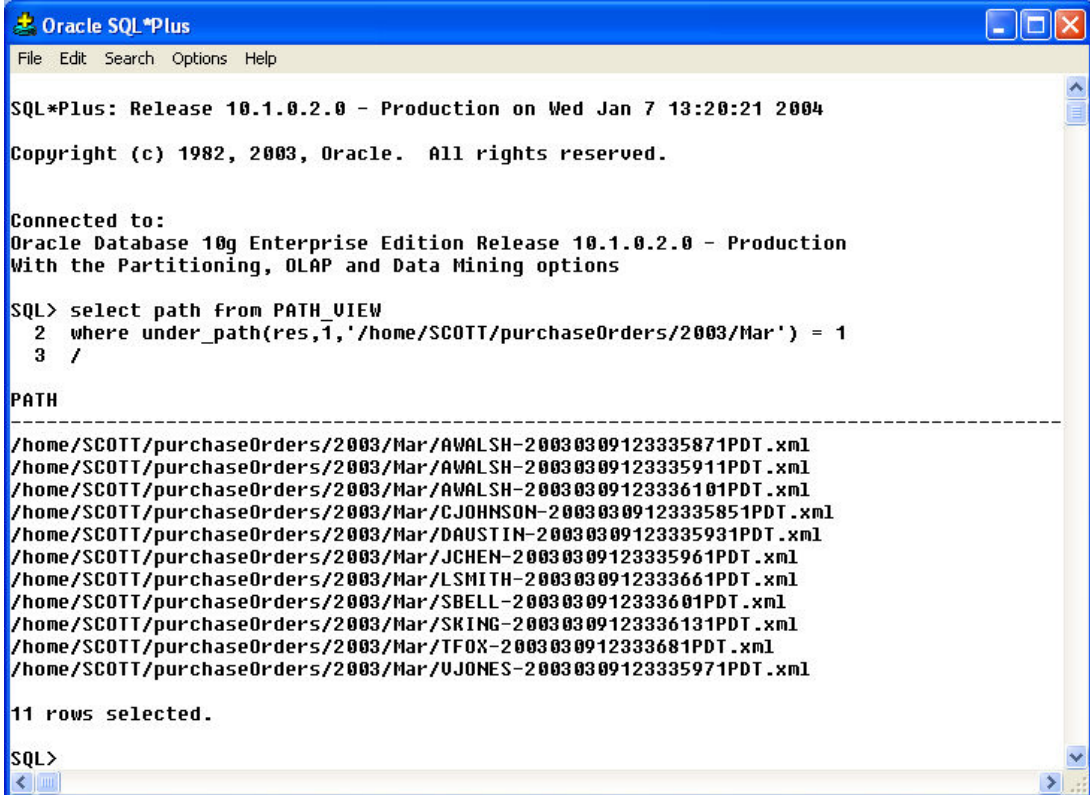
```

Figure XXXVII. Viewing a Resource

KEY POINTS:

- The resource XML contains standard meta-data such as Creation Date, Creator, Owner, Last Modification Date and Display Name.
- This content can be accessed and updated using the SQL/XML operators.

The following screen shot shows the **under_path()** operator and the **PATH_VIEW** being used to find the set of documents in a particular folder.



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 13:20:21 2004

Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select path from PATH_VIEW
  2  where under_path(res,1,'/home/SCOTT/purchaseOrders/2003/Mar') = 1
  3  /

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/LSHITH-2003030912333661PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/SBELL-2003030912333601PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/SKING-20030309123336131PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/TFOX-2003030912333681PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/UJONES-20030309123335971PDT.xml

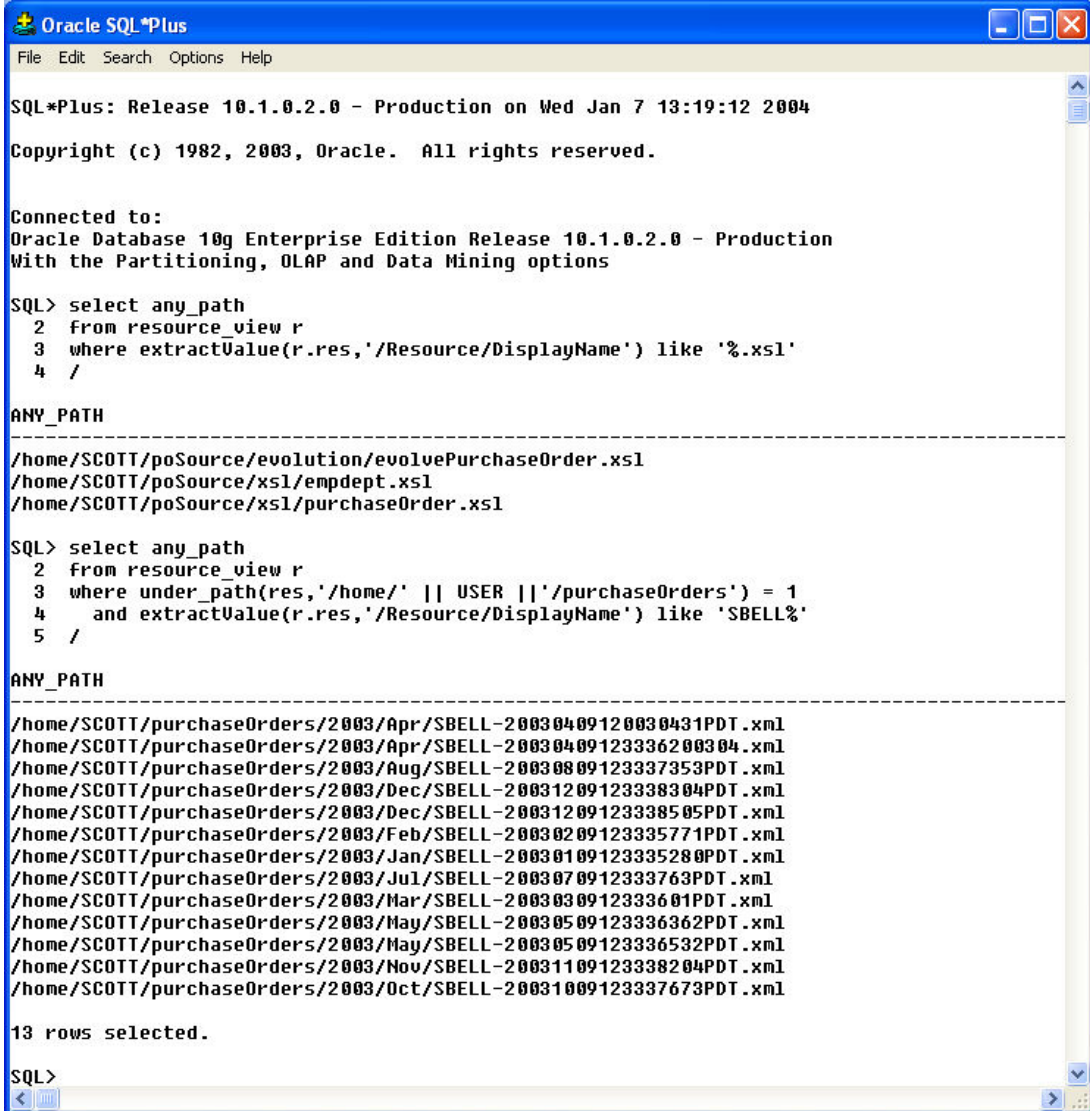
11 rows selected.

SQL>

```

Figure XXXVIII. Querying the Folder Hierarchy

The following screen shots show some simple queries against the meta-data contained in the **RESOURCE_VIEW**



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 13:19:12 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select any_path
  2 from resource_view r
  3 where extractValue(r.res,'/Resource/DisplayName') like '%.xsl'
  4 /

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

SQL> select any_path
  2 from resource_view r
  3 where under_path(res,'/home/' || USER || '/purchaseOrders') = 1
  4 and extractValue(r.res,'/Resource/DisplayName') like 'SBELL%'
  5 /

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-2003070912333763PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/SBELL-2003030912333601PDT.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

13 rows selected.

SQL>

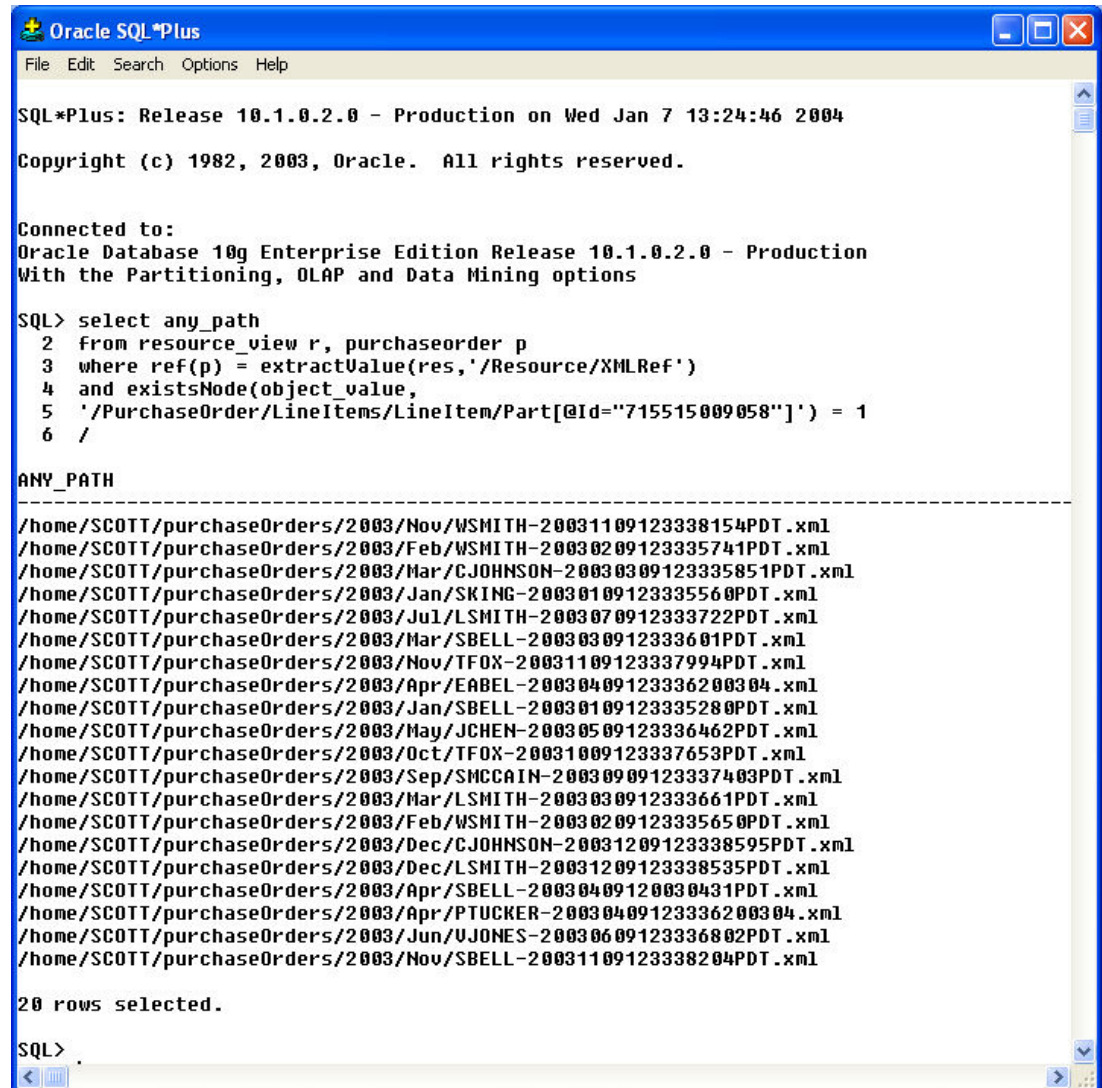
```

Figure XXXIX. Querying the Resource View

KEY POINTS:

- The first query finds a path to each XSL style sheet stored in the Oracle XML DB repository. It finds the set of documents by using a SQL 'like' query on the contents of the node `/Resource/DisplayName` to find documents with names that end with the file extension `'xsl'`.
- The second query finds the path to all the documents under the folder `/home/SCOTT/purchaseOrders` that starts with `'SBELL'`. In this case the `under_path()` operator restricts which tree is searched, and the like condition restricts which documents in the tree are returned.

The next screen shot shows a query that performs a join between the default table (that contains the content of a Schema based XML document) and the meta-data associated with the document.



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 13:24:46 2004

Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select any_path
2  from resource_view r, purchaseorder p
3  where ref(p) = extractValue(res, '/Resource/XMLRef')
4  and existsNode(object_value,
5  '/PurchaseOrder/LineItems/LineItem/Part[@Id='715515009058']) = 1
6  /

ANY_PATH
-----
/home/SCOTT/purchaseOrders/2003/Nov/WSMITH-20031109123338154PDT.xml
/home/SCOTT/purchaseOrders/2003/Feb/WSMITH-20030209123335741PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/CJOHNSON-20030309123335851PDT.xml
/home/SCOTT/purchaseOrders/2003/Jan/SKING-20030109123335560PDT.xml
/home/SCOTT/purchaseOrders/2003/Jul/LSMITH-2003070912333722PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/SBELL-2003030912333601PDT.xml
/home/SCOTT/purchaseOrders/2003/Nov/TFOX-20031109123337994PDT.xml
/home/SCOTT/purchaseOrders/2003/Apr/EABEL-20030409123336200304.xml
/home/SCOTT/purchaseOrders/2003/Jan/SBELL-20030109123335280PDT.xml
/home/SCOTT/purchaseOrders/2003/May/JCHEN-20030509123336462PDT.xml
/home/SCOTT/purchaseOrders/2003/Oct/TFOX-20031009123337653PDT.xml
/home/SCOTT/purchaseOrders/2003/Sep/SMCCAIN-20030909123337403PDT.xml
/home/SCOTT/purchaseOrders/2003/Mar/LSMITH-2003030912333661PDT.xml
/home/SCOTT/purchaseOrders/2003/Feb/WSMITH-20030209123335650PDT.xml
/home/SCOTT/purchaseOrders/2003/Dec/CJOHNSON-20031209123338595PDT.xml
/home/SCOTT/purchaseOrders/2003/Dec/LSMITH-20031209123338535PDT.xml
/home/SCOTT/purchaseOrders/2003/Apr/SBELL-20030409120030431PDT.xml
/home/SCOTT/purchaseOrders/2003/Apr/PTUCKER-20030409123336200304.xml
/home/SCOTT/purchaseOrders/2003/Jun/VJONES-20030609123336802PDT.xml
/home/SCOTT/purchaseOrders/2003/Nov/SBELL-20031109123338204PDT.xml

20 rows selected.

SQL> .

```

Figure XL. Joining meta-data and Content

KEY POINTS:

- In this example the query on the *PURCHASEORDER* table locates the set of documents that contain an order for the Part whose 'Id' is 75515009058. The results of this query are joined with the **RESOURCE_VIEW**, using the node /Resource/XMLRef. The **RESOURCE_VIEW** is then used to obtain a valid path to each of the documents that contained an order for the part.
- The ability to perform joins between the **RESOURCE_VIEW** and the default tables is extremely important as it allows a developer to create queries that include conditions based on both meta-data and content.
- In this example the path returned by the query forms the local part of a URL that can be used to access the document directly from a web browser.

Using conventional relational techniques, path-based queries become very inefficient as the depth of the hierarchy increases. The reason for this is that performing a path-based query in a relational database typically requires a `CONNECT BY` operation. A `CONNECT BY` query is difficult for a traditional relational database to resolve in an efficient manner. Oracle XML DB introduces a new index called the hierarchical index. This index allows the database to resolve a path-based query without using a 'connect by' operation. This means that path-based queries are resolved in an extremely efficient manner.

The following screen shot shows the explain plan output for a path-based query.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 13:28:48 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

Connected.
Connected.
SQL> explain plan for
2 select any_path
3 from resource_view r, purchaseorder p
4 where ref(p) = extractValue(res, '/Resource/XMLRef')
5 and existsNode(object_value,
6 '/PurchaseOrder/LineItems/LineItem/Part[@Id='715515009058']) = 1
7 /

Explained.

SQL> set echo off

PLAN_TABLE_OUTPUT
-----
Plan hash value: 2353940853

-----
| Id | Operation | Name | Rows | Bytes | Cost (%CPU)| Time |
-----
| 0 | SELECT STATEMENT | | 1 | 323 | 12 (0)| 00:00:01 |
|* 1 | HASH JOIN SEMI | | 1 | 323 | 12 (0)| 00:00:01 |
| 2 | NESTED LOOPS | | 1 | 203 | 6 (0)| 00:00:01 |
| 3 | TABLE ACCESS BY INDEX ROWID | XDB$RESOURCE | 2 | 292 | 4 (0)| 00:00:01 |
| 4 | DOMAIN INDEX | XDBHI_IDX | | | | |
|* 5 | TABLE ACCESS BY INDEX ROWID | PURCHASEORDER | 1 | 57 | 1 (0)| 00:00:01 |
|* 6 | INDEX UNIQUE SCAN | SYS_C002945 | 1 | | 0 (0)| 00:00:01 |
|* 7 | INDEX UNIQUE SCAN | LINEITEM_TABLE_DATA | 22 | 2640 | 6 (0)| 00:00:01 |
|* 8 | INDEX RANGE SCAN | IPARTNUMBER | 106 | | 2 (0)| 00:00:01 |
-----

Predicate Information (identified by operation id):
-----
1 - access("NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC0003400035$")
5 - filter(SYS_CHECKACL("ACLOID", "OWNERID", xmltype(' '<privilege
xmlns="http://xmlns.oracle.com/xdb/acl.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/xdb/acl.xsd
http://xmlns.oracle.com/xdb/acl.xsd DAU:http://xmlns.oracle.com/xdb/dav.xsd"><read-properties
/><read-contents/></privilege>'))=1)
6 - access("PURCHASEORDER"."SYS_NC_OID$"="P"."SYS_NC00024$")
7 - access("SYS_NC00011$"='715515009058')
8 - access("SYS_NC00011$"='715515009058')

28 rows selected.

SQL>

```

Figure XLI. Explain plan for path-based query showing use of Hierarchical Index

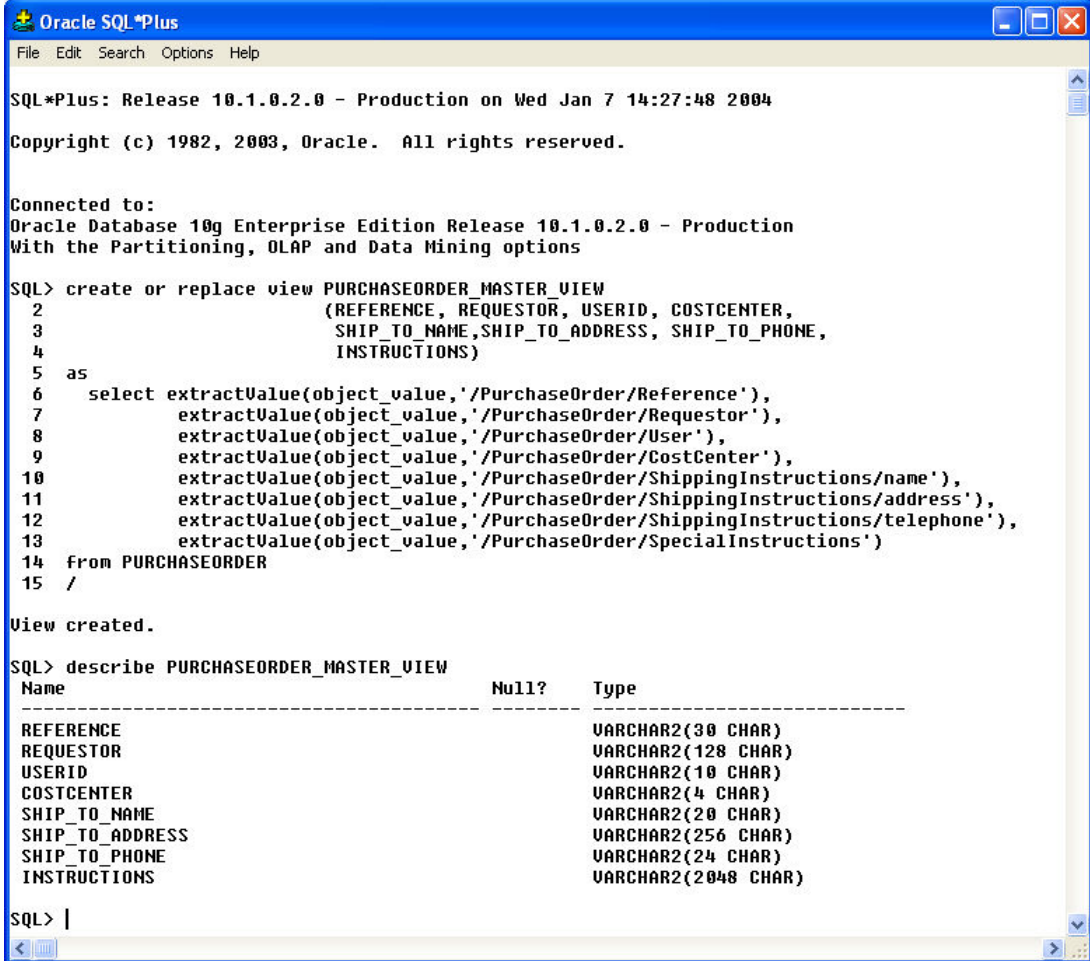
KEY POINTS:

- The hierarchical index is used to resolve the path-based query. No connect by processing is required to resolve the query.
- The hierarchical index is implemented as an Oracle domain index. This is the same technique that is used to add support for text indexing and many other advanced index types to the database.
- Oracle Database 10g Release 1 introduces cost-based optimization of queries on RESOURCE_VIEW and PATH_VIEW, making access to repository content up to 100 times faster than Oracle Database 9iR2.

RELATIONAL ACCESS TO XML CONTENT

Oracle XML DB allows XML content, stored in the database, to be exposed as conventional relational views. This means that tools, applications and programmers who have no understanding of XML, but understand the Oracle database, can work with XML content. The views use the SQL/XML operators and XPath expressions to map nodes in the XML document into columns in the view.

The following screen shot shows the definition of a relational view that will expose the header information contained in a PurchaseOrder document as a conventional relational view.



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 14:27:48 2004

Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> create or replace view PURCHASEORDER_MASTER_VIEW
2      (REFERENCE, REQUESTOR, USERID, COSTCENTER,
3      SHIP_TO_NAME, SHIP_TO_ADDRESS, SHIP_TO_PHONE,
4      INSTRUCTIONS)
5  as
6      select extractValue(object_value, '/PurchaseOrder/Reference'),
7             extractValue(object_value, '/PurchaseOrder/Requestor'),
8             extractValue(object_value, '/PurchaseOrder/User'),
9             extractValue(object_value, '/PurchaseOrder/CostCenter'),
10            extractValue(object_value, '/PurchaseOrder/ShippingInstructions/name'),
11            extractValue(object_value, '/PurchaseOrder/ShippingInstructions/address'),
12            extractValue(object_value, '/PurchaseOrder/ShippingInstructions/telephone'),
13            extractValue(object_value, '/PurchaseOrder/SpecialInstructions')
14  from PURCHASEORDER
15  /

View created.

SQL> describe PURCHASEORDER_MASTER_VIEW
Name                                     Null?      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)

SQL> |

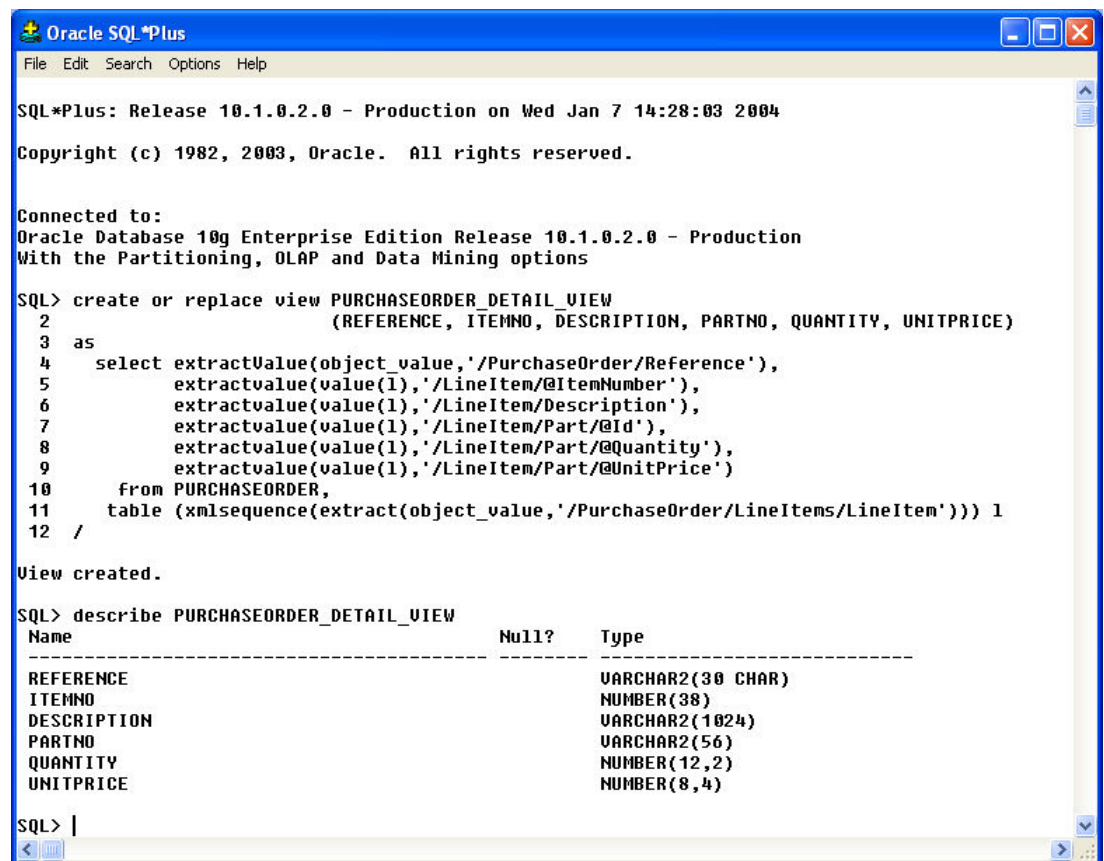
```

Figure XLII. creating a simple relational view over an XMLType table

KEY POINTS:

- The view defines the required set of columns. A set of XPath expressions and the **extractvalue()** operator are used to define which node in the document maps to which columns in the view.
- The view appears to be a standard relational view.
- This technique is used when there is a 1:1 relationship between the documents in the XMLType table and the rows in the view.

The following screen shot show how SQL/XML operators to can be used to expose the set of elements in a collection as a set of rows in a relational view.



```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 14:28:03 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> create or replace view PURCHASEORDER_DETAIL_VIEW
2      (REFERENCE, ITEMNO, DESCRIPTION, PARTNO, QUANTITY, UNITPRICE)
3  as
4      select extractValue(object_value, '/PurchaseOrder/Reference'),
5             extractvalue(value(1), '/LineItem/@ItemNumber'),
6             extractvalue(value(1), '/LineItem/Description'),
7             extractvalue(value(1), '/LineItem/Part/@Id'),
8             extractvalue(value(1), '/LineItem/Part/@Quantity'),
9             extractvalue(value(1), '/LineItem/Part/@UnitPrice')
10     from PURCHASEORDER,
11     table (xmlsequence(extract(object_value, '/PurchaseOrder/LineItems/LineItem'))) l
12 /

View created.

SQL> describe PURCHASEORDER_DETAIL_VIEW
Name                               Null?    Type
-----
REFERENCE                          VARCHAR2(30 CHAR)
ITEMNO                             NUMBER(38)
DESCRIPTION                        VARCHAR2(1024)
PARTNO                             VARCHAR2(56)
QUANTITY                           NUMBER(12,2)
UNITPRICE                          NUMBER(8,4)

SQL> |

```

Figure XLIII. creating a simple relational view over a XML collection

KEY POINTS:

- The view looks and behaves like a normal relational view.
- The create view statement defines the required set of columns. The view is based on a virtual table that is created using the SQL **table()**, **xmlsequence()** and **extract()** operators. The virtual table will contain one document for each element in the target collection.
- A set of XPath expressions and the **extractvalue()** operator are used to define which node in the virtual table maps to which column in the view.
- The virtual table is created as follows

The **extract()** operator is used to generate an XML fragment for each document in the *PURCHASEORDER* table. The fragment will contain the set of nodes that match the XPath expression passed to the **extract()** operator. In this case the **extract()** operator will generate an XML fragment that contains the set of *LineItem* elements present in the original *PurchaseOrder* document. Each *LineItem* will appear as a top level node.

The fragment is passed to the **xmlsequence()** operator. The **xmlsequence()** operator takes the XML fragment and generates a separate row from each top level node in the fragment. Each row will consist of a single *XMLType*.

The set of rows generated by the **xmlsequence()** operator are based to the SQL **table()** operator. The **table()** operator generates a virtual table from the set of rows generated by **xmlsequence()**.
- There is an implicit correlated join between the base table and the argument passed to **extract()** operator. This ensures that the query does not generate a Cartesian product.

The following screen shot shows how the views allow standard relational queries to be executed on XML content.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 14:28:21 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select      REFERENCE, COSTCENTER, SHIP_TO_NAME
2  from PURCHASEORDER_MASTER_VIEW
3  where USERID = 'KCHUNG'
4  /

REFERENCE                                COST SHIP_TO_NAME
-----
SBELL-2003030912333601PDT                S30 Sarah J. Bell

SQL> select d.REFERENCE, d.ITEMNO, d.PARTNO, d.DESRIPTION
2  from PURCHASEORDER_DETAIL_VIEW d, PURCHASEORDER_MASTER_VIEW m
3  where m.REFERENCE = d.REFERENCE
4  and m.USERID = 'KCHUNG'
5  /

REFERENCE                                ITEMNO PARTNO      DESCRIPTION
-----
SBELL-2003030912333601PDT                1 715515009058  A Night to Remember
SBELL-2003030912333601PDT                2 37429140222   The Unbearable Lightness Of Being
SBELL-2003030912333601PDT                3 715515011020  Sisters

SQL>

```

Figure XLIV. using a view to perform simple relational queries over XMLType content.

KEY POINTS:

- The first query shows a simple query against the master view. The second query shows a query based on a join between the master view and the detail view.
- Since the views look and act like standard relational views they can be queried using standard relational syntax. There is no XML specific syntax in either the query syntax or the generated result set.

The following screen shot shows a simple business intelligence query being executed on XML content.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 14:44:30 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> select partno, count(*) "No of Orders", quantity "No of Copies"
2  from purchaseorder_detail_view
3  where partno in ( 715515009126, 715515009058 )
4  group by rollup(partno, quantity)
5  /

PARTNO          No of Orders No of Copies
-----
715515009058          7             1
715515009058          8             2
715515009058          5             3
715515009058          2             4
715515009058         22             4
715515009126          4             1
715515009126          7             3
715515009126         11             3
715515009126         33             3

9 rows selected.

SQL>

```

Figure XLV. using a relational view to perform complex SQL queries on XML content.

KEY POINTS:

- The query performs an analysis of the PurchaseOrder documents to determine the number of copies of each title that are being ordered on each PurchaseOrder. For example, looking at PurchaseOrders that contain an order for part '715515009058', there are 7 PurchaseOrders where 1 copy of the item is ordered and 2 PurchaseOrders where 4 copies of the item are ordered.
- By exposing XML content as relational data Oracle XML DB allows advanced features of the Oracle Database 10g, such as the Business Intelligence and Analytic capabilities, to be applied to XML content. Even though the B.I. features themselves are not XML aware, the XML/SQL duality provided by Oracle XML DB makes it possible for these features to be applied to XML content.

XML ACCESS TO RELATIONAL CONTENT

SQL/XML OPERATORS

Oracle XML DB allows XML documents to be generated directly from relational content. Using The SQL/XML operators, developers can create SQL statements which generate a single XML document or a set of XML documents, rather than a conventional tabular result set.

Oracle XML DB also allows the results of a SQL statement that generates one or more XML documents can be persisted as an XMLType view. This makes it possible to use the SQL/XML operators to provide a persistent XML view of relational content.

The following screen shot shows how the contents of the relational tables *DEPARTMENTS*, *EMPLOYEES*, *JOBS*, *LOCATIONS* and *COUNTRIES*, located in the *HR* Schema can be exposed as a set of XML documents. In this case the view will contain one document for each row in the *DEPARTMENTS* table.

```

Oracle SQL*Plus
File Edit Search Options Help

SQL*Plus: Release 10.1.0.2.0 - Production on Wed Jan 7 14:48:45 2004
Copyright (c) 1982, 2003, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - Production
With the Partitioning, OLAP and Data Mining options

SQL> create or replace view DEPARTMENT_XML of xmltype
2 with object id
3 (
4 substr(extractValue(sys_nc_rowinfo$, '/Department/Name'),1,128)
5 ) from columns (d.DEPARTMENT_NAME)
6 as
7 select xmlElement
8 (
9 "Department",
10 xmlAttributes( d.DEPARTMENT_ID as "DepartmentId"),
11 xmlElement("Name", d.DEPARTMENT_NAME),
12 xmlElement
13 (
14 "Location",
15 xmlForest
16 (
17 STREET_ADDRESS as "Address", CITY as "City", STATE_PROVINCE as "State",
18 POSTAL_CODE as "Zip", COUNTRY_NAME as "Country"
19 )
20 ),
21 xmlElement
22 (
23 "EmployeeList",
24 (
25 select xmlAgg
26 (
27 xmlElement
28 (
29 "Employee",
30 xmlAttributes( e.EMPLOYEE_ID as "employeeNumber" ),
31 xmlForest
32 (
33 e.FIRST_NAME as "FirstName", e.LAST_NAME as "LastName", e.EMAIL as "EmailAddress",
34 e.PHONE_NUMBER as "Telephone", e.HIRE_DATE as "StartDate", j.JOB_TITLE as "JobTitle",
35 e.SALARY as "Salary", m.FIRST_NAME || ' ' || m.LAST_NAME as "Manager"
36 ),
37 xmlElement ( "Commission", e.COMMISSION_PCT )
38 )
39 )
40 from HR.EMPLOYEES e, HR.EMPLOYEES m, HR.JOBS j
41 where e.DEPARTMENT_ID = d.DEPARTMENT_ID
42 and j.JOB_ID = e.JOB_ID
43 and m.EMPLOYEE_ID = e.MANAGER_ID
44 )
45 )
46 ) as XML
47 from HR.DEPARTMENTS d, HR.COUNTRIES c, HR.LOCATIONS l
48 where d.LOCATION_ID = l.LOCATION_ID
49 and l.COUNTRY_ID = c.COUNTRY_ID
50 /

View created.

SQL>

```

Figure XLVI. Using SQL/XML operators to create an XMLType View over relational content

KEY POINTS:

- The SQL/XML standard defines a set of operators that allow any shape of XML to be generated from the tabular result set returned by a conventional relational query.
- SQL/XML defines operators that can be used to aggregate the results of sub-queries into XML collections.
- XMLType views allow an XML representation of the relational data to be persisted in the database.
- Oracle Database 10g Release 1 includes support for XPath re-write over XMLType views created using the SQL/XML operators. This means that the XPath based operators **extract()**, **extractValue()** and **existsNode()** can be used to efficiently query these views.

The following screen shot shows a simple query against an XMLType View. In this case the XPath expression restricts the result set to the node that contains the information related to the Department named “Executive”.

```

Oracle SQL*Plus
File Edit Search Options Help
With the Partitioning, OLAP and Data Mining options

SQL> select XMLTYPE.extract(object_value,'/*')
2   from DEPARTMENT_XML
3  where existsNode(object_value,'/Department[Name="Executive"]') = 1
4  /

XMLTYPE.EXTRACT(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>21-SEP-89</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>13-JAN-93</StartDate>
      <JobTitle>Administration Vice President</JobTitle>
      <Salary>17000</Salary>
      <Manager>Steven King</Manager>
      <Commission/>
    </Employee>
  </EmployeeList>
</Department>

SQL>

```

Figure XLVII. Querying an XMLType using XML metaphors.

KEY POINTS:

- The XMLType view allows relational data to be persisted as XML content.
- Rows in an XMLType view can be persisted as documents in the Oracle XML DB repository.
- Rows in an XMLType view can be queried using the SQL/XML operators.
- XPath re-write will translate the queries into the operations on the underlying tables.

THE DBURI SERVLET

The database-resident *DBUri Servlet* allows the functionality of the DBUri data-type to be accessed directly from any browser that supports XML. The *DBUri Servlet* allows a simple URL to be used to view the entire contents of any table in an Oracle database. The contents of the table are returned as a single XML Document. The local part of the URL takes the form of `/oradb/HR/DEPARTMENTS`, where *oradb* is the default virtual root of the *DBUri Servlet*, *HR* is the name of the target database schema, and *DEPARTMENTS* is the name of the table.

The following screen shot shows the results using the *DBUri Servlet* to access the contents of the *DEPARTMENTS* table:

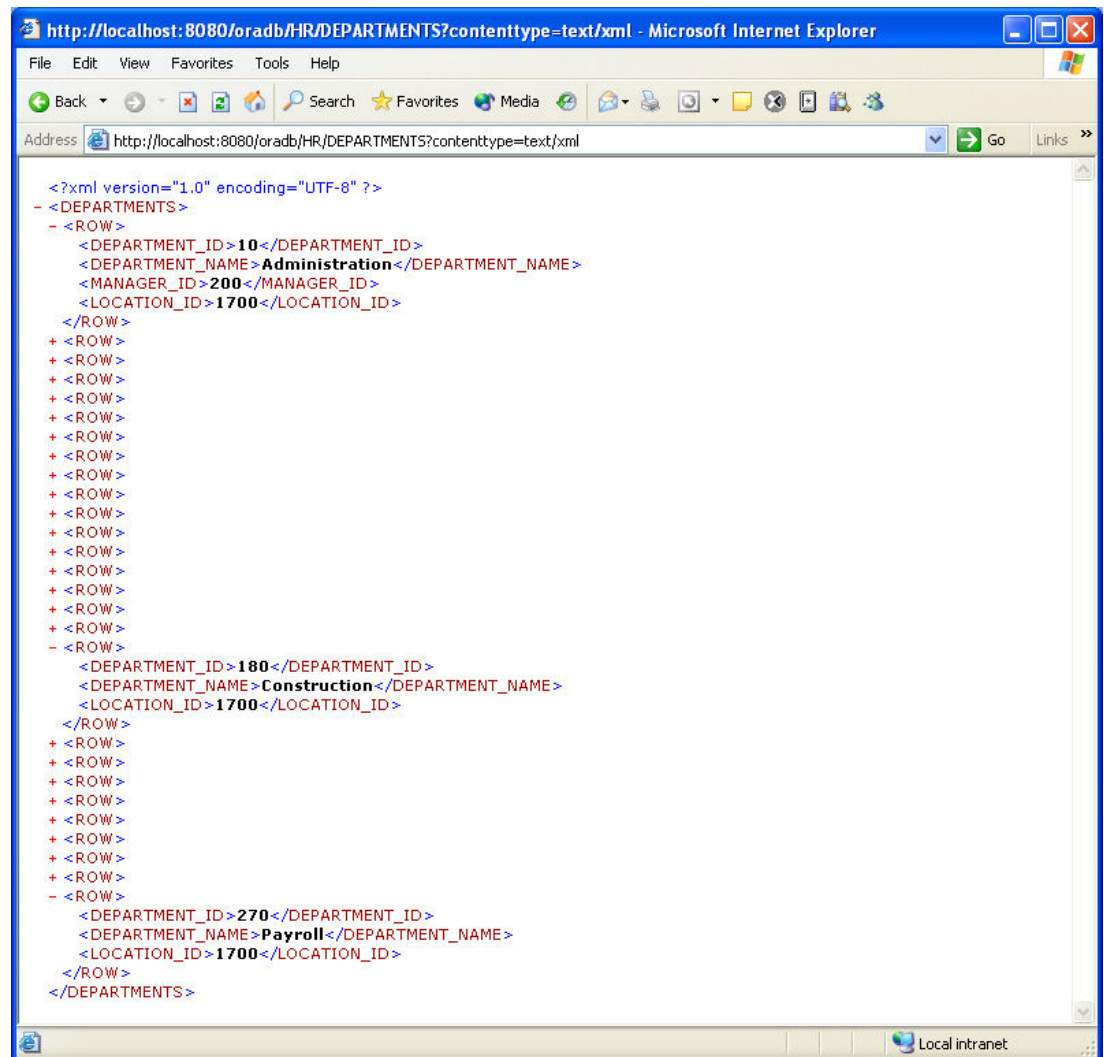


Figure XLVIII. Using the DBUri Servlet to access relational content.

KEY POINTS:

- Each row in the table becomes a complex element called ROW.
- The ROW element contains one element for each column in the table.
- The set of ROW elements is enclosed in a ROOT whose name is derived from the name of the table.

The URL passed to the *DBUri Servlet* controls the content of the generated XML document. The URL can contain XPath expressions that control the set of rows and columns that are included in the generated XML document. The URL passed to the *DBUri Servlet* can also include parameters that provide control over the features like the name of the root element, or the mime type of content being generated.

The following example shows how a more complex URL can be used to control the output of the *DBUri Servlet*. The complete URL used in this case is

```
http://localhost:8080/oradb/HR/DEPARTMENTS/ROW[LOCATION_ID="2400"%20or%20LOCATION_ID="1800"]/DEPARTMENT_NAME?contenttype=text/xml&rowsettag=DepartmentNames
```

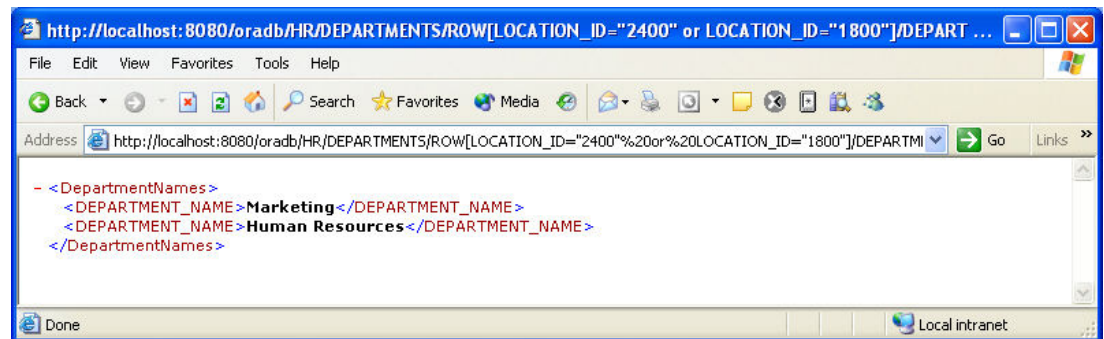


Figure XLIX. Using a complex URL to control content and formatting.

KEY POINTS:

- The URL uses the XPath notation [LOCATION_ID="2400" or LOCATION_ID="1800"] to limit which rows are included in the generated XML document.
- The URL uses the XPath notation ROW[.]/DEPARTMENT_NAME to limit which columns are included in the generated document.
- The URL uses the rowsettag parameter to specify the name of the root element of the generated document.

The *DBUri Servlet* can also be used to access XML content stored in an XMLType table or view. When accessing XML content the URL can contain XPath notations which control which documents are returned and which nodes are included. In the case of an XMLType the URL is allowed to reference any node in the document.

The following screen shot shows the *DBUri Servlet* being used to access a row in the *PURCHASEORDER* table.

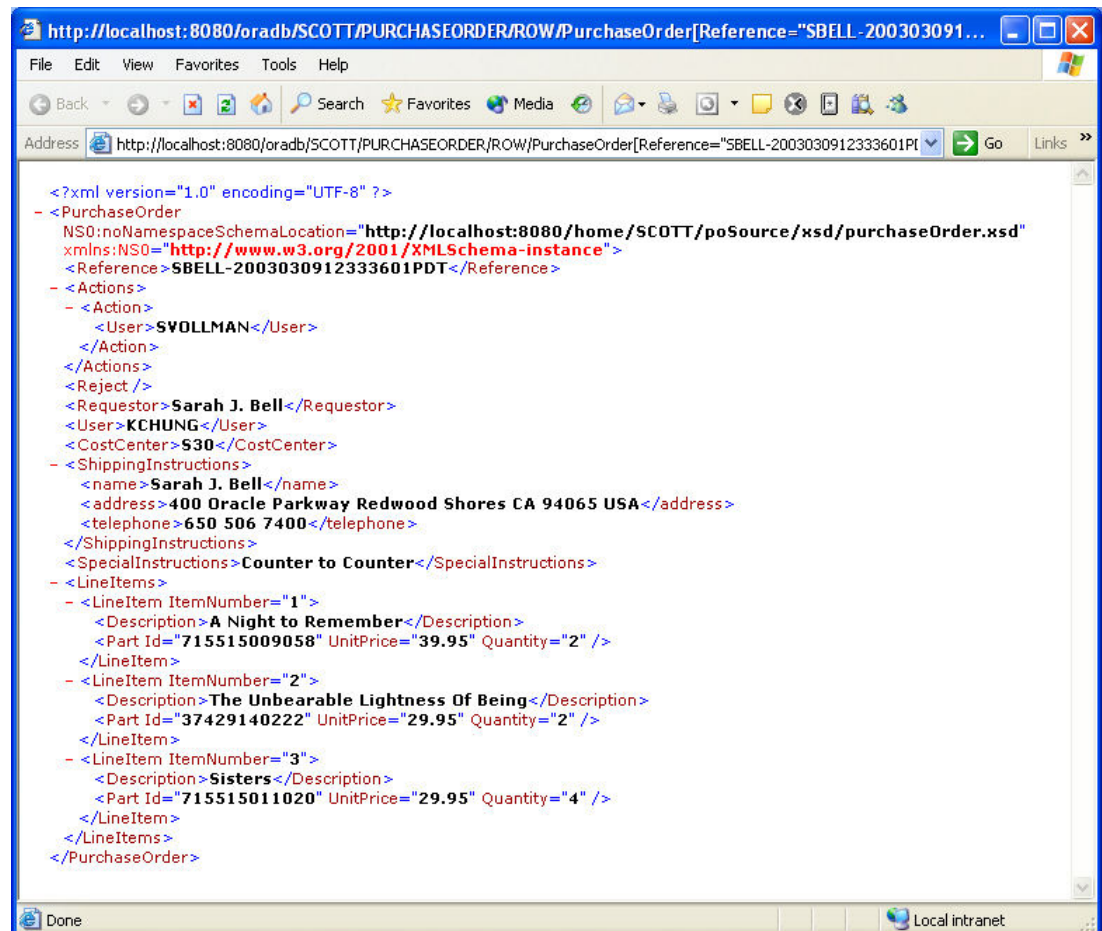


Figure L. Using the DBUri Servlet to access XML content.

KEY POINTS:

- The URL accesses the document as a row in the *PURCHASEORDER* table rather than as a resource in the XML DB repository.
- The generated document will include only documents where the node */PurchaseOrder/Reference/text()* contains the value specified in the condition.
- The *contenttype* parameter is used to set the mime type of the generated document to *text/xml*.

XSL TRANSFORMATION

Oracle XML DB provides support for performing XSL transformations inside the database. Traditionally, since XSL transformation is based on the DOM memory Model, XSL transformation has required very large amounts of memory. By performing XSL transformation inside the database, alongside the data, Oracle XML DB is able to use XML specific memory optimization to significantly reduce the amount of memory required to perform the transformation.

The SQL **xmlTransform()** operator provides the SQL programmer with access to the XSL transformation capability. The **xmlTransform()** operator takes two arguments; the first is the XML document to be transformed, the second is the XSL style sheet which defines the transformation. Both parameters are XMLType. The result of the transformation is also expected to be a valid XML document. This means that any HTML generated by the transformation must be XHTML, which is valid XML, as well as valid HTML.

The most efficient way to make a style sheet available to the **xmlTransform()** operator is to store it as a document in the Oracle XML DB repository. The following screen shot shows a standard style sheet stored in the Oracle XML DB repository being accessed from a Web browser.

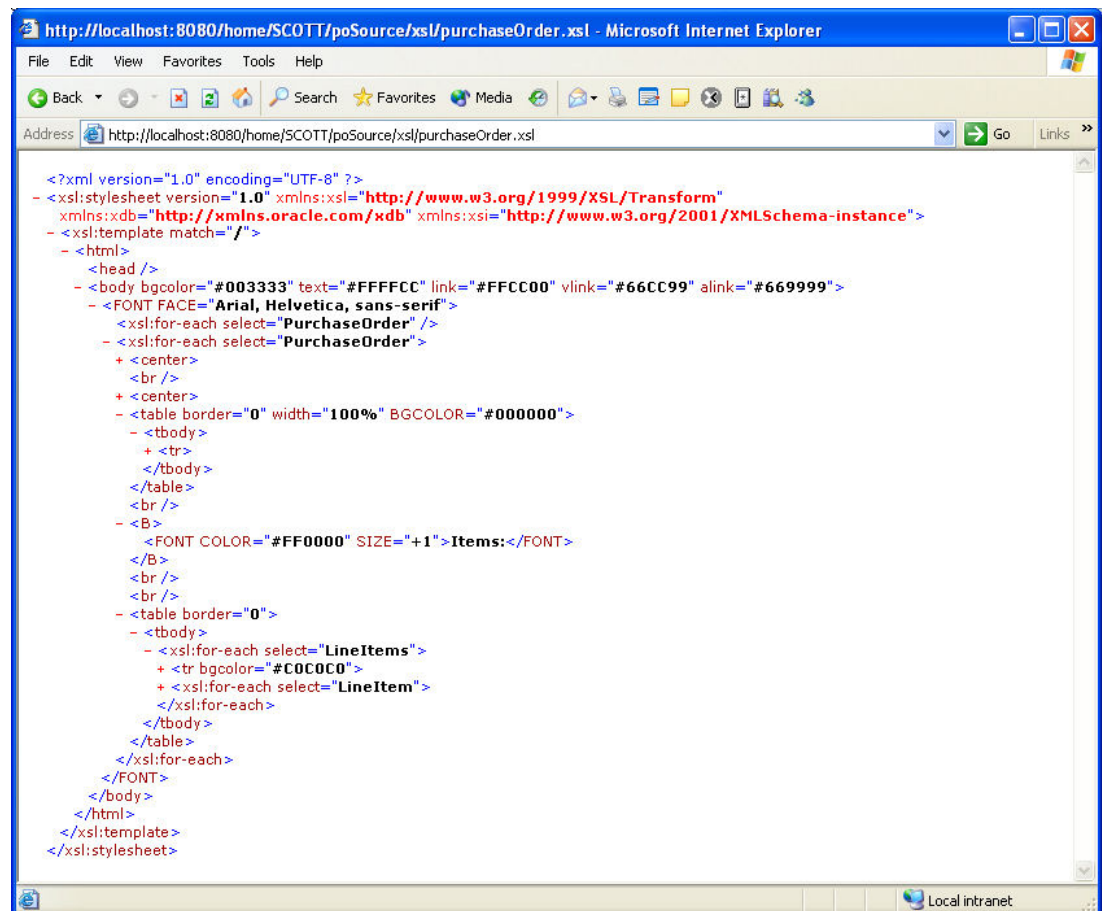


Figure LI. Standard W3C XSL style sheet stored in Oracle XML DB repository.

KEY POINTS:

- The style sheet is an absolutely standard XSL style sheet. There is nothing Oracle XML DB specific about the styles sheet
- The style sheet is simply stored as non-schema based XML inside the Oracle XML DB repository.

XSL TRANSFORMATION WITH THE DBUri SERVLET

The XML accessed by the *DBUri Servlet* can be transformed using the XSLT processor built into Oracle XML DB. This allows the XML generated by the *DBUri Servlet* to be presented to in a more user-friendly manner, such as HTML.

Style sheet processing is initiated by adding a transform parameter to the URL passed to the *DBUri Servlet*. The style sheet is specified using a URI that references a style sheet stored in the database. The style sheet is applied directly to the generated XML before the document is returned to the client. When using the *DBUri Servlet* to perform XSLT processing it is good practice to use the *contenttype* parameter to explicitly specify the mime type of the generated output.

The following screen shot shows how an XSL Transformation can be applied to XML content generated by the *DBUri Servlet*. In this example the complete URL is

```
http://localhost:8080/oradb/SCOTT/PURCHASEORDER/ROW/PurchaseOrder[Reference="SBELL-2003030912333601PDT"]?contenttype=text/html&transform=/home/SCOTT/poSource/xsl/purchaseOrder.xsl
```

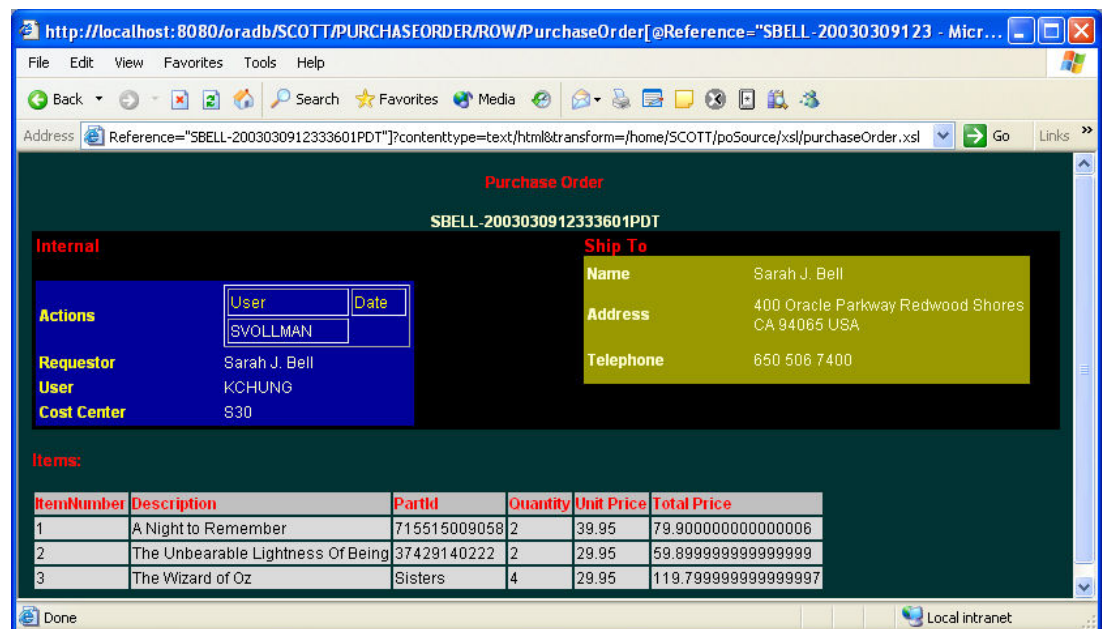


Figure LII.Database XSL transformation of a PurchaseOrder using the DBUri Servlet.

KEY POINTS:

- The URL passed to the *DBUri Servlet* will retrieve one PurchaseOrder document from the PURCHASEORDER table.
- The *DBUri Servlet* will apply the style sheet */home/SCOTT/poSource/xsl/purchaseOrder.xsl* to the PurchaseOrder document before returning the document to the browser. This style sheet will transform the XML into the HTML.
- The *contenttype* parameter is used to ensure that the mime type of the generated document is set to text/html.
- The XSLT processing is performed inside the database using the Oracle XML DB XSLT processor.

The following screen shot show how a combination of XMLType views and the *DBUri Servlet* can be used to simplify the process of exposing relational data as HTML. First a persistent XML representation of the relational content is created by using the SQL/XML operators to create an XMLType view. Next the *DBUri Servlet* is used to apply an XSL Transformation to the XMLType view, producing HTML that can be viewed directly from a web browser.

DEPARTMENT	LOCATION	EMPLOYEES
Administration	2004 Charade Rd Seattle Washington 98199 United States of America	Jennifer Whalen Administration Assistant 4400 17-SEP-87
Marketing	147 Spadina Ave Toronto Ontario M5V 2L7 Canada	Michael Hartstein Marketing Manager 13000 17-FEB-96
		Pat Fay Marketing Representative 6000 17-AUG-97
Purchasing	2004 Charade Rd Seattle Washington 98199 United States of America	Den Raphaely Purchasing Manager 11000 07-DEC-94
		Alexander Khoo Purchasing Clerk 3100 18-MAY-95
		Shelli Baida Purchasing Clerk 2900 24-DEC-97
		Sigal Tobias Purchasing Clerk 2800 24-JUL-97
		Guy Himuro Purchasing Clerk 2600 15-NOV-98
		Karen Colmenares Purchasing Clerk 2500 10-AUG-99
Human Resources	8204 Arthur St London United Kingdom	Susan Mavris Human Resources Representative 6500 07-JUN-94
Shipping	2011 Interiors Blvd South San Francisco California 99236 United States of America	Matthew Weiss Stock Manager 8000 18-JUL-96
		Adam Frip Stock Manager 8200 10-APR-97
		Peyam Kaufing Stock Manager 7900 01-MAY-95
		Shanta Vollman Stock Manager 6500 10-OCT-97
		Kevin Mourgos Stock Manager 5800 16-NOV-99
		Winston Taylor Shipping Clerk 3200 24-JAN-98
		Jean Fleaur Shipping Clerk 3100 23-FEB-98
		Martha Sullivan Shipping Clerk 2500 21-JUN-99

Figure LIII. Database XSL transformation of relational content using the *DBUri Servlet*.

KEY POINTS:

- Oracle XML DB makes it possible to expose the content of relational tables as an HTML document without very little code.
- All that was required was a simple XMLType view, based on SQL/XML operators, an industry standard XSL style sheet and the *DBUri Servlet*.

SUMMARY

Oracle XML DB adds native support for the emerging XML standards to the popular Oracle database, brings SQL and XML processing together, and introduces a number of innovations needed for efficient storage and retrieval of XML.

More information on XML DB can be found at the following Locations:

The Oracle Technology Network (OTN) page for XML DB:
<http://otn.oracle.com/tech/xml/xmlldb/content.html>,

The Oracle Technology Network (OTN) page for XML:
<http://otn.oracle.com/tech/xml/content.html>

The Oracle Documentation, in the book titled '[XML Database Developer's Guide - Oracle XML DB](#)'.



Oracle XML DB White Paper

May 2005

Author: Mark Drake

Contributing Authors: Sandeepan Banerjee, Geoff Lee

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
www.oracle.com

Oracle Corporation provides the software
that powers the internet.

Oracle is a registered trademark of Oracle Corporation. Various
product and service names referenced herein may be trademarks
of Oracle Corporation. All other product and service names
mentioned may be trademarks of their respective owners.
Copyright © 2005, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.