5

Query and Update of XML Data

There are many ways for applications to query and update XML data that is in Oracle Database, both XML schema-based and non-schema-based.

- Using XQuery with Oracle XML DB
  XQuery is a very general and expressive language, and SQL/XML functions `XMLQuery`, `XMLTable`, `XMLExists`, and `XMLCast` combine that power of expression and computation with the strengths of SQL.
- Querying XML Data Using SQL and PL/SQL
  You can query XML data from `XMLType` columns and tables in various ways.
- Using the SQL*Plus XQUERY Command
  You can evaluate an XQuery expression using the SQL*Plus `XQUERY` command.
- Using XQuery with PL/SQL, JDBC, and ODP.NET to Access Database Data
  You can use XQuery with the Oracle APIs for PL/SQL, JDBC, and Oracle Data Provider for .NET (ODP.NET).
- Updating XML Data
  There are several ways you can use Oracle XML DB features to update XML data, whether it is transient or stored in database tables.
- Performance Tuning for XQuery
  A SQL query that involves XQuery expressions can often be automatically rewritten (optimized) in one or more ways. This optimization is referred to as **XML query rewrite** or optimization. When this happens, the XQuery expression is, in effect, evaluated directly against the XML document without constructing a DOM in memory.

See Also:

- Overview of How To Use Oracle XML DB for `XMLType` storage recommendations
- XML Schema Storage and Query: Basic for how to work with XML schema-based `XMLType` tables and columns
- XQuery and Oracle XML DB for information about updating XML data using XQuery Update

Using XQuery with Oracle XML DB

XQuery is a very general and expressive language, and SQL/XML functions `XMLQuery`, `XMLTable`, `XMLExists`, and `XMLCast` combine that power of expression and computation with the strengths of SQL.

You typically use XQuery with Oracle XML DB in the following ways. The examples here are organized to reflect these different uses.

- Query XML data in Oracle XML DB Repository.

See Querying XML Data in Oracle XML DB Repository Using XQuery.

- Query a relational table or view as if it were XML data. To do this, you use XQuery function `fn:collection`, passing as argument a URI that uses the URI-scheme name `oradb` together with the database location of the data.

  See Querying Relational Data Using XQuery and URI Scheme oradb.

- Query `XMLType` data, possibly decomposing the resulting XML into relational data using function `XMLTable`.

  See Querying XMLType Data Using XQuery.

Example 5-1 creates Oracle XML DB Repository resources that are used in some of the other examples in this chapter.

Example 5-1 Creating Resources for Examples

```
DECLARE
  res BOOLEAN;
  empsxmlstring VARCHAR2(300):=
    '<?xml version="1.0"?>
     <emps>
       <emp empno="1" deptno="10" ename="John" salary="21000"/>
       <emp empno="2" deptno="10" ename="Jack" salary="310000"/>
       <emp empno="3" deptno="20" ename="Jill" salary="100001"/>
     </emps>';
  empsxmlnsstring VARCHAR2(300):=
    '<?xml version="1.0"?>
     <emps xmlns="http://example.com">
       <emp empno="1" deptno="10" ename="John" salary="21000"/>
       <emp empno="2" deptno="10" ename="Jack" salary="310000"/>
       <emp empno="3" deptno="20" ename="Jill" salary="100001"/>
     </emps>';
  deptsxmlstring VARCHAR2(300):=
    '<?xml version="1.0"?>
     <depts>
       <dept deptno="10" dname="Administration"/>
       <dept deptno="20" dname="Marketing"/>
       <dept deptno="30" dname="Purchasing"/>
     </depts>';
BEGIN
  res := DBMS_XDB_REPOS.createResource('/public/emps.xml',   empsxmlstring);
  res := DBMS_XDB_REPOS.createResource('/public/empsns.xml', empsxmlnsstring);
  res := DBMS_XDB_REPOS.createResource('/public/depts.xml',  deptsxmlstring);
END;
/
```

- XQuery Sequences Can Contain Data of Any XQuery Type
  XQuery is a general *sequence*-manipulation language. Its expressions and their results are not necessarily XML data. An XQuery sequence can contain items of any XQuery type, which includes numbers, strings, Boolean values, dates, and various types of XML node (`document-node()`, `element()`, `attribute()`, `text()`, `namespace()`, and so on).
- Querying XML Data in Oracle XML DB Repository Using XQuery
  Examples are presented that use XQuery with XML data in Oracle XML DB Repository. You use XQuery functions `fn:doc` and `fn:collection` to query file and folder resources in the repository, respectively.
- Querying Relational Data Using XQuery and URI Scheme oradb
  Examples are presented that use XQuery to query relational table or view data as if it were XML data. The examples use XQuery function `fn:collection`, passing as argument a URI that uses the URI-scheme name `oradb` together with the database location of the data.

- Querying XMLType Data Using XQuery
  Examples are presented that use XQuery to query `XMLType` data.
- Using Namespaces with XQuery
  You can use the XQuery `declare namespace` declaration in the prolog of an XQuery expression to define a namespace prefix. You can use `declare` **`default`** `namespace` to establish the namespace as the default namespace for the expression.

XQuery Sequences Can Contain Data of Any XQuery Type

XQuery is a general *sequence*-manipulation language. Its expressions and their results are not necessarily XML data. An XQuery sequence can contain items of any XQuery type, which includes numbers, strings, Boolean values, dates, and various types of XML node (`document-node()`, `element()`, `attribute()`, `text()`, `namespace()`, and so on).

Example 5-2 provides a sampling. It applies SQL/XML function `XMLQuery` to an XQuery sequence that contains items of several different kinds:

- an integer literal: `1`
- a arithmetic expression: `2 + 3`
- a string literal: `"a"`
- a sequence of integers: `100 to 102`
- a constructed XML element node: `<A>33</A>`

Example 5-2 also shows construction of a sequence using the comma operator (`,`) and parentheses (`(`, `)`) for grouping.

The sequence expression `100 to 102` evaluates to the sequence `(100, 101, 102)`, so the argument to `XMLQuery` here is a sequence that contains a nested sequence. The sequence argument is automatically flattened, as is always the case for XQuery sequences. The argument is, in effect, `(1, 5, "a", 100, 101, 102, <A>33</A>)`.

Example 5-2 XMLQuery Applied to a Sequence of Items of Different Types

```
SELECT XMLQuery('(1, 2 + 3, "a", 100 to 102, <A>33</A>)'
                RETURNING CONTENT) AS output
  FROM DUAL;

OUTPUT
--------------------------
1 5 a 100 101 102<A>33</A>

1 row selected.
```

Querying XML Data in Oracle XML DB Repository Using XQuery

Examples are presented that use XQuery with XML data in Oracle XML DB Repository. You use XQuery functions `fn:doc` and `fn:collection` to query file and folder resources in the repository, respectively.

The examples here use XQuery function `fn:doc` to obtain a repository file that contains XML data, and then bind XQuery variables to parts of that data using `for` and `let` FLWOR-expression clauses.

Example 5-3 queries two XML-document resources in Oracle XML DB Repository: `/public/emps.xml` and `/public/depts.xml`. It illustrates the use of `fn:doc` and each of the possible FLWOR-expression clauses.

Example 5-4 also uses each of the FLWOR-expression clauses. It shows the use of XQuery functions `doc`, `count`, `avg`, and `integer`, which are in the namespace for built-in XQuery functions, `http://www.w3.org/2003/11/xpath-functions`. This namespace is bound to the prefix `fn`.

Example 5-3 FLOWR Expression Using for, let, order by, where, and return

```
SELECT XMLQuery('for $e in doc("/public/emps.xml")/emps/emp
                 let $d :=
                   doc("/public/depts.xml")//dept[@deptno = $e/@deptno]/@dname
                 where $e/@salary > 100000
                 order by $e/@empno
                 return <emp ename="{$e/@ename}" dept="{$d}"/>'
                RETURNING CONTENT) FROM DUAL;

XMLQUERY('FOR$EINDOC("/PUBLIC/EMPS.XML")/EMPS/EMPLET$D:=DOC("/PUBLIC/DEPTS.XML")
--------------------------------------------------------------------------------
<emp ename="Jack" dept="Administration"></emp><emp ename="Jill" dept="Marketing"
></emp>

1 row selected.
```

In this example, the various FLWOR clauses perform these operations:

- **`for`** iterates over the `emp` elements in `/public/emps.xml`, binding variable `$e` to the value of each such element, in turn. That is, it iterates over a general list of employees, binding `$e` to each employee.
- **`let`** binds variable `$d` to a *sequence* consisting of all of the values of `dname` attributes of those `dept` elements in `/public/emps.xml` whose `deptno` attributes have the same value as the `deptno` attribute of element `$e` (this is a join operation). That is, it binds `$d` to the names of all of the departments that have the same department number as the department of employee `$e`. (It so happens that the `dname` value is unique for each `deptno` value in `depts.xml`.) Unlike `for`, `let` never iterates over values; `$d` is bound only once in this example.
- Together, `for` and `let` produce a stream of tuples (`$e`, `$d`), where `$e` represents an employee and `$d` represents the names of all of the departments to which that employee belongs —in this case, the unique name of the employee's unique department.
- **`where`** filters this tuple stream, keeping only tuples with employees whose salary is greater than 100,000.
- **`order by`** sorts the filtered tuple stream by employee number, `empno` (in ascending order, by default).
- **`return`** constructs `emp` elements, one for each tuple. Attributes `ename` and `dept` of these elements are constructed using attribute `ename` from the input and `$d`, respectively. The element and attribute names `emp` and `ename` in the output have no necessary connection with the same names in the input document `emps.xml`.

Example 5-4 FLOWR Expression Using Built-In Functions

```
SELECT XMLQuery('for $d in fn:doc("/public/depts.xml")/depts/dept/@deptno
                 let $e := fn:doc("/public/emps.xml")/emps/emp[@deptno = $d]
                 where fn:count($e) > 1
                 order by fn:avg($e/@salary) descending
                 return
                   <big-dept>{$d,
                              <headcount>{fn:count($e)}</headcount>,
                              <avgsal>{xs:integer(fn:avg($e/@salary))}</avgsal>}
                   </big-dept>'
               RETURNING CONTENT) FROM DUAL;

XMLQUERY('FOR$DINFN:DOC("/PUBLIC/DEPTS.XML")/DEPTS/DEPT/@DEPTNOLET$E:=FN:DOC("/P
--------------------------------------------------------------------------------
<big-dept deptno="10"><headcount>2</headcount><avgsal>165500</avgsal></big-dept>

1 row selected.
```

In this example, the various FLWOR clauses perform these operations:

- **`for`** iterates over `deptno` attributes in input document `/public/depts.xml`, binding variable `$d` to the value of each such attribute, in turn.
- **`let`** binds variable `$e` to a sequence consisting of all of the `emp` elements in input document `/public/emps.xml` whose `deptno` attributes have value `$d` (this is a join operation).
- Together, `for` and `let` produce a stream of tuples (`$d`, `$e`), where `$d` represents a department number and `$e` represents the set of employees in that department.
- **`where`** filters this tuple stream, keeping only tuples with more than one employee.
- **`order by`** sorts the filtered tuple stream by average salary in descending order. The average is computed by applying XQuery function `avg` (in namespace `fn`) to the values of attribute `salary`, which is attached to the `emp` elements of `$e`.
- **`return`** constructs `big-dept` elements, one for each tuple produced by `order by`. The `text()` node of `big-dept` contains the department number, bound to `$d`. A `headcount` child element contains the number of employees, bound to `$e`, as determined by XQuery function `count`. An `avgsal` child element contains the computed average salary.

Related Topics

- XQuery Functions fn:doc, fn:collection, and fn:doc-available
  Oracle XML DB supports XQuery functions `fn:doc`, `fn:collection`, and `fn:doc-available` for all resources in Oracle XML DB Repository.

Querying Relational Data Using XQuery and URI Scheme oradb

Examples are presented that use XQuery to query relational table or view data as if it were XML data. The examples use XQuery function `fn:collection`, passing as argument a URI that uses the URI-scheme name `oradb` together with the database location of the data.

Example 5-5 uses Oracle XQuery function `fn:collection` in a FLWOR expression to query two relational tables, `regions` and `countries`. Both tables belong to sample database schema `HR`. The example also passes scalar SQL value `Asia` to XQuery variable `$regionname`. Any SQL expression can be evaluated to produce a value passed to XQuery using `PASSING`. In this

case, the value comes from a SQL*Plus variable, `REGION`. You must cast the value to the scalar SQL data type expected, in this case, `VARCHAR2(40)`.

In Example 5-5, the various FLWOR clauses perform these operations:

- **`for`** iterates over sequences of XML elements returned by calls to `fn:collection`. In the first call, each element corresponds to a row of relational table `hr.regions` and is bound to variable `$i`. Similarly, in the second call to `fn:collection`, `$j` is bound to successive rows of table `hr.countries`. Since `regions` and `countries` are not `XMLType` tables, the top-level element corresponding to a row in each table is `ROW` (a wrapper element). Iteration over the row elements is unordered.
- **`where`** filters the rows from both tables, keeping only those pairs of rows whose `region_id` is the same for each table (it performs a join on `region_id`) and whose `region_name` is `Asia`.
- **`return`** returns the filtered rows from table `hr.countries` as an XML document containing XML fragments with `ROW` as their top-level element.

Example 5-6 uses `fn:collection` within nested FLWOR expressions to query relational data.

In Example 5-6, the various FLWOR clauses perform these operations:

- The outer **`for`** iterates over the sequence of XML elements returned by `fn:collection`: each element corresponds to a row of relational table `oe.warehouses` and is bound to variable `$i`. Since `warehouses` is not an `XMLType` table, the top-level element corresponding to a row is `ROW`. The iteration over the row elements is unordered.
- The inner **`for`** iterates, similarly, over a sequence of XML elements returned by `fn:collection`: each element corresponds to a row of relational table `hr.locations` and is bound to variable `$j`.
- **`where`** filters the tuples (`$i`, `$j`), keeping only those whose `location_id` child is the same for `$i` and `$j` (it performs a join on `location_id`).
- The inner **`return`** constructs an XQuery sequence of elements `STREET_ADDRESS`, `CITY`, and `STATE_PROVINCE`, all of which are children of locations-table `ROW` element `$j`; that is, they are the values of the locations-table columns of the same name.
- The outer **`return`** wraps the result of the inner `return` in a `Location` element, and wraps that in a `Warehouse` element. It provides the `Warehouse` element with an `id` attribute whose value comes from the `warehouse_id` column of table `warehouses`.

Example 5-7 uses SQL/XML function `XMLTable` to decompose the result of an XQuery query to produce virtual relational data. The XQuery expression used in this example is identical to the one used in Example 5-6; the result of evaluating the XQuery expression is a sequence of `Warehouse` elements. Function `XMLTable` produces a virtual relational table whose rows are those `Warehouse` elements. More precisely, in this example the value of pseudocolumn `COLUMN_VALUE` for each virtual-table row is an XML fragment (of type `XMLType`) with a single `Warehouse` element.

> **See Also:**
>
> - Example 5-41 for the execution plan of Example 5-6
> - Example 5-42 for the execution plan of Example 5-7

Example 5-5 Querying Relational Data as XML Using XMLQuery

```
DEFINE REGION = 'Asia'
SELECT XMLQuery('for $i in fn:collection("oradb:/HR/REGIONS"),
                     $j in fn:collection("oradb:/HR/COUNTRIES")
                   where $i/ROW/REGION_ID = $j/ROW/REGION_ID
                     and $i/ROW/REGION_NAME = $regionname
                   return $j'
                 PASSING CAST('&REGION' AS VARCHAR2(40)) AS "regionname"
                 RETURNING CONTENT) AS asian_countries
  FROM DUAL;
```

This produces the following result. (The result is shown here pretty-printed, for clarity.)

```
ASIAN_COUNTRIES
----------------------------------------
<ROW>
  <COUNTRY_ID>AU</COUNTRY_ID>
  <COUNTRY_NAME>Australia</COUNTRY_NAME>
  <REGION_ID>3</REGION_ID>
</ROW>
<ROW>
  <COUNTRY_ID>CN</COUNTRY_ID>
  <COUNTRY_NAME>China</COUNTRY_NAME>
  <REGION_ID>3</REGION_ID>
</ROW>
<ROW>
  <COUNTRY_ID>HK</COUNTRY_ID>
  <COUNTRY_NAME>HongKong</COUNTRY_NAME>
  <REGION_ID>3</REGION_ID>
</ROW>
<ROW>
  <COUNTRY_ID>IN</COUNTRY_ID>
  <COUNTRY_NAME>India</COUNTRY_NAME>
  <REGION_ID>3</REGION_ID>
</ROW>
<ROW>
  <COUNTRY_ID>JP</COUNTRY_ID>
  <COUNTRY_NAME>Japan</COUNTRY_NAME>
  <REGION_ID>3</REGION_ID>
</ROW>
<ROW>
  <COUNTRY_ID>SG</COUNTRY_ID>
  <COUNTRY_NAME>Singapore</COUNTRY_NAME>
  <REGION_ID>3</REGION_ID>
</ROW>

1 row selected.
```

Example 5-6 Querying Relational Data as XML Using a Nested FLWOR Expression

```
CONNECT hr
Enter password: password

Connected.
```

```
GRANT SELECT ON LOCATIONS TO OE
/
CONNECT oe
Enter password: password

Connected.

SELECT XMLQuery(
         'for $i in fn:collection("oradb:/OE/WAREHOUSES")/ROW
          return <Warehouse id="{$i/WAREHOUSE_ID}">
                   <Location>
                     {for $j in fn:collection("oradb:/HR/LOCATIONS")/ROW
                      where $j/LOCATION_ID eq $i/LOCATION_ID
                      return ($j/STREET_ADDRESS, $j/CITY, $j/STATE_PROVINCE)}
                   </Location>
                 </Warehouse>'
         RETURNING CONTENT) FROM DUAL;
```

This query is an example of using nested FLWOR expressions. It accesses relational table `warehouses`, which is in sample database schema `oe`, and relational table `locations`, which is in sample database schema `HR`. To run this example as user `oe`, you must first connect as user `hr` and grant permission to user `oe` to perform `SELECT` operations on table `locations`.

This produces the following result. (The result is shown here pretty-printed, for clarity.)

```
XMLQUERY('FOR$IINFN:COLLECTION("ORADB:/OE/WAREHOUSES")/ROWRETURN
----------------------------------------------------------------
<Warehouse id="1">
  <Location>
    <STREET_ADDRESS>2014 Jabberwocky Rd</STREET_ADDRESS>
    <CITY>Southlake</CITY>
    <STATE_PROVINCE>Texas</STATE_PROVINCE>
  </Location>
</Warehouse>
<Warehouse id="2">
  <Location>
    <STREET_ADDRESS>2011 Interiors Blvd</STREET_ADDRESS>
    <CITY>South San Francisco</CITY>
    <STATE_PROVINCE>California</STATE_PROVINCE>
  </Location>
</Warehouse>
<Warehouse id="3">
  <Location>
    <STREET_ADDRESS>2007 Zagora St</STREET_ADDRESS>
    <CITY>South Brunswick</CITY>
    <STATE_PROVINCE>New Jersey</STATE_PROVINCE>
  </Location>
</Warehouse>
<Warehouse id="4">
  <Location>
    <STREET_ADDRESS>2004 Charade Rd</STREET_ADDRESS>
    <CITY>Seattle</CITY>
    <STATE_PROVINCE>Washington</STATE_PROVINCE>
  </Location>
</Warehouse>
```

```
<Warehouse id="5">
  <Location>
    <STREET_ADDRESS>147 Spadina Ave</STREET_ADDRESS>
    <CITY>Toronto</CITY>
    <STATE_PROVINCE>Ontario</STATE_PROVINCE>
  </Location>
</Warehouse>
<Warehouse id="6">
  <Location>
    <STREET_ADDRESS>12-98 Victoria Street</STREET_ADDRESS>
    <CITY>Sydney</CITY>
    <STATE_PROVINCE>New South Wales</STATE_PROVINCE>
  </Location>
</Warehouse>
<Warehouse id="7">
  <Location>
    <STREET_ADDRESS>Mariano Escobedo 9991</STREET_ADDRESS>
    <CITY>Mexico City</CITY>
    <STATE_PROVINCE>Distrito Federal,</STATE_PROVINCE>
  </Location>
</Warehouse>
<Warehouse id="8">
  <Location>
    <STREET_ADDRESS>40-5-12 Laogianggen</STREET_ADDRESS>
    <CITY>Beijing</CITY>
  </Location>
</Warehouse>
<Warehouse id="9">
  <Location>
    <STREET_ADDRESS>1298 Vileparle (E)</STREET_ADDRESS>
    <CITY>Bombay</CITY>
    <STATE_PROVINCE>Maharashtra</STATE_PROVINCE>
  </Location>
</Warehouse>

1 row selected.
```

Example 5-7 Querying Relational Data as XML Using XMLTable

```
SELECT *
  FROM XMLTable(
         'for $i in fn:collection("oradb:/OE/WAREHOUSES")/ROW
          return <Warehouse id="{$i/WAREHOUSE_ID}">
                   <Location>
                     {for $j in fn:collection("oradb:/HR/LOCATIONS")/ROW
                      where $j/LOCATION_ID eq $i/LOCATION_ID
                      return ($j/STREET_ADDRESS, $j/CITY, $j/STATE_PROVINCE)}
                   </Location>
                 </Warehouse>');
```

This produces the same result as Example 5-6, except that each `Warehouse` element is output as a separate row, instead of all `Warehouse` elements being output together in a single row.

```
COLUMN_VALUE
------------------------------------------------------------
```

```
<Warehouse id="1">
  <Location>
    <STREET_ADDRESS>2014 Jabberwocky Rd</STREET_ADDRESS>
    <CITY>Southlake</CITY>
    <STATE_PROVINCE>Texas</STATE_PROVINCE>
  </Location>
</Warehouse>
<Warehouse id="2">
  <Location>
    <STREET_ADDRESS>2011 Interiors Blvd</STREET_ADDRESS>
    <CITY>South San Francisco</CITY>
    <STATE_PROVINCE>California</STATE_PROVINCE>
  </Location>
</Warehouse>
. . .

9 rows selected.
```

Querying XMLType Data Using XQuery

Examples are presented that use XQuery to query `XMLType` data.

The query in Example 5-8 passes an `XMLType` column, `warehouse_spec`, as *context* item to XQuery, using function `XMLQuery` with the `PASSING` clause. It constructs a `Details` element for each of the warehouses whose area is greater than 80,000: `/Warehouse/Area > 80000`.

In Example 5-8, function `XMLQuery` is applied to the `warehouse_spec` column in each row of table `warehouses`. The various FLWOR clauses perform these operations:

- **`for`** iterates over the `Warehouse` elements in each row of column `warehouse_spec` (the passed context item): each such element is bound to variable `$i`, in turn. The iteration is unordered.
- **`where`** filters the `Warehouse` elements, keeping only those whose `Area` child has a value greater than 80,000.
- **`return`** constructs an XQuery sequence of `Details` elements, each of which contains a `Docks` and a `Rail` child elements. The `num` attribute of the constructed `Docks` element is set to the `text()` value of the `Docks` child of `Warehouse`. The `text()` content of `Rail` is set to `true` or `false`, depending on the value of the `RailAccess` attribute of element `Warehouse`.

The `SELECT` statement in Example 5-8 applies to each row in table `warehouses`. The `XMLQuery` expression returns the *empty sequence* for those rows that do not match the XQuery expression. Only the warehouses in New Jersey and Seattle satisfy the XQuery query, so they are the only warehouses for which `<Details>...</Details>` is returned.

Example 5-9 uses SQL/XML function `XMLTable` to query an `XMLType` table, `oe.purchaseorder`, which contains XML Schema-based data. It uses the `PASSING` clause to provide the `purchaseorder` table as the context item for the XQuery-expression argument to `XMLTable`. Pseudocolumn `COLUMN_VALUE` of the resulting virtual table holds a constructed element, `A10po`, which contains the `Reference` information for those purchase orders whose `CostCenter` element has value `A10` and whose `User` element has value `SMCCAIN`. The query performs a join between the virtual table and database table `purchaseorder`.

The `PASSING` clause of function `XMLTable` passes the `OBJECT_VALUE` of `XMLType` table `purchaseorder`, to serve as the XPath context. The `XMLTable` expression thus *depends* on the `purchaseorder` table. Because of this, table `purchaseorder` must appear *before* the `XMLTable`

expression in the `FROM` list. This is a general requirement in any situation involving data dependence.

> **Note:**
>
> Whenever a `PASSING` clause refers to a column of an `XMLType` table in a query, that table ***must appear before*** the `XMLTable` expression in the query `FROM` list. This is because the `XMLTable` expression *depends* on the `XMLType` table — a *left lateral* (correlated) join is needed, to ensure a one-to-many (1:N) relationship between the `XMLType` table row accessed and the rows generated from it by `XMLTable`.

Example 5-10 is similar to Example 5-9 in its effect. It uses `XMLQuery`, instead of `XMLTable`, to query `oe.purchaseorder`. These two examples differ in their treatment of the empty sequences returned by the XQuery expression. In Example 5-9, these empty sequences are not joined with the `purchaseorder` table, so the overall SQL-query result set has only ten rows. In Example 5-10, these empty sequences are part of the overall result set of the SQL query, which contains 132 rows, one for each of the rows in table `purchaseorder`. All but ten of those rows are empty, and show up in the output as empty lines. To save space here, those empty lines have been removed.

> **See Also:**
>
> Example 5-43 for the execution plan of Example 5-10

Example 5-11 uses `XMLTable` clauses `PASSING` and `COLUMNS`. The XQuery expression iterates over top-level `PurchaseOrder` elements, constructing a `PO` element for each purchase order with cost center `A10`. The resulting `PO` elements are then passed to `XMLTable` for processing.

In Example 5-11, data from the children of `PurchaseOrder` is used to construct the children of `PO`, which are `Ref`, `Type`, and `Name`. The content of `Type` is taken from the content of `/PurchaseOrder/SpecialInstructions`, but the classes of `SpecialInstructions` are divided up differently for `Type`.

Function `XMLTable` breaks up the result of XQuery evaluation, returning it as three `VARCHAR2` columns of a virtual table: `poref`, `priority`, and `contact`. The `DEFAULT` clause is used to supply a default `priority` of `Regular`.

Example 5-11 does not use the clause `RETURNING SEQUENCE BY REF`, which means that the XQuery sequence returned and then used by the `COLUMNS` clause is passed by *value*, not by reference. That is, a copy of the targeted nodes is returned, not a reference to the actual nodes.

When the returned sequence is passed by value, the columns specified in a `COLUMNS` clause cannot refer to any data that is not in that returned copy. In particular, they cannot refer to data that *precedes* the targeted nodes in the source data.

To be able to refer to an arbitrary part of the source data from column specifications in a `COLUMNS` clause, you need to use the clause `RETURNING SEQUENCE BY REF`, which causes the sequence resulting from the XQuery expression to be returned by *reference*.

Example 5-12 shows the use of clause `RETURNING SEQUENCE BY REF`, which allows column `reference` to refer to a node that is outside the nodes targeted by the XQuery expression. Because the sequence of `LineItem` nodes is returned by reference, the code has access to the complete tree of nodes, so it can navigate upward and then back down to node `Reference`.

Clause `RETURNING SEQUENCE BY REF` lets you specify that the result of evaluating the top-level XQuery expression used to generate rows for `XMLTable` be returned by reference. The same kind of choice is available for the result of evaluating a `PATH` expression in a `COLUMNS` clause. To specify that such a result be returned by reference you use `XMLType (SEQUENCE) BY REF` as the column data type.

Example 5-13 illustrates this. It chains together two `XMLTable` tables, `t1` and `t2`, returning XML data from the source document by reference:

- For column `reference` of the top-level table, `t1`, because it corresponds to a node outside element `LineItem` (just as in Example 5-12)
- For column `part` of table `t1`, because it is passed to table `t2`, whose column `item` targets data outside node `Part`

In table `t1`, the type used for column `part` is `XMLType (SEQUENCE) BY REF`, so that the `part` data is a reference to the source data targeted by its `PATH` expression, `LineItem/Part`. This is needed because the `PATH` expression for column `item` in table `t2` targets attribute `ItemNumber` of the parent of element `Part`, `LineItem`. Without specifying that `part` is a reference, it would be a copy of just the `Part` element, so that using `PATH` expression `../@ItemNumber` would raise an error.

Example 5-14 uses SQL/XML function `XMLTable` to break up the XML data in an `XMLType` collection element, `LineItem`, into separate columns of a virtual table.

> **See Also:**
>
> - Example 5-44 for the execution plan of Example 5-14
> - Creating a Relational View over XML: Mapping XML Nodes to Columns, for an example of applying `XMLTable` to multiple document levels (multilevel chaining)

Example 5-8 Querying an XMLType Column Using XMLQuery PASSING Clause

```
SELECT warehouse_name,
       XMLQuery(
         'for $i in /Warehouse
          where  $i/Area > 80000
          return <Details>
                   <Docks num="{$i/Docks}"/>
                   <Rail>{if ($i/RailAccess = "Y")
                          then "true"
                          else "false"}
                   </Rail>
                 </Details>'
        PASSING warehouse_spec RETURNING CONTENT) big_warehouses
  FROM oe.warehouses;
```

This produces the following output:

```
WAREHOUSE_NAME
--------------
BIG_WAREHOUSES
--------------
Southlake, Texas


San Francisco


New Jersey
<Details><Docks num=""></Docks><Rail>false</Rail></Details>

Seattle, Washington
<Details><Docks num="3"></Docks><Rail>true</Rail></Details>

Toronto


Sydney


Mexico City


Beijing


Bombay


9 rows selected.
```

Example 5-9 Using XMLTABLE with XML Schema-Based Data

```
SELECT xtab.COLUMN_VALUE
  FROM purchaseorder, XMLTable('for $i in /PurchaseOrder
                                where $i/CostCenter eq "A10"
                                  and $i/User eq "SMCCAIN"
                                return <A10po pono="{$i/Reference}"/>'
                               PASSING OBJECT_VALUE) xtab;

COLUMN_VALUE
--------------------------------------------------
<A10po pono="SMCCAIN-20021009123336151PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336341PDT"></A10po>
<A10po pono="SMCCAIN-20021009123337173PDT"></A10po>
<A10po pono="SMCCAIN-20021009123335681PDT"></A10po>
<A10po pono="SMCCAIN-20021009123335470PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336972PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336842PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336512PDT"></A10po>
<A10po pono="SMCCAIN-20021009123333894PDT"></A10po>
```

```
<A10po pono="SMCCAIN-20021009123337403PDT"></A10po>

10 rows selected.
```

Example 5-10 Using XMLQUERY with XML Schema-Based Data

```
SELECT XMLQuery('for $i in /PurchaseOrder
                 where $i/CostCenter eq "A10"
                   and $i/User eq "SMCCAIN"
                 return <A10po pono="{$i/Reference}"/>'
                PASSING OBJECT_VALUE
                RETURNING CONTENT)
  FROM purchaseorder;

XMLQUERY('FOR$IIN/PURCHASEORDERWHERE$I/COSTCENTEREQ
--------------------------------------------------
<A10po pono="SMCCAIN-20021009123336151PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336341PDT"></A10po>
<A10po pono="SMCCAIN-20021009123337173PDT"></A10po>
<A10po pono="SMCCAIN-20021009123335681PDT"></A10po>
<A10po pono="SMCCAIN-20021009123335470PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336972PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336842PDT"></A10po>
<A10po pono="SMCCAIN-20021009123336512PDT"></A10po>
<A10po pono="SMCCAIN-20021009123333894PDT"></A10po>
<A10po pono="SMCCAIN-20021009123337403PDT"></A10po>

132 rows selected.
```

Example 5-11 Using XMLTABLE with PASSING and COLUMNS Clauses

```
SELECT xtab.poref, xtab.priority, xtab.contact
  FROM purchaseorder,
       XMLTable('for $i in /PurchaseOrder
                 let $spl := $i/SpecialInstructions
                 where $i/CostCenter eq "A10"
                 return <PO>
                          <Ref>{$i/Reference}</Ref>
                          {if ($spl eq "Next Day Air" or $spl eq "Expedite") then
                             <Type>Fastest</Type>
                           else if ($spl eq "Air Mail") then
                             <Type>Fast</Type>
                           else ()}
                          <Name>{$i/Requestor}</Name>
                        </PO>'
                PASSING OBJECT_VALUE
                COLUMNS poref    VARCHAR2(20) PATH 'Ref',
                        priority VARCHAR2(8)  PATH 'Type' DEFAULT 'Regular',
                        contact  VARCHAR2(20) PATH 'Name') xtab;

POREF                PRIORITY CONTACT
-------------------- -------- --------------------
SKING-20021009123336 Fastest  Steven A. King
SMCCAIN-200210091233 Regular  Samuel B. McCain
SMCCAIN-200210091233 Fastest  Samuel B. McCain
```

```
JCHEN-20021009123337 Fastest  John Z. Chen
JCHEN-20021009123337 Regular  John Z. Chen
SKING-20021009123337 Regular  Steven A. King
SMCCAIN-200210091233 Regular  Samuel B. McCain
JCHEN-20021009123338 Regular  John Z. Chen
SMCCAIN-200210091233 Regular  Samuel B. McCain
SKING-20021009123335 Regular  Steven X. King
SMCCAIN-200210091233 Regular  Samuel B. McCain
SKING-20021009123336 Regular  Steven A. King
SMCCAIN-200210091233 Fast     Samuel B. McCain
SKING-20021009123336 Fastest  Steven A. King
SKING-20021009123336 Fastest  Steven A. King
SMCCAIN-200210091233 Regular  Samuel B. McCain
JCHEN-20021009123335 Regular  John Z. Chen
SKING-20021009123336 Regular  Steven A. King
JCHEN-20021009123336 Regular  John Z. Chen
SKING-20021009123336 Regular  Steven A. King
SMCCAIN-200210091233 Regular  Samuel B. McCain
SKING-20021009123337 Regular  Steven A. King
SKING-20021009123338 Fastest  Steven A. King
SMCCAIN-200210091233 Regular  Samuel B. McCain
JCHEN-20021009123337 Regular  John Z. Chen
JCHEN-20021009123337 Regular  John Z. Chen
JCHEN-20021009123337 Regular  John Z. Chen
SKING-20021009123337 Regular  Steven A. King
JCHEN-20021009123337 Regular  John Z. Chen
SKING-20021009123337 Regular  Steven A. King
SKING-20021009123337 Regular  Steven A. King
SMCCAIN-200210091233 Fast     Samuel B. McCain

32 rows selected.
```

Example 5-12 Using XMLTABLE with RETURNING SEQUENCE BY REF

```
SELECT t.*
  FROM purchaseorder,
       XMLTable('/PurchaseOrder/LineItems/LineItem' PASSING OBJECT_VALUE
                RETURNING SEQUENCE BY REF
                COLUMNS reference   VARCHAR2(30) PATH '../../Reference',
                        item        VARCHAR2(4)  PATH '@ItemNumber',
                        description VARCHAR2(45) PATH 'Description') t
  WHERE item = 5;

REFERENCE                      ITEM DESCRIPTION
------------------------------ ---- -----------------------------------
AMCEWEN-20021009123336171PDT   5    Coup De Torchon (Clean Slate)
AMCEWEN-20021009123336271PDT   5    The Unbearable Lightness Of Being
PTUCKER-20021009123336191PDT   5    The Scarlet Empress
PTUCKER-20021009123336291PDT   5    The Unbearable Lightness Of Being
SBELL-20021009123336231PDT     5    Black Narcissus
SBELL-20021009123336331PDT     5    Fishing With John 1 -3
SKING-20021009123336321PDT     5    The Red Shoes
SMCCAIN-20021009123336151PDT   5    Wages of Fear
SMCCAIN-20021009123336341PDT   5    The Most Dangerous Game
VJONES-20021009123336301PDT    5    Le Trou
```

```
10 rows selected.
```

Example 5-13 Using Chained XMLTABLE with Access by Reference

```
SELECT t1.reference, t2.id, t2.item
  FROM purchaseorder,
       XMLTable('/PurchaseOrder/LineItems' PASSING OBJECT_VALUE
                RETURNING SEQUENCE BY REF
                COLUMNS part XMLType (SEQUENCE) BY REF
                           PATH 'LineItem/Part',
                         reference VARCHAR2(30)
                           PATH '../Reference') t1,
       XMLTable('.' PASSING t1.part
                RETURNING SEQUENCE BY REF
                COLUMNS id   VARCHAR2(12) PATH '@Id',
                        item NUMBER       PATH '../@ItemNumber') t2;
```

Example 5-14 Using XMLTABLE to Decompose XML Collection Elements into Relational Data

```
SELECT lines.lineitem, lines.description, lines.partid,
       lines.unitprice, lines.quantity
  FROM purchaseorder,
       XMLTable('for $i in /PurchaseOrder/LineItems/LineItem
                 where $i/@ItemNumber >= 8
                  and $i/Part/@UnitPrice > 50
                  and $i/Part/@Quantity > 2
                 return $i'
                PASSING OBJECT_VALUE
                COLUMNS
                  lineitem    NUMBER       PATH '@ItemNumber',
                  description VARCHAR2(30) PATH 'Description',
                  partid      NUMBER       PATH 'Part/@Id',
                  unitprice   NUMBER       PATH 'Part/@UnitPrice',
                  quantity    NUMBER       PATH 'Part/@Quantity') lines;

LINEITEM DESCRIPTION                         PARTID UNITPRICE QUANTITY
-------- ------------------------------ ------------ --------- --------
      11 Orphic Trilogy                 37429148327        80        3
      22 Dreyer Box Set                 37429158425        80        4
      11 Dreyer Box Set                 37429158425        80        3
      16 Dreyer Box Set                 37429158425        80        3
       8 Dreyer Box Set                 37429158425        80        3
      12 Brazil                         37429138526        60        3
      18 Eisenstein: The Sound Years    37429149126        80        4
      24 Dreyer Box Set                 37429158425        80        3
      14 Dreyer Box Set                 37429158425        80        4
      10 Brazil                         37429138526        60        3
      17 Eisenstein: The Sound Years    37429149126        80        3
      16 Orphic Trilogy                 37429148327        80        4
      13 Orphic Trilogy                 37429148327        80        4
      10 Brazil                         37429138526        60        4
      12 Eisenstein: The Sound Years    37429149126        80        3
      12 Dreyer Box Set                 37429158425        80        4
```

```
                13 Dreyer Box Set                    37429158425        80        4

17 rows selected.
```

Using Namespaces with XQuery

You can use the XQuery `declare namespace` declaration in the prolog of an XQuery expression to define a namespace prefix. You can use `declare` **`default`** `namespace` to establish the namespace as the default namespace for the expression.

> **Note:**
>
> Be aware of the following pitfall, if you use SQL*Plus: If the semicolon (`;`) at the end of a namespace declaration terminates a line, SQL*Plus interprets it as a SQL terminator. To avoid this, you can do one of the following:
>
> - Place the text that follows the semicolon on the same line.
> - Place a comment, such as `(: :)`, after the semicolon, on the same line.
> - Turn off the recognition of the SQL terminator with SQL*Plus command `SET SQLTERMINATOR`.

Example 5-15 illustrates use of a namespace declaration in an XQuery expression.

An XQuery namespace declaration has no effect outside of its XQuery expression. To declare a namespace prefix for use in an `XMLTable` expression outside of the XQuery expression, use the `XMLNAMESPACES` clause. This clause also covers the XQuery expression argument to `XMLTable`, eliminating the need for a separate declaration in the XQuery prolog.

In Example 5-16, `XMLNAMESPACES` is used to define the prefix `e` for the namespace `http://example.com`. This namespace is used in the `COLUMNS` clause and the XQuery expression of the `XMLTable` expression.

Example 5-15 Using XMLQUERY with a Namespace Declaration

```
SELECT XMLQuery('declare namespace e = "http://example.com";
ERROR:
ORA-01756: quoted string not properly terminated

                for $i in doc("/public/empsns.xml")/e:emps/e:emp
SP2-0734: unknown command beginning "for $i in ..." - rest of line ignored.
...

-- This works - do not end the line with ";".
SELECT XMLQuery('declare namespace e = "http://example.com"; for
                    $i in doc("/public/empsns.xml")/e:emps/e:emp
                let $d :=
                  doc("/public/depts.xml")//dept[@deptno=$i/@deptno]/@dname
                where $i/@salary > 100000
                order by $i/@empno
                return <emp ename="{$i/@ename}" dept="{$d}"/>'
               RETURNING CONTENT) FROM DUAL;
```

```
XMLQUERY('DECLARENAMESPACEE="HTTP://EXAMPLE.COM";FOR$IINDOC("/PUBLIC/EMPSNS.XML"
--------------------------------------------------------------------------------
<emp ename="Jack" dept=""></emp><emp ename="Jill" dept=""></emp>

-- This works too - add a comment after the ";".
SELECT XMLQuery('declare namespace e = "http://example.com";  (: :)
                 for $i in doc("/public/empsns.xml")/e:emps/e:emp
                 let $d := doc("/public/depts.xml")//dept[@deptno=$i/@deptno]/@dname
                 where $i/@salary > 100000
                 order by $i/@empno
                 return <emp ename="{$i/@ename}" dept="{$d}"/>'
                RETURNING CONTENT) FROM DUAL;

XMLQUERY('DECLARENAMESPACEE="HTTP://EXAMPLE.COM";(::)FOR$IINDOC("/PUBLIC/EMPSNS.
--------------------------------------------------------------------------------
<emp ename="Jack" dept=""></emp><emp ename="Jill" dept=""></emp>

1 row selected.

-- This works too - tell SQL*Plus to ignore the ";".

SET SQLTERMINATOR OFF

SELECT XMLQuery('declare namespace e = "http://example.com";
                 for $i in doc("/public/empsns.xml")/e:emps/e:emp
                 let $d :=
                   doc("/public/depts.xml")//dept[@deptno=$i/@deptno]/@dname
                 where $i/@salary > 100000
                 order by $i/@empno
                 return <emp ename="{$i/@ename}" dept="{$d}"/>'
                RETURNING CONTENT) FROM DUAL
/

XMLQUERY('DECLARENAMESPACEE="HTTP://EXAMPLE.COM";FOR$IINDOC("/PUBLIC/EMPSNS.XML"
--------------------------------------------------------------------------------
<emp ename="Jack" dept=""></emp><emp ename="Jill" dept=""></emp>
```

Example 5-16 Using XMLTABLE with the XMLNAMESPACES Clause

```
SELECT * FROM XMLTable(XMLNAMESPACES ('http://example.com' AS "e"),
                       'for $i in doc("/public/empsns.xml")
                        return $i/e:emps/e:emp'
                       COLUMNS name VARCHAR2(6) PATH '@ename',
                               id   NUMBER      PATH '@empno');
```

This produces the following result:

```
NAME           ID
------ ----------
John            1
Jack            2
Jill            3

3 rows selected.
```

It is the presence of qualified names `e:ename` and `e:empno` in the `COLUMNS` clause that necessitates using the `XMLNAMESPACES` clause. Otherwise, a prolog namespace declaration (`declare namespace e = "http://example.com"`) would suffice for the XQuery expression itself.

Because the same namespace is used throughout the `XMLTable` expression, a default namespace could be used: `XMLNAMESPACES (DEFAULT 'http://example.com')`. The qualified name `$i/e:emps/e:emp` could then be written without an explicit prefix: `$i/emps/emp`.

Querying XML Data Using SQL and PL/SQL

You can query XML data from `XMLType` columns and tables in various ways.

- Select `XMLType` data using SQL, PL/SQL, or Java.
- Query `XMLType` data using SQL/XML functions such as `XMLQuery`. See Querying XMLType Data Using XQuery.
- Perform full-text search using XQuery Full Text. See Support for XQuery Full Text and Indexes for XMLType Data.

The examples in this section illustrate different ways you can use SQL and PL/SQL to query XML data. Example 5-17 inserts two rows into table `purchaseorder`, then queries data in those rows using SQL/XML functions `XMLCast`, `XMLQuery`, and `XMLExists`.

Example 5-18 uses a PL/SQL cursor to query XML data. It uses a local `XMLType` instance to store transient data.

Example 5-19 and Example 5-20 both use SQL/XML function `XMLTable` to extract data from an XML purchase-order document. They then insert that data into a relational table. Example 5-19 uses SQL; Example 5-20 uses PL/SQL.

Example 5-20 defines and uses a PL/SQL procedure to extract data from an XML purchase-order document and insert it into a relational table.

Example 5-21 tabulates the purchase orders whose shipping address contains the string "`Shores`" and which were requested by customers whose names contain the string "`ll`" (double L). These purchase orders are grouped by customer and counted. The example uses XQuery Full Text to perform full-text search.

Example 5-22 extracts the fragments of a document that are identified by an XPath expression. The `XMLType` instance returned by `XMLQuery` can be a set of nodes, a singleton node, or a text value. Example 5-22 uses `XMLType` method `isFragment()` to determine whether the result is a fragment.

> **Note:**
>
> You cannot insert fragments into `XMLType` columns. You can use SQL/XML function `XMLQuery` to convert a fragment into a well-formed document.

Example 5-17 Querying XMLTYPE Data

```
INSERT INTO purchaseorder
  VALUES (XMLType(bfilename('XMLDIR', 'SMCCAIN-2002091213000000PDT.xml'),
                  nls_charset_id('AL32UTF8')));
```

```
INSERT INTO purchaseorder
  VALUES (XMLType(bfilename('XMLDIR', 'VJONES-20020916140000000PDT.xml'),
                  nls_charset_id('AL32UTF8')));

SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Reference'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
               AS VARCHAR2(30)) reference,
       XMLCast(XMLQuery('$p/PurchaseOrder/*//User'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
               AS VARCHAR2(30)) userid,
       CASE
         WHEN XMLExists('$p/PurchaseOrder/Reject/Date'
                        PASSING po.OBJECT_VALUE AS "p")
           THEN 'Rejected'
           ELSE 'Accepted'
       END "STATUS",
       XMLCast(XMLQuery('$p//Date'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
               AS VARCHAR2(12)) status_date
  FROM purchaseorder po
  WHERE XMLExists('$p//Date' PASSING po.OBJECT_VALUE AS "p")
  ORDER BY XMLCast(XMLQuery('$p//Date' PASSING po.OBJECT_VALUE AS "p"
                                       RETURNING CONTENT)
                   AS VARCHAR2(12));



REFERENCE                       USERID   STATUS   STATUS_DATE
------------------------------- -------- -------- ------------
VJONES-20020916140000000PDT     SVOLLMAN Accepted 2002-10-11
SMCCAIN-2002091213000000PDT     SKING    Rejected 2002-10-12

2 rows selected.
```

Example 5-18 Querying Transient XMLTYPE Data Using a PL/SQL Cursor

```
DECLARE
  xNode      XMLType;
  vText      VARCHAR2(256);
  vReference VARCHAR2(32);
  CURSOR getPurchaseOrder(reference IN VARCHAR2) IS
           SELECT OBJECT_VALUE XML
             FROM purchaseorder
             WHERE XMLExists('$p/PurchaseOrder[Reference=$r]'
                             PASSING OBJECT_VALUE AS "p",
                                     reference    AS "r");
BEGIN
  vReference := 'EABEL-20021009123335791PDT';
  FOR c IN getPurchaseOrder(vReference) LOOP
    xNode := c.XML.extract('//Requestor');
    SELECT XMLSerialize(CONTENT
                        XMLQuery('//text()'
                                 PASSING xNode RETURNING CONTENT))
           INTO vText FROM DUAL;
```

```
    DBMS_OUTPUT.put_line('The Requestor for Reference '
                         || vReference || ' is '|| vText);
  END LOOP;
  vReference := 'PTUCKER-20021009123335430PDT';
  FOR c IN getPurchaseOrder(vReference) LOOP
    xNode := c.XML.extract('//LineItem[@ItemNumber="1"]/Description');
    SELECT XMLSerialize(CONTENT
                        XMLQuery('//text()' PASSING xNode RETURNING CONTENT))
           INTO vText FROM DUAL;
    DBMS_OUTPUT.put_line('The Description of LineItem[1] for Reference '
                         || vReference || ' is '|| vText);
  END LOOP;
END;
/
The Requestor for Reference EABEL-20021009123335791PDT is Ellen S. Abel
The Description of LineItem[1] for Reference PTUCKER-20021009123335430PDT is
 Picnic at
Hanging Rock

PL/SQL procedure successfully completed.
```

Example 5-19 Extracting XML Data and Inserting It into a Relational Table Using SQL

```
CREATE TABLE purchaseorder_table (reference           VARCHAR2(28) PRIMARY KEY,
                                  requestor           VARCHAR2(48),
                                  actions             XMLType,
                                  userid              VARCHAR2(32),
                                  costcenter          VARCHAR2(3),
                                  shiptoname          VARCHAR2(48),
                                  address             VARCHAR2(512),
                                  phone               VARCHAR2(32),
                                  rejectedby          VARCHAR2(32),
                                  daterejected        DATE,
                                  comments            VARCHAR2(2048),
                                  specialinstructions VARCHAR2(2048));

CREATE TABLE purchaseorder_lineitem (reference,
                                     FOREIGN KEY ("REFERENCE")
                                       REFERENCES "PURCHASEORDER_TABLE" ("REFERENCE") ON DELETE CASCADE,
                                     lineno      NUMBER(10), PRIMARY KEY ("REFERENCE", "LINENO"),
                                     upc         VARCHAR2(14),
                                     description VARCHAR2(128),
                                     quantity    NUMBER(10),
                                     unitprice   NUMBER(12,2));



INSERT INTO purchaseorder_table (reference, requestor, actions, userid, costcenter, shiptoname, address,
                                 phone, rejectedby, daterejected, comments, specialinstructions)
  SELECT t.reference, t.requestor, t.actions, t.userid, t.costcenter, t.shiptoname, t.address,
         t.phone, t.rejectedby, t.daterejected, t.comments, t.specialinstructions
    FROM purchaseorder p,
         XMLTable('/PurchaseOrder' PASSING p.OBJECT_VALUE
                  COLUMNS reference           VARCHAR2(28)   PATH 'Reference',
                          requestor           VARCHAR2(48)   PATH 'Requestor',
                          actions             XMLType        PATH 'Actions',
                          userid              VARCHAR2(32)   PATH 'User',
                          costcenter          VARCHAR2(3)    PATH 'CostCenter',
                          shiptoname          VARCHAR2(48)   PATH 'ShippingInstructions/name',
                          address             VARCHAR2(512)  PATH 'ShippingInstructions/address',
                          phone               VARCHAR2(32)   PATH 'ShippingInstructions/telephone',
                          rejectedby          VARCHAR2(32)   PATH 'Reject/User',
                          daterejected        DATE           PATH 'Reject/Date',
```

```
                        comments           VARCHAR2(2048) PATH 'Reject/Comments',
                        specialinstructions VARCHAR2(2048) PATH 'SpecialInstructions') t
    WHERE t.reference = 'EABEL-20021009123336251PDT';

INSERT INTO purchaseorder_lineitem (reference, lineno, upc, description, quantity, unitprice)
  SELECT t.reference, li.lineno, li.upc, li.description, li.quantity, li.unitprice
    FROM purchaseorder p,
         XMLTable('/PurchaseOrder' PASSING p.OBJECT_VALUE
                  COLUMNS reference VARCHAR2(28) PATH 'Reference',
                          lineitem XMLType PATH 'LineItems/LineItem') t,
         XMLTable('LineItem' PASSING t.lineitem
                  COLUMNS lineno      NUMBER(10)    PATH '@ItemNumber',
                          upc         VARCHAR2(14)  PATH 'Part/@Id',
                          description VARCHAR2(128) PATH 'Description',
                          quantity    NUMBER(10)    PATH 'Part/@Quantity',
                          unitprice   NUMBER(12,2)  PATH 'Part/@UnitPrice') li
    WHERE t.reference = 'EABEL-20021009123336251PDT';



SELECT reference, userid, shiptoname, specialinstructions FROM purchaseorder_table;

REFERENCE                       USERID   SHIPTONAME                                      SPECIALINSTRUCTIONS
------------------------------- -------- ----------------------------------------------- -------------------
EABEL-20021009123336251PDT      EABEL    Ellen S. Abel                                   Counter to Counter

SELECT reference, lineno, upc, description, quantity FROM purchaseorder_lineitem;

REFERENCE                          LINENO UPC           DESCRIPTION                         QUANTITY
------------------------------- ---------- ------------- ----------------------------------- ----------
EABEL-20021009123336251PDT              1 37429125526   Samurai 2: Duel at Ichijoji Temple          3
EABEL-20021009123336251PDT              2 37429128220   The Red Shoes                               4
EABEL-20021009123336251PDT              3 715515009058  A Night to Remember                         1
```

Example 5-20 Extracting XML Data and Inserting It into a Table Using PL/SQL

```
CREATE OR REPLACE PROCEDURE insertPurchaseOrder(purchaseorder XMLType) AS reference VARCHAR2(28);
BEGIN
  INSERT INTO purchaseorder_table (reference, requestor, actions, userid, costcenter, shiptoname, address,
                                   phone, rejectedby, daterejected, comments, specialinstructions)
    SELECT * FROM XMLTable('$p/PurchaseOrder' PASSING purchaseorder AS "p"
                           COLUMNS reference           VARCHAR2(28)   PATH 'Reference',
                                   requestor           VARCHAR2(48)   PATH 'Requestor',
                                   actions             XMLType        PATH 'Actions',
                                   userid              VARCHAR2(32)   PATH 'User',
                                   costcenter          VARCHAR2(3)    PATH 'CostCenter',
                                   shiptoname          VARCHAR2(48)   PATH 'ShippingInstructions/name',
                                   address             VARCHAR2(512)  PATH 'ShippingInstructions/address',
                                   phone               VARCHAR2(32)   PATH 'ShippingInstructions/telephone',
                                   rejectedby          VARCHAR2(32)   PATH 'Reject/User',
                                   daterejected        DATE           PATH 'Reject/Date',
                                   comments            VARCHAR2(2048) PATH 'Reject/Comments',
                                   specialinstructions VARCHAR2(2048) PATH 'SpecialInstructions');

  INSERT INTO purchaseorder_lineitem (reference, lineno, upc, description, quantity, unitprice)
    SELECT t.reference, li.lineno, li.upc, li.description, li.quantity, li.unitprice
    FROM XMLTable('$p/PurchaseOrder' PASSING purchaseorder AS "p"
                  COLUMNS reference VARCHAR2(28) PATH 'Reference',
                          lineitem XMLType PATH 'LineItems/LineItem') t,
         XMLTable('LineItem' PASSING t.lineitem
                  COLUMNS lineno NUMBER(10)    PATH '@ItemNumber',
                          upc VARCHAR2(14)  PATH 'Part/@Id',
                          description VARCHAR2(128) PATH 'Description',
                          quantity NUMBER(10)    PATH 'Part/@Quantity',
                          unitprice NUMBER(12,2)  PATH 'Part/@UnitPrice') li;
END;
```

```
CALL insertPurchaseOrder(XMLType(bfilename('XMLDIR', 'purchaseOrder.xml'), nls_charset_id('AL32UTF8')));


SELECT reference, userid, shiptoname, specialinstructions FROM purchaseorder_table;

REFERENCE                      USERID   SHIPTONAME                                        SPECIALINSTRUCTIONS
------------------------------ -------- ------------------------------------------------- -------------------
SBELL-20021009123336O1PDT      SBELL    Sarah J. Bell                                     Air Mail


SELECT reference, lineno, upc, description, quantity FROM purchaseorder_lineitem;

REFERENCE                LINENO UPC          DESCRIPTION                       QUANTITY
------------------------ ------ ------------ --------------------------------- --------
SBELL-20021009123336O1PDT      1 715515009058 A Night to Remember                      2
SBELL-20021009123336O1PDT      2 37429140222  The Unbearable Lightness Of Being        2
SBELL-20021009123336O1PDT      3 715515011020 Sisters                                  4
```

Example 5-21 Searching XML Data Using SQL/XML Functions

```
SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                 AS VARCHAR2(128)) name,
       count(*)
  FROM purchaseorder po
  WHERE
    XMLExists(
      'declare namespace ora="http://xmlns.oracle.com/xdb"; (: :)
       $p/PurchaseOrder/ShippingInstructions[address/text() contains text "Shores"]'
      PASSING po.OBJECT_VALUE AS "p")
    AND XMLCast(XMLQuery('$p/PurchaseOrder/Requestor/text()'
                         PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                  AS VARCHAR2(128))
        LIKE '%ll%'
  GROUP BY XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
                            PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                   AS VARCHAR2(128));

NAME                 COUNT(*)
-------------------- ----------
Allan D. McEwen               9
Ellen S. Abel                 4
Sarah J. Bell                13
William M. Smith              7
```

Example 5-22 Extracting Fragments Using XMLQUERY

```
SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Reference' PASSING po.OBJECT_VALUE AS "p"
                                                     RETURNING CONTENT)
               AS VARCHAR2(30)) reference,
       count(*)
  FROM purchaseorder po, XMLTable('$p//LineItem[Part/@Id="37429148327"]' PASSING OBJECT_VALUE AS "p")
  WHERE XMLQuery('$p/PurchaseOrder/LineItems/LineItem[Part/@Id="37429148327"]'
                 PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT).isFragment() = 1
  GROUP BY XMLCast(XMLQuery('$p/PurchaseOrder/Reference' PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                   AS VARCHAR2(30))
  ORDER BY XMLCast(XMLQuery('$p/PurchaseOrder/Reference' PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                   AS VARCHAR2(30));

REFERENCE                      COUNT(*)
------------------------------ ----------
TFOX-20021009123337784PDT               3
```

Using the SQL*Plus XQUERY Command

You can evaluate an XQuery expression using the SQL*Plus `XQUERY` command.

Example 5-23 shows how you can enter an XQuery expression directly at the SQL*Plus command line, by preceding the expression with the SQL*Plus command **`XQUERY`** and following it with a slash (`/`) on a line by itself. Oracle Database treats XQuery expressions submitted with this command the same way it treats XQuery expressions in SQL/XML functions `XMLQuery` and `XMLTable`. Execution is identical, with the same optimizations.

There are also a few SQL*Plus `SET` commands that you can use for settings that are specific to XQuery. Use `SHOW XQUERY` to see the current settings.

- **`SET XQUERY BASEURI`** – Set the base URI for `XQUERY`. URIs in XQuery expressions are relative to this URI.
- **`SET XQUERY CONTEXT`** – Specify a context item for subsequent `XQUERY` evaluations.

> **See Also:**
>
> *SQL*Plus User's Guide and Reference*

Example 5-23 Using the SQL*Plus XQUERY Command

```
SQL> XQUERY for $i in fn:collection("oradb:/HR/DEPARTMENTS")
  2  where $i/ROW/DEPARTMENT_ID < 50
  3  return $i
  4  /

Result Sequence
--------------------------------------------------------------------------------
<ROW><DEPARTMENT_ID>10</DEPARTMENT_ID><DEPARTMENT_NAME>Administration</DEPARTMEN
T_NAME><MANAGER_ID>200</MANAGER_ID><LOCATION_ID>1700</LOCATION_ID></ROW>

<ROW><DEPARTMENT_ID>20</DEPARTMENT_ID><DEPARTMENT_NAME>Marketing</DEPARTMENT_NAM
E><MANAGER_ID>201</MANAGER_ID><LOCATION_ID>1800</LOCATION_ID></ROW>

<ROW><DEPARTMENT_ID>30</DEPARTMENT_ID><DEPARTMENT_NAME>Purchasing</DEPARTMENT_NA
ME><MANAGER_ID>114</MANAGER_ID><LOCATION_ID>1700</LOCATION_ID></ROW>

<ROW><DEPARTMENT_ID>40</DEPARTMENT_ID><DEPARTMENT_NAME>Human Resources</DEPARTME
NT_NAME><MANAGER_ID>203</MANAGER_ID><LOCATION_ID>2400</LOCATION_ID></ROW>
```

Using XQuery with PL/SQL, JDBC, and ODP.NET to Access Database Data

You can use XQuery with the Oracle APIs for PL/SQL, JDBC, and Oracle Data Provider for .NET (ODP.NET).

Example 5-24 shows how to use XQuery with PL/SQL, in particular, how to bind *dynamic variables* to an XQuery expression using the `XMLQuery PASSING` clause. The bind variables `:1` and `:2` are bound to the PL/SQL bind arguments `nbitems` and `partid`, respectively. These are then passed to XQuery as XQuery variables `itemno` and `id`, respectively.

Example 5-25 shows how to use XQuery with JDBC, binding variables by position with the `PASSING` clause of SQL/XML function `XMLTable`.

Example 5-26 shows how to use XQuery with ODP.NET and the C# language. The C# input parameters `:nbitems` and `:partid` are passed to XQuery as XQuery variables `itemno` and `id`, respectively.

Example 5-24 Using XQuery with PL/SQL

```
DECLARE
  sql_stmt VARCHAR2(2000); -- Dynamic SQL statement to execute
  nbitems  NUMBER := 3; -- Number of items
  partid   VARCHAR2(20):= '715515009058'; -- Part ID
  result   XMLType;
  doc      DBMS_XMLDOM.DOMDocument;
  ndoc     DBMS_XMLDOM.DOMNode;
  buf      VARCHAR2(20000);
BEGIN
  sql_stmt :=
    'SELECT XMLQuery(
              ''for $i in fn:collection("oradb:/OE/PURCHASEORDER") ' ||
               'where count($i/PurchaseOrder/LineItems/LineItem) = $itemno ' ||
                 'and $i/PurchaseOrder/LineItems/LineItem/Part/@Id = $id ' ||
               'return $i/PurchaseOrder/LineItems'' ' ||
              'PASSING :1 AS "itemno", :2 AS "id" ' ||
              'RETURNING CONTENT) FROM DUAL';

  EXECUTE IMMEDIATE sql_stmt INTO result USING nbitems, partid;
  doc  := DBMS_XMLDOM.newDOMDocument(result);
  ndoc := DBMS_XMLDOM.makeNode(doc);
  DBMS_XMLDOM.writeToBuffer(ndoc, buf);
  DBMS_OUTPUT.put_line(buf);
END;
/
```

This produces the following output:

```
<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>
<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>

PL/SQL procedure successfully completed.
```

Example 5-25 Using XQuery with JDBC

```
import java.sql.*;
import oracle.sql.*;
import oracle.jdbc.*;
import oracle.xdb.XMLType;
import java.util.*;

public class QueryBindByPos
{
  public static void main(String[] args) throws Exception, SQLException
  {
    System.out.println("*** JDBC Access of XQuery using Bind Variables ***");
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    OracleConnection conn
      = (OracleConnection)
        DriverManager.getConnection("jdbc:oracle:oci8:@localhost:1521:ora11gR1", "oe", "oe");
    String xqString
      = "SELECT COLUMN_VALUE" +
          "FROM XMLTable('for $i in fn:collection(\"oradb:/OE/PURCHASEORDER\") " +
                         "where $i/PurchaseOrder/Reference= $ref " +
                         "return $i/PurchaseOrder/LineItems' " +
                        "PASSING ? AS \"ref\")";
    OraclePreparedStatement stmt = (OraclePreparedStatement)conn.prepareStatement(xqString);
    String refString = "EABEL-20021009123336251PDT"; // Set the filter value
    stmt.setString(1, refString); // Bind the string
    ResultSet rs = stmt.executeQuery();
    while (rs.next())
    {
       SQLXML sqlXml = rs.getSQLXML(1);
       System.out.println("LineItem Description: " + sqlXml.getString());
       sqlXml.free();
    }
    rs.close();
    stmt.close();
  }
}
```

This produces the following output:

```
*** JDBC Access of Database XQuery with Bind Variables ***
LineItem Description: Samurai 2: Duel at Ichijoji Temple
LineItem Description: The Red Shoes
LineItem Description: A Night to Remember
```

Example 5-26 Using XQuery with ODP.NET and C#

```
using System;
using System.Data;
using System.Text;
using System.IO;
using System.Xml;
using Oracle.DataAccess.Client;
using Oracle.DataAccess.Types;

namespace XQuery
{
  /// <summary>
```

```
  /// Demonstrates how to bind variables for XQuery calls
  /// </summary>
  class XQuery
  {
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    static void Main(string[] args)
    {
      int rows = 0;
      StreamReader sr = null;

      // Create the connection.
      string constr = "User Id=oe;Password=***********;Data Source=ora11gr2"; // Replace with real password.
      OracleConnection con = new OracleConnection(constr);
      con.Open();

      // Create the command.
      OracleCommand cmd = new OracleCommand("", con);

      // Set the XML command type to query.
      cmd.CommandType   = CommandType.Text;

      // Create the SQL query with the XQuery expression.
      StringBuilder blr = new StringBuilder();
      blr.Append("SELECT COLUMN_VALUE FROM XMLTable");
      blr.Append("(\'for $i in fn:collection(\"oradb:/OE/PURCHASEORDER\") ");
      blr.Append("   where count($i/PurchaseOrder/LineItems/LineItem) = $itemno ");
      blr.Append("      and $i/PurchaseOrder/LineItems/LineItem/Part/@Id = $id ");
      blr.Append("   return $i/PurchaseOrder/LineItems\' ");
      blr.Append("  PASSING :nbitems AS \"itemno\", :partid AS \"id\")");

      cmd.CommandText = blr.ToString();
      cmd.Parameters.Add(":nbitems", OracleDbType.Int16, 3, ParameterDirection.Input);
      cmd.Parameters.Add(":partid", OracleDbType.Varchar2, "715515009058", ParameterDirection.Input);

      // Get the XML document as an XmlReader.
      OracleDataReader dr = cmd.ExecuteReader();
      dr.Read();

      // Get the XMLType column as an OracleXmlType
      OracleXmlType xml = dr.GetOracleXmlType(0);

      // Print the XML data in the OracleXmlType object
      Console.WriteLine(xml.Value);
      xml.Dispose();

      // Clean up.
      cmd.Dispose();
      con.Close();
      con.Dispose();
    }
  }
}
```

This produces the following output:

```
<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>
```

Related Topics

- PL/SQL APIs for XMLType
  There are several PL/SQL packages that provide APIs for `XMLType`.
- Java DOM API for XMLType
  The Java DOM API for `XMLType` lets you operate on `XMLType` instances using a DOM. You can use it to manipulate XML data in Java, including fetching it through Java Database Connectivity (JDBC).
- Oracle XML DB and Oracle Data Provider for .NET
  Oracle Data Provider for Microsoft .NET (ODP.NET) is an implementation of a data provider for Oracle Database. It uses Oracle native APIs to offer fast and reliable access to Oracle data and features from any .NET application.

Updating XML Data

There are several ways you can use Oracle XML DB features to update XML data, whether it is transient or stored in database tables.

- Updating an Entire XML Document
  To update an entire XML document, use a SQL `UPDATE` statement.
- Replacing XML Nodes
  You can use XQuery Update with a SQL `UPDATE` statement to update an existing XML document instead of creating a new document. The entire document is updated, not just the part of it that is selected.
- Inserting Child XML Nodes
  You can use XQuery Update to insert new children (either a single attribute or one or more elements of the same type) under parent XML elements. The XML document that is the target of the insertion can be schema-based or non-schema-based.
- Deleting XML Nodes
  An example uses XQuery Update to delete XML nodes.
- Creating XML Views of Modified XML Data
  You can use XQuery Update to create new views of XML data.

Updating an Entire XML Document

To update an entire XML document, use a SQL `UPDATE` statement.

The right side of the `UPDATE` statement `SET` clause must be an `XMLType` instance. This can be created in any of the following ways:

- Use SQL functions or XML constructors that return an XML instance.
- Use the PL/SQL DOM APIs for `XMLType` that change and bind an existing XML instance.
- Use the Java DOM API that changes and binds an existing XML instance.

Updates for non-schema-based documents stored as binary XML can be made in a piecewise manner.

Example 5-27 updates an `XMLType` instance using a SQL `UPDATE` statement.

Example 5-27 Updating XMLType Data Using SQL UPDATE

```
SELECT t.reference, li.lineno, li.description
  FROM purchaseorder po,
       XMLTable('$p/PurchaseOrder' PASSING po.OBJECT_VALUE AS "p"
                COLUMNS reference VARCHAR2(28) PATH 'Reference',
                        lineitem  XMLType      PATH 'LineItems/LineItem') t,
       XMLTable('$l/LineItem' PASSING t.lineitem AS "l"
                COLUMNS lineno      NUMBER(10)    PATH '@ItemNumber',
                        description VARCHAR2(128) PATH 'Description') li
  WHERE t.reference = 'DAUSTIN-20021009123335811PDT' AND ROWNUM < 6;

REFERENCE                      LINENO DESCRIPTION
------------------------------ ------- -----------------
DAUSTIN-20021009123335811PDT           1 Nights of Cabiria
DAUSTIN-20021009123335811PDT           2 For All Mankind
DAUSTIN-20021009123335811PDT           3 Dead Ringers
DAUSTIN-20021009123335811PDT           4 Hearts and Minds
DAUSTIN-20021009123335811PDT           5 Rushmore

UPDATE purchaseorder po
  SET po.OBJECT_VALUE = XMLType(bfilename('XMLDIR','NEW-DAUSTIN-20021009123335811PDT.xml'),
                                nls_charset_id('AL32UTF8'))
  WHERE XMLExists('$p/PurchaseOrder[Reference="DAUSTIN-20021009123335811PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

SELECT t.reference, li.lineno, li.description
  FROM purchaseorder po,
       XMLTable('$p/PurchaseOrder' PASSING po.OBJECT_VALUE AS "p"
                COLUMNS reference VARCHAR2(28) PATH 'Reference',
                        lineitem  XMLType      PATH 'LineItems/LineItem') t,
       XMLTable('$l/LineItem' PASSING t.lineitem AS "l"
                COLUMNS lineno      NUMBER(10)    PATH '@ItemNumber',
                        description VARCHAR2(128) PATH 'Description') li
  WHERE t.reference = 'DAUSTIN-20021009123335811PDT';

REFERENCE                      LINENO DESCRIPTION
------------------------------ ------- --------------------------------
DAUSTIN-20021009123335811PDT           1 Dead Ringers
DAUSTIN-20021009123335811PDT           2 Getrud
DAUSTIN-20021009123335811PDT           3 Branded to Kill
```

Replacing XML Nodes

You can use XQuery Update with a SQL `UPDATE` statement to update an existing XML document instead of creating a new document. The entire document is updated, not just the part of it that is selected.

In Example 5-28 we pass the SQL string literal `'SKING'` to the XQuery expression as a variable (`$p2`). In this simple example, since the value is a string literal, we could have simply used `replace value of node $j with "SKING"`. That is, you can just use a literal XQuery string here, instead of passing a literal string from SQL to XQuery. In real-world examples you will typically pass a value that is available only at runtime; Example 5-28 shows how to do that. This is also true of other examples.

Example 5-29 updates multiple text nodes and attribute nodes.

Example 5-30 updates selected nodes within a collection.

Example 5-31 illustrates the common mistake of using an XQuery Update replace-value operation to update a *node that occurs multiple times* in a collection. The `UPDATE` statement sets the value of the text node of a `Description` element to `The Wizard of Oz`, where the current value of the text node is `Sisters`. The statement includes an `XMLExists` expression in the `WHERE` clause that identifies the set of nodes to be updated.

Instead of updating only the intended node, Example 5-31 updates the values of *all* text nodes that belong to the `Description` element. This is not what was intended.

*A `WHERE` clause can be used only to identify which **documents** must be updated, not which **nodes** within a document must be updated*.

After the document has been selected, the *XQuery expression* passed to XQuery Update determines which *nodes* within the document must be updated. In this case, the XQuery expression identifies all three `Description` nodes, so all three of the associated text nodes were updated.

To correctly update a node that occurs multiple times within a collection, use the XQuery expression passed XQuery Update to identify which nodes in the XML document to update. By introducing the appropriate predicate into the XQuery expression, you can limit which nodes in the document are updated. Example 5-32 illustrates the correct way to update one node within a collection.

Example 5-28 Updating XMLTYPE Data Using SQL UPDATE and XQuery Update

```
SELECT XMLQuery('$p/PurchaseOrder/Actions/Action[1]' PASSING po.OBJECT_VALUE AS "p"
                                                     RETURNING CONTENT) action
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

ACTION
--------------------------------
<Action>
  <User>SVOLLMAN</User>
</Action>

UPDATE purchaseorder po
  SET po.OBJECT_VALUE =
    XMLQuery('copy $i := $p1 modify
               (for $j in $i/PurchaseOrder/Actions/Action[1]/User
                return replace value of node $j with $p2)
              return $i' PASSING po.OBJECT_VALUE AS "p1",
                                 'SKING' AS "p2" RETURNING CONTENT)
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

SELECT XMLQuery('$p/PurchaseOrder/Actions/Action[1]' PASSING po.OBJECT_VALUE AS "p"
                                                     RETURNING CONTENT) action
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

ACTION
```

```
---------------------------------
<Action>
  <User>SKING</User>
</Action>
```

Example 5-29 Updating Multiple Text Nodes and Attribute Nodes

```
SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                AS VARCHAR2(30)) name,
       XMLQuery('$p/PurchaseOrder/LineItems'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT) lineitems
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

NAME             LINEITEMS
---------------- --------------------------------------------------------------------------
Sarah J. Bell    <LineItems>
                   <LineItem ItemNumber="1">
                     <Description>A Night to Remember</Description>
                     <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
                   </LineItem>
                   <LineItem ItemNumber="2">
                     <Description>The Unbearable Lightness Of Being</Description>
                     <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/>
                   </LineItem>
                   <LineItem ItemNumber="3">
                     <Description>Sisters</Description>
                     <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/>
                   </LineItem>
                 </LineItems>

UPDATE purchaseorder
  SET OBJECT_VALUE =
    XMLQuery('copy $i := $p1 modify
                ((for $j in $i/PurchaseOrder/Requestor
                  return replace value of node $j with $p2),
                 (for $j in $i/PurchaseOrder/LineItems/LineItem[1]/Part/@Id
                  return replace value of node $j with $p3),
                 (for $j in $i/PurchaseOrder/LineItems/LineItem[1]/Description
                  return replace value of node $j with $p4),
                 (for $j in $i/PurchaseOrder/LineItems/LineItem[3]
                  return replace node $j with $p5))
               return $i'
             PASSING OBJECT_VALUE AS "p1",
                     'Stephen G. King' AS "p2",
                     '786936150421' AS "p3",
                     'The Rock' AS "p4",
                     XMLType('<LineItem ItemNumber="99">
                                <Description>Dead Ringers</Description>
                                <Part Id="715515009249" UnitPrice="39.95" Quantity="2"/>
                              </LineItem>') AS "p5"
             RETURNING CONTENT)
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING OBJECT_VALUE AS "p");

SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                AS VARCHAR2(30)) name,
       XMLQuery('$p/PurchaseOrder/LineItems'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT) lineitems
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

NAME             LINEITEMS
---------------- --------------------------------------------------------------------------
```

```
Stephen G. King  <LineItems>
                   <LineItem ItemNumber="1">
                     <Description>The Rock</Description>
                     <Part Id="786936150421" UnitPrice="39.95" Quantity="2"/>
                   </LineItem>
                   <LineItem ItemNumber="2">
                     <Description>The Unbearable Lightness Of Being</Description>
                     <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/>
                   </LineItem>
                   <LineItem ItemNumber="99">
                     <Description>Dead Ringers</Description>
                     <Part Id="715515009249" UnitPrice="39.95" Quantity="2"/>
                   </LineItem>
                 </LineItems>
```

Example 5-30 Updating Selected Nodes within a Collection

```
SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                AS VARCHAR2(30)) name,
       XMLQuery('$p/PurchaseOrder/LineItems'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT) lineitems
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-20021009123336O1PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

NAME             LINEITEMS
---------------- --------------------------------------------------------------
Sarah J. Bell    <LineItems>
                   <LineItem ItemNumber="1">
                     <Description>A Night to Remember</Description>
                     <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
                   </LineItem>
                   <LineItem ItemNumber="2">
                     <Description>The Unbearable Lightness Of Being</Description>
                     <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/>
                   </LineItem>
                   <LineItem ItemNumber="3">
                     <Description>Sisters</Description>
                     <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/>
                   </LineItem>
                 </LineItems>

UPDATE purchaseorder
  SET OBJECT_VALUE =
      XMLQuery(
        'copy $i := $p1 modify
           ((for $j in $i/PurchaseOrder/Requestor
             return replace value of node $j with $p2),
            (for $j in $i/PurchaseOrder/LineItems/LineItem/Part[@Id="715515009058"]/@Quantity
             return replace value of node $j with $p3),
            (for $j in $i/PurchaseOrder/LineItems/LineItem
                        [Description/text()="The Unbearable Lightness Of Being"]
             return replace node $j with $p4))
           return $i'
        PASSING OBJECT_VALUE AS "p1",
                'Stephen G. King' AS "p2",
                25 AS "p3",
```

```
                XMLType('<LineItem ItemNumber="99">
                           <Part Id="786936150421" Quantity="5" UnitPrice="29.95"/>
                           <Description>The Rock</Description>
                         </LineItem>') AS "p4"
        RETURNING CONTENT)
      WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                      PASSING OBJECT_VALUE AS "p");

SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                AS VARCHAR2(30)) name,
       XMLQuery('$p/PurchaseOrder/LineItems'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT) lineitems
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

NAME             LINEITEMS
---------------- ------------------------------------------------------------
Stephen G. King  <LineItems>
                   <LineItem ItemNumber="1">
                     <Description>A Night to Remember</Description>
                     <Part Id="715515009058" UnitPrice="39.95" Quantity="25"/>
                   </LineItem>
                   <LineItem ItemNumber="99">
                     <Part Id="786936150421" Quantity="5" UnitPrice="29.95"/>
                     <Description>The Rock</Description>
                   </LineItem>
                   <LineItem ItemNumber="3">
                     <Description>Sisters</Description>
                     <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/>
                   </LineItem>
                 </LineItems>
```

Example 5-31 Incorrectly Updating a Node That Occurs Multiple Times in a Collection

```
SELECT XMLCast(des.COLUMN_VALUE AS VARCHAR2(256))
  FROM purchaseorder,
       XMLTable('$p/PurchaseOrder/LineItems/LineItem/Description'
                PASSING OBJECT_VALUE AS "p") des
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING OBJECT_VALUE AS "p");

XMLCAST(DES.COLUMN_VALUEASVARCHAR2(256))
----------------------------------------
The Lady Vanishes
The Unbearable Lightness Of Being
Sisters

3 rows selected.

UPDATE purchaseorder
  SET OBJECT_VALUE =
        XMLQuery('copy $i := $p1 modify
                    (for $j in $i/PurchaseOrder/LineItems/LineItem/Description
```

```
                   return replace value of node $j with $p2)
                 return $i'
                PASSING OBJECT_VALUE AS "p1", 'The Wizard of Oz' AS "p2"
                RETURNING CONTENT)
        WHERE XMLExists('$p/PurchaseOrder/LineItems/LineItem[Description="Sisters"]'
                        PASSING OBJECT_VALUE AS "p")
          AND XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                        PASSING OBJECT_VALUE AS "p");

1 row updated.

SELECT XMLCast(des.COLUMN_VALUE AS VARCHAR2(256))
  FROM purchaseorder,
       XMLTable('$p/PurchaseOrder/LineItems/LineItem/Description'
                PASSING OBJECT_VALUE AS "p") des
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING OBJECT_VALUE AS "p");

XMLCAST(DES.COLUMN_VALUEASVARCHAR2(256))
----------------------------------------
The Wizard of Oz
The Wizard of Oz
The Wizard of Oz

3 rows selected.
```

Example 5-32 Correctly Updating a Node That Occurs Multiple Times in a Collection

```
SELECT XMLCast(des.COLUMN_VALUE AS VARCHAR2(256))
  FROM purchaseorder,
       XMLTable('$p/PurchaseOrder/LineItems/LineItem/Description'
                PASSING OBJECT_VALUE AS "p") des
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING OBJECT_VALUE AS "p");

XMLCAST(DES.COLUMN_VALUEASVARCHAR2(256))
----------------------------------------

A Night to Remember
The Unbearable Lightness Of Being
Sisters

3 rows selected.


UPDATE purchaseorder
 SET OBJECT_VALUE =
       XMLQuery('copy $i := $p1 modify
                   (for $j in $i/PurchaseOrder/LineItems/LineItem/Description
                               [text()="Sisters"]
                    return replace value of node $j with $p2)
                 return $i'
                PASSING OBJECT_VALUE       AS "p1",
                        'The Wizard of Oz' AS "p2" RETURNING CONTENT)
       WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
```

```
                        PASSING OBJECT_VALUE AS "p");

1 row updated.

SELECT XMLCast(des.COLUMN_VALUE AS VARCHAR2(256))
  FROM purchaseorder,
       XMLTable('$p/PurchaseOrder/LineItems/LineItem/Description'
                PASSING OBJECT_VALUE AS "p") des
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-20021009123336O1PDT"]'
                  PASSING OBJECT_VALUE AS "p");

XMLCAST(DES.COLUMN_VALUEASVARCHAR2(256))
----------------------------------------
A Night to Remember
The Unbearable Lightness Of Being
The Wizard of Oz

3 rows selected.
```

- Updating XML Data to NULL Values
  Certain considerations apply to updating XML data to `NULL` values.

Updating XML Data to NULL Values

Certain considerations apply to updating XML data to `NULL` values.

- If you update an XML *element* to `NULL`, the attributes and children of the element are removed, and the element becomes empty. The type and namespace properties of the element are retained. See Example 5-33.
- If you update an *attribute* value to `NULL`, the value appears as the empty string. See Example 5-33.
- If you update the *text* node of an element to `NULL`, the content (text) of the element is removed. The element itself remains, but it is empty. See Example 5-34.

Example 5-33 updates all of the following to `NULL`:

- The `Description` element and the `Quantity` attribute of the `LineItem` element whose `Part` element has attribute `Id` value 715515009058.
- The `LineItem` element whose `Description` element has the content (text) "The Unbearable Lightness Of Being".

Example 5-33 shows two different but equivalent ways to remove the value of a node. For element `Description` and attribute `Quantity`, a literal XQuery empty sequence, `()`, replaces the existing value directly. For element `LineItem`, SQL **`NULL`** is passed into the XQuery expression to provide the empty node value. Since the value used is literal, it is simpler not to pass it from SQL to XQuery. But in real-world examples you will often pass a value that is available only at runtime. Example 5-33 shows how to do this for an empty XQuery sequence: pass a SQL `NULL` value.

Example 5-34 updates the text node of a `Part` element whose `Description` attribute has value "`A Night to Remember`" to `NULL`. The XML data for this example corresponds to a different, revised purchase-order XML schema – see Scenario for Copy-Based Evolution. In that XML schema, `Description` is an attribute of the `Part` element, not a sibling element.

> **See Also:**
>
> Example 3-28

Example 5-33 NULL Updates – Element and Attribute

```
SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                AS VARCHAR2(30)) name,
       XMLQuery('$p/PurchaseOrder/LineItems'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT) lineitems
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

NAME             LINEITEMS
---------------- --------------------------------------------------------------------
Sarah J. Bell    <LineItems>
                   <LineItem ItemNumber="1">
                     <Description>A Night to Remember</Description>
                     <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
                   </LineItem>
                   <LineItem ItemNumber="2">
                     <Description>The Unbearable Lightness Of Being</Description>
                     <Part Id="37429140222" UnitPrice="29.95" Quantity="2"/>
                   </LineItem>
                   <LineItem ItemNumber="3">
                     <Description>Sisters</Description>
                     <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/>
                   </LineItem>
                 </LineItems>

UPDATE purchaseorder
  SET OBJECT_VALUE =
      XMLQuery(
        'copy $i := $p1 modify
           ((for $j in $i/PurchaseOrder/LineItems/LineItem[Part/@Id="715515009058"]/Description
             return replace value of node $j with ()) ,
            (for $j in $i/PurchaseOrder/LineItems/LineItem/Part[@Id="715515009058"]/@Quantity
             return replace value of node $j with ()) ,
            (for $j in $i/PurchaseOrder/LineItems/LineItem
                        [Description/text()= "The Unbearable Lightness Of Being"]
             return replace node $j with $p2))
          return $i'
         PASSING OBJECT_VALUE AS "p1", NULL AS "p2"
         RETURNING CONTENT)
      WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                      PASSING OBJECT_VALUE AS "p");

SELECT XMLCast(XMLQuery('$p/PurchaseOrder/Requestor'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
                AS VARCHAR2(30)) name,
       XMLQuery('$p/PurchaseOrder/LineItems'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT) lineitems
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="SBELL-2002100912333601PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

NAME             LINEITEMS
```

```
---------------- ------------------------------------------------------------
Sarah J. Bell    <LineItems>
                   <LineItem ItemNumber="1">
                     <Description/>
                     <Part Id="715515009058" UnitPrice="39.95" Quantity=""/>
                   </LineItem>
                   <LineItem/>
                   <LineItem ItemNumber="3">
                     <Description>Sisters</Description>
                     <Part Id="715515011020" UnitPrice="29.95" Quantity="4"/>
                   </LineItem>
                 </LineItems>
```

Example 5-34 NULL Updates – Text Node

```
SELECT XMLCast(XMLQuery('$p/PurchaseOrder/LineItems/LineItem/Part[@Description="A Night to Remember"]'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
               AS VARCHAR2(128)) part
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[@Reference="SBELL-20030309123336O1PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

PART
----
<Part Description="A Night to Remember" UnitCost="39.95">715515009058</Part>

UPDATE purchaseorder
  SET OBJECT_VALUE =
      XMLQuery(
        'copy $i := $p1 modify
           (for $j in $i/PurchaseOrder/LineItems/LineItem/Part[@Description="A Night to Remember"]
            return replace value of node $j with $p2)
         return $i
        PASSING OBJECT_VALUE AS "p1", NULL AS "p2" RETURNING CONTENT)
  WHERE XMLExists('$p/PurchaseOrder[@Reference="SBELL-20030309123336O1PDT"]'
                  PASSING OBJECT_VALUE AS "p");

SELECT XMLCast(XMLQuery('$p/PurchaseOrder/LineItems/LineItem/Part[@Description="A Night to Remember"]'
                        PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
               AS VARCHAR2(128)) part
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[@Reference="SBELL-20030309123336O1PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

PART
----
<Part Description="A Night to Remember" UnitCost="39.95"/>
```

Inserting Child XML Nodes

You can use XQuery Update to insert new children (either a single attribute or one or more elements of the same type) under parent XML elements. The XML document that is the target of the insertion can be schema-based or non-schema-based.

Example 5-35 inserts a new `LineItem` element as a child of element `LineItems`. It uses the Oracle XQuery pragma `ora:child-element-name` to specify the name of the inserted child element as `LineItem`.

If the XML data to be updated is XML schema-based and it refers to a namespace, then the data to be inserted must also refer to the same namespace. Otherwise, an error is raised because the inserted data does not conform to the XML schema.

> **Note:**
>
> Be aware that using XQuery Update to update XML schema-based data results in an error being raised if you try to store the updated data back into an XML schema-based column or table. To prevent this, use XQuery pragma `ora:transform_keep_schema`. See Oracle XQuery Extension-Expression Pragmas.

Example 5-36 is the same as Example 5-35, except that the `LineItem` element to be inserted refers to a namespace. This assumes that the relevant XML schema requires a namespace for this element.

Example 5-37 inserts a `LineItem` element before the first `LineItem` element.

Example 5-38 inserts a `Date` element as the last child of an `Action` element.

Example 5-35 Inserting an Element into a Collection

```
SELECT XMLQuery('$p/PurchaseOrder/LineItems/LineItem[@ItemNumber=222]'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

XMLQUERY('$P/PURCHASEORDER/LINEITEMS/LINEITEM[@ITEMNUMBER=222]'
--------------------------------------------------------------

1 row selected.

UPDATE purchaseorder
  SET OBJECT_VALUE =
      XMLQuery('copy $i := $p1 modify
                  (for $j in $i/PurchaseOrder/LineItems
                   return (# ora:child-element-name LineItem #)
                          {insert node $p2 into $j})
                return $i'
               PASSING OBJECT_VALUE AS "p1",
                       XMLType('<LineItem ItemNumber="222">
                                  <Description>The Harder They Come</Description>
                                  <Part Id="953562951413" UnitPrice="22.95" Quantity="1"/>
                                </LineItem>') AS "p2"
               RETURNING CONTENT)
      WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                      PASSING OBJECT_VALUE AS "p");

SELECT XMLQuery('$p/PurchaseOrder/LineItems/LineItem[@ItemNumber=222]'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

XMLQUERY('$P/PURCHASEORDER/LINEITEMS/LINEITEM[@ITEMNUMBER=222]'
--------------------------------------------------------------
<LineItem ItemNumber="222">
  <Description>The Harder They Come</Description>
  <Part Id="953562951413" UnitPrice="22.95" Quantity="1"/>
</LineItem>

1 row selected.
```

Example 5-36 Inserting an Element that Uses a Namespace

```
UPDATE purchaseorder
  SET OBJECT_VALUE =
      XMLQuery('declare namespace e = "films.xsd"; (: :)
                copy $i := $p1 modify
                  (for $j in $i/PurchaseOrder/LineItems
                   return (# ora:child-element-name e:LineItem #)
                          {insert node $p2 into $j})
                return $i'
               PASSING OBJECT_VALUE AS "p1",
                       XMLType('<e:LineItem ItemNumber="222">
                                  <Description>The Harder They Come</Description>
                                  <Part Id="953562951413" UnitPrice="22.95" Quantity="1"/>
                                </e:LineItem>') AS "p2"
               RETURNING CONTENT)
      WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                      PASSING OBJECT_VALUE AS "p");
```

Example 5-37 Inserting an Element Before an Element

```
SELECT XMLQuery('$p/PurchaseOrder/LineItems/LineItem[1]'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                   PASSING po.OBJECT_VALUE AS "p");

XMLQUERY('$P/PURCHASEORDER/LINEITEMS/LINEITEM[1]'PASSINGPO.OBJECT_
--------------------------------------------------------------------
<LineItem ItemNumber="1">
  <Description>Salesperson</Description>
  <Part Id="37429158920" UnitPrice="39.95" Quantity="2"/>
</LineItem>

UPDATE purchaseorder
  SET OBJECT_VALUE =
      XMLQuery('copy $i := $p1 modify
                  (for $j in $i/PurchaseOrder/LineItems/LineItem[1]
                   return insert node $p2 before $j)
                return $i'
               PASSING OBJECT_VALUE AS "p1",
                       XMLType('<LineItem ItemNumber="314">
                                  <Description>Brazil</Description>
                                  <Part Id="314159265359" UnitPrice="69.95"
                                        Quantity="2"/>
                                </LineItem>') AS "p2"
               RETURNING CONTENT)
      WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                      PASSING OBJECT_VALUE AS "p");

SELECT XMLQuery('$p/PurchaseOrder/LineItems/LineItem[position() <= 2]'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                   PASSING po.OBJECT_VALUE AS "p");

XMLQUERY('$P/PURCHASEORDER/LINEITEMS/LINEITEM[POSITION()<=2]'PASSINGPO.OBJECT_
--------------------------------------------------------------------------------
<LineItem ItemNumber="314">
  <Description>Brazil</Description>
  <Part Id="314159265359" UnitPrice="69.95" Quantity="2"/>
</LineItem>
<LineItem ItemNumber="1">
```

```
  <Description>Salesperson</Description>
  <Part Id="37429158920" UnitPrice="39.95" Quantity="2"/>
</LineItem>
```

Example 5-38 Inserting an Element as the Last Child Element

```
SELECT XMLQuery('$p/PurchaseOrder/Actions/Action[1]'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

XMLQUERY('$P/PURCHASEORDER/ACTIONS/ACTION[1]'PASSINGPO.OBJECT_VALUE
--------------------------------------------------------------------
<Action>
  <User>KPARTNER</User>
</Action>

UPDATE purchaseorder
  SET OBJECT_VALUE =
      XMLQuery('copy $i := $p1 modify
                  (for $j in $i/PurchaseOrder/Actions/Action[1]
                   return insert nodes $p2 as last into $j)
                return $i'
               PASSING OBJECT_VALUE AS "p1",
                       XMLType('<Date>2002-11-04</Date>') AS "p2"
               RETURNING CONTENT)
      WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                      PASSING OBJECT_VALUE AS "p");

SELECT XMLQuery('$p/PurchaseOrder/Actions/Action[1]'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

XMLQUERY('$P/PURCHASEORDER/ACTIONS/ACTION[1]'PASSINGPO.OBJECT_VALUE
--------------------------------------------------------------------
<Action>
  <User>KPARTNER</User>
  <Date>2002-11-04</Date>
</Action>
```

Deleting XML Nodes

An example uses XQuery Update to delete XML nodes.

Example 5-39 deletes the `LineItem` element whose `ItemNumber` attribute has value `222`.

Example 5-39 Deleting an Element

```
SELECT XMLQuery('$p/PurchaseOrder/LineItems/LineItem[@ItemNumber=222]'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

XMLQUERY('$P/PURCHASEORDER/LINEITEMS/LINEITEM[@ITEMNUMBER=222]'PASSINGPO
--------------------------------------------------------------------------
<LineItem ItemNumber="222">
```

```
  <Description>The Harder They Come</Description>
  <Part Id="953562951413" UnitPrice="22.95" Quantity="1"/>
</LineItem>

UPDATE purchaseorder
  SET OBJECT_VALUE =
      XMLQuery('copy $i := $p modify
                  delete nodes $i/PurchaseOrder/LineItems/LineItem[@ItemNumber="222"]
                return $i'
               PASSING OBJECT_VALUE AS "p" RETURNING CONTENT)
      WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                      PASSING OBJECT_VALUE AS "p");

SELECT XMLQuery('$p/PurchaseOrder/LineItems/LineItem[@ItemNumber=222]'
                PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
  FROM purchaseorder po
  WHERE XMLExists('$p/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]'
                  PASSING po.OBJECT_VALUE AS "p");

XMLQUERY('$P/PURCHASEORDER/LINEITEMS/LINEITEM[@ITEMNUMBER=222]'PASSINGPO
--------------------------------------------------------------------------

1 row selected.
```

Creating XML Views of Modified XML Data

You can use XQuery Update to create new views of XML data.

Example 5-40 creates a view of table `purchaseorder`.

Example 5-40 Creating a View Using Updated XML Data

```
CREATE OR REPLACE VIEW purchaseorder_summary OF XMLType AS
  SELECT XMLQuery('copy $i := $p1 modify
                     ((for $j in $i/PurchaseOrder/Actions
                       return replace value of node $j with ()),
                      (for $j in $i/PurchaseOrder/ShippingInstructions
                       return replace value of node $j with ()),
                      (for $j in $i/PurchaseOrder/LineItems
                       return replace value of node $j with ()))
                   return $i'
                  PASSING OBJECT_VALUE AS "p1" RETURNING CONTENT)
    FROM purchaseorder p;

SELECT OBJECT_VALUE FROM purchaseorder_summary
  WHERE XMLExists('$p/PurchaseOrder[Reference="DAUSTIN-20021009123335811PDT"]'
                  PASSING OBJECT_VALUE AS "p");

OBJECT_VALUE
--------------------------------------------------------------------------------
<PurchaseOrder
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation=
      "http://localhost:8080/source/schemas/poSource/xsd/purchaseOrder.xsd">
  <Reference>DAUSTIN-20021009123335811PDT</Reference>
  <Actions/>
```

```
  <Reject/>
  <Requestor>David L. Austin</Requestor>
  <User>DAUSTIN</User>
  <CostCenter>S30</CostCenter>
  <ShippingInstructions/>
  <SpecialInstructions>Courier</SpecialInstructions>
  <LineItems/>
</PurchaseOrder>
```

Performance Tuning for XQuery

A SQL query that involves XQuery expressions can often be automatically rewritten (optimized) in one or more ways. This optimization is referred to as **XML query rewrite** or optimization. When this happens, the XQuery expression is, in effect, evaluated directly against the XML document without constructing a DOM in memory.

XPath expressions are a proper subset of XQuery expressions. **XPath rewrite** is a subset of XML query rewrite that involves rewriting queries that involve XPath expressions.

XPath rewrite includes all of the following:

- Single-pass streaming of `XMLType` data stored as binary XML – A set of XPath expressions is evaluated in a single scan of the data.
- `XMLIndex` optimizations – A SQL statement that uses an XPath expression is rewritten to an equivalent SQL statement that does not use it but which instead references the relational `XMLIndex` tables. The rewritten SQL statement can also make use of any B-tree indexes on the underlying `XMLIndex` tables.
- Optimizations for `XMLType` data stored object-relationally and for `XMLType` views – A SQL statement that uses an XPath expression is rewritten to an equivalent SQL statement that does not use it but which instead references the object-relational or relational data structures that underly the `XMLType` data. The rewritten SQL statement can also make use of any B-tree indexes on the underlying data structures. This can take place for both queries and update operations.

Just as query tuning can improve SQL performance, so it can improve XQuery performance. You tune XQuery performance by choosing appropriate XML storage models and indexes.

As with database queries generally, you determine whether tuning is required by examining the execution plan for a query. If the plan is not optimal, then consult the following documentation for specific tuning information:

- For object-relational storage: XPath Rewrite for Object-Relational Storage
- For compact schema-aware binary XML and transportable binary XML storage: Indexes for XMLType Data

In addition, be aware that the following expressions can be expensive to process, so they might add performance overhead when processing large volumes of data:

- XQuery expressions that use the following axes (use forward and descendent axes instead):
  - `ancestor`
  - `ancestor-or-self`
  - `descendant-or-self`
  - `following`

  - following-sibling
  - namespace
  - parent
  - preceding
  - preceding-sibling
- XQuery expressions that involve node identity (for example, using the order-comparison operators << and >>)

Topics in this section present execution plans for some of the examples shown in XQuery and Oracle XML DB, to indicate how they are executed.

- Rule-Based and Cost-Based XQuery Optimization
  Several competing optimization possibilities can exist for queries with XQuery expressions, depending on various factors such as the `XMLType` storage model and indexing that are used.
- XQuery Optimization over Relational Data
  Use of SQL/XML functions `XMLQuery` and `XMLTable` over relational data can be optimized. Examples are included that use XQuery expressions that target XML data created on the fly using `fn:collection` together with URI scheme `oradb`.
- XQuery Optimization over XML Schema-Based XMLType Data
  Use of SQL/XML functions `XMLQuery` and `XMLTable` XML Schema-based data can be optimized. Examples are included that use XQuery expressions that target an XML schema-based `XMLType` table stored object-relationally.
- Diagnosis of XQuery Optimization: XMLOptimizationCheck
  You can examine an execution plan for your SQL code to determine whether XQuery optimization occurs or the plan is instead suboptimal.
- Performance Improvement for fn:doc and fn:collection on Repository Data
  You can improve the performance of `fn:doc` and `fn:collection` queries over the Oracle XML DB Repository, by linking them to the actual database tables that hold the repository data being queried.

Related Topics

- Oracle XML DB Support for XQuery
  Oracle XML DB support for the XQuery language includes SQL support and support for XQuery functions and operators.

Rule-Based and Cost-Based XQuery Optimization

Several competing optimization possibilities can exist for queries with XQuery expressions, depending on various factors such as the `XMLType` storage model and indexing that are used.

By default, Oracle XML DB follows a prioritized set of rules to determine which of the possible optimizations should be used for any given query and context. This behavior is referred to as **rule-based** XML query rewrite.

Alternatively, Oracle XML DB can use **cost-based** XML query rewrite. In this mode, Oracle XML DB estimates the performance of the various XML optimization possibilities for a given query and chooses the combination that is expected to be most performant.

You can impose cost-based optimization for a given SQL statement by using the optimizer hint `/*+ COST_XML_QUERY_REWRITE */`.

XQuery Optimization over Relational Data

Use of SQL/XML functions `XMLQuery` and `XMLTable` over relational data can be optimized. Examples are included that use XQuery expressions that target XML data created on the fly using `fn:collection` together with URI scheme `oradb`.

Example 5-41 shows the optimization of `XMLQuery` over relational data accessed as XML. Example 5-42 shows the optimization of `XMLTable` in the same context.

Example 5-41 Optimization of XMLQuery over Relational Data

Here again is the query of Example 5-6, together with its execution plan, which shows that the query has been optimized.

```
SELECT XMLQuery(
         'for $i in fn:collection("oradb:/OE/WAREHOUSES")/ROW
          return <Warehouse id="{$i/WAREHOUSE_ID}">
                   <Location>
                     {for $j in fn:collection("oradb:/HR/LOCATIONS")/ROW
                      where $j/LOCATION_ID eq $i/LOCATION_ID
                      return ($j/STREET_ADDRESS, $j/CITY, $j/STATE_PROVINCE)}
                   </Location>
                 </Warehouse>'
        RETURNING CONTENT) FROM DUAL;

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 3341889589

-------------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |     1 |       |     2   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE              |            |     1 |    41 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| LOCATIONS  |     1 |    41 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN         | LOC_ID_PK  |     1 |       |     0   (0)| 00:00:01 |
|   4 |  SORT AGGREGATE              |            |     1 |     6 |            |          |
|   5 |   TABLE ACCESS FULL          | WAREHOUSES |     9 |    54 |     2   (0)| 00:00:01 |
|   6 |  FAST DUAL                   |            |     1 |       |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("LOCATION_ID"=:B1)

18 rows selected.
```

Example 5-42 Optimization of XMLTable over Relational Data

Here again is the query of Example 5-7, together with its execution plan, which shows that the query has been optimized.

```
SELECT *
  FROM XMLTable(
         'for $i in fn:collection("oradb:/OE/WAREHOUSES")/ROW
          return <Warehouse id="{$i/WAREHOUSE_ID}">
                   <Location>
                     {for $j in fn:collection("oradb:/HR/LOCATIONS")/ROW
                      where $j/LOCATION_ID eq $i/LOCATION_ID
                      return ($j/STREET_ADDRESS, $j/CITY, $j/STATE_PROVINCE)}
                   </Location>
                 </Warehouse>');
```

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 1021775546

------------------------------------------------------------------------------------------
| Id  | Operation                     | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |     9 |    54 |     2   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE               |            |     1 |    41 |            |          |
|   2 |   TABLE ACCESS BY INDEX ROWID| LOCATIONS  |     1 |    41 |     1   (0)| 00:00:01 |
|*  3 |    INDEX UNIQUE SCAN          | LOC_ID_PK  |     1 |       |     0   (0)| 00:00:01 |
|   4 |  TABLE ACCESS FULL            | WAREHOUSES |     9 |    54 |     2   (0)| 00:00:01 |
------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("LOCATION_ID"=:B1)

16 rows selected.
```

XQuery Optimization over XML Schema-Based XMLType Data

Use of SQL/XML functions `XMLQuery` and `XMLTable` XML Schema-based data can be optimized. Examples are included that use XQuery expressions that target an XML schema-based `XMLType` table stored object-relationally.

Example 5-43 shows the optimization of `XMLQuery` over an XML schema-based `XMLType` table. Example 5-44 shows the optimization of `XMLTable` in the same context.

Example 5-43 Optimization of XMLQuery with Schema-Based XMLType Data

Here again is the query of Example 5-10, together with its execution plan, which shows that the query has been optimized.

```
SELECT XMLQuery('for $i in /PurchaseOrder
                 where $i/CostCenter eq "A10"
                   and $i/User eq "SMCCAIN"
                 return <A10po pono="{$i/Reference}"/>'
                PASSING OBJECT_VALUE
                RETURNING CONTENT)
  FROM purchaseorder;


PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 3611789148

------------------------------------------------------------------------------------------
| Id  | Operation           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |     1 |   530 |     5   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE     |               |     1 |       |            |          |
|*  2 |   FILTER            |               |       |       |            |          |
|   3 |    FAST DUAL        |               |     1 |       |     2   (0)| 00:00:01 |
|*  4 |    TABLE ACCESS FULL| PURCHASEORDER |     1 |   530 |     5   (0)| 00:00:01 |
------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
```

```
   2 - filter(:B1='SMCCAIN' AND :B2='A10')
   4 - 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 DAV:http://xmlns.oracle.com/xdb/dav.xsd">
              <read-properties/><read-contents/></privilege>'))=1)

22 rows selected.
```

Example 5-44 Optimization of XMLTable with Schema-Based XMLType Data

Here again is the query of Example 5-14, together with its execution plan, which shows that the query has been optimized. The XQuery result is never materialized. Instead, the underlying storage columns for the XML collection element `LineItem` are used to generate the overall result set.

```
SELECT lines.lineitem, lines.description, lines.partid,
       lines.unitprice, lines.quantity
  FROM purchaseorder,
       XMLTable('for $i in /PurchaseOrder/LineItems/LineItem
                 where $i/@ItemNumber >= 8
                   and $i/Part/@UnitPrice > 50
                   and $i/Part/@Quantity > 2
                 return $i'
                PASSING OBJECT_VALUE
                COLUMNS lineitem    NUMBER       PATH '@ItemNumber',
                        description VARCHAR2(30) PATH 'Description',
                        partid      NUMBER       PATH 'Part/@Id',
                        unitprice   NUMBER       PATH 'Part/@UnitPrice',
                        quantity    NUMBER       PATH 'Part/@Quantity') lines;
```

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                |     4 |   384 |     7   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                 |                |       |       |            |          |
|   2 |   NESTED LOOPS                |                |     4 |   384 |     7   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL          | PURCHASEORDER  |     1 |    37 |     5   (0)| 00:00:01 |
|*  4 |    INDEX RANGE SCAN           | SYS_C005478    |    17 |       |     1   (0)| 00:00:01 |
|*  5 |   TABLE ACCESS BY INDEX ROWID| LINEITEM_TABLE |     3 |   177 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - 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
              DAV:http://xmlns.oracle.com/xdb/dav.xsd"><read-prop
              erties/><read-contents/></privilege>'))=1)
   4 - access("NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC0003400035$")
   5 - filter("SYS_NC00013$">50 AND "SYS_NC00012$">2 AND "ITEMNUMBER">=8 AND
              "SYS_NC_TYPEID$" IS NOT NULL)

25 rows selected.
```

This example traverses table `oe.purchaseorder` completely. The `XMLTable` expression is evaluated for each purchase-order document. It is more efficient to have the `XMLTable` expression, not the `purchaseorder` table, drive the SQL-query execution.

Although the XQuery expression has been rewritten to relational expressions, you can improve this optimization by creating an *index* on the underlying relational data — you can optimize this query in the same way that you would optimize a purely SQL query. That is always the case with XQuery in Oracle XML DB: the optimization techniques you use are the same as those you use in SQL.

The `UnitPrice` attribute of collection element `LineItem` is an appropriate index target. The governing XML schema specifies that an ordered collection table (OCT) is used to store the `LineItem` elements.

However, the name of this OCT was generated by Oracle XML DB when the XML purchase-order documents were decomposed as XML schema-based data. Instead of using table `purchaseorder` from sample database schema `HR`, you could manually create a new `purchaseorder` table (in a different database schema) with the same properties and same data, but having OCTs with user-friendly names.

Assuming that this has been done, the following statement creates the appropriate index:

```
CREATE INDEX unitprice_index ON lineitem_table("PART"."UNITPRICE");
```

With this index defined, the query of Example 5-14 results in the following execution plan, which shows that the `XMLTable` expression has driven the overall evaluation.

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 1578014525

-------------------------------------------------------------------------------------------
| Id  | Operation          | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |                   |     3 |   624 |     8   (0)| 00:00:01 |
|   1 |  NESTED LOOPS      |                   |     3 |   624 |     8   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN| SYS_IOT_TOP_49323 |     3 |   564 |     5   (0)| 00:00:01 |
|*  3 |    INDEX RANGE SCAN| UNITPRICE_INDEX   |    20 |       |     2   (0)| 00:00:01 |
|*  4 |   INDEX UNIQUE SCAN| SYS_C004411       |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("SYS_NC00013$">50)
       filter("ITEMNUMBER">=8 AND "SYS_NC00012$">2)
   3 - access("SYS_NC00013$">50)
   4 - access("NESTED_TABLE_ID"="PURCHASEORDER"."SYS_NC0003400035$")

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

23 rows selected.
```

Diagnosis of XQuery Optimization: XMLOptimizationCheck

You can examine an execution plan for your SQL code to determine whether XQuery optimization occurs or the plan is instead suboptimal.

In the latter case, a note such as the following appears immediately after the plan:

```
Unoptimized XML construct detected (enable XMLOptimizationCheck
for more information)
```

You can also compare the execution plan output with the plan output that you see after you use the optimizer hint `NO_XML_QUERY_REWRITE`, which turns off XQuery optimization.

In addition, you can use the SQL*Plus `SET` command with system variable `XMLOptimizationCheck` to turn on an **XML diagnosability mode** for SQL:

```
SET XMLOptimizationCheck ON
```

When this mode is on, the plan of execution is automatically checked for XQuery optimization, and if the plan is suboptimal then an error is raised and diagnostic information is written to the trace file indicating which operators are not rewritten.

The main advantage of `XMLOptimizationCheck` is that it brings a potential problem to your attention immediately. For this reason, you might find it preferable to leave it turned on at all times. Then, if an application change or a database change for some reason prevents a SQL operation from rewriting, execution is stopped instead of performance being negatively impacted without your being aware of the cause.

Note:

- `XMLOptimizationCheck` was not available prior to Oracle Database 11g Release 2 (11.2.0.2). Users of older releases directly manipulated event 19201 to obtain XQuery optimization information.
- OCI users can use `OCIStmtExecute` or event 19201. Only the event is available to Java users.

See Also:

Turning Off Use of XMLIndex for information about optimizer hint `NO_XML_QUERY_REWRITE`

Performance Improvement for fn:doc and fn:collection on Repository Data

You can improve the performance of `fn:doc` and `fn:collection` queries over the Oracle XML DB Repository, by linking them to the actual database tables that hold the repository data being queried.

In Oracle XML DB, you can use XQuery functions `fn:doc` and `fn:collection` to reference documents and collections in Oracle XML DB Repository.

When repository XML data is stored object-relationally or as binary XML, queries that use `fn:doc` and `fn:collection` are evaluated functionally; that is, they are not optimized to access the underlying storage tables directly. To improve the performance of such queries, you must

link them to the actual database tables that hold the repository data being queried. You can do that in either of the following ways:

- Join view `RESOURCE_VIEW` with the `XMLType` table that holds the data, and then use the Oracle SQL functions `equals_path` and `under_path` instead of the XQuery functions `fn:doc` and `fn:collection`, respectively. These SQL functions reference repository resources in a performant way.
- Use the Oracle XQuery extension-expression pragma `ora:defaultTable`.

Both methods have the same effect. Oracle recommends that you use the `ora:defaultTable` pragma because it lets you continue to use the XQuery standard functions `fn:doc` and `fn:collection` and it simplifies your code.

These two methods are illustrated in the examples of this section.

- Use EQUALS_PATH and UNDER_PATH Instead of fn:doc and fn:collection
  Using Oracle SQL functions `equals_path` and `under_path` instead of XQuery functions `fn:doc` and `fn:collection` can improve performance.
- Using Oracle XQuery Pragma ora:defaultTable
  You can use Oracle XQuery extension-expression pragma `ora:defaultTable` to improve the performance of querying repository data.

Use EQUALS_PATH and UNDER_PATH Instead of fn:doc and fn:collection

Using Oracle SQL functions `equals_path` and `under_path` instead of XQuery functions `fn:doc` and `fn:collection` can improve performance.

SQL function `equals_path` references a resource located at a specified repository path, and SQL function `under_path` references a resource located under a specified repository path. Example 5-45 and Example 5-46 illustrate this for functions `fn:doc` and `equals_path`; functions `fn:collection` and `under_path` are treated similarly.

Example 5-45 Unoptimized Repository Query Using fn:doc

```
SELECT XMLQuery(
         'let $val :=
               fn:doc("/home/OE/PurchaseOrders/2002/Sep/VJONES-20021009123337583PDT.xml")
               /PurchaseOrder/LineItems/LineItem[@ItemNumber =19]
          return $val' RETURNING CONTENT)
  FROM DUAL;
```

Example 5-46 Optimized Repository Query Using EQUALS_PATH

```
SELECT XMLQuery('let $val := $DOC/PurchaseOrder/LineItems/LineItem[@ItemNumber = 19]
                 return $val' PASSING OBJECT_VALUE AS "DOC" RETURNING CONTENT)
  FROM RESOURCE_VIEW rv, purchaseorder p
  WHERE ref(p) = XMLCast(XMLQuery('declare default element namespace
                                   "http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :)
                                   fn:data[1](/Resource/XMLRef)'
                                  PASSING rv.RES RETURNING CONTENT)
                         AS REF XMLType)
    AND equals_path(rv.RES, '/home/OE/PurchaseOrders/2002/Sep/VJONES-20021009123337583PDT.xml')
        = 1;
```

[1] XQuery function `fn:data` is used here to atomize its argument, in this case returning the `XMLRef` node's typed atomic value.

Using Oracle XQuery Pragma ora:defaultTable

You can use Oracle XQuery extension-expression pragma `ora:defaultTable` to improve the performance of querying repository data.

Oracle XQuery extension-expression pragma `ora:defaultTable` lets you specify the default table used to store repository data that you query. The query is rewritten to automatically join the default table to view `RESOURCE_VIEW` and use Oracle SQL functions `equals_path` and `under_path` instead of XQuery functions `fn:doc` and `fn:collection`, respectively. The effect is thus the same as coding the query manually to use an explicit join and `equals_path` or `under_path`. Example 5-47 illustrates this; the query is rewritten automatically to what is shown in Example 5-46.

For clarity of scope Oracle recommends that you apply pragma `ora:defaultTable` directly to the relevant document or collection expression, `fn:doc` or `fn:collection`, rather than to a larger expression.

Example 5-47 Repository Query Using Oracle XQuery Pragma ora:defaultTable

```
SELECT XMLQuery('for $doc in (#ora:defaultTable PURCHASEORDER #)
                            {fn:doc("/home/OE/PurchaseOrders/2002/Sep/VJONES-20021009123337583PDT.xml")}
                  let $val := $doc/PurchaseOrder/LineItems/LineItem[@ItemNumber = 19]
                    return $val}'
                RETURNING CONTENT)
  FROM DUAL;
```