A Hitchhikers Guide to XQUERY SQL Developer and 11g

Tuesday, December 17th, 2013
1:30PM – 2:30PM

Coleman Leviter, OCP
Arrow Electronics/IOUG
IT Software Systems Engineer
cleviter@ieee.org

CV

- WMS Group – Fifteen years
- VAX Rewrite (.for) to UNIX (.c, .pc)
- VAX Forms to Oracle Forms
- TIFF file migration to Oracle
- XML Development
- WMS Development and Support
- IOUG Select Contributor
- IOUG Tips & Best Practices
- ODTUG Journal
- NYOUG WEB SIG Chair, Steering Committee
- IOUG Collaborate
- Oracle Open World
- IOUG Collaborate Conference Chair
- IOUG Board of Directors

Certified Professional

Presentation Objectives

- XQuery History
- Current Use
- Architecture, Applications
- Syntax
- Examples - Predefined Namespaces and Prefixes for XQuery
- FLWOR Overview
- FLWOR Examples
- Queries
- Load DOC Resources in Oracle
- TOAD, SQL*PLUS, SQL Developer (version 10g vs 11gR2)
- Production Examples - Replace Current Implementation

XQuery History

- XQuery is a query and functional programming language designed to query collections of XML data.
- XQuery 1.0 was developed by the XML Query working group of the W3C (w3c.org)
- XQuery 1.0 became a W3C Recommendation on January 23, 2007
- Oracle XML DB supports the latest version of the XQuery language specification, W3C XQuery 1.0 Recommendation. See http://www.w3c.org
- The mission of the XML Query project is to provide flexible query facilities to extract data from real and virtual documents on the World Wide Web, therefore finally providing the needed interaction between the Web world and the database world. Ultimately, collections of XML files will be accessed like databases.

Current Use

XQuery provides the means to extract and manipulate data from XML documents or any data source that can be viewed as XML, such as relational databases or office documents.

XQuery uses XPath expression syntax to address specific parts of an XML document. It supplements this with a SQL-like "FLWOR expression" for performing joins. A FLWOR expression is constructed from the five clauses after which it is named: FOR, LET, WHERE, ORDER BY, RETURN.

The language also provides syntax allowing new XML documents to be constructed. Where the element and attribute names are known in advance, an XML-like syntax can be used; in other cases, expressions referred to as dynamic node constructors are available.

All these constructs are defined as expressions within the language, and can be arbitrarily nested.

Architecture

The language is based on a tree-structured model of the information content of an XML document, containing seven kinds of node: document nodes, elements, attributes, text nodes, comments, processing instructions, and namespaces.

The type system of the language models all values as sequences (a singleton value is considered to be a sequence of length one).

The items in a sequence can either be nodes or atomic values.

Atomic values may be integers, strings, booleans, and so on: the full list of types is based on the primitive types defined in XML Schema.

XQuery 1.0 does not include features for updating XML documents or databases; it also lacks full text search capability. These features are both under active development for a subsequent version of the language. However, the new standards such as XQuery Update Facility 1.0 supports update feature and XQuery and XPath Full Text 1.0 support full text search in XML documents.

Applications

Examples using XQuery:

- Extracting information from a database for use in a web service.
- Generating summary reports on data stored in an XML database.
- Searching textual documents on the Web for relevant information and compiling the results.
- Selecting and transforming XML data to XHTML to be published on the Web.
- Pulling data from databases to be used for the application integration.
- Splitting up an XML document that represents multiple transactions into multiple XML documents.
- Shred XML Documents for relational table storage

Syntax

XQuery is case-sensitive - examples

XQuery elements, attributes, and variables must be valid XML names

An XQuery string value can be in single or double quotes

An XQuery variable is defined with a $ followed by a name, e.g. $house

XQuery comments are delimited by (: and :), e.g. (: XQuery Comment :)

Direct element constructor, curly braces {} delimit enclosed expressions, distinguishing them from literal text. Enclosed expressions are evaluated. Braces ({, }) are used to mark off an XQuery expression to be evaluated.

FLWOR expression has at least one “for” or “let” clause and a “return” clause; single “where” and “order by” clauses are optional.

XQuery Functions

ora:view lets you query existing database tables or views inside an XQuery expression, as if they were XML documents. In effect, ora:view creates XML views over the relational data, on the fly.

You can thus use ora:view to avoid explicitly creating XML views on top of relational data.

ora:view embeds each row in a ROW element

XQuery: case-sensitive (sql developer)

xquery for $i **In** ora:view("my_relat_table") return $i (: **In** s/b small in :)

(select * from my_relat_table)

SQL Error: ORA-19114: XPST0003 - error during parsing the XQuery expression: LPX-00801: XQuery syntax error at '**In**'

1 for $i In ora:view("my_relat_table") return $i - - sql developer

xquery for $i **in** ora:view ("my_relat_table") return $i

Example <ROW> Xquery (xquery_1)

select **xmlquery** ('for $i in ora:view("my_relat_table") return $i' RETURNING CONTENT) aa FROM DUAL ;

xquery for $i in ora:view("my_relat_table") return $i
(: observe **xmlquery** vs **xquery** :)

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 dbms treats XQuery expressions submitted with this command the same way it treats XQuery expressions in SQL functions XMLQuery and XMLTable.

Execution is identical, with the same optimizations.

```
SQL> xquery for $i in ora:view("my_relat_table") return $
 2  /
```

XQuery Functions (xquery_2)

Predefined Namespaces and Prefixes for XQuery

fn: interact with the local environment

ora: Oracle XML DB namespace

xs: XML Schema namespace

a) xs:integer - xquery xs:integer(23.4) = 23

b) ora:view - creates XML over relational table - without root node

c) xs:date("2013-08-25") - constructor function return

d) fn:string-to-codepoints("Thérèse") - returns a sequence of xs:integer values representing the Unicode code points.
xquery fn:string-to-codepoints("ML2 OEG")
xquery fn:string-to-codepoints("ML2 0EG")

e) fn:floor(-10.5) - smallest value = -11

f) concat($v1, $v2) - return $v1, $v2 concatenated

g) xquery fn:substring("motor car", 6)

Examples - SQL Developer (run script)

FLWOR (xquery_3)

"Watering the FLWOR"

For - Bind one or more variables each to any number of values

Let - Declares a variable and assigns it a value(s), can be a list, all at once

Where - Predicate, specifies criteria for filtering query results

Order By - Specifies the sort order of the result

Return - Defines the result to be returned

FLWOR expression has at least one "for" or "let" clause and a "return" clause;

Single "where" and "order by" clauses are optional.

Basic statement:

```
xquery "Hello World"  (: use SQL Developer or SQLPlus, run script :)
select xmlquery (' "Hello World" ' returning content ) aa from dual
```

FLWOR Examples (xquery_4)

```
SQL> xquery  for $i in (1 to 10) return ($i)
  2  /
  For - for $i in (1 to 10) return ($i)
  xquery for $i in (1 to 4), $j in (11 to 14)  return ($i * $j)
  Let – xquery let $x := 1 return <test>{$x}</test>
  xquery let $x := (1 to 5) return <test>{$x}</test>
  Where - for  $x in (1 to 5) where ($x < 3) return
          <test>{$x}</test>
Not valid for  let $x := (1 to 5) where ($x < 3) return
          <test>{$x}</test>


  Order By – xquery  for  $x in (1 to 5)
                where ($x < 3)
                order by $x descending
                return <test>{$x}</test>
```

FLWOR Examples (xquery_5)

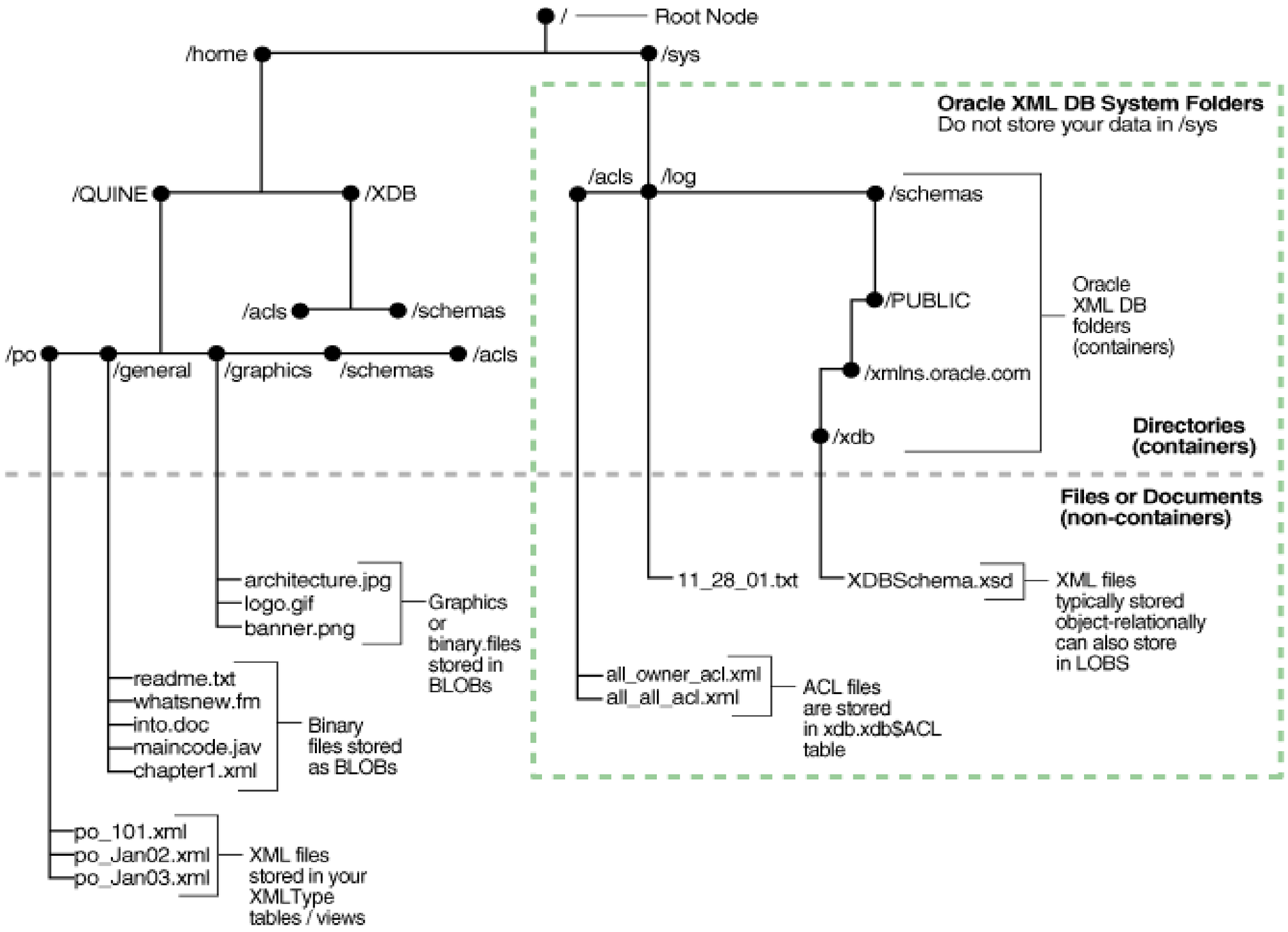
```
select xmlquery('for $i in  (: no root node F5 :)
  ora:view("scott","my_relat_table") return $i'
    returning content) from dual

xquery for $i in ora:view("scott","my_relat_table")
      where $i//NAME = "den"
      return $i

xquery for $i in 1 to 5
      for $j in 3 to 4
      return  <pair i="{$i}" j="{$j}" />

xquery    (: add root node :)  <result>{ for $i in 1 to 5
                                  for $j in 3 to 4
                                  return <pair i="{$i}" j="{$j}" />
                                     }</result>
```

FLWOR Result (xquery_6)

```
SELECT XMLQUERY('  (: run statement SQL Developer :)
                    <result>{
                    for $i in ora:view("my_relat_table") return $i
                   }</result>'
                   RETURNING CONTENT) aa from DUAL
```

```
SELECT XMLQUERY('  (: run statement SQL Developer :)
                     <result>{
                     for $i in ora:view("my_relat_table")
                     where $i/ROW/NAME eq "kitchen" return $i
                    }</result>'
                    RETURNING CONTENT) aa from DUAL
```

A Folder Tree, Showing Hierarchical Structures in the Repository*



* http://docs.oracle.com/cd/B28359_01/appdev.111/b28369/xdb16fol.htm

Load XML Document - Create Resource (xquery_7)

```
-- example creates “home” XML in resource

DECLARE

  res BOOLEAN;

  roomxmlstring VARCHAR2(2000):=
      '<?xml version="1.0"?
     ><root><ROW><NAME>kitchen</NAME><ITEM>sink</ITEM></ROW><ROW><NAME>kitchen</NAME><ITEM>table</ITEM>
     </ROW><ROW><NAME>kitchen</NAME><ITEM>counter</ITEM></ROW><ROW><NAME>kitchen</NAME><ITEM>microwav
     e</ITEM></ROW><ROW><NAME>kitchen</NAME><ITEM>oven</ITEM></ROW><ROW><NAME>kitchen</NAME><ITEM>ra
     nge</ITEM></ROW><ROW><NAME>kitchen</NAME><ITEM>refrigerator</ITEM></ROW><ROW><NAME>living_room</NAM
     E><ITEM>pictures</ITEM></ROW><ROW><NAME>living_room</NAME><ITEM>chair</ITEM></ROW><ROW><NAME>living
     _room</NAME><ITEM>television</ITEM></ROW><ROW><NAME>living_room</NAME><ITEM>couch</ITEM></ROW><ROW
     ><NAME>dining_room</NAME><ITEM>table</ITEM></ROW><ROW><NAME>dining_room</NAME><ITEM>etagere</ITEM><
     /ROW><ROW><NAME>dining_room</NAME><ITEM>chair</ITEM></ROW><ROW><NAME>dining_room</NAME><ITEM>cha
     ndelier</ITEM></ROW><ROW><NAME>den</NAME><ITEM>fireplace</ITEM></ROW><ROW><NAME>den</NAME><ITEM>t
     able</ITEM></ROW><ROW><NAME>den</NAME><ITEM>chair</ITEM></ROW><ROW><NAME>den</NAME><ITEM>lamp<
     /ITEM></ROW><ROW><NAME>den</NAME><ITEM>étagère</ITEM></ROW></root>';

  BEGIN
    res := DBMS_XDB.createResource('/public/room.xml',   roomxmlstring);
    IF res = TRUE THEN
          dbms_output.put_line('1');
    ELSE
          dbms_output.put_line('0');
    END IF;
  END;
  /
```

XML Document – Resource Queries (xquery_8)

```
   select * from path_view;          -- sql developer view all resource paths
-- requires data installed
-- VALID QUERY
    select xmlquery(' <root_result>{
                    for $e in fn:doc("/public/room.xml")//root/*
                    where $e//NAME eq "kitchen"
                    return $e
                    }</root_result>'
     returning content) aa from dual;

  -- show metadata for resource
SELECT r.RES.getCLOBVal()
FROM resource_view r
WHERE equals_path(r.res, '/public/room.xml') = 1;

 http://www.oracle-developer.net/display.php?id=416
 -- sql developer, remove resource
 select schema_url, schema
 FROM   dba_xml_schemas

 – remove resource
 DECLARE
   res BOOLEAN;
 BEGIN
    DBMS_XDB.DELETERESOURCE('/public/room.xml',   4 /*DELETE_RECURSIVE_FORCE*/);
 END;
```

FLWOR Examples: row, * (xquery_9 wait)

```
-- works, use script - room with ROW -

        select xmlquery(' <root_result>{
                    for $e in fn:doc("/public/room.xml")//root/ROW
                    order by $e/ITEM descending
                    return  <all_codes> {$e/ITEM}   </all_codes>
                    }</root_result>'
       returning content) aa from dual;
```

```
-- works use script - room with wildcard

        select xmlquery(' <root_result>{
                    for $e in fn:doc("/public/room.xml")//root/*
                    order by $e/ITEM descending
                    return  <all_codes> {$e//ITEM}   </all_codes>
                    }</root_result>'
       returning content) aa from dual;
```

FLWOR Examples: row, * (xquery_9)

```
select xmlquery(' <root_result>{
        for $e in fn:doc("/public/room.xml")//root/ROW
        let $room_name:= (<room_name> {$e/NAME/text() } </room_name>)
        where count($e/NAME) > 0
        order by $e/ITEM descending
        return  <EACH_ROOM_CONTAINS> {$e/ITEM} {$room_name}
                </EACH_ROOM_CONTAINS>

          }</root_result>'  returning content) aa from dual;
```

FLWOR Examples: attribute, sort (xquery_10)

```
-- works use script observe attribute notation

 SELECT XMLQuery(' <root_result>{
            for $e in fn:doc("/public/room.xml")//root/*
            return <room_code cname="{$e/NAME}" />
            }</root_result>'
RETURNING content) aa from dual;
```

```
--  works ROOM use script observe attribute notation, two asc.desc
SELECT xmlquery(' <root_result>{
            for $e in fn:doc("/public/room.xml")//root/*
            order by $e/NAME ascending, $e/ITEM descending
            return  <all_rooms room_name="{$e/NAME}"
                        item_name="{$e/ITEM}" />
            }</root_result>'
RETURNING content) aa from dual;
```

FLWOR Examples: namespace* (xquery_13)

Natural language and, therefore, tag names can be imprecise.
Two different tags can have identical names and yet hold entirely different sorts of information. Namespaces are intended to resolve any such sources of confusion.

```
– ns explicit – node not found
SELECT mxt.xmlcol, mxt.ref_id, -- working
XMLQuery( 'xquery version "1.0"; (: :)   (:  optional  :)
        declare  namespace obj2="http://www.w3.org/2001/XMLSchema/obj_2"; (: :)
        for $i in //obj2:object   (: working :)
        where $i/obj2:thing="apple"
        return <Root>
                        <name1 thing="{$i/obj2:thing}"/>
                </Root>'
        PASSING mxt.xmlcol RETURNING CONTENT) records
FROM my_xml_table mxt
WHERE existsnode(mxt.xmlcol, '//obj2:object/obj2:thing = "frisbee" ',
         'xmlns:obj2="http://www.w3.org/2001/XMLSchema/obj_2"') = 1;
```

*http://docs.oracle.com/cd/E17276_01/html/gsg_xml/java/xquery.html

FLWOR Examples: namespace (xquery_11)

```
<?xml version="1.0" encoding="UTF-8"?>
  <ns1:object_group
  xmlns:ns1="http://www.w3.org/2001/XMLSchema/sample_namespace_1
  " xmlns:obj1="http://www.w3.org/2001/XMLSchema/obj_1">
    <obj1:object0>
      <obj1:thing>ball</obj1:thing>
      <obj1:thing>key</obj1:thing>
      <obj1:thing>table</obj1:thing>
    </obj1:object>
  <obj2:object xmlns:obj2="http://www.w3.org/2001/XMLSchema/obj_2">
      <obj2:thing>frisbee</obj2:thing>
      <obj2:thing>bbq</obj2:thing>
      <obj2:thing>switch</obj2:thing>
    </obj2:object>
  </ns1:object_group>
```

FLWOR Examples: namespace (xquery_12)

– ns wildcard

```
select mxt.xmlcol, mxt.ref_id
from my_xml_table mxt
where existsnode(mxt.xmlcol, '//*:object/*:thing = "frisbee" ') = 1;
```

– ns explicit

```
select mxt.xmlcol, mxt.ref_id
from my_xml_table mxt
where existsnode(mxt.xmlcol, '//obj2:object/obj2:thing = "frisbee" ',
'xmlns:obj2="http://www.w3.org/2001/XMLSchema/obj_2"') = 1;
```

```
SELECT mxt.xmlcol, mxt.ref_id, -- working
   XMLQuery('xquery version "1.0"; (: :)   (:  optional  :)
   declare  namespace obj2="http://www.w3.org/2001/XMLSchema/obj_2"; (: :)
     for $i in //obj2:object   (: working :)
     (: where $i//obj2:thing="bbq"  :)
     return <Root>
           <name1 thing="{$i//obj2:thing}"/>
           </Root>'
      PASSING mxt.xmlcol RETURNING CONTENT) records
   FROM my_xml_table mxt
   WHERE existsnode(mxt.xmlcol, '//obj2:object/obj2:thing = "frisbee" ',
        'xmlns:obj2="http://www.w3.org/2001/XMLSchema/obj_2"') = 1;
```

FOR/LET (xquery_13)

```
-- difference between for and let

– let: $s is evaluated before return
select xmlquery (
        'let $s := (<one/>, <two/>, <three/>)
        return <out>{$s}</out>    '
        RETURNING CONTENT ) aa FROM DUAL
        <out>
                <one></one>
                <two></two>
                <three></three>
        </out>

-- for: $s is evaluated each time, i.e for - return
select xmlquery (
        'for $s in(<one/>, <two/>, <three/>)
        return <out>{$s}</out>    '
        RETURNING CONTENT ) aa FROM DUAL
        <out><one/></out>
        <out><two/></out>
        <out><three/></out>
```

FLWOR - FOR LET Example (xquery_14)

```
select xmlquery
      (' <root_result>{    (: items descending order :)
            for $e in fn:doc("/public/room.xml")//root/*
            let $f := (<test2/>, <test3/>, <test4/>)
            where $e//NAME gt "a"
            order by $e//ITEM descending
            return  <all_codes> {$e/NAME} {$e/ITEM} {$f}
</all_codes>
             }</root_result>'
returning content) aa from dual;
 xquery
            for $e in doc("/public/room.xml")//root
            let $f := (<test2/>, <test3/>, <test4/>)
            where $e gt "a"
            order by $e descending
            return  <all_codes> {$e} {$f}   </all_codes>
```

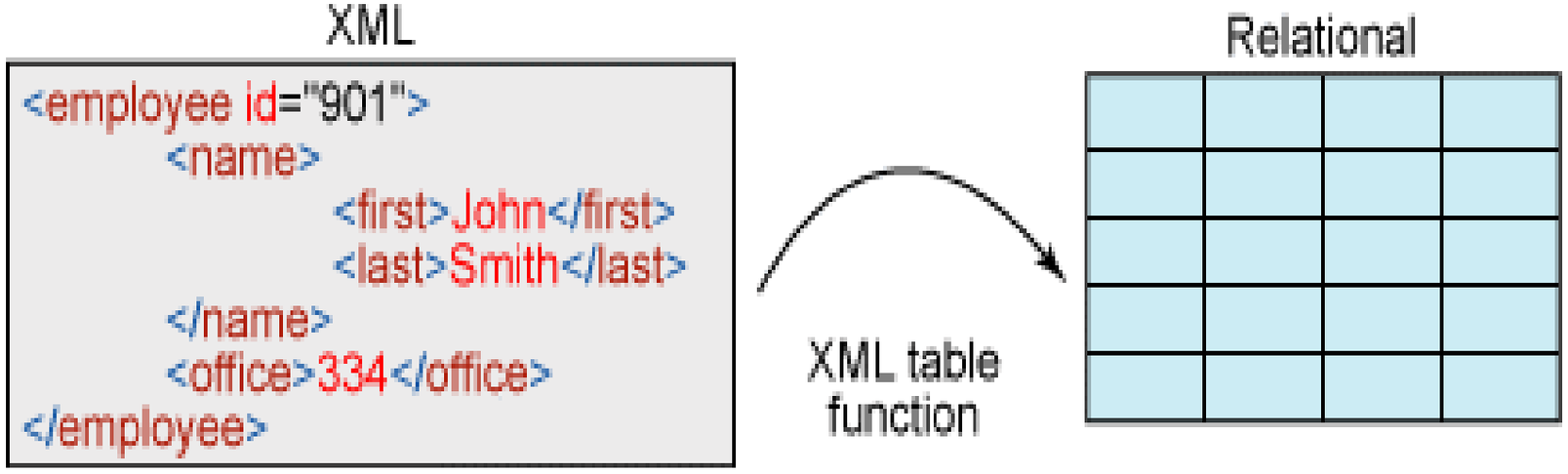
XMLTable (xquery_15)

Use XMLTABLE to decompose XML data into relational format

```
SELECT *
 FROM XMLTable('for $i in fn:doc("/public/room.xml"
                (: from XML DOC :)  )
 return $i')  ;


SELECT *
FROM XMLTable('for $i in ora:view("my_xml_table")
               (: from relational table :)
return $i')  ;

-- use COLUMNS on XML table to produce individual data items
   SELECT x.*
   FROM XMLTable(
              'for $i in ora:view("my_xml_table")
               return $i'
               COLUMNS
                           ref_id   VARCHAR(10) PATH '/ROW/REF_ID' ,
                           house  VARCHAR(15) PATH '//house') AS x
```

XML Document Shreding to Relational*

- XML Document stored in XMLTYPE object
- Using XQUERY XTABLE create relational table
- Shred XML Document



*http://www.ibm.com/developerworks/data/library/techarticle/dm-0708nicola/

XMLTable (xquery_16)

```
-- use COLUMNS on XML Document to produce individual data items
   http://www.ibm.com/developerworks/data/library/techarticle/dm-0708nicola/

   SELECT x.*
   FROM XMLTable(
        'for $i in doc("/public/room.xml")//ROW
         return $i'
         COLUMNS
                    "name"    VARCHAR(15) PATH 'NAME' ,
                     item      VARCHAR(15) PATH 'ITEM') AS x

-- finally, create a table from the XML Document stored as a resource
        CREATE TABLE del_room as
                    SELECT x.*
                    FROM XMLTable(
                            'for $i in doc("/public/room.xml")//ROW
                            return $i'
                            COLUMNS
                                        "name"   VARCHAR(15) PATH 'NAME' ,
                                         item     VARCHAR(15) PATH 'ITEM') AS x

        SELECT * FROM del_room
        DROP TABLE del_room
```

XMLTable – Create Relational Table (xquery_17)

-- or, create a table from the XML Document from a view

```
SELECT * FROM my_relat_table

 SELECT *
 FROM XMLTable('for $i in ora:view("my_relat_table")
                  (: from relational table :)
                    return $i')  ;
```

```
CREATE TABLE del_my_relat_table as
          SELECT x.*
          FROM XMLTable(
                              'for $i in ora:view("my_relat_table")
                               return $i'
                               COLUMNS
                                          "name"   VARCHAR(15) PATH '/ROW/NAME' ,
                                           item      VARCHAR(15) PATH '/ROW/ITEM') AS x

SELECT * FROM del_my_relat_table

DROP TABLE del_my_relat_table
```

XMLTable – Create Relational Table from XML Document (xquery_18)

```
create table del_house as
      select house_items.lcl_name, house_items.lcl_item
         from my_xml_table ,
                xmltable('for $i in //house/room
                   return $i'
                passing xmlcol
                     columns lcl_name varchar2(30) path 'name',
                                lcl_item  varchar2(30) path 'item' ) house_items
 WHERE ref_id = 36

SELECT * FROM del_house
DESCRIBE del_house
DROP TABLE del_house
```

Shred data using refcursor (xquery_19)

```
CREATE TABLE MY_RELAT_TABLE_XMLTABLE ( NAME  VARCHAR2(20 CHAR),ITEM  VARCHAR2(20
    CHAR) )
DECLARE
        lcl_name         VARCHAR2 (20 CHAR) ;
        lcl_ITEM         VARCHAR2 (20 CHAR)  ;
        lcl_stmt         VARCHAR2 (4000 CHAR ) ;
        lcl_cursor       SYS_REFCURSOR;
   BEGIN
          lcl_stmt := 'select data.name, data.item
                       from    xmltable("for $i in doc("/public/room.xml")/root/ROW
                             return $i"
                             columns
                             name varchar2(20) PATH "/ROW/NAME",
                             item varchar2(20) PATH "/ROW/ITEM") as data ';
      OPEN lcl_cursor FOR lcl_stmt;
          LOOP
             FETCH lcl_cursor INTO lcl_name, lcl_item;
             EXIT WHEN  lcl_cursor %NOTFOUND;
             INSERT INTO  my_relat_table_xmltable (name, item)
             VALUES (lcl_name, lcl_item);
          END LOOP;
      CLOSE lcl_cursor;
   COMMIT;
   END;

SELECT * from MY_RELAT_TABLE_XMLTABLE
```

Query using ROW (xquery_20)

```
xquery  (: sql developer only  ROW Element :)
for $e in doc("/public/room.xml")//root
return   $e
xquery  (: sql developer only  ROW omitted :)
for $e in doc("/public/room.xml")//root/ROW/*
return   $e

-- compare query, error in TOAD due to version  étagère
SELECT XMLQuery(' <root_result>{
                for $e in doc("/public/room.xml")/root/ROW
                return $e
                }</root_result>'
RETURNING CONTENT) aa FROM DUAL;

-- compare query, error in TOAD due to version étagère, child nodes
SELECT XMLQuery(' <root_result>{
                for $e in doc("/public/room.xml")//root/ROW/*
                return $e
                }</root_result>'
RETURNING CONTENT) aa FROM DUAL;
```

Hamlet – Table from XML:1 (xquery_hamlet_2)

```
select * from all_directories

CREATE DIRECTORY XML_DIR_DOS  AS 'c:\xml_example';

drop directory XML_DIR_DOS

DECLARE
   xbfile  BFILE;
   RET     BOOLEAN;
BEGIN
   xbfile := bfilename('XML_DIR_DOS', 'hamlet_xml.xml'); -- 279,639
   ret := DBMS_XDB.createResource('/public/hamlet.xml', xbfile);
   commit;
END;

select * from path_view where path like '%public%'
```

Hamlet – Table from XML:2 (xquery_hamlet_2)

```
DECLARE
  res BOOLEAN;
BEGIN
   DBMS_XDB.DELETERESOURCE('/public/hamlet.xml',   4
                                                    /*DELETE_RECURSIVE_FORCE*/);
   COMMIT;
END;

        select xmlquery(' <root_result>{
                        for $e in fn:doc("/public/hamlet.xml")//*
                        return $e
                        }</root_result>'
        returning content) aa from dual;

-- view entire resource
        SELECT r.RES.getCLOBVal()
        from resource_view r
        where equals_path(r.res, '/public/hamlet.xml') = 1;
```

Hamlet* – Table from XML:3 (xquery_hamlet_2)

```
select xmlquery(
'<html><head/><body>
 {
   for $act in fn:doc("/public/hamlet.xml")//ACT
   let $speakers := distinct-values($act//SPEAKER)
   return
     <div>
       <h1>{ string($act/TITLE) }</h1>
       <ul>
       {
         for $speaker in $speakers
         return <li>{ $speaker }</li>
       }
       </ul>
     </div>
 }
 </body></html>'
returning content) aa from dual;
```

*http://en.wikipedia.org/wiki/XQuery

Sources

http://en.wikipedia.org/wiki/XQuery

http://docs.oracle.com/cd/B28359_01/appdev.111/b28369/xdb16fol.htm

http://docs.oracle.com/cd/E14072_01/appdev.112/e10492/xdb_xquery.htm#

http://www.ibm.com/developerworks/data/library/techarticle/dm-0708nicola/

http://docs.oracle.com/cd/E17276_01/html/gsg_xml/java/xquery.html

Upcoming Conference - ioug.org

Collaborate 14 – Las Vegas - April 7-11, 2014

DBA
Development
Performance and scalability
Manageability
High Availability
Storage
Data Warehousing
Engineered Systems
Big Data
Business Intelligence
Cloud Computing

Questions

Coleman Leviter, OCP
Arrow Electronics/IOUG
IT Software Systems Engineer
cleviter@ieee.org