MONASH University
Information Technology

FIT3176 Advanced Database Design

Topic 9 Tutorial/Lab - XML in the Database

Task 1 - XPATH

Obtain a copy of the publications 3a XML document used last week, open this in oXygen and find the XPATH for the following (record the XPATH expression for each item requested):

- All direct child nodes of the root
- All direct child nodes of the book element
- All titles elements
- The text of all title elements
- The names of all hard cover books
- The book elements which have a price >= 100
- The title of all texts written by Patrick Carey
- In the publications document you have two books of hard cover type, select one of these and write the XPath to find the title for this book based on the type of cover (hard) and the price - write the XPath expression using a single predicate.

Task 2 - XML in Oracle

Obtain the schema file used in topic 3 for the Large Company model and recreate the necessary tables.

2.1 Generating XML data from relational data

Note, this week's lab has text available so that you can copy and paste to save time during the lab, please do not just 'blindly' copy and paste, understand what the commands are doing.

XMLELEMENT can be used to **generate XML data from a relational table** - the XMLELEMENT SQL function:

XMLELEMENT("Name", contents)

produces an XML element with name as its root and contents as the content. By nesting this command we can build a complete XML document.

For example using the BRAND table:

```
select xmlelement("Brand",
    xmlelement("Name", brand_name),
    xmlelement("Type", brand_type))
FROM
    brand;
```

Run this command and produce some rows of XML content, copy and paste the output to a text editor (use run script to get the output). Save one instance to a separate XML document and write a basic schema file to validate the document **use oXygen** for both the instance document and the schema.

There is also an XMLATTRIBUTE, XMLFOREST and XMLAGG functions **which can be used to generate XML data,** test the following scenarios to be familiar with the different approaches.

```
set echo on

-- XMLELEMENT with keyword NAME
SELECT XMLELEMENT(NAME "Department",
    XMLELEMENT(NAME "Name", DEPT_NAME),
    xmlelement(NAME "MailBox", dept_mail_box))
FROM
    department;

-- The keyword NAME is not required, add an attribute
SELECT XMLELEMENT("Department", XMLATTRIBUTES(dept_num as "deptno"),
    XMLELEMENT("Name", DEPT_NAME),
    xmlelement("MailBox", dept_mail_box))
FROM
    DEPARTMENT;

-- Use any select
SELECT XMLELEMENT("Department", XMLATTRIBUTES(d.dept_num as "deptno"),
    XMLELEMENT("Name", DEPT_NAME),
    XMLELEMENT("MailBox", DEPT_MAIL_BOX),
    XMLELEMENT("Manager", to_char(e.emp_num) || ' ' || emp_fname || ' ' ||
emp_lname))
FROM
    DEPARTMENT D,
    EMPLOYEE E
WHERE
    D.EMP_NUM = E.EMP_NUM;
```

```
SELECT XMLELEMENT("Department",
    XMLELEMENT ("No", D.DEPT_NUM),
    XMLELEMENT ("Name", DEPT_NAME),
    XMLELEMENT ("EmployeeCount", COUNT(*)))
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.DEPT_NUM = D.DEPT_NUM
GROUP BY D.DEPT_NUM, DEPT_NAME;

-- XMLFOREST is a simpler alternative saving the need to nest elements
SELECT XMLELEMENT(NAME "Department", XMLFOREST(DEPT_NUM AS "DeptNumber",
DEPT_NAME))
FROM DEPARTMENT D;

-- XMLAGG - what does this do?
SELECT XMLELEMENT("Departments",
    XMLAGG(XMLELEMENT(NAME "Department",
    XMLELEMENT(NAME "Name", DEPT_NAME),
    XMLELEMENT(NAME "MailBox", DEPT_MAIL_BOX))))
FROM
    DEPARTMENT;
```



```
set echo off
```


2.2 Register a schema

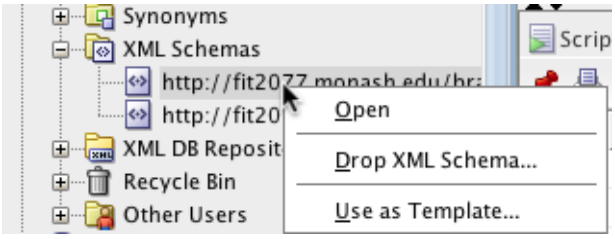
Details about the schemas that you currently have registered are available in the table user_xml_schemas. You can see the tables structure by using the command:


```
DESC user_xml_schemas
```

Due to the to the JDBC drivers (thin drivers) that SQL Developer uses we are not able to directly show LOB values, so we need to exclude the actual schema by using the command

```
SELECT
    schema_url,
    local,
    binary
FROM
    user_xml_schemas;
```

An alternative way of looking at/dropping your registered schema is by using the "XML Schemas" node in the left hand Connections panel of SQL Developer.



During the lecture a technique for registering a schema was introduced that works for all versions of Oracle. This approach requires the addition of appropriate triggers to enforce the schema for any inserted/updated data. In the recent versions of Oracle a new storage form for XML has been added called "binary storage" - storing a schema in this manner has significant benefits in that the triggers to validate inserts/updates against the schema are not required, the registered binary schema does this automatically.

To store a schema as a non-binary schema (the approach used in the lecture) we use:

```
-- Register a schema (non-binary)
BEGIN
DBMS_XMLSCHEMA.REGISTERSCHEMA(SCHEMAURL => 'schema URL goes here',
 schemadoc => 'schema content goes here',
 LOCAL => TRUE,
 GENTYPES => TRUE,
 GENBEAN => FALSE,
 GENTABLES => FALSE);
 end;
/
```


To store a schema as a binary schema (our preferred approach for FIT3176 prac work)

```
-- Register a schema (binary)
BEGIN
DBMS_XMLSCHEMA.REGISTERSCHEMA(SCHEMAURL => 'schema URL goes here',
 schemadoc => 'schema content goes here',
 LOCAL => TRUE,
 GENTYPES => FALSE,
 GENTABLES => FALSE,
 FORCE => FALSE,
 options => DBMS_XMLSCHEMA.REGISTER_BINARYXML);
 end;
/
```


Given our study to date, you should be able to recognise each of the above as anonymous PL/SQL blocks.

Using the BRAND schema that you created in oXygen, register it as both a non-binary and binary schema. The schema URL's (which you are free to modify if you wish) could be:

- Non binary: http://fit3176.monash.edu/brand.xsd
- Binary: http://fit3176.monash.edu/brandB.xsd

From the Oracle documentation: "SCHEMAURL – the XML schema URL. This is a unique identifier for the XML schema within Oracle XML DB. It is conventionally in the form of a URL,

but this is not a requirement. The XML schema URL is used with Oracle XML DB to identify instance documents, by making the schema location hint identical to the XML schema URL. Oracle XML DB never tries to access a Web server identified by the specified URL."

For your work with FIT3176 please ensure you use a conventional URL form, even though it is not a requirement.

We can drop a schema in one of two ways:

- Drop it via right click in the schema node of the left connections panel, or
- Via the DBMS_XMLSCHEMA packages DELETESCHEMA procedure (the preferred approach)

```
BEGIN
DBMS_XMLSCHEMA.DELETESCHEMA(
   'schema URL goes here',
   dbms_xmlschema.DELETE_CASCADE_FORCE);
END;
/
```

Take a few moments and drop and then recreate each of your schema - be sure that you are confident with this process before proceeding with the rest of the lab exercises. If you run into problems with testing any of the following ideas simply, drop everything (including the schemas) and recreate as needed.

Before running any of the further exercise it is recommend that you add the following to your script:


```
SET LINESIZE 32000;
SET PAGESIZE 40000;
SET LONG 50000;
```

2.3 Create an XMLType table and column and insert some data

The example below makes use of a CLOB store for the XML data - enter these commands and note the results:

```
create table BRAND_XML_TAB of xmltype
XMLTYPE STORE AS CLOB;


-- Test to run on each table type (note what occurs)

SELECT * FROM USER_XML_TABLES;

INSERT INTO BRAND_XML_TAB VALUES ('<Brand>
    <Name>Test Insert</Name>
    <Type>Test</Type>
</Brand>
');

-- NOT Well formed
INSERT INTO BRAND_XML_TAB VALUES ('<Brand>
    <Name>FORESTERS BEST</Names>
    <Types>VALUE</Type>
</Brand>
');

-- Well formed but different element Name
INSERT INTO BRAND_XML_TAB VALUES ('<Brand>
    <names>FIT3176</names>
    <Types>VALUE</Types>
</Brand>
');

-- add multiple rows
INSERT INTO BRAND_XML_TAB
select xmlelement("Brand",
    xmlelement("Name", brand_name),
    xmlelement("Type", brand_type)).getStringVal()
FROM
    BRAND;

-- Output (object) is character
-- object_value is a psuedo column
SELECT object_value FROM BRAND_XML_TAB;


-- End of test to run on each table
```

In inserting data you may see an error - "String literal too long" - this occurs because an SQL insert string is limited to 4000 characters. You can work around this issue by inserting the document via a PL/SQL anonymous block:

```
DECLARE
 xmlval clob := '... your XML document goes here ....';
BEGIN
  INSERT INTO tablename VALUES (......,XMLTYPE(xmlval));
END;
```

In particular note the way each of the XML documents you inserted by the different techniques is presented on select (for a CLOB white space is maintained).

Repeat the steps on page 6 but this time create the table as BINARY XML i.e., use the syntax:

```
CREATE TABLE BRAND_XML_TAB OF XMLTYPE
XMLTYPE STORE AS BINARY XML;
```

This is the default storage type used for XMLType. If you do not specify a type BINARY XML will be used in the **current** versions of Oracle. Despite this being the case please ***ensure you always specify the storage type***.

Both of the last two tables you created were not attached to schemas. Create a new table attached to your non binary schema. To attach a schema to a table use:


```
-- use non binary schema for the CLOB table
CREATE TABLE BRAND_XML_TAB OF XMLTYPE
XMLTYPE STORE AS CLOB
XMLSCHEMA
  "schema URL goes here"
  ELEMENT "Brand";
```


Repeat the steps on page 6 and again note what happens with each test.

Create a new table attached to your binary schema:

```
-- schema must be registered for binary usage
CREATE TABLE BRAND_XML_TAB OF XMLTYPE
XMLTYPE STORE AS BINARY XML
XMLSCHEMA
  "schema URL goes here"
  ELEMENT "Brand";
```

Repeat the steps on page 6 and again note what happens with each test.

2.4 Select XML data

```
-- object_value pseudocolumn = value of a row in an object table
SELECT OBJECT_VALUE FROM BRAND_XML_TAB B;

-- getClobVal() returns xmltype as CLOB
-- note correlation variable B here is *required* to execute xmltype
-- member functions
SELECT B.OBJECT_VALUE.GETCLOBVAL() FROM BRAND_XML_TAB B;

-- getStringVal
SELECT B.OBJECT_VALUE.GETSTRINGVAL() FROM BRAND_XML_TAB B;

-- EXTRACT function, return is XML
SELECT EXTRACT(B.OBJECT_VALUE,'/Brand/Type').GETSTRINGVAL() FROM
BRAND_XML_TAB B;

-- EXTRACTVALUE function
SELECT EXTRACTVALUE(B.OBJECT_VALUE,'/Brand/Type') FROM BRAND_XML_TAB B;
```


XMLQUERY returns XMLData - the thin drivers which SQL Developer use prevent the display of this data (you will see "Only LOB or String Storage is supported in Thin XMLType"). To deal with this limitation we add `getStringVal()` to each output display the XML output as string.


```
-- XMLQUERY Query XML data - returns xmldata
-- FLWOR (pronounced "flower" -for -let [-where -order by] -return
-- for: analogous to FROM
-- let: analogous to SET
-- where: filters for and let
-- order by: arranges output
-- return: constructs a result from above
-- Used to execute XQUERY inside an XML statement
SELECT XMLQUERY('for $b in /Brand/Type
                 return $b/text()'
                 PASSING OBJECT_VALUE
                 RETURNING CONTENT).getStringVal() AS BRANDS
FROM BRAND_XML_TAB;

-- Note all rows appear, if not premium then display null
SELECT XMLQUERY('for $b in /Brand
                 where $b/Type[text()="PREMIUM"]
                 return $b/Name/text()'
                 PASSING OBJECT_VALUE
                 RETURNING CONTENT).getStringVal() AS PREMIUM_BRANDS
FROM BRAND_XML_TAB;
```

```
--XMLEXISTS
SELECT XMLQUERY('for $b in /Brand/Type/text()
                 return $b'
                 PASSING OBJECT_VALUE
                 RETURNING CONTENT).getStringVal() AS PREMIUM_BRANDS
FROM BRAND_XML_TAB
where XMLEXISTS('/Brand/Type[text()="PREMIUM"]' passing object_value);


-- Using return to build a different XML tag
SELECT XMLQUERY('for $b in /Brand
                 where $b/Type[text()="PREMIUM"]
                 return
                    <Newtag>{$b/Name/text()}</Newtag>'
                 PASSING OBJECT_VALUE
                 RETURNING CONTENT) AS PREMIUM_BRANDS
FROM BRAND_XML_TAB
where XMLEXISTS('/Brand/Type[text()="PREMIUM"]' passing object_value);
```


2.5 Using XMLTable to represent XML data as relational data

Run the following command to select data from your table as standard relational row and column data


```
SELECT OB.BRANDTYPE,
            OB.BRANDNAME
     FROM BRAND_XML_TAB B,
          XMLTABLE('$r/Brand'
                      PASSING B.OBJECT_VALUE AS "r"
                      COLUMNS BRANDTYPE VARCHAR(10) PATH 'Type',
                              BRANDNAME VARCHAR(50) PATH 'Name') AS OB;
```


Note the output which is produced. The attributes listed here could, for example, be used as part of a join to another relational table, or any such standard SQL command:

```
SELECT ob.brandtype, count(*) as "Count of BrandTypes"
     FROM BRAND_XML_TAB B,
          XMLTABLE('$r/Brand'
                      PASSING B.OBJECT_VALUE AS "r"
                      COLUMNS brandtype VARCHAR(10) path 'Type') AS ob
    GROUP BY ob.brandtype
    order by ob.brandtype;
```