Table of Contents

Oracle has a native data type called **xmltype** which stores well-formed XML. The alternative in the past was to store the XML as a string (varchar2 or clob) and you had to handle all of the XML rules for the data being stored. That can make sense if you are simply storing the data 'raw"- and using the database table as a storage container. But if you want to work with the inner structures of the XML document, it is better to have a native xml type.

We can use the Oracle XmlType as the data type for a column in a table, as the data type for a table, and we can use it as a data type in PL/SQL. The first thing to understand about this is that xml data is considered to be an object in Oracle. It has an internal structure that can be accessed. So the syntax will vary a bit from the SQL syntax that we are used to- but it is not a lot different.

We will start with creating three tables that can store XML data using the native XML data type.

- XmlAnimals will have an integer column (pk) and an xml column
- XmlBooks will have an integer column (pk) and an xml column
- XmlOnlyAnimals will be an xml only table

The pk column in the first two tables is mostly used as a way to filter for specific rows to display for certain techniques. The pk value is not related to the data in the XML column.

1. Creating a table with an XML column

1.1. Create

Demo 01: create table with an xml column

```
create table XmlAnimals (
  id    number(5) primary key
, anXmlData xmltype);
```

Here we just use the keyword xmltype as the data type for the column. The table has a numeric primary key to make it easier to deal with the rows. anXmlData is not a key word- it is just the name I use for that column.

1.2. Describe

Demo 02:

```
desc XmlAnimals
```

```
Name                                      Null?    Type
----------------------------------------- -------- ------------
ID                                        NOT NULL NUMBER(5)
ANXMLDATA                                          PUBLIC.XMLTYPE
```

1.3. Inserting data into XmlAnimals

Demo 03: insert data

```
insert into XmlAnimals  (id, anXmlData) values (
1,
xmltype.createXml(
'<animal>
 <an_id>136</an_id>
 <an_type>bird</an_type>
 <an_name>ShowBoat</an_name>
 <an_price>75</an_price>
 <an_dob>2000-01-15</an_dob>
 </animal>
')
);
```

There is a method, createXml, that is part of the xmltype that creates an xml data value from a string. The above insert has two values- 1 for an id and the xml data for the second column. Since whitespace outside of the elements is not significant in our use of xml, I can build the xml string one element at a time. Note the quote delimiters for the string. The method can be invoked as

```
xmltype.createXml('MyStringLiteral')
```

You know that Oracle does a lot of implicit casting for you. You can use that here where the string is implicitly cast to xml because the column's data type is xmltype.

```
insert into XmlAnimals  (id, anXmlData) values (
2,
'<animal>Fluffy</animal>'
);
```

If you do a Select * from the table, you will get a result set such as this. The second column will be very wide. The XML values are each valid, but they do not have the same pattern.

```
SQL> select * from XmlAnimals;

        ID
----------
ANXMLDATA
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
         1
<animal>
 <an_id>136</an_id>
 <an_type>bird</an_type>
 <an_name>ShowBoat</an_name>
 <an_price>75</an_price>
 <an_dob>2000-01-15</an_dob>
 </animal>

         2
<animal>Fluffy</animal>


2 rows selected.
```

Demo 04: Try doing an insert where the xml string is not valid xml. The string will be rejected. You should be able to see the error.

```
insert into XmlAnimals  (id, anXmlData) values (3,'<animal>Fluffy</Animal>');
```

```
insert into XmlAnimals  (id, anXmlData) values (3,'<animal>Fluffy</Animal>')
            *
ERROR at line 1:
ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00225: end-element tag "Animal" does not match start-element tag "animal"
Error at line 1
```

```
insert into XmlAnimals  (id, anXmlData) values (3,'<animal>Fluffy<animal>');
```

```
insert into XmlAnimals  (id, anXmlData) values (3,'<animal>Fluffy<animal>')
            *
ERROR at line 1:
ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00007: unexpected end-of-file encountered
```

Demo 05: This animal has two names- how would you handle that in a regular database table?
We can use the id column in a filer as before.

```
insert into XmlAnimals  (id, anXmlData) values
(3,'<animal><an_name>Fluffy</an_name><an_name>Snookums</an_name></animal>');
```

1 row created.

```
SQL> select * from XmlAnimals where id = 3;

        ID
----------
ANXMLDATA
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
-------------------------------------
         3
<animal><an_name>Fluffy</an_name><an_name>Snookums</an_name></animal>

1 row selected.
```

Demo 06: More invalid inserts

```
insert into XmlAnimals  (id, anXmlData) values
(4,'<animal>Fluffy</animal><animal>Snookums</animal>');
```

```
insert into XmlAnimals  (id, anXmlData) values
(4,'<animal>Fluffy</animal><animal>Snookums</animal>')
            *
ERROR at line 1:
ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00245: extra data after end of document
Error at line 1
```

```
insert into XmlAnimals  (id, anXmlData) values
(5,'<animal>Fluffy</animal><pet><an_price>75</an_price></pet>');
```

```
insert into XmlAnimals  (id, anXmlData) values
(5,'<animal>Fluffy</animal><pet><an_price>75</an_price></pet>')
            *
ERROR at line 1:
ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00245: extra data after end of document
Error at line 1
```

What we want now is rows that do have the same pattern. If we had associated this column with an XML schema, we could enforce a pattern for the data. Since we are not going into that topic, we will just take care that our data follows a pattern that an animal element has one of each of the following subelements: an_id, an_type, an_name (all of those are strings), an_price (a number) and an_dob (a date value). The date value literal will use an Xml date format YYYY-MM-DD

Demo 07: Delete rows to clear the table. This is the same as any SQL delete.

```
delete from XmlAnimals ;
```

Demo 08: insert three rows

```
insert into XmlAnimals  (id, anXmlData) values (1,
xmltype.createXml(
'<animal>
 <an_id>136</an_id>
 <an_type>bird</an_type>
 <an_name>ShowBoat</an_name>
 <an_price>75</an_price>
 <an_dob>2000-01-15</an_dob>
 </animal>
')
);

insert into XmlAnimals  (id, anXmlData) values (9,
'<animal>
 <an_id>137</an_id>
 <an_type>bird</an_type>
 <an_name>Mr. Peanut</an_name>
 <an_price>150</an_price>
 <an_dob>2008-01-12</an_dob>
 </animal>');

insert into XmlAnimals  (id, anXmlData) values (14,
'<animal>
    <an_id>1201</an_id>
    <an_type>cat</an_type>
    <an_name>Ursula</an_name>
    <an_price>500</an_price>
    <an_dob>2007-12-15</an_dob>
  </animal>');
```

1.4. Displaying the data

Demo 09: Start with a plain select. I have reformatted the data a bit to fit on the page.
Start with the set commands shown or the output may be truncated

```
set Long 999999
set feedback 1
select * from XmlAnimals ;
```

```
   ID      ANXMLDATA
-----      -------------------------------------
    1      <animal>
            <an_id>136</an_id>
            <an_type>bird</an_type>
            <an_name>ShowBoat</an_name>
            <an_price>75</an_price>
            <an_dob>2000-01-15</an_dob>
            </animal>

    9      <animal>
            <an_id>137</an_id>
            <an_type>bird</an_type>
            <an_name>Mr. Peanut</an_name>
            <an_price>150</an_price>
            <an_dob>2008-01-12</an_dob>
            </animal>

   14      <animal>
               <an_id>1201</an_id>
               <an_type>cat</an_type>
               <an_name>Ursula</an_name>
               <an_price>500</an_price>
               <an_dob>2007-12-15</an_dob>
              </animal>
```

2. The second demo table

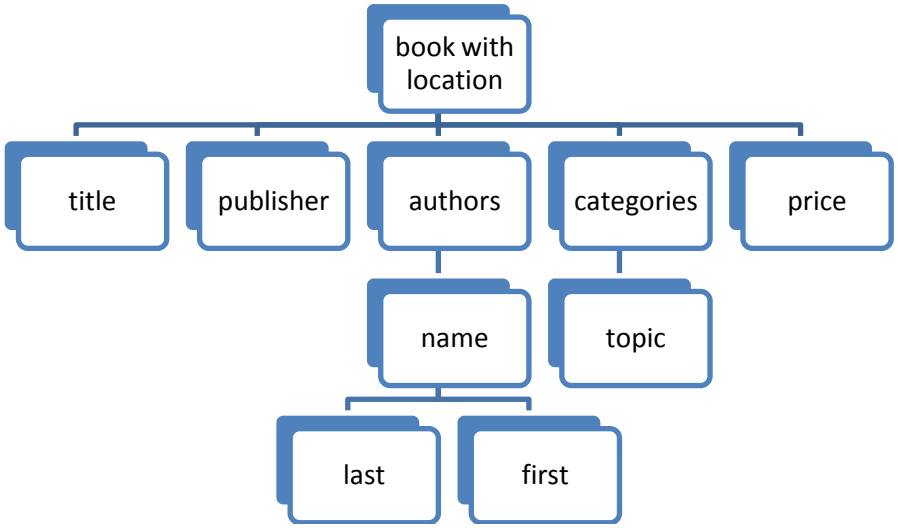
This set of XML has a deeper structure. For each book, we start with a publisher with a name; then we have a book element with a book name; then an author element with possible multiple author names with a first and last name element, then a categories element with possible multiple topics. The book element has an attribute named location. This is storing the same type of data we stored in multiple tables using traditional database tables.

There is another document which has a generated schema for this xml document.

This is one XML string

```
<book location="275">
  <title>SQL is Fun</title>
  <publisher>Addison Wesley</publisher>
  <authors>
    <name>
      <last>Collins</last>
      <first>Joan</first>
    </name>
    <name>
      <last>Effingham</last>
      <first>Jill</first>
    </name>
  </authors>
  <categories>
    <topic>SQL</topic>
    <topic>DB</topic>
  </categories>
  <price>29.95</price>
</book>
```

You can use this picture to help see the various paths through the data. Each of these boxes can be considered an element node in the tree



Demo 10: create the table

```
create table XmlBooks (
  id    int primary key
, book_xml xmltype)
;
```

Demo 11: These are the first two inserts- see the demo for the rest- there are 10 inserts.

```
insert into XMLBooks values (1,
'<book  location="275">
   <title>SQL is Fun</title>
   <publisher>Addison Wesley</publisher>
   <authors>
      <name>
         <last>Collins</last><first>Joan</first>
      </name>
      <name>
         <last>Effingham</last><first>Jill</first>
      </name>
   </authors>
   <categories>
      <topic>SQL</topic>
      <topic>DB</topic>
   </categories>
   <price>29.95</price>
</book>'
);

insert into XMLBooks values (2,
'<book  location="275">
   <title>Databases: an Introduction</title>
   <publisher>Addison Wesley</publisher>
   <authors>
      <name>
         <last>Malone</last>
         <first>Mike</first>
      </name>
      <name>
         <last>Effingham</last>
         <first>Jill</first>
      </name>
   </authors>
   <categories>
      <topic>SQL</topic>
      <topic>DB</topic>
      <topic>XML</topic>
   </categories>
   <price>79.95</price>
</book>'
);
```

Demo 12: Displaying the table- this is a partial display

```
select * from XmlBooks;
```

SQL Plus

```
SQL> select * from XmlBooks;

        ID
BOOK_XML
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------

         1
<book  location="275">
   <title>SQL is Fun</title>
   <publisher>Addison Wesley</publisher>
   <authors>
      <name>
         <last>Collins</last><first>Joan</first>
      </name>
      <name>
         <last>Effingham</last><first>Jill</first>
      </name>
   </authors>
   <categories>
      <topic>SQL</topic>
      <topic>DB</topic>
   </categories>
   <price>29.95</price>
</book>

         2
<book  location="275">
   <title>Databases: an Introduction</title>
   <publisher>Addison Wesley</publisher>
   <authors>
      <name>
         <last>Malone</last><first>Mike</first>
      </name>
      <name>
         <last>Effingham</last><first>Jill</first>
      </name>
   </authors>
   <categories>
      <topic>SQL</topic>
      <topic>DB</topic>
      <topic>XML</topic>
   </categories>
   <price>79.95</price>
</book>

         3
<book  location="313">
   <title>C# and MySQL</title>
   <publisher>Addison Wesley</publisher>
   <authors>
      <name><last>Horn</last><first>Shirley</first></name>
      <name><last>Cocker</last><first>Joe</first></name>
      <name><last>Shorter</last><first>Wayne</first></name>
   </authors>
   <categories>
      <topic>SQL</topic>
      <topic>DB</topic>
      <topic>XML</topic>
      <topic>C#</topic>
      <topic>MySQL</topic>
   </categories>
</book>

         4
<book location="493:75">
   <title>Databases- Design and Logic</title>
   <publisher>McGraw Hill</publisher>
   <authors>
      <name><last>LaVette</last><first>Betty</first></name>
   </authors>
   <categories>
   </categories>
   <price>185.95</price>
</book>

         5
<book location="550:0">
   <title>The truth about everything</title>
   <publisher>Addison Wesley</publisher>
   <authors>
      <name><last>Del Ansem</last></name>
   </authors>
   <categories>
      <topic></topic>
   </categories>
   <price>79.95</price>
</book>
```

3. Creating an XML table

The previous tables had an integer column and an xml column. We can also create a table that stores only xml- one xml value per row.

3.1. Create

xmltype is an object type so we can create a table of that type, rather than just defining a column of the xml type.

Demo 13:

```
create table XmlOnlyAnimals of XmlType;
```

3.2. Describe

Demo 14: Describing the table

```
desc XmlOnlyAnimals
```

```
Name                                      Null?    Type
----------------------------------------- -------- ------------
TABLE of PUBLIC.XMLTYPE
```

If you do this in SQL Developer, the display is a bit different

```
Name                          Null     Type
----------------------------- -------- --------------------------------
SYS_NC_ROWINFO$                        XMLTYPE()
```

3.3. Inserting data into XmlOnlyAnimals

Use the same inserts as before except insert one value only; we do not have an ID column in this table. We also do not have a column name. You can use the createXml method or let Oracle handle the cast.

Demo 15: I am inserting three rows.

```
insert into XmlOnlyAnimals values (
xmltype.createXml(
'<animal>
 <an_id>406</an_id>
 <an_type>bird</an_type>
 <an_name>ShowBoat</an_name>
 <an_price>75</an_price>
 <an_dob>2000-01-15</an_dob>
 </animal>
')
);

insert into XmlOnlyAnimals values (
'<animal>
 <an_id>407</an_id>
 <an_type>bird</an_type>
 <an_name>Mr. Peanut</an_name>
 <an_price>150</an_price>
 <an_dob>2008-01-12</an_dob>
 </animal>')
;
```

```
insert into XmlOnlyAnimals values (
'<animal>
 <an_id>401</an_id>
 <an_type>cat</an_type>
 <an_name>Ursula</an_name>
 <an_price>500</an_price>
 <an_dob>2007-12-15</an_dob>
 </animal>')
;
```

3.4. Displaying the data

Demo 16: Note the name of the column for the xml data; this is a system name

```
select *
from XmlOnlyAnimals;
```

```
SYS_NC_ROWINFO$
------------------------------------
<animal>
 <an_id>406</an_id>
 <an_type>bird</an_type>
 <an_name>ShowBoat</an_name>
 <an_price>75</an_price>
 <an_dob>2000-01-15</an_dob>
 </animal>

<animal>
 <an_id>407</an_id>
 <an_type>bird</an_type>
 <an_name>Mr. Peanut</an_name>
 <an_price>150</an_price>
 <an_dob>2008-01-12</an_dob>
 </animal>

<animal>
 <an_id>401</an_id>
 <an_type>cat</an_type>
 <an_name>Ursula</an_name>
 <an_price>500</an_price>
 <an_dob>2007-12-15</an_dob>
 </animal>

3 rows selected.
```

Demo 17: We can use the term object_value to refer to this column which is easier to remember and easier to type

```
select object_value
from XmlOnlyAnimals;
```