Table of Contents

So far, we have seen xml and inserted it into tables but we have not done anything useful with it. To do anything with the contents of the xml values, we need to query the data and that requires

- some methods to get to the data (Extract, ExtractValue, ExistsNode).
- a knowledge of how to write a path telling the system how to get to the part of the data we want.
- writing tests along that path.

This document is the crux of this unit and it takes some time to understand- this is a different way to think of data. Some of the output will be from SQL*Plus and some from SQL Developer.

1. Querying the XML data

XQuery is a language used to query XML data. XQuery is part of the W3C standards.

Oracle supports techniques that provide similar functionality.

XPath is part of XQuery and is used in querying XML data. When you use traditional relational SQL queries, you access data by using the tableName.ColumnName syntax (such as vt_animals. an_type). This SQL query syntax is designed for two-dimensional table access.

XML data is tree-structured so you access data values by specifying the path through the tree to the data. XML data is structured differently than relational data so the access paths will be different.

Node: An XML document is composed of nodes organized in a hierarchical/tree fashion. With the XML documents we are using only some types of nodes.

- The entire document is a document node
- Every XML element is an element node
- The text in the XML elements are text nodes
- Every attribute is an attribute node

The plan for this document:

- We are going to start by looking at a few simple paths through our xml document..
- We will use the path with the Extract()function.
- Then we will look at various paths to get to the data we want.
- After that, we will add predicates to the path; the predicates act as filters

Then we will look at two other functions

- ExtractValue() to get the scalar data out of the XML document
- ExistsNode() which we use as a test for data.

2. Simple XPath expressions

The paths used by XML are similar to the paths used in hierarchical file systems. We start at the root and navigate down the notes.

The symbol `/` has two meanings. It can identify the root of the XML tree. It is also used as a path separator.

The symbol `//` refers to all descendents of the current node.

The symbol `.` refers the current node in the tree structure.

The symbol `..` refers to going up one level in the tree structure.

The first function we will use is the extract () function. The function is passed an xml expression and an XPath string expression. The syntax is extract (xmlData, 'path as a string literal')

Demo 01: Here are several examples of queries that use the extract function (discussed in the next section) to illustrate several path expression with our tables. I am using a Where clause to get only one row from the table. I will not show the column headers for these.

Starts at the root; we get an xml document starting with the animal element.

```
select extract(anXmlData,'/animal') as "XMLData"
from XmlAnimals
where id = 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>

1 row selected.
```

Since animal is the top level element, you could skip the leading / but it is probably clearer with it. This is considered a relative path.

```
select extract(anXmlData,'animal') as "XMLData"
from XmlAnimals
where id = 1;
```

Now I walk down the path from animal to an_name. I get the xml elements for the animal's name.

```
select extract(anXmlData,'/animal/an_name') as "XMLData"
from XmlAnimals
where id = 1;
```

```
<an_name>ShowBoat</an_name>

1 row selected.
```

Demo 02: This uses the // symbol to go down the path until you get to the categories element. This book has one categories element with two topic subelements.

```
select extract(book_xml,'//categories') as "XMLData"
from XmlBooks
where id = 1;
```

```
<categories><topic>SQL</topic><topic>DB</topic></categories>

1 row selected.
```

What happens if I use //name? I got the author name - there are two authors.

```
   select extract(book_xml,'//name') as "XMLData"
   from XmlBooks
   where id = 1;
```

```
<name><last>Collins</last><first>Joan</first></name><name><last>Effingham</last>
<first>Jill</first></name>
1 row selected.
```

This says to get the authors' last name.

```
   select extract(book_xml,'book/authors/name/last') as "XMLData"
   from XmlBooks
   where id = 1;
```

```
<last>Collins</last><last>Effingham</last>
```

This also says to get the authors' last name because there is only one element in the xml with the element name last.

```
   select extract(book_xml,'//last') as "XMLData"
   from XmlBooks
   where id = 1;
```

```
<last>Collins</last><last>Effingham</last>
```

Demo 03:   data that is there and data that is not there

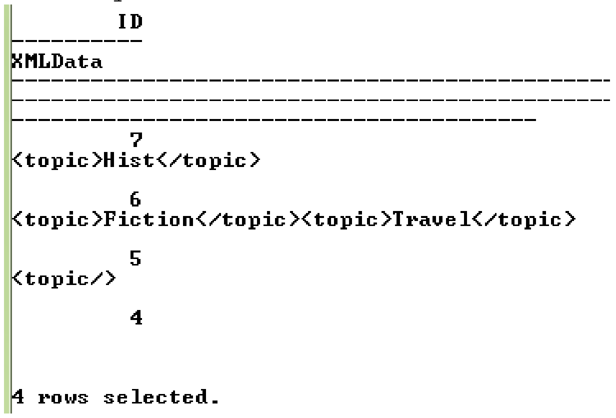
You can filter on the id column to get several rows.

```
   select extract(book_xml,'//last') as "XMLData"
   from XmlBooks
   where id between 2 and 5;
```

```
<last>Malone</last><last>Effingham</last>
<last>Horn</last><last>Cocker</last><last>Shorter</last>
<last>LaVette</last>
<last>Del Ansom</last>

4 rows selected.
```

What if we look for topics? Compare these to the inserts for these rows.

```
   select id, extract(book_xml,'//topic') as "XMLData"
   from XmlBooks
   where id between 4 and 7
   order by id desc;
```



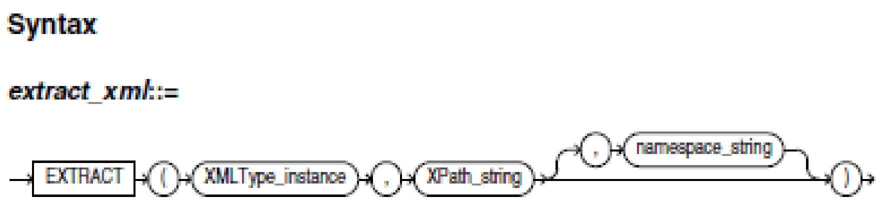
```
        ID
----------
XMLData
--------------------------------------------------
--------------------------------------------------
------------------------------------------
         7
<topic>Hist</topic>

         6
<topic>Fiction</topic><topic>Travel</topic>

         5
<topic/>

         4


4 rows selected.
```

3. The extract Function

The first technique we will look at is the extract() function. You probably have a good feeling for this function by now. You provide an xml source- I have been using the xmltype column in the table and an XML path expression as a string literal' to the element you want returned as XML.

The previous set of demos often used the ID column in the table to get a single row- that was an integer column. If I do not have a Where filter, then I get all of the rows. The Select clause contains the function so that determines what is returned with each row.

Sometimes the row returned will be an empty result. The SQL function extract returns null if its XPath-expression argument targets no nodes. An error is never raised if no nodes are targeted.



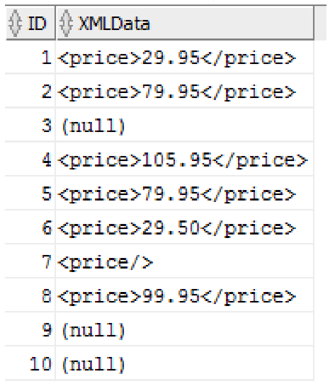
Demo 04:   We get one row in the result set for each row in the table.

```
select extract(book_xml,'//publisher') as "XMLData"
from XmlBooks;
```

```
<publisher>Addison Wesley</publisher>
<publisher>Addison Wesley</publisher>
<publisher>Addison Wesley</publisher>
<publisher>McGraw Hill</publisher>
<publisher>Addison Wesley</publisher>
<publisher>Addison Wesley</publisher>
<publisher>Addison Wesley</publisher>
<publisher>Jones Books</publisher>
<publisher>Addison Wesley</publisher>
<publisher>McGraw Hill</publisher>

10 rows selected.
```

Demo 05:   Book id 4 had an empty price element and that is what we get. If that book had no price element, then that column would be empty. select id, substr(extract(book_xml,'//price'), 1,22) as "XMLData"

```
select id, extract(book_xml,'//price') as "XMLData"
from XmlBooks;
```



ID	XMLData
1	`<price>29.95</price>`
2	`<price>79.95</price>`
3	(null)
4	`<price>105.95</price>`
5	`<price>79.95</price>`
6	`<price>29.50</price>`
7	`<price/>`
8	`<price>99.95</price>`
9	(null)
10	(null)

4. Bad path expressions

Demo 06: If you use an XMLPath expression that does not work with your XML document, you get empty rows. You do not get an error. I am using a Where clause just to reduce the number of rows returned. These all return three empty rows.

```
select extract(book_xml,'/pubisher/name') as "XMLData"
from XmlBooks
where id in (1,2,3);
```

```
-------------------------




3 rows selected.
```

```
select extract(book_xml,'/last') as "XMLData"
from XmlBooks
where id in (1,2,3);

select extract(book_xml,'//Last') as "XMLData"
from XmlBooks
where id in (1,2,3);
```

5. Including a predicate/test on the path

We can include a test on the path by including the test in square brackets. XML has a somewhat different syntax for writing tests; this is an xml test included in an sql query. The test is case specific.

The test is written inside square brackets [ ] and is written along the path. In the first example, we are testing the an_name which is on the animal path. The string literal is in double quotes.

The predicate becomes: an_name = "Ursula"] Check if the first an_name element value is the string "Ursula"

In the books table, we can have multiple topics; we can use [1] as an index.

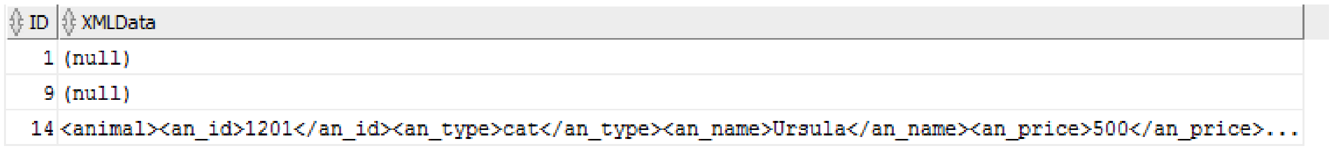
If the XPath-expression argument targets no nodes, a null is returned.

For not equals use !=; for inequality use > <

Demo 07:

We have one animal named Ursula.

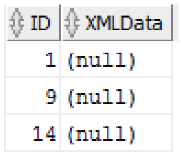
```
select id, extract (anXmlData, '/animal[an_name = "Ursula"]') as "XMLData"
from XmlAnimals;
```



ID	XMLData
1	(null)
9	(null)
14	<animal><an_id>1201</an_id><an_type>cat</an_type><an_name>Ursula</an_name><an_price>500</an_price>...

We have no animal named Fluffy.

```
select id,  extract (anXmlData, '/animal[an_name = "Fluffy"]') as "XMLData"
from XmlAnimals;
```



ID	XMLData
1	(null)
9	(null)
14	(null)

We have two animals with a price over 100.

```
   select id, extract (anXmlData, '/animal[an_price > 100]') as "XMLData"
   from XmlAnimals;
```

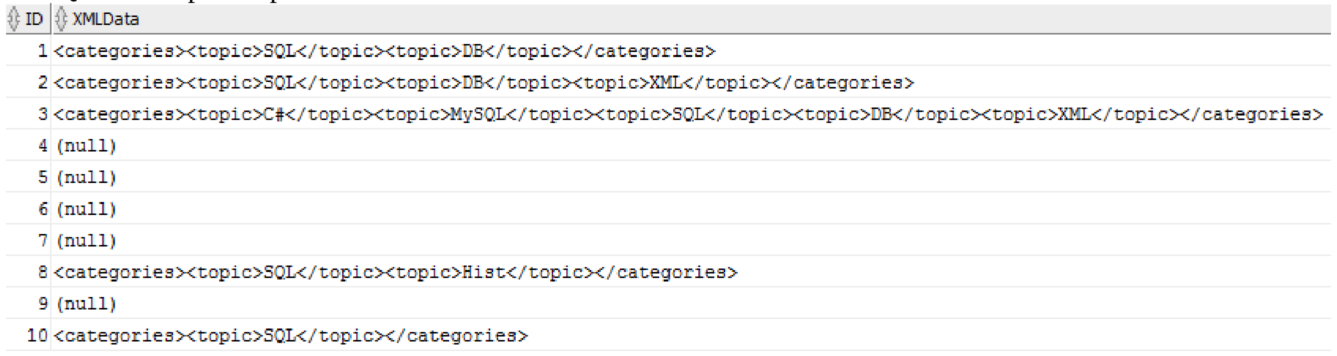
```
  1
  2 <animal><an_id>137</an_id><an_type>bird</an_type><an_name>Mr.
   Peanut</an_name><an_price>150</an_price><an_dob>2008-01-12</an_dob></animal>
 14
   <animal><an_id>1201</an_id><an_type>cat</an_type><an_name>Ursula</an_name><an_price>500<
   /an_price><an_dob>2007-12-15</an_dob></animal>

 3 rows selected.
```

Demo 08: using an index

We have five books with a topic of SQL. We get 10 rows back because there is no Where clause.

```
   select id, extract(book_xml,'//categories[topic= "SQL"]') as "XMLData"
   from XmlBooks;
```

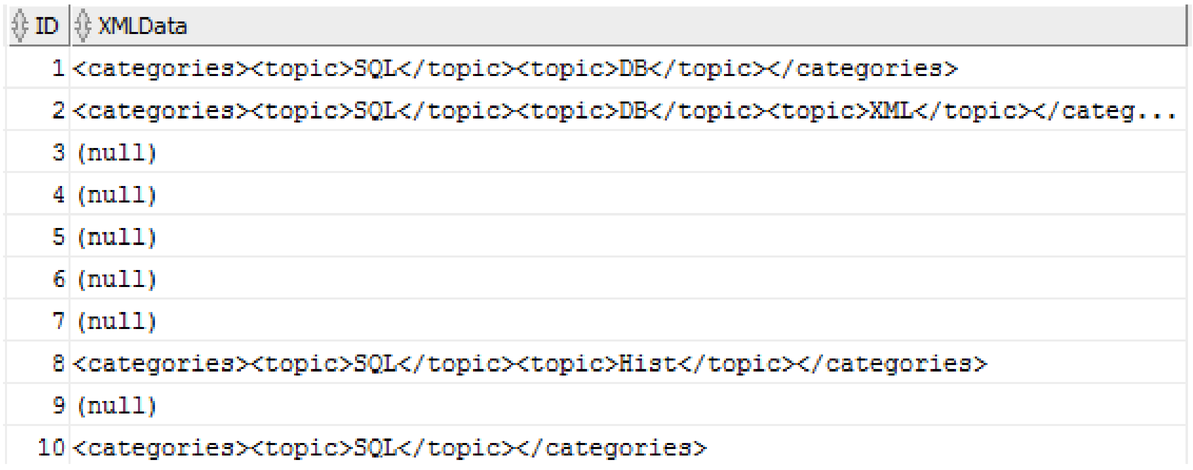
This is SQL Developer output.



ID	XMLData
1	<categories><topic>SQL</topic><topic>DB</topic></categories>
2	<categories><topic>SQL</topic><topic>DB</topic><topic>XML</topic></categories>
3	<categories><topic>C#</topic><topic>MySQL</topic><topic>SQL</topic><topic>DB</topic><topic>XML</topic></categories>
4	(null)
5	(null)
6	(null)
7	(null)
8	<categories><topic>SQL</topic><topic>Hist</topic></categories>
9	(null)
10	<categories><topic>SQL</topic></categories>

We have three books with the **first** topic is SQL.

```
   select id, extract(book_xml,'//categories[topic[1]= "SQL"]') as "XMLData"
   from XmlBooks;
```



ID	XMLData
1	<categories><topic>SQL</topic><topic>DB</topic></categories>
2	<categories><topic>SQL</topic><topic>DB</topic><topic>XML</topic></categ...
3	(null)
4	(null)
5	(null)
6	(null)
7	(null)
8	<categories><topic>SQL</topic><topic>Hist</topic></categories>
9	(null)
10	<categories><topic>SQL</topic></categories>

We have no books with the s**econd** topic is SQL.

```
   select id, extract(book_xml,'//categories[topic[2]= "SQL"]') as "XMLData"
   from XmlBooks;
```

6. Coalesce and CTEs

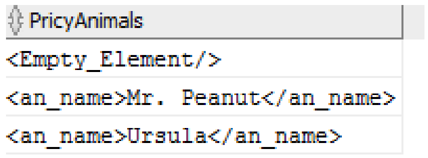
Demo 09: Trying to use coalesce. Coalesce does not work if you try to use it to display an alternate value. The extract function returns xml, not scalars. And this gives the coalesce expression two inconsistent types.

```
select coalesce(extract(anXmlData, '//animal[an_price[1] > 100]'),
                'no return value')
from XmlAnimals;
```

```
SQL Error: ORA-00932: inconsistent datatypes: expected - got CHAR
```

You can use two xml values with coalesce. You would need to know that the element you just created makes sense in how this is used.

```
select coalesce(extract(anXmlData, '//animal[an_price[1] > 100]/an_name'),
                xmltype.createXML('<Empty_Element/>')) as "PricyAnimals"
from XmlAnimals;
```



PricyAnimals
<Empty_Element/>
<an_name>Mr. Peanut</an_name>
<an_name>Ursula</an_name>

(NullIf does not work here.)

You can take the XMLtype expression and cast as a varchar2 and then use coalesce. I cast the xml as a shorter string; you would need to take care with any size limit depending on your xml size. This return is more than 100 characters long and would have been truncated at that length if I used varchar2(100)

```
select coalesce(
         cast(
           extract(anXmlData, '//animal[an_price[1] > 100]')
           as varchar2(200))
       , 'no return value') as "XMLData"
from XmlAnimals;
```

```
----------------------------------------
no return value

<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>

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

3 rows selected.
```

Some string functions can work with xmltype data.

Demo 10: Using CTEs. This is easier to see if you use a CTE to do the extract and then handle the format issues in the main query

```
with CTE as
    (select extract(anXmlData, '//animal[an_price[1] > 100]') as rtnValue
     from XmlAnimals)
select coalesce(cast(rtnValue as varchar2(200)), 'no return value') as
"XMLData"
from CTE;
```

```
no return value
<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>
<animal><an_id>1201</an_id><an_type>cat</an_type><an_name>Ursula</an_name><an_price>500</a
n_price><an_dob>2007-12-15</an_dob></animal>
```

Demo 11: You also can use a CTE to filter more easily on the original return set; the CTE subquery is the query from a previous demo which returned 3 rows but only one matched the animal name.

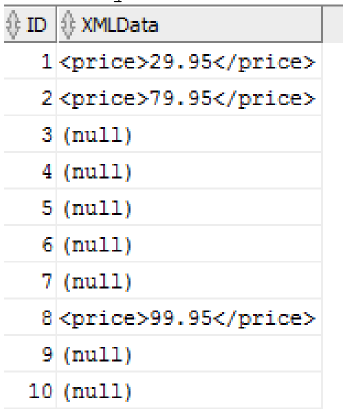
```
With CTE as (
  select ID, extract(anXmlData, '//animal[an_name[1] = "Ursula"]') as XMLData
  from XmlAnimals
)
select ID, cast(XMLData as varchar2(200)) as "XMLData"
from CTE
where XMLData is not null
;
```

```
ID    XMLData
----- -------------------------------------------------------
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>
```

7. Wandering around the path after a test

Demo 12: This time we have a test along the path and then use `..` to go back up the path one level to find the price. The syntax `/../` is walking up the tree just as it does in a file path

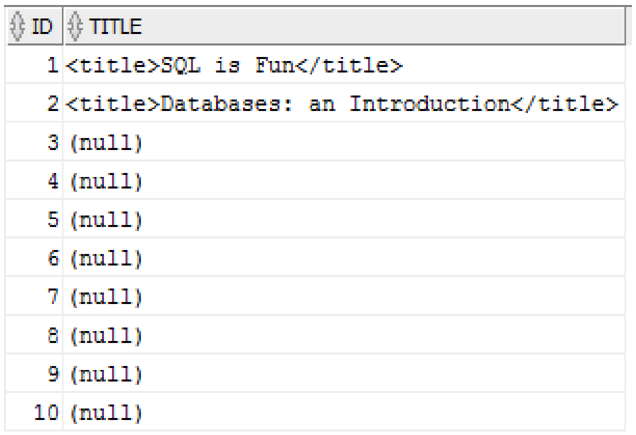
```
select id, extract(book_xml, '/book/categories[topic[1]= "SQL"]/../price ')
as "XMLData"
from XmlBooks
order by id;
```



```
ID XMLData
 1 <price>29.95</price>
 2 <price>79.95</price>
 3 (null)
 4 (null)
 5 (null)
 6 (null)
 7 (null)
 8 <price>99.95</price>
 9 (null)
10 (null)
```

Demo 13: Two tests along the path and then go up to get the book title

```
select id,
extract(book_xml, '/book[price[1] < 85]/categories[topic[1]="SQL"]/../title')
     as title
from XmlBooks
order by id;
```



ID	TITLE
1	<title>SQL is Fun</title>
2	<title>Databases: an Introduction</title>
3	(null)
4	(null)
5	(null)
6	(null)
7	(null)
8	(null)
9	(null)
10	(null)

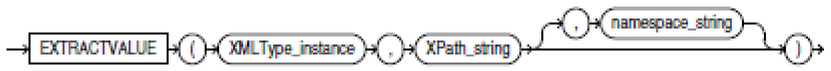
8. The ExtractValue Technique -- Getting inside the xmldata

Obviously we will want a way to get into the xml data and pull out the piece of data we want. We should be able to do that based on the tag names and specify that we want the contents. The function that we want is called ExtractValue; we give it two arguments- the first is the xmltype instance and the second argument is a string representing the XPath expresion to the value we want.

The result has to be a single node (use the [9] syntax).The node has to give you a scalar. That could be a text node or an element with a single text node. If the function call does not follow these rules, you get an error.

The return will be a varchar2 value unless you have an attached Schema (we don't have one).

Syntax



Demo 14: extractValue from xml

```
select    extractvalue(anXmlData,'/animal/an_name[1]')  as "AnimalName"
from      XmlAnimals
;
```

```
AnimalName
-------------------------------------------------------------
ShowBoat
Mr. Peanut
Ursula
```

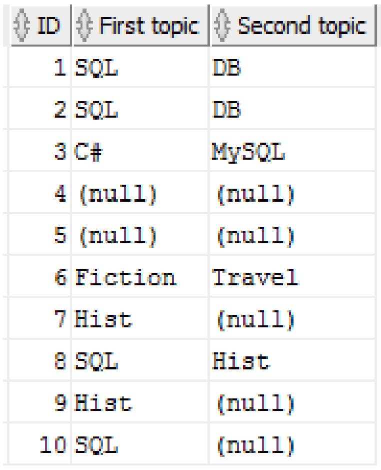
Demo 15: I can return the an_dob and then cast it to a date value. The format returned by extractValue is the standard that XML uses. We can format it to the Oracle version.

```
select extractvalue(anXmlData,'(//animal/an_dob)') as "AnimalDOB "
, to_date(extractvalue(anXmlData,'(//animal/an_dob)') , 'yyyy-mm-dd') as
"AnimalDOB_Oracle"
from XmlAnimals;
```

```
AnimalDOB  AnimalDob_Oracle
---------- -----------------------------------------------------------------
2000-01-15 15-JAN-00
2008-01-12 12-JAN-08
2007-12-15 15-DEC-07
```

Demo 16: The use of //topic and index lets us find the first two topics

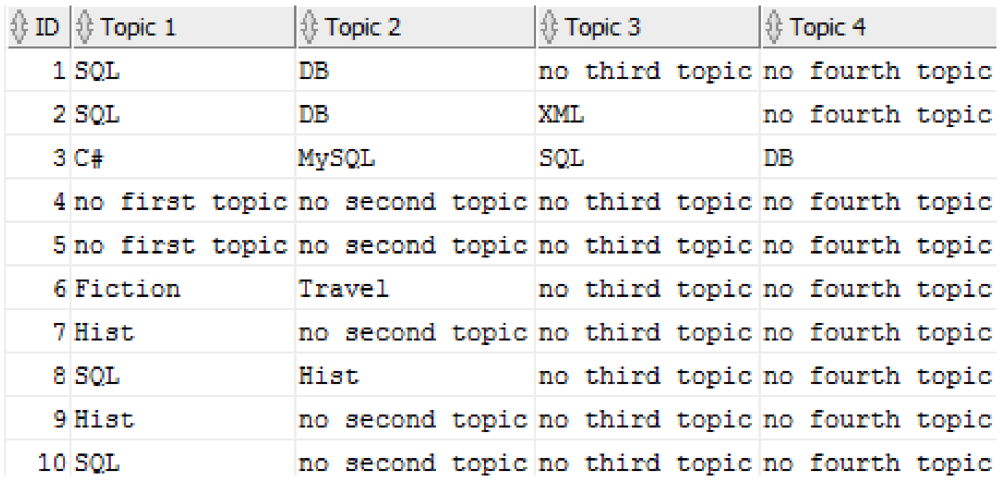
```
select id
,  extractvalue(book_xml,'(//topic)[1]') as "First topic"
,  extractvalue(book_xml,'(//topic)[2]') as "Second topic"
from XmlBooks;
```



ID	First topic	Second topic
1	SQL	DB
2	SQL	DB
3	C#	MySQL
4	(null)	(null)
5	(null)	(null)
6	Fiction	Travel
7	Hist	(null)
8	SQL	Hist
9	Hist	(null)
10	SQL	(null)

Demo 17: The following gets up to four topics for each book. With this query you can use coalesce directly with the extractValue expression, because this is a scalar value, not an xml element

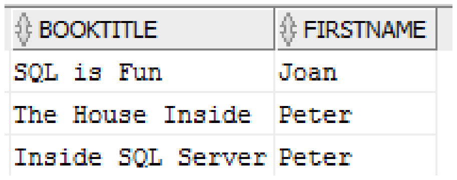
```
select  id
,  coalesce(extractvalue(book_xml,'(//topic)[1]'), 'no first topic')  as "Topic 1"
,  coalesce(extractvalue(book_xml,'(//topic)[2]'), 'no second topic') as "Topic 2"
,  coalesce(extractvalue(book_xml,'(//topic)[3]'), 'no third topic')  as "Topic 3"
,  coalesce(extractvalue(book_xml,'(//topic)[4]'), 'no fourth topic') as "Topic 4"
from XmlBooks;
```



ID	Topic 1	Topic 2	Topic 3	Topic 4
1	SQL	DB	no third topic	no fourth topic
2	SQL	DB	XML	no fourth topic
3	C#	MySQL	SQL	DB
4	no first topic	no second topic	no third topic	no fourth topic
5	no first topic	no second topic	no third topic	no fourth topic
6	Fiction	Travel	no third topic	no fourth topic
7	Hist	no second topic	no third topic	no fourth topic
8	SQL	Hist	no third topic	no fourth topic
9	Hist	no second topic	no third topic	no fourth topic
10	SQL	no second topic	no third topic	no fourth topic

Demo 18: We can use the extractValue method in the Where clause to filter on the contents of an xml value. Here we compare the scalar returned by the function ot a string literal. Which books are written by author Collins? I am using an index here also.

```
select extractvalue(book_xml,'(/book/title)[1]')   as BookTitle,
       extractvalue(book_xml,'(//authors/name/first)[1]') as FirstName
from XmlBooks
where extractvalue(book_xml,'(//authors/name/last)[1]') ='Collins' ;
```



BOOKTITLE	FIRSTNAME
SQL is Fun	Joan
The House Inside	Peter
Inside SQL Server	Peter

Demo 19: Once you extract the values of the data you can use regular functions and other techniques. What is the average price for books published by Addison Wesley?

```
select count(*) as "Addison Wesley Count"
, avg(extractvalue(book_xml,'(//price)') )  as "AvgPrice"
from XmlBooks
where extractvalue(book_xml,'(//publisher)[1]') ='Addison Wesley' ;
```

```
Addison Wesley Count   AvgPrice
-------------------- ----------
                   7    54.8375
```

Demo 20: Supposed you wanted the average price by publisher? That is a group by query. It probably makes sense to do this one step at a time and use CTE to work it out.

Step A get the data out; you can test this by itself; supply simple names for the columns (sql friendly column aliases)

```
select cast(extractvalue(book_xml,'(//publisher)[1]') as varchar2(25)) as Publ
,  extractvalue(book_xml,'(//price)')   as price
from XmlBooks;
```

Step B do the grouping

```
with ExtractedData as (
   select cast(extractvalue(book_xml,'(//publisher)[1]') as varchar2(25)) as Publ
  ,  extractvalue(book_xml,'(//price)')   as price
   from XmlBooks
   )
   select Publ, count(*) as NumBk, avg(price) as avgPrice
   From ExtractedData
   group by Publ ;
```

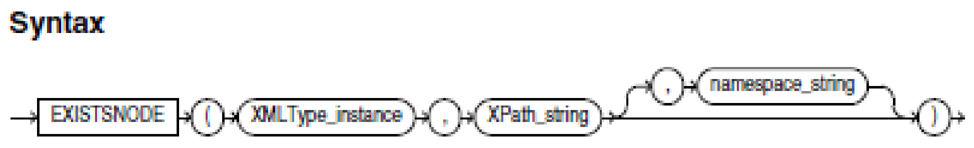
Step C do any final formatting and supply user friendly column aliases ; You can combine steps B and C if there is no particular formatting to do in the last step

```
with ExtractedData as (
   select cast(extractvalue(book_xml,'(//publisher)[1]') as varchar2(25)) as Publ
  ,  extractvalue(book_xml,'(//price)')   as price
   from XmlBooks
   )
, GroupedData as (
   select Publ
   , count(*) as NumBk
   , avg(price) as avgPrice
   From ExtractedData
   group by Publ
   )
select
  Publ     as "Publisher"
, numBk    as "BookCount"
, Round( Avgprice,2) as  "AveragePrice"
from GroupedData
;
```

```
Publisher                  BookCount AveragePrice
------------------------- ---------- ------------
McGraw Hill                        2       105.95
Addison Wesley                     7        54.84
Jones Books                        1        99.95
```

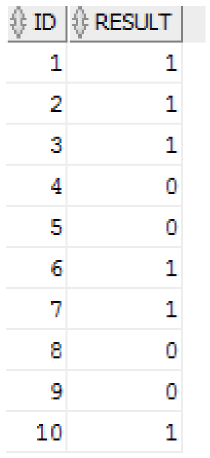
9. The existsNode()Technique --Does this node exist?

existsNodes is a function with two parameters- an xmlType instance and an xPath string. The function returns a 0 or a 1. If there are no matches for the xPath then the function returns 0; if there is one or more matches, then the function returns 1. **existsNode is generally used in the Where clause of a query.**



Demo 21: Using exists in the Select we get a return of 1 if there is a second author for a book

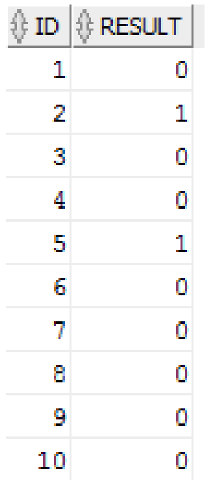
```
select id, existsNode(book_xml, '(//authors//last)[2]') as result
from XmlBooks;
```



ID	RESULT
1	1
2	1
3	1
4	0
5	0
6	1
7	1
8	0
9	0
10	1

Demo 22: Using exists in the Select to determine if a book price is 79.95

```
select id, existsNode(book_xml, '/book[price[1] = 79.95]') as Result
from XmlBooks;
```



ID	RESULT
1	0
2	1
3	0
4	0
5	1
6	0
7	0
8	0
9	0
10	0

Demo 23: You are more likely to use ExistsNode in the Where clause. The predicate is in the path.

```
select extract(book_xml, '(/book/title)[1]')  as "XMLData"
from XmlBooks
where existsNode(book_xml, '/book[price[1] = 79.95]') = 1
;
```

```
XMLData
-------------------------------------------------
<title>Databases: an Introduction</title>
<title>The truth about everything</title>
```

What do these find?

```
   Select  id, extract(book_xml, '//authors//last')  as "XMLData"
   From XmlBooks
   where existsNode(book_xml, '(//authors//last)[2]') =1;

   Select  id, extract(book_xml, '//authors//last')  as "XMLData"
   From XmlBooks
   where existsNode(book_xml, '(//authors//last)[2]') = 0;
```

10. Using the XML table

We have a table XmlOnlyAnimals which is a pure xml table. We can use the same methods with the table.

Demo 24: extractvalue

```
   select extractvalue(sys_nc_rowinfo$, '/animal/an_name') As BirdNames
   from XmlOnlyAnimals
   where extractvalue(sys_nc_rowinfo$, '/animal/an_type') = 'bird';
```

```
BIRDNAMES
-------------------------------------------------
ShowBoat
Mr. Peanut
```

Demo 25: You can also use the word object_value instead of SYS_NC_ROWINFO$, which will be easier to remember and easier to type.

```
   select extractvalue(object_value, '/animal/an_name') As BirdNames
   from XmlOnlyAnimals
   where extractvalue(object_value, '/animal/an_type') = 'bird';
```