Table of Contents

1. Working with XML interactively

For continuing with the demos, if your client session was opened with the --xml option, close it and start another client session. DO NOT USE the --xml option for these demos and do not use it for the assignment.

1.1. Creating a table with a character column that holds xml

We are going to work mostly with the following table which has only a few rows and a fairly simple structure.

Demo 01: The table has a simple integer Id column as the pk and a column that holds the XML strings; this is declared as Text so that it can hold long strings.

```
create table xml_bk_1(
  id    int primary key
, datax TEXT);
```

The pattern for our data is that we have a root element of book with a catalog_id attribute. We have sub-elements of title, price and publisher- these occur only once. We have a subelement of authors which contains zero or more name subelements; each of these has a first and last element. We have a subelement of categories which contains zero or more topic subelements.

```
<book  catlog_id="99999">
    <title>. . .</title>
    <authors>
    <name>                                          ------ optional and can repeat
      <first>. . .</first><last>. . .</last>
    </name>
    </authors>
    <categories>
      <topic>. . .</topic>                          ------ optional and can repeat
    </categories>
    <price>999.99</price>
    <publisher>. . .</publisher>
</book>
```

Demo 02: inserting data- these are the first two rows. Consult the demo file for the full set of inserts

```
insert into xml_bk_1 values (1,
'<book  catlog_id="124">
    <title>SQL is Fun</title>
    <authors>
    <name>
      <first>Joan</first><last>Collins</last>
    </name>
    <name>
      <first>Jill</first><last>Effingham</last>
    </name>
    </authors>
    <categories>
      <topic>SQL</topic>
      <topic>DB</topic>
    </categories>
    <price>29.95</price>
    <publisher>Addison Wesley</publisher>
</book>'
);

insert into xml_bk_1 values (2,
'<book  catlog_id="275">
    <title>Databases: an Introduction</title>
    <authors>
      <name>
        <first>Mike</first><last>Malone</last>
     </name>
     <name>
         <first>Jill</first><last>Effingham</last>
       </name>
    </authors>
    <categories>
      <topic>SQL</topic>
      <topic>DB</topic>
     <topic>XML</topic>
    </categories>
    <price>79.95</price>
    <publisher>Addison Wesley</publisher>
</book>'
);
```

Demo 03:

```
Select * from xml_bk_1 limit 2\G
mysql> Select * from xml_bk_1 limit 2\G
*************************** 1. row ***************************
   id: 1
datax: <book  catlog_id="124">
    <title>SQL is Fun</title>
    <authors>
    <name>
      <first>Joan</first><last>Collins</last>
    </name>
    <name>
      <first>Jill</first><last>Effingham</last>
    </name>
    </authors>
```

```
    <categories>
      <topic>SQL</topic>
      <topic>DB</topic>
    </categories>
    <price>29.95</price>
    <publisher>Addison Wesley</publisher>
</book>
*************************** 2. row ***************************
   id: 2
datax: <book  catlog_id="275">
    <title>Databases: an Introduction</title>
    <authors>
      <name>
             <first>Mike</first><last>Malone</last>
          </name>
          <name>
         <first>Jill</first><last>Effingham</last>
       </name>
    </authors>
    <categories>
      <topic>SQL</topic>
      <topic>DB</topic>
          <topic>XML</topic>
    </categories>
    <price>79.95</price>
    <publisher>Addison Wesley</publisher>
</book>
```

1.2. Not an XML type

It is critical at this point to note that the table has a TEXT column and I can insert any string into that column. There is nothing in the table definition that checks if the string is a valid or even a well-formed xml document. If you have worked with XML data in other databases that have an xml data type- that is a completely different story. MySQL does not currently have an XML type. It is up to you to be sure that your strings are well formed XML documents.

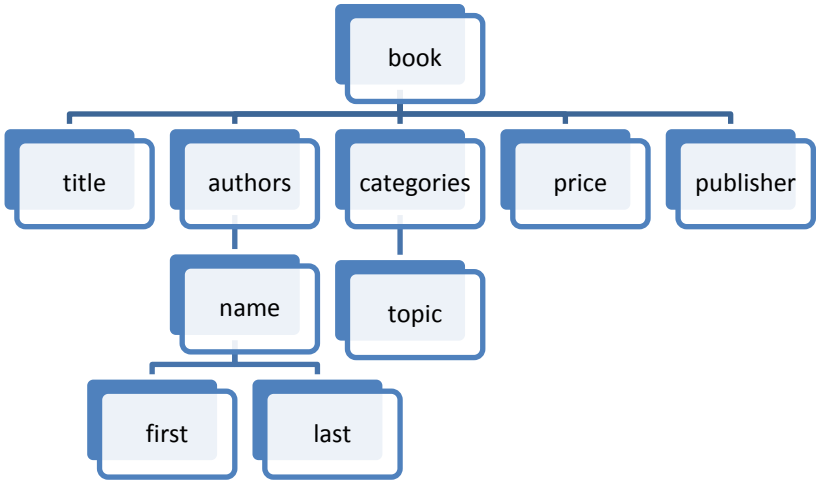
You might find it worthwhile to take the string you are using for the xml data and save it as an xml file and try opening it in a browser or a xml editor which will usually find errors in the xml structure.

Also you would not normally write XML documents longhand like this. You would have an application layer program that gets the data values and build the XML string. So we are keeping these short.

1.3. Hierarchical structure of the xml strings and XPath

What can we do with that datax column? MySQL does have 2 XMLfunctions. One of these is ExtractValue which uses XPath notation to get a value from an XML string. Theother is an Update function we are not discussing.

The ExtractValue function takes two arguments. The first is the XML string and the second is an XPath expression. The XPath expression relies on the hierarchical structure of the xml string. This is the structure used for the book strings



2. The ExtractValue function

We could use string functions to work with that text value but we do have some ability to work with that string as if it is xml data. The ExtractValue function takes two arguments; the first should be an xml fragment (an xml; fragment is well formed with the exception that it does not need a root element). The second argument is an XPath expression. We will look at various examples of XPath expressions

2.1. Using paths

The xpath expression are similar to file name paths in that they can start at a root and traverse a tree structure using the / character to indicate that we are traveling to the next node.

Demo 04: Walking down the path from the root book node to the price node

```
Select id, extractValue(datax, '/book/price' )
from xml_bk_1;
+----+-------------------------------------+
| id | extractValue(datax, '/book/price' ) |
+----+-------------------------------------+
|  1 | 29.95                               |
|  2 | 79.95                               |
|  3 | 58.95                               |
|  4 | 105.95                              |
|  5 | 79.95                               |
|  6 | 29.50                               |
|  7 | 99.95                               |
+----+-------------------------------------+

Select id, extractValue(datax, '/book/title' )
from xml_bk_1;
+----+-------------------------------------+
| id | extractValue(datax, '/book/title' ) |
+----+-------------------------------------+
|  1 | SQL is Fun                          |
|  2 | Databases: an Introduction          |
|  3 | Visual Basic and MySQL              |
|  4 | Databases- Design and Logic         |
|  5 | The truth about everything          |
|  6 | Alice Goes to Oz                    |
|  7 | The House Inside                    |
+----+-------------------------------------+
```

We can use ExtractValue to go into our document and retrieve data values.

These look pretty clear; we go the price and then we got the title. Sometimes the result may not be as helpful.

Demo 05: Since there can be multiple authors, we get all last name values as a single string value with spaces as delimiters. How many authors do we have for each book? You cannot really tell from this display that book 5 has only one author whose last name is 'Del Ansom"

```
Select id
, extractValue(datax, '/book/authors/name/last' )
from xml_bk_1;
+----+-------------------------------------------------+
| id | extractValue(datax, '/book/authors/name/last' ) |
+----+-------------------------------------------------+
|  1 | Collins Effingham                               |
|  2 | Malone Effingham                                |
|  3 | Horn Cocker Shorter                             |
|  4 | LaVette                                         |
|  5 | Del Ansom                                       |
|  6 | Dodgson Baum                                    |
|  7 | Collins Collins                                 |
+----+-------------------------------------------------+
```

First- a few things to watch out for. I don't normally like to show a lot of error possibilities but you will probably do some of these.

Demo 06: Error 1- the path /book/price is not the same as /book/Price; this is case specific. But we do not get an error message such as "path not found" instead we get output with no warning messages.

```
Select id, extractValue(datax, '/book/Price' )
from xml_bk_1;
+----+--------------------------------------+
| id | extractValue(datax, '/book/Price' ) |
+----+--------------------------------------+
|  1 |                                      |
|  2 |                                      |
|  3 |                                      |
|  4 |                                      |
|  5 |                                      |
|  6 |                                      |
|  7 |                                      |
+----+--------------------------------------+
```

Demo 07: Error 2- the path /book/price is not the same as /book/pice. But we do not get an error message such as "no elements named pice" instead we get output with no warning messages.

```
Select id, extractValue(datax, '/book/pice' )
from xml_bk_1;
```

Demo 08: Error 3- the path is wrong here also. /price is not the root. And we cannot say just look for the price element. But we do not get an error message such as "no elements named price" instead we get the same output with no warning messages.

```
Select id, extractValue(datax, '/price' )
from xml_bk_1;
Select id, extractValue(datax, 'price' )
```

```
from xml_bk_1;
```

Note that we have 7 rows in our table and the sql query does not include a Where clause so the result set has 6 rows. If I run the query shown here I get 6 rows of output

```
Select id, extractValue(datax, '/price' )
from xml_bk_1;
+-------------------------------+
| extractValue(datax, 'price' ) |
+-------------------------------+
|                               |
|                               |
|                               |
|                               |
|                               |
|                               |
|                               |
+-------------------------------+
7 rows in set (0.00 sec)
```

These rows are not null, they contain an empty string- a zero-length string. If we use coalesce we do not get the second argument since the first argument is not null

We can use concat.

Demo 09:

```
Select  coalesce(extractValue(datax, 'price' )  , 'Nothing')
from xml_bk_1;
+-------------------------------------------------------+
| coalesce(extractValue(datax, 'price' )  , 'Nothing') |
+-------------------------------------------------------+
|                                                       |
|                                                       |
|                                                       |
|                                                       |
|                                                       |
|                                                       |
|                                                       |
+-------------------------------------------------------+
7 rows in set (0.00 sec)

Select  concat(extractValue(datax, 'price' )  , 'Nothing')
from xml_bk_1;
+-----------------------------------------------------+
| concat(extractValue(datax, 'price' )  , 'Nothing') |
+-----------------------------------------------------+
| Nothing                                             |
| Nothing                                             |
| Nothing                                             |
| Nothing                                             |
| Nothing                                             |
| Nothing                                             |
| Nothing                                             |
+-----------------------------------------------------+
7 rows in set (0.00 sec)
```

These are a few helpful variations on the path expression which do work.

Demo 10: If we start the path with the root ,we can skip the leading slash/

```
Select id, extractValue(datax, 'book/price' )
from xml_bk_1;
+----+-----------------------------------+
| id | extractValue(datax, 'book/price' ) |
+----+-----------------------------------+
|  1 | 29.95                             |
|  2 | 79.95                             |
|  3 | 58.95                             |
|  4 | 105.95                            |
|  5 | 79.95                             |
|  6 | 29.50                             |
|  7 | 99.95                             |
+----+-----------------------------------+
```

Demo 11: We can use // to tell the path evaluator to just go down the path till it finds the named element. This can create problems we will see soon.

```
Select id
, extractValue(datax, '//price' )  as Price
, Concat(extractValue(datax, '//first' ) , ' ',
         extractValue(datax, '//last' ) )   as Authors
from xml_bk_1;
+----+--------+---------------------------------------+
| id | Price  | Authors                               |
+----+--------+---------------------------------------+
|  1 | 29.95  | Joan Jill Collins Effingham           |
|  2 | 79.95  | Mike Jill Malone Effingham            |
|  3 | 58.95  | Shirley Joe Wayne Horn Cocker Shorter |
|  4 | 105.95 | Betty LaVette                         |
|  5 | 79.95  | June Del Ansom                        |
|  6 | 29.50  | Lewis L. Frank Dodgson Baum           |
|  7 | 99.95  | Peter Henry Collins Collins           |
+----+--------+---------------------------------------+
```

Demo 12: Since you are just retrieving a value, you can use the result of the function in a Where clause filter

```
Select id, extractValue(datax, '/book/title' )  As Title
from xml_bk_1
where extractValue(datax, '/book/price' ) > 50;
+----+-----------------------------+
| id | Title                       |
+----+-----------------------------+
|  2 | Databases: an Introduction  |
|  3 | Visual Basic and MySQL      |
|  4 | Databases- Design and Logic |
|  5 | The truth about everything  |
|  7 | The House Inside            |
+----+-----------------------------+
```

We can use the function on data that is not in a table. We did this with other functions. You will see this often when people are explaining a feature of extractValue and don't want to set up a table.

Demo 13: Here the first argument is a literal string for the xml fragment

```
select extractValue (
'<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>'
, 'animal/an_name')  as Result;
+----------+
| Result   |
+----------+
| ShowBoat |
+----------+
```

Demo 14: With some errors in the string that make it invalid xml, the result of the function will be a null and you get a warning. Some other errors do not seem to bother the function!

```
select extractValue (
'<animal>
 <an_id>136</an_id>
 <an_type>bird</an_type>
 <an_name>ShowBoat</an_name>
 <an_price>75
 <an_dob>2000-01-15</an_dob>
 </animal>'
, 'animal/an_name')  as Result;
+--------+
| Result |
+--------+
| NULL   |
+--------+

| Warning | 1525 | Incorrect XML value: 'parse error at line 7 pos 11:
'</animal>' unexpected ('</an_price>' wanted)' |
```

Demo 15: You can combine two path expressions as a union to join the result sets; this uses the pipe character | . This is matching any title element and combining with any topic element. But the output is not particularly intuitive. Maybe the title of the first book is "SQL is" and the topics are Fun, SQL, and DB

```
Select id
, extractValue(datax, '//price' )              as Price
, extractValue(datax, '//title  | //topic'  )  as "Title and  Topics"
from xml_bk_1;
+----+--------+---------------------------------------+
| id | Price  | Title and  Topics                     |
+----+--------+---------------------------------------+
|  1 | 29.95  | SQL is Fun SQL DB                     |
|  2 | 79.95  | Databases: an Introduction SQL DB XML |
|  3 | 58.95  | Visual Basic and MySQL VB MySQL SQL DB |
|  4 | 105.95 | Databases- Design and Logic           |
|  5 | 79.95  | The truth about everything            |
|  6 | 29.50  | Alice Goes to Oz Fiction Travel       |
|  7 | 99.95  | The House Inside Hist                 |
+----+--------+---------------------------------------+
```

Examine the inserts for id 4 and 5.

Demo 16: Using concat and two ExtractValue function calls; we are concatenating 4 pieces: the extracted title, a literal , the extracted title, and a literal for the closing parentheses.

```
Select id
, extractValue(datax, '//price' )  as Price
, concat( extractValue(datax, '//title') ,
          ' (categories: ' ,
          extractValue(datax, '//topic'  )
          , ')'
         )  as "Title and  Topics"
from xml_bk_1;
+----+--------+-----------------------------------------------------+
| id | Price  | Title and  Topics                                   |
+----+--------+-----------------------------------------------------+
|  1 | 29.95  | SQL is Fun (categories: SQL DB)                     |
|  2 | 79.95  | Databases: an Introduction (categories: SQL DB XML) |
|  3 | 58.95  | Visual Basic and MySQL (categories: VB MySQL SQL DB)|
|  4 | 105.95 | Databases- Design and Logic (categories: )          |
|  5 | 79.95  | The truth about everything (categories: )           |
|  6 | 29.50  | Alice Goes to Oz (categories: Fiction Travel)       |
|  7 | 99.95  | The House Inside (categories: Hist)                 |
+----+--------+-----------------------------------------------------+
```

2.2. Attributes

Demo 17: Getting an attribute- use the @ symbol to indicate an attribute in the XML string

```
Select id
, extractValue(datax, 'book/@catalog_id' )  as "catalog id"
from xml_bk_1;
+----+------------+
| id | catalog id |
+----+------------+
|  1 | 124        |
|  2 | 275        |
|  3 | 313        |
|  4 | 49375      |
|  5 | 5508       |
|  6 | 6006       |
|  7 | 125        |
+----+------------+
```

3. Locating specific data within multiple elements

3.1. Sequence number [1]

Demo 18: We have multiple topic sub-elements. You can specify the one you want with the number in square brackets. It is not an error to ask for a sequence number that does not exist

```
Select id
, extractValue(datax, '//title ') as Title
, extractValue(datax, '//topic[1]'  )   as "tpc 1"
, extractValue(datax, '//topic[2]'  )   as "tpc 2"
, extractValue(datax, '//topic[3]'  )   as "tpc 3"
```

```
, extractValue(datax, '//topic[4]'  )   as "tpc 4"
, extractValue(datax, '//topic[5]'  )   as "tpc 5"
from xml_bk_1;
+----+-----------------------------+---------+--------+-------+-------+-------+
| id | Title                       | tpc 1   | tpc 2  | tpc 3 | tpc 4 | tpc 5 |
+----+-----------------------------+---------+--------+-------+-------+-------+
|  1 | SQL is Fun                  | SQL     | DB     |       |       |       |
|  2 | Databases: an Introduction  | SQL     | DB     | XML   |       |       |
|  3 | Visual Basic and MySQL      | VB      | MySQL  | SQL   | DB    |       |
|  4 | Databases- Design and Logic |         |        |       |       |       |
|  5 | The truth about everything  |         |        |       |       |       |
|  6 | Alice Goes to Oz            | Fiction | Travel |       |       |       |
|  7 | The House Inside            | Hist    |        |       |       |       |
+----+-----------------------------+---------+--------+-------+-------+-------+
```

3.2. Last() function

Demo 19: Getting the last of the sub elements using the last() function in the XPath expression. Note that the function is inside the square brackets.

```
Select id
, extractValue(datax, '//title ') as Title
, extractValue(datax, '//topic[last()]'  )   as "last topic"
 from xml_bk_1;
+----+-----------------------------+------------+
| id | Title                       | last topic |
+----+-----------------------------+------------+
|  1 | SQL is Fun                  | DB         |
|  2 | Databases: an Introduction  | XML        |
|  3 | Visual Basic and MySQL      | DB         |
|  4 | Databases- Design and Logic |            |
|  5 | The truth about everything  |            |
|  6 | Alice Goes to Oz            | Travel     |
|  7 | The House Inside            | Hist       |
+----+-----------------------------+------------+
```

Demo 20: Arithmetic with the result of last()- this uses [last()-2] to get the element that is the third from the end

```
Select id
, cast(extractValue(datax, '//title ') as char(10))as Title
, extractValue(datax, '//topic[last()]'  )   as "last topic"
, extractValue(datax, '//topic[last()-1]'  ) as "almost topic"
, extractValue(datax, '//topic[last()-2]'  ) as "third from the end"
 from xml_bk_1;
+----+------------+------------+--------------+--------------------+
| id | Title      | last topic | almost topic | third from the end |
+----+------------+------------+--------------+--------------------+
|  1 | SQL is Fun | DB         | SQL          |                    |
|  2 | Databases: | XML        | DB           | SQL                |
|  3 | Visual Bas | DB         | SQL          | MySQL              |
|  4 | Databases- |            |              |                    |
|  5 | The truth  |            |              |                    |
|  6 | Alice Goes | Travel     | Fiction      |                    |
|  7 | The House  | Hist       |              |                    |
+----+------------+------------+--------------+--------------------+
```

4. Using the XPath count function

You can use the count function to determine how many child elements are on the path. In these demos the xml string might have 1, 2, or no animal name sub-elements.

Demo 21: We have one <an_name> element

```
SELECT
ExtractValue('<animal><an_name>ShowBoat</an_name></animal>', '/animal/an_name') as name
,
ExtractValue('<animal><an_name>ShowBoat</an_name></animal>', 'count(/animal/an_name)')
as CountValue;
+----------+------------+
| name     | CountValue |
+----------+------------+
| ShowBoat | 1          |
+----------+------------+
```

Demo 22: We have two <an_name> elements for this animal element

```
SELECT
ExtractValue('<animal><an_name>Tweetie</an_name><an_name>Sylvester</an_name></animal>',
'/animal/an_name') as name
,
ExtractValue('<animal><an_name>Tweetie</an_name><an_name>Sylvester</an_name></animal>',
'count(/animal/an_name)') as CountValue;
+-------------------+------------+
| name              | CountValue |
+-------------------+------------+
| Tweetie Sylvester | 2          |
+-------------------+------------+
```

Demo 23: We have one <an_name> element- it still counts even if it is empty

```
select
ExtractValue('<animal><an_name></an_name></animal>', '/animal/an_name') as name
,
ExtractValue('<animal><an_name></an_name></animal>', 'count(/animal/an_name)') as
CountValue;
+------+------------+
| name | CountValue |
+------+------------+
|      | 1          |
+------+------------+
```

Demo 24: Now we do not have an <an_name> element and the count is 0

```
SELECT
ExtractValue('<animal><an_price>75</an_price></animal>', '/animal/an_name') as name
,
ExtractValue('<animal><an_price>75</an_price></animal>', 'count(/animal/an_name)') as
CountValue;
+------+------------+
| name | CountValue |
+------+------------+
|      | 0          |
+------+------------+
```

Demo 25: How many authors for each book?

```
Select id
, Concat(extractValue(datax, '//first' ) , ' ',
         extractValue(datax, '//last' ) )   as Authors
,  ExtractValue(datax, 'count(//last)') as authorCount
from xml_bk_1
;
+----+---------------------------------------+-------------+
| id | Authors                               | authorCount |
+----+---------------------------------------+-------------+
|  1 | Joan Jill Collins Effingham           | 2           |
|  2 | Mike Jill Malone Effingham            | 2           |
|  3 | Shirley Joe Wayne Horn Cocker Shorter | 3           |
|  4 | Betty LaVette                         | 1           |
|  5 | June Del Ansom                        | 1           |
|  6 | Lewis L. Frank Dodgson Baum           | 2           |
|  7 | Peter Henry Collins Collins           | 2           |
+----+---------------------------------------+-------------+
```

Demo 26: Books with a single author

```
Select id
, extractValue(datax, '//title' )  as Title
, Concat(extractValue(datax, '//first' ) , ' ',
         extractValue(datax, '//last' ) )   as Authors
from xml_bk_1
where ExtractValue(datax, 'count(//last)') =1
;
+----+-----------------------------+----------------+
| id | Title                       | Authors        |
+----+-----------------------------+----------------+
|  4 | Databases- Design and Logic | Betty LaVette  |
|  5 | The truth about everything  | June Del Ansom |
+----+-----------------------------+----------------+
```

Demo 27: Setting a variable for the xml fragment

```
set @x1 ='<book  catalog_id="624">
    <authors>
    <name><first>Joan</first><last></last></name>
    <name><first></first><last>Effingham</last></name>
    </authors>
</book>';
```

We can use that variable for the xml fragment

```
select
extractvalue(@x1, '//first');
+------------------------------+
| extractvalue(@x1, '//first') |
+------------------------------+
| Joan                         |
+------------------------------+
```

Demo 28: As always you need to take care with your expressions; it might look like we have an author named Joan Effingham, but we actually have two authors.

```
select concat(
 extractvalue(@x1, '//first')
, ' '
, extractvalue(@x1, '//last')
) as author
, ExtractValue(@x1, 'count(//last)') as authorCount
;
+----------------+-------------+
| author         | authorCount |
+----------------+-------------+
| Joan Effingham | 2           |
+----------------+-------------+
```

MySQL does not support the proper use of a variable for the XPath expression. If you tired to use something like the following, you get a result set with no warning and the second argument is treated as a null. (that is considered a bug.)

```
set @x2 ='//first';
select extractvalue(@x1, @x2);
+------------------------+
| extractvalue(@x1, @x2) |
+------------------------+
|                        |
+------------------------+
```

5. Adding predicates

These demos are using string literals for the first argument to ExtractValue and the count function in the XPath expression, but I have also added a predicate. The following expression says to count the occurrences of an animal element with an an_price subelement with a value equal to 75. With this literal, we either have such a an_price value and then count =-1 or we don't and then count = 0.

```
'count(/animal[an_price=75])'
```

The predicate (the test expression) is enclosed in square brackets. We can use test for equals =, not equals !=, inequality >, < , >=, <=

Demo 29:

```
select
  ExtractValue('<animal><an_price>75</an_price></animal>',
'count(/animal[an_price=75])') as CountValue1

 , ExtractValue('<animal><an_price>75</an_price></animal>',
'count(/animal[an_price=85])') as CountValue2

  , ExtractValue('<animal><an_price>75</an_price></animal>',
'count(/animal[an_price!=75])') as CountValue3
  ;
+-------------+-------------+-------------+
| CountValue1 | CountValue2 | CountValue3 |
+-------------+-------------+-------------+
| 1           | 0           | 0           |
+-------------+-------------+-------------+
```

Demo 30:

```
select
  ExtractValue('<animal><an_price>75</an_price></animal>',
'count(/animal[an_price>=50])') as CountValue1

 , ExtractValue('<animal><an_price>75</an_price></animal>',
'count(/animal[an_price>100])') as CountValue2;
 +-------------+-------------+
| CountValue1 | CountValue2 |
+-------------+-------------+
| 1           | 0           |
+-------------+-------------+
```

Demo 31: string comparison

```
select
  ExtractValue('<animal><an_name>ShowBoat</an_name></animal>',
'count(/animal[an_name="ShowBoat"])') as CountValue1

 , ExtractValue('<animal><an_name>ShowBoat</an_name></animal>',
'count(/animal[an_name="Showboat"])') as CountValue2;
+-------------+-------------+
| CountValue1 | CountValue2 |
+-------------+-------------+
| 1           | 1           |
+-------------+-------------+
```

Demo 32: show the books that have a price of 29.95

```
select id
, extractValue(datax, '/book/price' ) As Price
, extractValue(datax, '/book/title' ) as Title
from xml_bk_1
where extractValue (datax, 'count(/book[price= 29.95])')  > 0;
+----+-------+-----------------------+
| id | Price | Title                 |
+----+-------+-----------------------+
|  1 | 29.95 | SQL is Fun            |
+----+-------+-----------------------+
```

Demo 33: Display books that cost between 29.95 and 60.

```
select id
, extractValue(datax, '/book/price' ) As Price
, extractValue(datax, '/book/title' ) as Title
from xml_bk_1
where extractValue (datax, 'count(/book[price>= 29.95 and price<= 60 ])')  > 0;
+----+-------+-----------------------+
| id | Price | Title                 |
+----+-------+-----------------------+
|  1 | 29.95 | SQL is Fun            |
|  3 | 58.95 | Visual Basic and MySQL |
+----+-------+-----------------------+
```

5.1. Contains

Demo 34: using the Xpath contains function- this returns a True/False response

```
select
ExtractValue('<animal><an_name>ShowBoat</an_name></animal>',
'contains(/animal/an_name, "Showboat")')  as test_1
,
ExtractValue('<animal><an_name>ShowBoat</an_name></animal>',
'contains(/animal/an_name, "oa")') as test_2
,
ExtractValue('<animal><an_name>ShowBoat</an_name></animal>',
'contains(/animal/an_name, "oo")') as test_3;
+--------+--------+--------+
| test_1 | test_2 | test_3 |
+--------+--------+--------+
| 1      | 1      | 0      |
+--------+--------+--------+
```

6. XPath extended notation

There are some additional notations you might want to use in XPath expressions.

6.1. Self

self:text() references the value of the current node

Demo 35: finding books with a topic of DB

```
Select id
, extractValue(datax, '/book/title' ) as title
, extractValue(datax, '//topic' ) as topiclist
from   xml_bk_1
where  extractValue(datax,'count(//categories/topic[self:text()="DB"])') > 0;
+----+----------------------------+-----------------+
| id | title                      | topiclist       |
+----+----------------------------+-----------------+
|  1 | SQL is Fun                 | SQL DB          |
|  2 | Databases: an Introduction | SQL DB XML      |
|  3 | Visual Basic and MySQL     | VB MySQL SQL DB |
+----+----------------------------+-----------------+
```

Demo 36: we want books with a topic of XML or Travel

```
select id
, extractValue(datax,'/book/title') as title
, extractValue(datax, '//topic' ) as topiclist
from xml_bk_1
where extractValue(datax,'count(//categories/topic[self:text()="XML"
      or self:text()="Travel"])') > 0;
+----+----------------------------+----------------+
| id | title                      | topiclist      |
+----+----------------------------+----------------+
|  2 | Databases: an Introduction | SQL DB XML     |
|  6 | Alice Goes to Oz           | Fiction Travel |
+----+----------------------------+----------------+
```

6.2. child

Demo 37: child element of the categories element

```
select extractValue(datax,'/book/categories/child::*')
from  xml_bk_1
limit 2;
+-------------------------------------------------+
| extractValue(datax,'/book/categories/child::*') |
+-------------------------------------------------+
| SQL DB                                          |
| SQL DB XML                                      |
+-------------------------------------------------+
```

Demo 38: This could be more useful- we want the child elements of name and we have first and last subelements. We still have issues with the spaces within the data values and the spaces between the data values

```
select extractValue(datax,'//name/child::*')
from  xml_bk_1
;
+--------------------------------------+
| extractValue(datax,'//name/child::*') |
+--------------------------------------+
| Joan Collins Jill Effingham          |
| Mike Malone Jill Effingham           |
| Shirley Horn Joe Cocker Wayne Shorter |
| Betty LaVette                        |
| June Del Ansom                       |
| Lewis Dodgson L. Frank Baum          |
| Peter Collins Henry Collins          |
+--------------------------------------+
```

Demo 39: Using child::* and child::an_name. The literal has two types of child elements for <animal> . One of the function calls retrieves all subelements and the other only the an_name subelements.

```
select
ExtractValue('<animal><an_name>Tweetie</an_name><an_name>Sylvester</an_name><pr
   ice>24</price></animal>' ,'/animal/child::an_name')  as col_1
,

   ExtractValue('<animal><an_name>Tweetie</an_name><an_name>Sylvester</an_name>
   <price>24</price></animal>' ,'/animal/child::*')  as Col_2
;
+------------------+----------------------+
| col_1            | Col_2                |
+------------------+----------------------+
| Tweetie Sylvester | Tweetie Sylvester 24 |
+------------------+----------------------+
```

Demo 40: child::* says to retrieve all child subelements. here we get all child elements of books This output is not very appealing

```
select extractValue(datax,'/book/child::*')
from  xml_bk_1
limit 2
;
```

```
+-------------------------------------------------------------------------------
----------------------+
| extractValue(datax,'/book/child::*')
                      |
+-------------------------------------------------------------------------------
----------------------+
| SQL is Fun




     29.95 Addison Wesley                                |
| Databases: an Introduction




     79.95 Addison Wesley |
+-------------------------------------------------------------------------------
----------------------+
2 rows in set (0.00 sec)
```

6.3. Testing along the path

It is possible to do a test in the path and then move up the path to get the data to extract. We want to test of the topic for SQL books and then walk back up the path to books and then down to price to display the price for SQL books.

Demo 41:

```
 Select id
, extractValue(datax,'(//title)') as "All Titles"
, extractValue(datax,'(//categories/topic[self:text()="SQL"]/../../price)') as
"SQL Title Prices"
from   xml_bk_1
;
+----+------------------------------+------------------+
| id | All Titles                   | SQL Title Prices |
+----+------------------------------+------------------+
|  1 | SQL is Fun                   | 29.95            |
|  2 | Databases: an Introduction   | 79.95            |
|  3 | Visual Basic and MySQL       | 58.95            |
|  4 | Databases- Design and Logic  |                  |
|  5 | The truth about everything   |                  |
|  6 | Alice Goes to Oz             |                  |
|  7 | The House Inside             |                  |
+----+------------------------------+------------------+
```

Demo 42: Two ways to get the first name of the author with the last name of Effingham.
The elements first and last are subelements of name, so we can test for last while we go down the path (version1)
or we can go down to last and do a self test and then go back up to name and down to first.

```
select id
, extractValue (datax, '(//name[last="Effingham"]/first)') As
Effingham_FirstName_v1
, extractValue (datax, '(//name/last[self:text()="Effingham"]/../first)') As
Effingham_FirstName_v2
from xml_bk_1
;
+----+------------------------+------------------------+
| id | Effingham_FirstName_v1 | Effingham_FirstName_v2 |
+----+------------------------+------------------------+
|  1 | Jill                   | Jill                   |
|  2 | Jill                   | Jill                   |
|  3 |                        |                        |
|  4 |                        |                        |
|  5 |                        |                        |
|  6 |                        |                        |
|  7 |                        |                        |
+----+------------------------+------------------------+
```

Demo 43: Getting the name of the first and the second author

```
Select id
, Concat(extractValue(datax, '//name[1]/first' )  , ' '
      ,  extractValue(datax, '//name[1]/last' )  )  as FirstAuthor
, Concat(extractValue(datax, '//name[2]/first' )  , ' '
      ,  extractValue(datax, '//name[2]/last' )  )  as SecondAuthor
from xml_bk_1;
+----+----------------+----------------+
| id | FirstAuthor    | SecondAuthor   |
+----+----------------+----------------+
|  1 | Joan Collins   | Jill Effingham |
|  2 | Mike Malone    | Jill Effingham |
|  3 | Shirley Horn   | Joe Cocker     |
|  4 | Betty LaVette  |                |
|  5 | June Del Ansom |                |
|  6 | Lewis Dodgson  | L. Frank Baum  |
|  7 | Peter Collins  | Henry Collins  |
+----+----------------+----------------+
```

6.4. Trying to clean up the demo with all the extra space.

Why did we get all of that extra space in a previous demo? If you go back to the inserts you might notice there is a lot of whitespace and new line characters in the string literals we are inserting. That is what is showing up in the result.

Demo 44: replacing the new line characters using \n

```
select replace(extractValue(datax,'/book/child::*'), '\n', '')   as Result
from  xml_bk_1
limit 2;

+-------------------------------------------------------------------------------
---------------+
| Result
               |
+-------------------------------------------------------------------------------
```

```
---------------+
| SQL is Fun                                    29.95 Addison Wesley
               |
| Databases: an Introduction                                          79.95
Addison Wesley |
+-------------------------------------------------------------------------
---------------+
2 rows in set (0.00 sec)
```

Demo 45: This nests replacing the new line characters and the spaces but it also messes up our data

```
select
replace(replace(extractValue(datax,'/book/child::*'), '\n', ''), ' ' ,'')
   as Result
from  xml_bk_1
limit 2;
+-----------------------------------------+
| Result                                  |
+-----------------------------------------+
| SQLisFun29.95AddisonWesley              |
| Databases:anIntroduction79.95AddisonWesley |
+-----------------------------------------+
2 rows in set (0.00 sec)
```

Demo 46: This is better since it replaces any 2 blank pattern; we are less likely to have two blank patterns in the data values. But we might and lost the data value integrity

```
select
replace(replace(extractValue(datax,'/book/child::*'), '\n', ''), '  ' ,'')
   as Result
from  xml_bk_1
limit 2;
+-------------------------------------------------+
| Result                                          |
+-------------------------------------------------+
| SQL is Fun 29.95 Addison Wesley                 |
| Databases: an Introduction79.95 Addison Wesley |
+-------------------------------------------------+
```

It would be better to store the data without the blanks and new lines commonly used to format the data for human reading; store the data as data and then format when it is being displayed.

Demo 47: Using version 2 of the file: the inserts are in the demo but this is the create table statement and the first insert. The xml string is a string with no whitespace padding. Harder to read for us; easier for us when dealing with the computer system.

```
create table xml_bk_2(
  id    int primary key
, datax TEXT)
;
truncate table xml_bk_2;
insert into xml_bk_2 values (1,
'<book><title>SQL is
Fun</title><authors><name><first>Joan</first><last>Collins</last></name><name><
first>Jill</first><last>Effingham</last></name></authors><categories><topic>SQL
```

```
</topic><topic>DB</topic></categories><price>29.95</price><publisher>Addison
Wesley</publisher></book>'
);
```

Demo 48: Using the query on this table without any replace cleanup.

```
select extractValue(datax,'/book/child::*')
from  xml_bk_2
limit 2;
+-------------------------------------------------+
| extractValue(datax,'/book/child::*')            |
+-------------------------------------------------+
| SQL is Fun 29.95 Addison Wesley                 |
| Databases: an Introduction 79.95 Addison Wesley |
+-------------------------------------------------+
```

6.5. Having duplicate element names

In a table each column has a unique name, but we can use the same attribute name in different tables. In an xml document we might use the same element identifier for different purposes. We would want to avoid that but is might happen.

This is an xml string where the person creating the string decided that books have names and authors have names and publishers have names.

```
<book>
   <name>SQL is Fun</name>
   <authors>
      <name>Joan Collins</name>
      <name>Jill Effingham</name>
   </authors>
   <categories>
   <topic>SQL</topic><topic>DB</topic></categories>
   <price>29.95</price>
   <publisher>
      <name>Addison Wesley</name>
   </publisher>
</book>
```

Demo 49: What do we get with that example if we ask for the name attributes using //name as our XPath expression?
We get all of the names in one string separated by spaces.

```
select extractValue(
 '<book><name>SQL is Fun</name><authors><name>Joan Collins</name><name>Jill
Effingham</name></authors><categories><topic>SQL</topic><topic>DB</topic></cate
gories><price>29.95</price><publisher><name>Addison
Wesley</name></publisher></book>', '//name') as Result;

+-----------------------------------------------------+
| Result                                              |
+-----------------------------------------------------+
| SQL is Fun Joan Collins Jill Effingham Addison Wesley |
+-----------------------------------------------------+
```