IDUG Solutions Journal

IDUG Solutions Journal

Table of Contents

List of Examples

From The Editor's Desk

Philip Nelson

Welcome to the new look IDUG Solutions Journal, developed in response to your feedback received over the last few months.

Our main feature article this month is a case study in producing and consuming documents in ODF (Open Document Format) using DB2 pureXML.

We are also pleased to have a full complement of columns from our regular contributors.

Using DB2 pureXML and ODF Spreadsheets

Abstract

ODF (Open Document Format) is a file format for office documents. At a high level, the ODF specification requires five mandatory XML documents and other optional items. This article will describe how DB2 pureXML could be used to handle ODF spreadsheet documents. Other ODF documents such as word documents, presentations, formulas, etc. could also be handled by DB2 because the ODF specification refers mainly to an archive of XML documents. However, for the sake of this article, we are considering only spreadsheets because it is easier to co-relate the rows and columns of the spreadsheet document and that of a DB2 table. This article can even be further extended into an on-line document editing software with .ods as file format and DB2 pureXML as the database.

Introduction

The ODF specification and DB2 pureXML are introduced below.

The ODF specification

The Open Document Format specification was originally developed by Sun [http://www.sun.com/] and the standard was developed by OASIS Open Document Format for Office Applications (Open Document) TC – OASIS ODF TC. This standard is based on XML format originally created and implemented by the OpenOffice.org office suite. In addition to being a free and open OASIS standard, it is published (in one of its version 1.0 manifestations) as an ISO/IEC international standard, ISO/IEC 26300:2006 Open Document Format for Office Applications (OpenDocument) v1.0 [http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=43485] .

DB2 pureXML

DB2 pureXML [http://www-01.ibm.com/software/data/db2/xml/] is IBM software for management of XML data. It eliminates much of the work typically involved in the management of XML data. From validation of XML documents, to generation of documents and support for querying XML data in the form of XQuery [http://www.w3.org/TR/xquery/] and traditional SQL, DB2 pureXML is a complete solution to manage XML data in enterprise-scale databases.

The ODF document

An ODF document is a zipped file of five mandatory XML files and other optional components. This can be seen by running your favorite un-zip utility to list the components. Here is the result after running the unzip command on an .ods file from command line to list the files. The XML files are highlighted below.

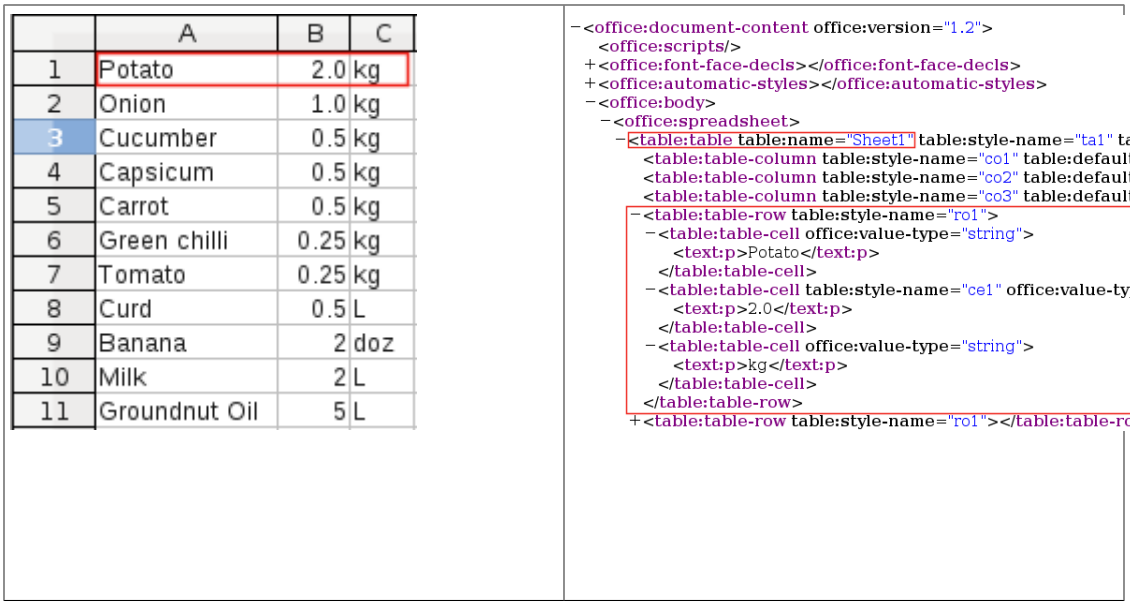
```
nsubrahm@NSUBRAHM:~/Desktop/dW$ unzip -l Shop.ods
Archive:  Shop.ods
  Length     Date   Time    Name
 --------    ----   ----    ----
       46  05-13-09 09:44   mimetype
        0  05-13-09 09:44   Configurations2/statusbar/
        0  05-13-09 09:44   Configurations2/accelerator/current.xml
        0  05-13-09 09:44   Configurations2/floater/
        0  05-13-09 09:44   Configurations2/popupmenu/
        0  05-13-09 09:44   Configurations2/progressbar/
        0  05-13-09 09:44   Configurations2/menubar/
        0  05-13-09 09:44   Configurations2/toolbar/
        0  05-13-09 09:44   Configurations2/images/Bitmaps/
     7931  05-13-09 09:44   content.xml
     6482  05-13-09 09:44   styles.xml
      777  05-13-09 09:44   meta.xml
     5459  05-13-09 09:44   Thumbnails/thumbnail.png
     7228  05-13-09 09:44   settings.xml
     1896  05-13-09 09:44   META-INF/manifest.xml
 --------                   -------
    29819                   15 files
```

Here is a short description of XML files as seen above.

File name	Usage	Location in zipped file
manifest.xml	Information about the files contained in the package.	/META-INF
styles.xml	Styles used in the document content and automatic styles used in the styles themselves.	/ (root)
settings.xml	Application-specific settings, such as the window size or printer information.	/ (root)
content.xml	Document content and automatic styles used in the content.	/ (root)
meta.xml	Document meta information, such as the author or the time of the last save action.	/ (root)

Table 1: XML files of an ODF document

For this article, we are interested in content.xml file which will hold the contents of the document. The document used for this article is a simple shopping list named Shop.ods. The below table shows the way the first row was entered in the spreadsheet document using OpenOffice and as seen by viewing the corresponding content.xml file in Firefox browser.



	A	B	C
1	Potato	2.0	kg
2	Onion	1.0	kg
3	Cucumber	0.5	kg
4	Capsicum	0.5	kg
5	Carrot	0.5	kg
6	Green chilli	0.25	kg
7	Tomato	0.25	kg
8	Curd	0.5	L
9	Banana	2	doz
10	Milk	2	L
11	Groundnut Oil	5	L

Table 2: Row in a ODF spreadsheet and as saved in content.xml file

A bit of hacking

Now that we know a bit about an ODF document, let us try to create one without actually running a standard software dealing with ODF documents. As noted above, the content.xml file holds the data entered while creating the spreadsheet. So, if this file is changed and zipped along with the rest of the files in the original archive (that is, the .ods file), we will be having a “legal” ODF spreadsheet. To test this, follow the steps below.

1. Open the content.xml file in a regular text editor.

2. In Table 2, the element <table:table-row> has been highlighted. Type in the contents of the highlighted section immediately as a sibling of another <table:table-row> element.

3. In the pasted element, make a change in any of the table:table-cell/text:p element say, something like this.

```
nsubrahm@NSUBRAHM:~/Desktop/dW$ unzip -l Shop.ods
Archive:  Shop.ods
  Length      Date    Time    Name
 ---------  ---------- -----   ----
       46  05-13-09 09:44   mimetype
        0  05-13-09 09:44   Configurations2/statusbar/
        0  05-13-09 09:44   Configurations2/accelerator/current.xml
        0  05-13-09 09:44   Configurations2/floater/
        0  05-13-09 09:44   Configurations2/popupmenu/
        0  05-13-09 09:44   Configurations2/progressbar/
        0  05-13-09 09:44   Configurations2/menubar/
        0  05-13-09 09:44   Configurations2/toolbar/
        0  05-13-09 09:44   Configurations2/images/Bitmaps/
     7931  05-13-09 09:44   content.xml
     6482  05-13-09 09:44   styles.xml
      777  05-13-09 09:44   meta.xml
     5459  05-13-09 09:44   Thumbnails/thumbnail.png
     7228  05-13-09 09:44   settings.xml
     1896  05-13-09 09:44   META-INF/manifest.xml
 ---------                     -------
    29819                     15 files
```

1. Save the edited content.xml file in the same folder where the Shop.ods file was unzipped into replacing the older one.

2. Select all the contents (that is, the ones mentioned in the manifest.xml file) of the unzipped folder and zip them. Name the zipped file as NewShop.ods.

3. Now, open NewShop.ods in OpenOffice or IBM Lotus Symphony and confirm that the hacking is successful. Note that, OpenOffice successfully opens a file with a .zip extension, so long it is a valid ODF document whereas, Symphony requires the extension to be .ods.

It is now clear that, to “read” an existing ODF spreadsheet in its simplest form, we have to navigate the path of the content.xml XML document. Similarly, to “modify” an existing ODF spreadsheet, we should generate the content.xml XML document correctly. One can, of course, extend this to create and/or modify other XML documents in the .ods archive. But, for the sake of this article, we will concentrate only on the creation of content.xml XML document.

Scheme of this article

With this knowledge of XML documents in an ODF document archive, we will now proceed to demonstrate the usage of DB2 pureXML with ODF documents. We consider the scheme below.

1. We consider an ODF spreadsheet of a simple shopping list where the item name, unit of measure (UOM) and the units required are provided.

2. The content.xml document of this ODF spreadsheet is imported into a DB2 table with an XML column.

3. For the sake of this article we will use the LOAD statement in DB2 to load the content.xml document and ignore any other methods of inserting XML documents into the DB2 table.

4. We will refer to another traditional table which stores the price list.

5. Using the table for price list and the table with XML column for content.xml, we will generate another content.xml XML document that will hold the items with their corresponding prices.

Inserting XML document into DB2 table

We will consider a table to demonstrate DB2 pureXML capabilities. The first table will be keyed via an ID column to differentiate multiple documents. The second column of the table will hold the content.xml file of the ODF spreadsheet document in a XML column. The DDL for this table is reproduced here :

Example 1. DB2 table to hold content.xml file of the ODF spreadsheet

```
-- DOCCONTENTXML table to hold the
content.xml file of a ODF document.
-- The table is keyed on DOCID a
running sequence number.
CREATE TABLE ODF.DOCCONTENTXML (
      DOCID
INTEGER NOT NULL,
      DOCCONTENT XML NOT NULL
) ;
```

IMPORT statements

Inserting the XML document into the table can be done using a simple INSERT SQL statement or using IMPORT statement. However, since we have a considerably sized XML document, we will use IMPORT statement from the command line. To use IMPORT statement, first, save the content.xml in a folder as say, ODSFolder and create a flat file with a single record. Save the flat file as load.txt. This record will have two values separated by a comma as shown below.

Example 2. Contents of flat file to be used for loading DOCCONTENTXML table

```
1,<XDS FIL='content.xml'/>
```

Example 3. IMPORT statement to load content.xml into DOCCONTENTXML table

```
IMPORT FROM load.txt OF DEL XML FROM
     ODSFolder INSERT INTO ODF.DOCCONTENTXML;
```

Alternatively, IBM Data Studio Developer can be used to load into XML column as it offers a convenient GUI.

Converting the content.xml into row-column format

Getting the cell values from the XML file

To get to the cell values of the spreadsheet, we will start with the required XPath expression. Then, we will develop a working query around the XPath expression to get the results.

The XPath expression

Going back to Table 2, the left column shows the first row of the spreadsheet highlighted. The right column is a part of the generated content.xml file and some of the elements collapsed. The relevant part are put in red boxes. The first box shows the table:table element with an attribute table:name whose value is “Sheet1”. It is the name of the sheet in the complete spreadsheet. This element is repeated as many times as there are sheets in the spreadsheet with the table:name attribute set to the names as set by the user. The second box is the representation of the highlighted row in the left column. Thus, to get to, say, column B of the first row, the path to be navigated (or, the XPath expression) from the root of XML document would be as below.

Example 4. XPath expression to get first row, second column value

```
/office:document-content/office:body/office:spreadsheet/table:table[1]/table:
```

In other words, we start from the office:document element to office:body and then to office:spreadsheet to look for the first table:table element. The first element represents the first sheet of the spreadsheet document. Then, to get to the second column of first row we navigate to table:row[1]/table:cell[2] and finally to text:p to get the actual text of the cell.

With Firefox Mozilla browser, the above XPath expression can be easily tested using the add-on Xpather (see Resources below). We follow the steps below :

1. Open the content.xml in Firefox Mozilla browser. Right-click on the page and select 'Show in XPather'.

1. The XPather dialog box opens up. Enter the XPath expression in the dialog box and click 'Eval'.

The required query

To wrap the XPath expression into a query, we first note the XML namespaces used. The content.xml file has the namespaces defined as shown below :

Prefix	Namespace
`office`	`“urn:oasis:names:tc:opendocument:xmlns:office:1`
`table`	`“urn:oasis:names:tc:opendocument:xmlns:table:1.`
`text`	`“urn:oasis:names:tc:opendocument:xmlns:text:1.0`

Table 7: Prefix and namespace URI used in content.xml document

The query shown below will use the XML namespaces and XPath expression (that we obtained in Section 6.2) to extract the cell values from the loaded content.xml file. The text in bold-blue shows the XPath expression used to extract cell values.

Example 5. Query to extract cell values from content.xml into tabular format

```
SELECT
X.*
FROM
XMLTABLE(
XMLNAMESPACES('urn:oasis:names:tc:opendocument:xmlns:office:1.0' AS
"office",
'urn:oasis:names:tc:opendocument:xmlns:table:1.0' AS "table",
'urn:oasis:names:tc:opendocument:xmlns:text:1.0' AS "text"),
'db2-fn:sqlquery("SELECT
DOCCONTENT
FROM
ODF.DOCCONTENTXML
WHERE DOCID
= 1")
/office:document-content/office:body/office:spreadsheet/table:table[1]/table:
COLUMNS
"ITEMNAME" CHARACTER (20)  PATH
'table:table-cell[1]/text:p',
"QUANTITY" DECIMAL   (4,2) PATH
'table:table-cell[2]/text:p',
"UOM"      CHARACTER (5)   PATH
'table:table-cell[3]/text:p'
) AS X;
```

To see the content.xml document in traditional rows and columns format, we create a view using the query above.

Example 6. View on the content.xml document

```
CREATE VIEW ODF.V_CONTENT AS
(
SELECT
X.*
FROM
XMLTABLE(
XMLNAMESPACES('urn:oasis:names:tc:opendocument:xmlns:office:1.0' AS
"office",
'urn:oasis:names:tc:opendocument:xmlns:table:1.0' AS "table",
'urn:oasis:names:tc:opendocument:xmlns:text:1.0' AS "text"),
'db2-fn:sqlquery("SELECT
DOCCONTENT
FROM
ODF.DOCCONTENTXML
WHERE
DOCID =
1"
)/office:document-content/office:body/office:spreadsheet/table:table[1]/table
COLUMNS
"ITEMNAME" CHARACTER (20)  PATH
'table:table-cell[1]/text:p',
"QUANTITY" DECIMAL   (4,2) PATH
'table:table-cell[2]/text:p',
"UOM"      CHARACTER (5)   PATH
'table:table-cell[3]/text:p'
) AS X
);
```

A simple SELECT query on this view shows that we have successfully 'converted' the content.xml document in the ODF archive for spreadsheet into a row-column format.

Example 7. Result of SELECT on the view V_CONTENT.

```
SELECT * FROM ODF.V_CONTENT ;

ITEMNAME
      QUANTITY UOM
-------------------- -------- ----
Potato
            2.00 kg
Onion                    1.00 kg
Cucumber
            0.50 kg
Capsicum                 0.50 kg
Carrot
            0.50 kg
Green chilli             0.25 kg
Tomato
            0.25 kg
Curd                     0.50 L
Banana
              2.00 doz
Milk                     2.00 L
Groundnut Oil            5.00 L

11 record(s) selected.
```

Shopping prices

We will now build the content.xml for an ODF spreadsheet that will hold the shopping list – discussed above – with an additional column for cost of items in the shopping list. We will also provide an additional row for the grand total of the items in shopping list.

Building the list of shopping prices

We create a simple table that holds the unit price of shopping items with the DDL shown below.

Example 8. DDL for table to hold the shopping prices

```
--Table to hold the unit price of shopping
items
CREATE TABLE ODF.SHOPPER
(ITEMNAME CHARACTER(50),
UNIT_PRICE
DECIMAL (4,2));
```

This table is inserted with values such that all the item names are available in the SHOPPER table. To obtain the list of item names and the cost of items, we use the following query. Note that, this query uses the view V_CONTENT. We could have used the base table directly. Usage of the view makes the query 'compact' and easier to follow.

Example 9. Query to obtain list and cost of items with grand total

```
SELECT
COALESCE(P.ITEMNAME,' TOTAL')
ITEMNAME,
SUM(P.QUANTITY) QTY,
COALESCE(P.UNIT_PRICE, 0) UNIT_PRICE,
SUM(P.COST) COST
FROM
(SELECT
A.ITEMNAME,
A.QUANTITY,
B.UNIT_PRICE,
A.QUANTITY*B.UNIT_PRICE COST
FROM
ODF.V_CONTENT A,
ODF.SHOPPER B
WHERE
UPPER(A.ITEMNAME) = B.ITEMNAME
) P
GROUP BY
ROLLUP(P.ITEMNAME,P.QUANTITY,P.UNIT_PRICE)
HAVING (P.ITEMNAME IS
NULL OR
(P.ITEMNAME IS NOT NULL AND
P.QUANTITY IS NOT NULL AND
P.UNIT_PRICE IS NOT NULL AND
SUM(P.COST) IS NOT NULL AND
SUM(P.QUANTITY) IS NOT NULL
)
)
ORDER BY COALESCE(P.ITEMNAME,' ') DESC
```

The result of the above query is shown below.

Example 10. Result of query for obtaining list and cost of items

```
ITEMNAME             QTY    UNIT_PRICE COST

-------------------- ------ ---------- --------
Tomato
      0.25       6.00 1.50
Potato                  2.00       5.50
 11.00
Onion                   1.00       4.00     4.00
Milk
        2.00      40.00    80.00
Groundnut Oil           5.00
 30.00   150.00
Green chilli            0.25       8.00   2.00
Curd
                0.50      20.00    10.00
Cucumber                0.50
     10.00   5.00
Carrot                  0.50      15.00     7.50
Capsicum                0.50      20.00    10.00
Banana
  2.00       6.00    12.00
TOTAL                   14.50       0.00
293.00
```

View with list and cost of items

Now that we have the result of computation of cost of items, we will take this into a view. This is purely for convenience. We can proceed without using this view.

Example 11. View with list and cost of items in shopping list

```
CREATE VIEW ODF.V_SHOP_COST AS
(SELECT
COALESCE(P.ITEMNAME,' TOTAL') ITEMNAME,
SUM(P.QUANTITY) QTY,
COALESCE(P.UNIT_PRICE, 0) UNIT_PRICE,
SUM(P.COST) COST
FROM
(SELECT
A.ITEMNAME,
A.QUANTITY,
B.UNIT_PRICE,
A.QUANTITY*B.UNIT_PRICE COST
FROM
ODF.V_CONTENT A,
 ODF.SHOPPER B
 WHERE
UPPER(A.ITEMNAME) =
B.ITEMNAME
) P
GROUP BY
ROLLUP(P.ITEMNAME,P.QUANTITY,P.UNIT_PRICE)
HAVING (P.ITEMNAME IS NULL OR
(P.ITEMNAME IS NOT NULL AND
P.QUANTITY
IS NOT NULL AND
P.UNIT_PRICE IS NOT NULL AND
SUM(P.COST) IS NOT NULL
AND
SUM(P.QUANTITY) IS NOT NULL
)
)
);
```

XQuery Update to build content.xml document

At this point, we have 'shredded' the shopping list (from the content.xml of ODF spreadsheet), joined with the relational table having the unit prices of items and generated a relational output of the cost of items in shopping list with the grand total. Our aim is now to convert this relational output into the format of content.xml so that the result is a complete ODF spreadsheet.

One way of building the content.xml is to start from the scratch and we build the whole of the document. This approach would be quite tedious. A better way would be to generate the relevant part in the content.xml part and then insert into an otherwise empty content.xml document. This approach will also allow us to make use of XQuery Update facility in DB2 pureXML.

A quick side note on XQuery Update

XQuery is a query and functional programming language designed to query XML documents. It became a W3C Candidate Recommendation on 23rd January, 2007. XQuery Update facility is an extension to XQuery that allows update (insert, modify and delete) of the XML documents. It became a W3C Candidate Recommendation on 14th March, 2008.

Generating the content.xml for the shopping prices

To start with, we will create a dummy content.xml that is complete and ODF compliant in all respects except it would resemble an empty spreadsheet. Next, we will generate XML sequences that is correct for a row and it's columns of a spreadsheet. Then, we will collect all of these XML sequences and insert into the dummy content.xml document.

To create the dummy document, we simply open the spreadsheet for shopping list, delete all rows and columns and save it with another name. Then, we unzip this document and refer the content.xml document. Here is a screenshot of how this document would look like.

Insert this dummy document into the ODF.DOCCONTENTXML table with DOCID as zero using the IMPORT method described before. Our aim now is to update this document such that, <table:table-row> elements are added as children of <table:table> and as siblings of <table:table-column>.

Building the <table:table-row> elements

In our example, each row of spreadsheet would be having four columns. Therefore, each row element – as required in content.xml – would have four children for the number of columns. A sample of <table:table-row> is shown below. The XML namespace declarations have been omitted for the sake of brevity.

Example 12. A sample <table:table-row> element

```
<table:table-row
table:style-name="ro1">
<table:table-cell
office:value-type="string"><text:p>Tomato</text:p></table:table-cell>
<table:table-cell
office:value-type="string"><text:p>.25</text:p></table:table-cell>
<table:table-cell
office:value-type="string"><text:p>6.00</text:p></table:table-cell>
<table:table-cell
office:value-type="string"><text:p>1.5000</text:p></table:table-cell>
</table:table-row>
```

To build the <table:table-row> elements, then, we need to generate <table:table-cell> for each column of the row in the relational table and then aggregate them to a single <table:table-row> element. We will use the XMLAGG function to build the elements and store in a table the generated elements so that it is easier to use while doing the update of XML document. The table to hold the generated elements is defined as shown below.

Example 13. Table to hold generated <table:table-row> elements

```
CREATE TABLE ODF.SHOP_COST_XML (SHOP_COST
XML);
```

Example 14. Insert generated <table:table-row> elements with <dummy> parent

```
INSERT INTO ODF.SHOP_COST_XML (SHOP_COST)
(SELECT
XMLDOCUMENT(
XMLELEMENT(NAME "dummy",
XMLNAMESPACES('urn:oasis:names:tc:opendocument:xmlns:office:1.0' AS
"office",
'urn:oasis:names:tc:opendocument:xmlns:table:1.0' AS
"table",
'urn:oasis:names:tc:opendocument:xmlns:text:1.0' AS "text"
),
XMLAGG(
XMLELEMENT(NAME "table:table-row",
XMLATTRIBUTES('ro1' AS
"table:style-name"),
XMLELEMENT(NAME "table:table-cell",
XMLATTRIBUTES('string' AS "office:value-type"),
XMLELEMENT(NAME "text:p", STRIP(O.ITEMNAME)
)
),
XMLELEMENT(NAME
"table:table-cell",
XMLATTRIBUTES('string' AS "office:value-type"),
XMLELEMENT(NAME "text:p", O.QTY )
),
XMLELEMENT(NAME
"table:table-cell",
XMLATTRIBUTES('string' AS "office:value-type"),
XMLELEMENT(NAME "text:p", O.UNIT_PRICE )
),
XMLELEMENT(NAME
"table:table-cell",
XMLATTRIBUTES('string' AS "office:value-type"),
XMLELEMENT(NAME "text:p", O.COST)
)
)
ORDER BY O.ITEMNAME DESC
)
)
)
FROM
ODF.V_SHOP_COST O
);
```

Finally, we are now ready to update our dummy XML document. The query to do the update is shown below. Note that, this query is not inserting the resulting into any table. It is left to the reader to route the output to a table column, file, message queue, etc.

Example 15. Updated content.xml document

```
VALUES XMLQUERY(
'declare namespace
office="urn:oasis:names:tc:opendocument:xmlns:office:1.0" ;
declare
namespace table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"
;
transform
copy $dummy := db2-fn:sqlquery("SELECT DOCCONTENT FROM
ODF.DOCCONTENTXML WHERE DOCID = 0") ,
$rows :=
db2-fn:sqlquery("SELECT * FROM ODF.SHOP_COST_XML")
modify do
insert
$rows/dummy/* as last into
$dummy/office:document-content/office:body/office:spreadsheet/table:table
return $dummy '
);
```

To test that our update has worked successfully run, we route the output of query above to a file called content.xml. Replace the content.xml file in the unzipped archive of our original shopping list spreadsheet with the one created now. Zip the files back and save it with name as ShopPrices.ods and open with OpenOffice. Here is the screenshot.

And, we are done !

Next steps

Further enhancements may involve manipulating other XML documents in a ODF compliant archive. For example, the meta.xml document holds information about the document itself. This information can be accessed and set by going to File -> Properties... dialog box. For example, for the document in this article (Shop.ods), this is how the dialog box looks like.

The corresponding meta.xml document (partial snapshot) is shown below.

Thus, while generating the spreadsheet, one can write queries to generate the meta.xml document with correct values for creator, creation time-stamp, etc.

Resources

Important resources :

1. DB2 Express-C [http://www-01.ibm.com/software/data/db2/express/download.html] – Free for download, develop, deploy and distribute.

2. DB2 v9 pureXML [http://www.ibm.com/db2/9] – All information about DB2 V9 pureXML

3. DB2 v9 Product Manuals [http://www.ibm.com/support/docview.wss?rs=71&uid=swg27009552] – Especially, refer the ones for XQuery and XML.

4. IBM Data Studio [http://www-01.ibm.com/software/data/studio/] – Govern, design, develop and deploy databases and data driven applications.

5. ODF Specification [http://www.oasis-open.org/committees/office/] – Link to the complete specification

6. OpenOffice.org XML essentials [http://books.evc-cit.info/book.php] – Nice explanation of XML data formats.

7. Thinking XML : The open office file format [http://www.ibm.com/developerworks/xml/library/x-think15/] – An earlier developerWorks article discussing the file formats of OpenOffice.

8. Manage ODF and Microsoft Office 2007 documents with DB2 9 pureXML [http://www.ibm.com/developerworks/db2/library/techarticle/dm-0705gruber/] – An earlier developerWorks article on using ODF and Microsoft Office documents with DB2 9 pureXML.

9. XPather add-on [https://addons.mozilla.org/addon/1192] – Nifty tool for Firefox Mozilla users to quickly check an XPath expression.

10. Update XML in DB2 9.5 [http://www.ibm.com/developerworks/db2/library/techarticle/dm-0710nicola/%5C] – Nice article on developerWorks explaining update of XML documents.

Biography

Nagesh Subrahmanyam is a Software Developer in IBM Global Services, India.....

Giving You A Reason

Compelling Reasons to Upgrade DB2

Roger Miller

Is your current bowl getting tight? What is limiting you? Is it CPU? Virtual storage? Latching? Catalog and directory? Utilities? Are you currently running DB2 9? V8? V7? Upgrading to a bigger bowl may be just what you need.

Should you upgrade to DB2 10?

To 10, or not to 10, when and how are the questions. Whether 'tis nobler in the mind to suffer the slings and arrows of outrageous limits, Or to take arms against a sea of troubles, And by upgrading, end them? To die: to sleep; No more; and by a sleep to say we end the heart-aches and the thousand natural shocks that old versions are heir to. ‘tis a consummation devoutly to be wished. To die, to sleep. To sleep: perchance to dream: ay, there’s the rub; For in that sleep of death what dreams may come when versions have shuffled off this mortal coil, must give us pause: There’s the respect that makes calamity of too long life for old versions. [With abject apologies to the Bard and to Hamlet act 3 scene 1]

The answer to upgrading to 10 is a definite Yes. The question is not so much whether to upgrade as when and how to upgrade. If you are running DB2 9 today, then DB2 10 is in your near future, giving you more room to grow, with higher limits, lower costs, and more for less. If you are running DB2 V8 today, then you have a choice of jumping to DB2 9 or directly to DB2 10. So the key question is …

When and how should I upgrade to DB2 10?

As of early August 2010, DB2 10 is in beta. Some of the key information for making this decision is known, but some is not yet. DB2 for z/OS V8 end of service is set for April 2012, 21 months from now. The unknown information includes the date for DB2 10 general availability, V8 extended service, and pricing, which will come in later announcements.

While DB2 10 is expected to be better than prior versions, it will have maturity, stability, and service delivery similar to other software and versions, with more defects at first, then fewer as the software matures. Determining when the software is ready for a specific customer and when the customer is ready for the software depends upon the specific customer resources, prior experience, and the value for the improvements versus the need for stability. Many customers depend upon tools or other software, and having that software that works with DB2 is a prerequisite. When this information is known, we can help answer the question. This web page can help. http://www.ibm.com/support/docview.wss?uid=swg21006951 Content of the two versions is available, although the details of DB2 10 are still not public. Here is a summary of the two versions -

DB2 9: Robust, Scalable, Available and Easily Manageable

DB2 9 delivers CPU reductions for utilities that are generally in the range of 20% to 30%. Customers report saving terabytes of disk space using index compression. More CPU time is shifted to zIIP processors, reducing costs. Security is improved with more flexible trusted contexts and roles. Resilience is improved as more changes can be made while applications keep running. One table can be replaced quickly with a clone. Indexes and columns can be renamed. Many more utilities can be online.

DB2 9 delivers seamless integration of XML and relational data with pureXML and makes big strides in SQL for productivity and portability of applications. A new storage structure is introduced for large tables. Today's complex applications include both transactions and reporting, so performing both well is required. The key improvements for reporting are optimization enhancements to improve query and reporting performance and ease of use. More queries can be expressed in SQL with new SQL enhancements. Improved data is provided for the optimizer, with improved algorithms. Improved CPU and elapsed times can be achieved with the FETCH FIRST clause specified on a subquery. The INTERSECT and EXCEPT clauses make SQL easier to write.

DB2 10: Cut costs and improve performance

DB2 10 for z/OS provides the best reduction in CPU for transactions and batch in over 20 years. We expect most customers to reduce CPU times between 5% and 10% initially, with the opportunity for much more. Applications which can take advantage of additional benefits, such as hash access, can have larger CPU and memory reductions.

Scalability is the second major benefit, with the ability to run five to ten times as many threads in a single subsystem by moving 80% to 90% of the virtual storage above the bar. Schema evolution or data definition on demand enhancements improves availability. SQL and pureXML improvements extend usability and application portability for this platform. Productivity improvements for application developers and for database administrators are very important as data grows in scale and complexity.

Questions for you

The right answer is not "One size fits all." If we know the key factors for you, we can help you make a better choice. Here are some of the key objectives. Which ones are most important for you?

- Performance improves in both DB2 9 and 10, with larger CPU reductions in DB2 10.
- Scalability is improved a little in DB2 9 and a lot in DB2 10.
- Availability is enhanced in both, with more online changes in both.

- Security is made stronger and more flexible with roles in DB2 9 and with more options in DB2 10.
- Productivity is helped in both releases, with more improvements in DB2 10. Upgrading to DB2 9 is easier than to DB2 10 from DB2 V8.
- Stability is better in more mature versions.
- Skills: What skill set is available within your organization? Do you have people with the right skills and time to plan and run a project? DB2 planning workshops can help with education. Transition classes provide more education for one or both versions. Services could be used if the skills are not presently available.
- Technology adoption model: Are you using the latest technology that is being shipped, or are the operating system and hardware back-level? Is the technology one level back, two or more? This question tells both how much work will be required and the comfort level of your organization for running the latest version.
- Platform management practices: What are your platform management practices? What type of change management practices are in place? How robust is your testing for new software? What is the inventory of software and tools? How many vendors are involved? Which ones? Almost every vendor has software ready for DB2 9, but DB2 10 may take some time after general availability.
- Numbers of servers: How many LPARs and subsystems does the organization have? An organization that has 100 subsystems has a different set of challenges than does one that has 5 subsystems.
- Organizational considerations: What additional organizational factors, such as politics and policies, must be considered?

What version are you running?

Here are the primary recommendations for customers who are running various DB2 versions.

DB2 9 If you are on DB2 9 today, then you are a good candidate for an early upgrade to DB2 10, especially if your custom is to move in the first year after general availability. Listen to reports from early customers and upgrade for the value.

V8 If you are on DB2 V8 today, then the next questions are on timing for you and readiness for the new version. How soon after general availability do you normally upgrade? Are you still in the process of moving to NFM or have you recently finished V8 upgrade? If you just finished, then you probably will wait a few years and use the skip. If you have resources for an upgrade, but DB2 10 is too new, then DB2 9 is probably your next move. If you have the resources and can work with a new version, then skipping to DB2 10 may work for you.

V7 If you are currently on DB2 V7, then upgrade to DB2 V8. Then you can use the skip version upgrade to DB2 10 in a few years.

DB2 has several new versions and upgrade paths for you to consider. This story will be changing, but you can hear the latest at IDUG conferences and on the web. DB2 9 is ready for you now. DB2 10 is still in beta, but is delivering higher limits, lower costs, and more for less.

- http://www.ibm.com/software/data/db2/zos/db2-10/
- http://www.ibm.com/data/db2/zos