IBM DB2 z/OS pureXML

Jane Man, janeman@us.ibm.com

Data Management Solutions
Information Management
IBM Software Group
Data Management – DB2 pureXML
2009
Agenda

• Introduction
• Create a Table – the XML Storage Model
• Insert a Row – Storing XML Data
• Create an Index – Index Structure and EXPLAIN, DML
• Running Utilities – LOAD, UNLOAD, others
• XML Schema – Registration, Validation, Decomposition
• RUNSTATS – XML Support (Koshy John)
pureXML on DB2 for z/OS

- **XML Storage:**
  - Mature, optimized storage infrastructure
    - compact storage with optional high-ratio compression (up to 70% savings)
  - Partitioned table space and data sharing

- **XML Parsing:**
  - Synergy with z/OS XML system services for highly efficient parsing
  - Schema validation exploiting new next-generation XML validating parser (XLXP-C);

- **Specialty Engines:**
  - Redirection of XML processing to the zIIP from DRDA
  - XML Parsing (non-validating) is 100% eligible for redirection to zIIP

- **Query Capability, SQL/XML + XPath:**
  - >95% of the XQuery Use Cases can be rewritten to run in DB2 for z/OS
  - Patent pending highly efficient XPath algorithm for query performance
# System Configuration

## XML Parsing
- Non-Validating Parsing: z/OS XMLSS: z/OS 1.8 (or z/OS 1.7 with APAR OA16303)

## XML Schema Objects
- DSNTIJNX, DSNTIJSG migration/install job
- Creates XML Schema Repository (XSR) tables.
- Creates XML Schema Registration Stored Procedures
- Creates XML Schema Validation UDF (DSN_XMLVALIDATE)

## DB2 Configuration
- XML Storage Pool ZPARMs: XMLVALA (default 200 MB), XMLVALS (10 GB)
- WLM Stored Procedure setup for XML Schema Registration Stored Procedures and DSN_XMLVALIDATE
- Java Stored Procedures setup for XML Schema Registration Stored Procedures
- Bufferpool BP16K0-9 for tables with XML column data
Agenda

• Introduction
• Create a Table – the XML Storage Model
• Insert a Row – Storing XML Data
• Create an Index – Index Structure and EXPLAIN
• Running Utilities – LOAD, UNLOAD, others
• XML Schema – Registration, Validation, Decomposition
CREATE TABLE with an XML Column

CREATE DATABASE BOOKSDB;

CREATE TABLESPACE BOOKSTS IN BOOKSDB;

CREATE TABLE JASON.BOOKS (ID INTEGER, DATE_ADDED DATE) IN BOOKSDB.BOOKSTS;

ALTER TABLE BOOKS ADD COLUMN BOOK XML;

BOOKS:

<table>
<thead>
<tr>
<th>ID</th>
<th>DATE_ADDED</th>
<th>BOOK</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-byte integer</td>
<td>4-byte date</td>
<td>????</td>
</tr>
</tbody>
</table>
XML Storage

1. ALTER ADD an XML column: a DOCID (BIGINT) column is implicitly generated.

2. An XML Table (BIGINT, VARBINARY, VARBINARY) to store XML data is implicitly generated.

3. A NodeID index is created to get from the base table to the XML Table.

4. A DOCID index gets from the XML table to the base table for use with a user-created index.
XML Objects for Non-partitioned Base Table

BOOKSTS
Non-Range-Partitioned (simple, segmented, PBG) Table Space

DOCID INDEX

BOOKS
(ID
DATE_ADDED
DOCID
BOOK (indicator)
BIBLIO (indicator))

BOOKSDB.XBOO0000
PBG TS for BOOKS

XBOOKS
(DOCID
MIN_NODEID
XMLDATA)

BOOKSDB.XBOO0001
PBG TS for BIBLIO

XBOOKS001
(DOCID
MIN_NODEID
XMLDATA)

XBOOKS000 Table

Suppose we added one more XML column (BIBLIO) for illustrative purposes...
DISPLAY DATABASE

-DISPLAY DATABASE (BOOKSDB)

DSNT360I ) ********************************************
DSNT361I ) * DISPLAY DATABASE SUMMARY
   * GLOBAL
DSNT360I ) ********************************************
DSNT362I ) DATABASE = BOOKSDB  STATUS = RW
            DBD LENGTH = 8066

DSNT397I )
NAME    TYPE PART STATUS          PHYERRLO PHYERRHI CATALOG PIECE
-------- ---- ----- ----------------- -------- -------- -------- -----
BOOKSTS TS    RW,AREO*
XBO00000 XS   0001 RW
XBO00001 XS   0001 RW
IRDOCIDB IX    RW
IRNO14H6 IX    L*   RW
IRNODEID IX    L*   RW

******* DISPLAY OF DATABASE BOOKSDB ENDED  ***********************************

Implicitly Generated Objects
SELECT from the DB2 Catalog

SYSXMLRELS shows the relationship between the base table and the XML auxiliary table.

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>XMLTBOWNER</th>
<th>XMLTBNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOK</td>
<td>JASON</td>
<td>XBOOKS</td>
</tr>
<tr>
<td>BIBLIO</td>
<td>JASON</td>
<td>XBOOKS000</td>
</tr>
</tbody>
</table>

Join SYSXMLRELS to SYSTABLES and SYSTABLESPACE to get the name of the auxiliary table space.

<table>
<thead>
<tr>
<th>COLNAME</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOK</td>
<td>XBOO0000</td>
</tr>
<tr>
<td>BIBLIO</td>
<td>XBOO00001</td>
</tr>
</tbody>
</table>

SELECT XR.COLNAME, TS.NAME
FROM SYSIBM.SYSTABLES ST, SYSIBM.SYSXMLRELS XR, SYSIBM.SYSTABLESPACE TS
WHERE XR.TBNAME='BOOKS' AND XR.TBOWNER='JASON' AND ST.NAME = XR.XMLTBNAME AND ST.CREATOR = XR.XMLTBOWNER AND ST.TSNAME = TS.NAME;
TABLESPACE SET REPORT:

TABLESPACE : BOOKSDB.BOOKSTS
TABLE : JASON.BOOKS
INDEXSPACE : BOOKSDB.IRDOCIDB
INDEX : JASON.I_DOCIDBOOKS

XML TABLESPACE SET REPORT:

TABLESPACE : BOOKSDB.BOOKSTS
BASE TABLE : JASON.BOOKS
COLUMN : BOOK
XML TABLESPACE : BOOKSDB.XBOO0000
XML TABLE : JASON.XBOOKS
XML NODEID INDEXSPACE : BOOKSDB.IRNODEID
XML NODEID INDEX : JASON.I_NODEIDXBOOKS
COLUMN : BIBLIO
XML TABLESPACE : BOOKSDB.XBOO0001
XML TABLE : JASON.XBOOKS000
XML NODEID INDEXSPACE : BOOKSDB.IRN014H6
XML NODEID INDEX : JASON.I_NODEIDXBOOKS000

Implicitly Generated Objects
## Implicitly Created Objects

<table>
<thead>
<tr>
<th>Create Table option</th>
<th>Base table</th>
<th>XML table</th>
<th>DocID index</th>
<th>NodeID index</th>
<th>XML Index</th>
</tr>
</thead>
</table>

NM: Name, TS: Table Space, ST: Stogroup, TB: Table, IS: Index Space, G: Generated.
CREATE TABLESPACE _BKPARTTS_
IN BOOKSDB
NUMPARTS 2;

CREATE TABLE JASON._BOOKSPART_
(ID INTEGER,
 DATE_ADDED DATE,
 BOOK XML,
 BIBLIO XML)
PARTITION BY (ID)
 (PART 0001 VALUES (50),
  PART 0002 VALUES (100))
;
XML Objects for Partitioned Base Table

**BKPARTTS**
Range-Partitioned Table Space (2 partitions)

**BOOKSDB.XBOO0002**
Partitioned Table Space (2 parts)

**BOOKSDB.XBOO0003**
Partitioned Table Space (2 parts)

**BOOKSPART**
(part 1)
(ID
DATE_ADDED
DOCID
BOOK (indicator)
BIBLIO (indicator))

**BOOKSPART**
(part 2)
(ID
DATE_ADDED
DOCID
BOOK (indicator)
BIBLIO (indicator))
Customize Your Objects

- **BP16K0** – default 16k bufferpool for implicitly created XML table spaces
- **Storage Group** specification is inherited from the base table, database, or SYSDEFLT
- **LOG, LOCKMAX, COMPRESS** attributes are inherited from the base table space
- **Default values** are provided for many attributes: segsize, pctfree, priqty, secqty, maxrow, freepage
XML Table Space Size

- Basic XML storage after parsing:
  - 1-to-1 average is a good rule of thumb
  - as small as 0.3 (strip ws w/ compression) of the original document
  - as large as 1.5 (preserve ws w/o compression) of original document

- An XML table space always use 16KB pages.

- For non range-partitioned base table spaces, PBG table space is used for XML.

- Range-partitioned base table spaces: XML partitioning follows base table partitioning.

- The number of rows to fit into a relational partition is limited by the number of documents to fit into an XML partition.

  - For example, average 4K doc size, 32GB partition can roughly store 8M documents (or 6M to be safe).
Agenda

• Introduction
• Create a Table – the XML Storage Model
• Insert a Row – Storing XML Data
• Create an Index – Index Structure and EXPLAIN, DML
• Running Utilities – LOAD, UNLOAD, others
• XML Schema – Registration, Validation, Decomposition
Insert a Small Document

INSERT INTO BOOKS VALUES
(103, CURRENT_DATE, XMLPARSE(DOCUMENT '<book><title>Effective XML</title>
    <author>Jason Cu</author>
    </book>'));

1. Non-XML data is inserted as expected. A unique DOCID is generated.

2. XML data is parsed by XMLSS and tree nodes are produced.

3. The XMLDATA is stored along with the base table DOCID and a generated NODEID.

4. (not shown) The DOCID and NODEID indexes are populated.
Insert a Larger Document

```sql
INSERT INTO BOOKS VALUES
(104,
 CURRENT DATE,
XMLPARSE(DOCUMENT 'book><title>Effective XML</title>
<author>Jason Cu</author>
<notes>
    .... lots of data ...
</notes>
</book>'));
```

**Books**

<table>
<thead>
<tr>
<th>ID</th>
<th>DATE_ADDED</th>
<th>DOCID</th>
<th>BOOK</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>2008-06-10</td>
<td>1000</td>
<td>***</td>
</tr>
<tr>
<td>104</td>
<td>2008-06-10</td>
<td>1001</td>
<td>***</td>
</tr>
</tbody>
</table>

**Xbooks (XML Aux Table)**

<table>
<thead>
<tr>
<th>DOCID</th>
<th>MIN_NODEID</th>
<th>XMLDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>02</td>
<td>**********</td>
</tr>
<tr>
<td>1001</td>
<td>02</td>
<td>**********</td>
</tr>
<tr>
<td>1001</td>
<td>020202</td>
<td>**********</td>
</tr>
</tbody>
</table>

*If the XML tree nodes cannot fit on one 16K record, they are split into multiple records.*

*The NODEID index enables fast traversal of the records in document order.*
Parsed XML tree nodes are packed into VARBINARY records by sub-tree.

The NodeID index determines the relationship between records.

In this example, RID1 is the parent record, and RID2 is the descendant record.

A special “proxy node” (node 2) exists in RID1 to indicate that there is a child beneath.
STRINGID compression

```xml
<book>
  <title>Effective XML</title>
  <author>Jason Cu</author>
</book>
```

- **Names** (element, attribute, namespace URI/prefix, PI Targets) are replaced by a 4-byte integer STRINGID.

- The **SYSIBM.SYSXMLSTRINGS** catalog table provides the mapping between STRINGID and string.

- SYSXMLSTRINGS is **cached for fast access**. Once the cache is primed, the catalog is rarely accessed.
Agenda

- Introduction
- Create a Table – the XML Storage Model
- Insert a Row – Storing XML Data
- Create an Index – Index Structure and EXPLAIN, DML
- Running Utilities – LOAD, UNLOAD, others
- XML Schema – Registration, Validation, Decomposition
CREATE INDEX

CREATE INDEX BOOKAUTHORIDX
ON JASON.BOOKS(BOOK)
GENERATE KEY USING
XMLPATTERN '/book/author'
AS SQL VARCHAR(40);

BOOKAUTHORIDX index:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>DOCID</th>
<th>NODEID</th>
</tr>
</thead>
<tbody>
<tr>
<td>John Smith</td>
<td>1006</td>
<td></td>
</tr>
<tr>
<td>George Doe</td>
<td>1006</td>
<td></td>
</tr>
<tr>
<td>Mary Lee</td>
<td>1006</td>
<td></td>
</tr>
<tr>
<td>Patricia G</td>
<td>1008</td>
<td></td>
</tr>
</tbody>
</table>

Row #1, DOCID=1006
<book>
  <title>Who’s Who?</title>
  <author>John Smith</author>
  <author>George Doe</author>
  <author>Mary Lee</author>
</book>

Row #2, DOCID=1007
<book>
  <title>That’s That</title>
</book>

Row #3, DOCID=1008
<book>
  <title>Who’s That</title>
  <author>Patricia G</author>
</book>
EXPLAINing Index Access

CREATE INDEX BOOKTITLEIDX
ON JASON.BOOKS(BOOK)
GENERATE KEY USING
XMLPATTERN '/book/title'
AS SQL VARCHAR(40);

CREATE INDEX BOOKAUTHORIDX
ON JASON.BOOKS(BOOK)
GENERATE KEY USING
XMLPATTERN '/book/author'
AS SQL VARCHAR(40);

Single Index Access

EXPLAIN ALL SET QUERYNO=123 FOR
SELECT * FROM JASON.BOOKS
WHERE XMLEXISTS
('/book[title="Effective XML"]'
PASSING BOOK);

Multiple Index Access (for AND)

EXPLAIN ALL SET QUERYNO=124 FOR
SELECT * FROM JASON.BOOKS
WHERE XMLEXISTS
('/book[title="Effective XML" and
author="Jason Cu"]'
PASSING BOOK);

<table>
<thead>
<tr>
<th>QUERYNO</th>
<th>ACCESTYPE</th>
<th>ACCESSNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>DX</td>
<td>BOOKTITLEIDX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QUERYNO</th>
<th>ACCESTYPE</th>
<th>ACCESSNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>124</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>124</td>
<td>DX</td>
<td>BOOKTITLEIDX</td>
</tr>
<tr>
<td>124</td>
<td>DX</td>
<td>BOOKAUTHORIDX</td>
</tr>
<tr>
<td>124</td>
<td>DI</td>
<td></td>
</tr>
</tbody>
</table>
SELECT the Document Back (No Index Access)

```sql
SELECT XMLQUERY('/book/author' PASSING BOOK)
FROM JASON.BOOKS
WHERE ID=104;
```

**Books**

<table>
<thead>
<tr>
<th>ID</th>
<th>DATE_ADDED</th>
<th>DOCID</th>
<th>BOOK</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>2008-06-10</td>
<td>1000</td>
<td>***</td>
</tr>
<tr>
<td>104</td>
<td>2008-06-10</td>
<td>1001</td>
<td>***</td>
</tr>
</tbody>
</table>

**Xbooks (XML Aux Table)**

<table>
<thead>
<tr>
<th>DOCID</th>
<th>MIN_NODEID</th>
<th>XMLDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>02</td>
<td>**************</td>
</tr>
<tr>
<td>1001</td>
<td>02</td>
<td>**************</td>
</tr>
<tr>
<td>1001</td>
<td>020202</td>
<td>**************</td>
</tr>
</tbody>
</table>

1. The base table is accessed first and base table predicates are evaluated (possibly using indexes on the base table).
2. The DOCID from the base table row is used as a key into the NODEID index.
3. Records are retrieved in document order through the NODEID index as XMLQUERY is evaluated and the XML nodes are serialized.
**SELECT the Document Back (Index Access)**

```sql
SELECT XMLQUERY('/book/author' PASSING BOOK) FROM JASON.books
WHERE DATE_ADDED = CURRENT DATE AND
XML EXISTS('/book[title='Effective XML']')
```

**Books**

<table>
<thead>
<tr>
<th>ID</th>
<th>DATE_ADDED</th>
<th>DOCID</th>
<th>BOOK</th>
</tr>
</thead>
<tbody>
<tr>
<td>103</td>
<td>2008-06-10</td>
<td>1000</td>
<td>***</td>
</tr>
<tr>
<td>104</td>
<td>2008-06-10</td>
<td>1001</td>
<td>***</td>
</tr>
</tbody>
</table>

**Xbooks (XML Aux Table)**

<table>
<thead>
<tr>
<th>DOCID</th>
<th>MIN_NODEID</th>
<th>XMLDATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>02</td>
<td>******************</td>
</tr>
<tr>
<td>1001</td>
<td>02</td>
<td>**********</td>
</tr>
<tr>
<td>1001</td>
<td>020202</td>
<td>**********</td>
</tr>
</tbody>
</table>

1. The XML Index is used to produce a list of qualifying documents (DOCIDs).
2. The DOCID list is used against the base table to determine the qualifying rows.
3. If multiple indexes are involved, the DOCID lists are intersected and unioned together.
4. The base table is accessed in DOCID list order (using the DOCID index), and base table predicates are evaluated.
Update and Delete

• Update and Delete are full-document operations.

• Update is a DELETE followed by an INSERT (requires a new parse).

  UPDATE JASON.BOOKS SET BOOK=:new_document
   WHERE ID=123;

• DELETE simply deletes the base table row and the related rows from the XML auxiliary table.

  DELETE FROM JASON.BOOKS WHERE ID=123;
Agenda

• Introduction
• Create a Table – the XML Storage Model
• Insert a Row – Storing XML Data
• Create an Index – Index Structure and EXPLAIN, DML
• Running Utilities – LOAD, UNLOAD, others
• XML Schema – Registration, Validation, Decomposition
Utilities and XML Objects

- The XML storage model does not introduce new kinds of objects, so there are no new utilities.
- **Existing utilities are enhanced** to support XML.
- Some utilities have new XML keywords:
  - CHECK DATA – XMLERROR, XMLONLY
  - LISTDEF – “XML” only may be specified
  - LOAD – XML may be used as a field specification with white space parsing options
  - UNLOAD – XML may be used as a field specification
- **Other utilities work as expected** against XML table spaces and implicitly created indexes (ie. REORG, RUNSTATS, COPY, RECOVER, REBUILD INDEX, CHECK, etc.)
UNLOAD

- Use UNLOAD and LOAD on the base table (space) only.
- XML is allowed as a field specification.
- XML is serialized during UNLOAD, and parsed during LOAD.
  - Note: DSN1COPY does not serialize XML data.
- The (UN)LOAD dataset has a maximum 32k width.
- 2 options for UNLOADing XML columns
  - Specify XML: UNLOAD XML directly into the SYSREC dataset
  - UNLOAD using a File Reference Variable (FRV)
UNLOAD

UNLOAD with an XML field specification:

UNLOAD DATA FROM TABLE JASON.BOOKS
(ID POSI ON (*) INTEGER
, DATE_ADDED Position (*) DATE EXTERN AL
, BOOK Position (*) XML
)

SYSREC dataset:

00000103_2008-06-10_</xml version="1.0" encoding="IBM037">book>...
00000104_2008-06-10_</xml version="1.0" encoding="IBM037">book>...
00000105_2008-06-10_</xml version="1.0" encoding="IBM037">book>...
...

XML data is serialized inline in the SYSREC dataset, and is subject to the 32k width restriction.
UNLOAD

UNLOAD with a FRV:
TEMPLATE CLOBFVCB UNIT(SYSDA) DISP(NEW,CATLG,CATLG)
DSN(JASON.UNLOAD1.CLOBFVCB) DSNTYPE(PDS) DIR(15)
VOLUMES(SCR03)

UNLOAD DATA FROM TABLE JASON.BOOKS
   (ID POSITION(*) INTEGER
    ,DATE_ADDED POSITION(*) DATE EXTERNAL
    ,BOOK    POSITION(*) VARCHAR CLOBF   CLOBFVCB
   )

SYSREC dataset:
00000103_2008-06-10_JASON.UNLOAD1.CLOBFVCB(BW1R3NCU)
00000104_2008-06-10_JASON.UNLOAD1.CLOBFVCB(BW1R506E)
00000105_2008-06-10_JASON.UNLOAD1.CLOBFVCB(BW1R506O)
...

The file name is specified in the SYSREC dataset, which indicates which files hold the UNLOADED XML.
LOAD

LOAD statement generated from UNLOAD with an XML field specification:

```
LOAD DATA INDDN SYSREC   LOG NO RESUME YES
  EBCDIC    CCSID(00037,00000,00000)
  SORTKEYS   1
  INTO TABLE "JASON"."BOOKS"
  WHEN(00001:00002) = X'0003'
  ( "DSN_NULL_IND_00001" POSITION(  00003)     CHAR(1)
    , "ID"
    POSITION(  00004:00007) INTEGER
      NULLIF(DSN_NULL_IND_00001)=X'FF'
    , "DSN_NULL_IND_00002" POSITION(  00008)     CHAR(1)
    , "DATE_ADDED"
    POSITION(  00009:00018) DATE EXTERNAL
      NULLIF(DSN_NULL_IND_00002)=X'FF'
    , "DSN_NULL_IND_00003" POSITION(   *)        CHAR(1)
    , "BOOK"
    POSITION(   *) XML PRESERVE WHITESPACE
      NULLIF(DSN_NULL_IND_00003)=X'FF'
  )
```

PRESERVE WHITESPACE is a parsing option that may be specified in the LOAD control statement.
LOAD DATA INDDN SYSREC LOG NO RESUME YES
EBCDIC CCSID(00037,00000,00000)
SORTKEYS 1
INTO TABLE "JASON"."BOOKS"
WHEN(00001:00002) = X'0003'
  ( "DSN_NULL_IND_00001" POSITION( 00003) CHAR(1)
    , "ID"
      POSITION( 00004:00007) INTEGER
        NULLIF(DSN_NULL_IND_00001)=X'FF'
    , "DSN_NULL_IND_00002" POSITION( 00008) CHAR(1)
    , "DATE_ADDED"
      POSITION( 00009:00018) DATE EXTERNAL
        NULLIF(DSN_NULL_IND_00002)=X'FF'
    , "DSN_NULL_IND_00003" POSITION( 00019) CHAR(1)
    , "BOOK"
      POSITION( 00020) VARCHAR CLOBF PRESERVE WHITESPACE
        NULLIF(DSN_NULL_IND_00003)=X'FF'
)
Like Referential Integrity

- The relationship between the base table (space) and the XML auxiliary table (space) is like Referential Integrity.

- CHECK DATA can be used on the base table space to determine consistency between the base table space and the XML table space.

- CHECK INDEX can be used to find problems with the DOCID and NODEID indexes. REBUILD INDEX can be used to repair the problems.

- COPY/RECOVER dependant objects (XML table space, DOCID/NODEID indexes) together. QUIESCE TABLESPACESET can achieve a single point of consistency.

- REORG and RUNSTATS XML table spaces independently from the base table space (or use LISTDEF).
Agenda

- Introduction
- Create a Table – the XML Storage Model
- Insert a Row – Storing XML Data
- Create an Index – Index Structure and EXPLAIN, DML
- Running Utilities – LOAD, UNLOAD, others
- XML Schema – Registration, Validation, Decomposition
XML Schema

- XML Schema adds constraints on XML data.
- Register a schema in XML Schema Repository (XSR)
  - XSR is created during install or migration.
- External names
  - target namespace: e.g., "http://www.ibm.com/software/catalog"
  - schema location: e.g., "http://www.ibm.com/schemas/software/catalog.xsd"
- SQL identifier - used to reference schemas in SQL
  - unique identifier in DB, e.g., SYSXSR.BOOKSCHEMA
Registering an XML Schema

Orderschema

Namespace: Order

Namespace: Lineitem


XSR_COMPLETE ('SYSXSR', 'ORDERSCHEMA', :schemaproperty, 0)
Using XML Schemas

- Schema validation using the DSN_XMLValidate UDF:

  ```sql
  INSERT into Books
  VALUES( 108, CURRENT DATE, XMLPARSE( DOCUMENT SYSFUN.DSN_XMLValidate(:book,'SYSXSR.BOOKSCHEMA')) );
  ```

- Annotated schema-based decomposition – XDBDECOMPXXML stored procedure

  E.g. authorname -> AUTHORS.NAME

  ```xml
  <attribute name="authorname" type="xs:string"
  db2-xdb:rowSet = "AUTHORS"
  db2-xdb:column = "NAME" />
  ```
Well-formedness vs. Validation

- All XML data stored in XML columns must be well-formed.

- Validation CPU time is 2-3 times as expensive as well-formedness parsing only.

- There is no difference in stored XML data whether you validate or not (unless there are default values defined in the XML schema).

- Carefully consider the costs of validation during application design.
Conclusion

• Introduction

• Create a Table – the XML Storage Model

• Insert a Row – Storing XML Data

• Create an Index – Index Structure and EXPLAIN, DML

• Running Utilities – LOAD, UNLOAD, others

• XML Schema – Registration, Validation, Decomposition
Where to get More Information

• DB2 9 Redbook (has a long chapter on XML):

• DB2 9 XML Guide (manual):

• DB2 9 for z/OS XSR Setup and Troubleshooting
Backup slides
### Platform Similarities

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>LUW</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Data Type &amp; DDL</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>(same syntax)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query language</td>
<td>SQL/XML with XPath</td>
<td>XQuery &amp; SQL/XML with XQuery</td>
</tr>
<tr>
<td>Indexing (same</td>
<td>XPath value index</td>
<td>XPath value index</td>
</tr>
<tr>
<td>syntax)</td>
<td>string and numeric (DECFLOAT). No</td>
<td>string, numeric (double), date, timestamp</td>
</tr>
<tr>
<td></td>
<td>hashed.</td>
<td></td>
</tr>
<tr>
<td>Validation</td>
<td>XML Schema, DSN_XMLVALIDATE() UDF,</td>
<td>XML Schema, XMLVALIDATE() BIF, type</td>
</tr>
<tr>
<td></td>
<td>type annotation not kept</td>
<td>annotation kept</td>
</tr>
<tr>
<td>I/U/D/M (same</td>
<td>Whole doc, no versioning</td>
<td>Whole doc w/ versioning</td>
</tr>
<tr>
<td>syntax)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Host languages and</td>
<td>JDBC, ODBC, C/C++, Java, COBOL,</td>
<td>JDBC, ODBC, C/C++, Java, Cobol, .Net</td>
</tr>
<tr>
<td>APIs</td>
<td>PL/I, Assembly, .Net</td>
<td></td>
</tr>
<tr>
<td>Decomposition</td>
<td>Annotated schema-based</td>
<td>Annotated schema-based</td>
</tr>
</tbody>
</table>
## Platform Differences

<table>
<thead>
<tr>
<th></th>
<th>z/OS</th>
<th>LUW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query language</td>
<td>XPath</td>
<td>XQuery</td>
</tr>
<tr>
<td>FETCH CONTINUE</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>XMLTable/XMLCast</td>
<td>No (working on, post GA)</td>
<td>Yes</td>
</tr>
<tr>
<td>Encoding</td>
<td>Table of any Encoding, UTF-8 internal for XML</td>
<td>Unicode or Non unicode database</td>
</tr>
<tr>
<td>Stored proc</td>
<td>Use xLOB or string for XML in parms</td>
<td>+XML as CLOB in parms</td>
</tr>
<tr>
<td>Partitioned tables</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Compression</td>
<td>Yes</td>
<td>Yes, with in lining</td>
</tr>
<tr>
<td>Data sharing</td>
<td>Yes</td>
<td>No MPP</td>
</tr>
<tr>
<td>Utilities</td>
<td>Almost all</td>
<td>Fewer utilities</td>
</tr>
</tbody>
</table>
XPath support in DB2 9 z/OS

- Used in XMLEXISTS, XMLQUERY, XMLTABLE, indexing

- XPath 1.0 + -
  - XPath 1.0 constructs in XPath 2.0 semantics (no value comp)
  - + more numeric data types and some generic types.
  - + namespace declaration from XQuery prolog
  - - Axes: only forward axes (child, descendant, descendant-or-self, self, attribute, //, /, @) & parent axis (..) are supported.

- Types supported in XPath expressions:
  - xs:boolean, xs:integer, xs:decimal, xs:double, xs:string

IBM DB2 z/OS pureXML