



XML Support in DB2 for z/OS

Guogen Zhang (张国根)



DB2 for z/OS Technical Conference
Sept 05-07, 2007 - Shanghai, China
Sept 10-12, 2007 - Beijing, China

Agenda

- Why XML and XML Databases
- Comparing pureXML with existing approaches
- pureXML features
 - XML data type, DDL, DML
 - XML Query Languages and API
 - Indexing and access methods
 - Schema support
 - Utilities
- Performance
- Tools
- Summary

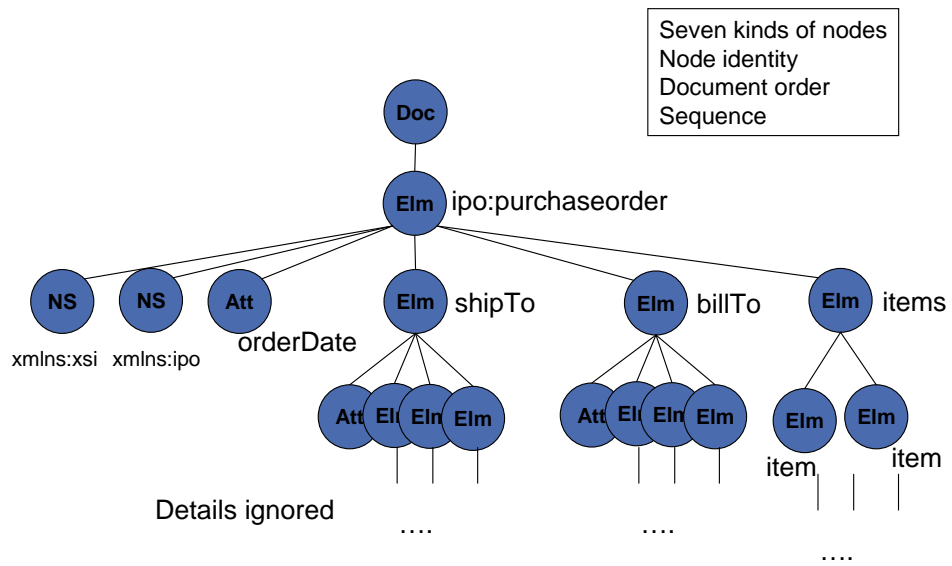


An XML Purchase Order

```
<?xml version="1.0" encoding="UTF-8"?>
<ipo:purchaseOrder
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ipo="http://www.example.com/IPO" orderDate="1999-12-01">
  <shipTo exportCode="1" xsi:type="ipo:UKAddress">
    <name>Helen Zoe</name>
    <street>47 Eden Street</street>
    <city>Cambridge</city>
    <postcode>CB1 1JR</postcode>
  </shipTo>
  <billTo xsi:type="ipo:USAddress">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <items>
    <item partNum="833-AA">
      <productName>Lapis necklace</productName>
      <quantity>1</quantity>
      <USPrice>99.95</USPrice>
      <comment>Want this for the
      holidays!</comment>
      <shipDate>1999-12-05</shipDate>
    </item>
    <item partNum="926-AA">
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-12-21</shipDate>
    </item>
  </items>
</ipo:purchaseOrder>
```



XML Data Model (XDM)



XML Characteristics

- XML supports flexible structured data in text
 - Nesting
 - Repeating
 - Self-describing
- XML is a universal language to represent e-Business data and transactions.
- Platform-independent, and Unicode compliant
- Easy to understand and easy to process (with the right tools)



Why Hybrid XML Databases?

- Businesses need to manage XML data w/ ACID properties, auditing and regulatory compliance, together with relational data.
- XML can be used as a powerful data model, with powerful declarative query language.
- Managing large volumes of XML data is a DB problem
 - ...all the same reasons as for relational data!
- Integration
 - Integrate new XML data with existing relational data
 - Publish (relational) data as XML
 - Database support for web applications, SOA, web services (SOAP)



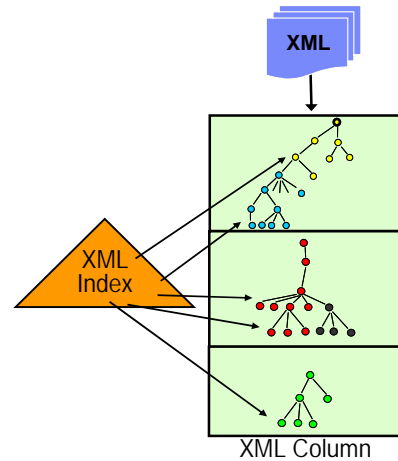
pureXML in DB2 9

- SQL XML data type and native storage
- Designed specifically for XML from the ground up
 - Supports XML hierarchical structure storage
 - Native operations and languages: XPath, SQL/XML, (XQuery)
- Not transformation into relational
- Not using objects or nested tables
- Not using LOBs
- Integrated with relational engine, with all the utilities and tools support

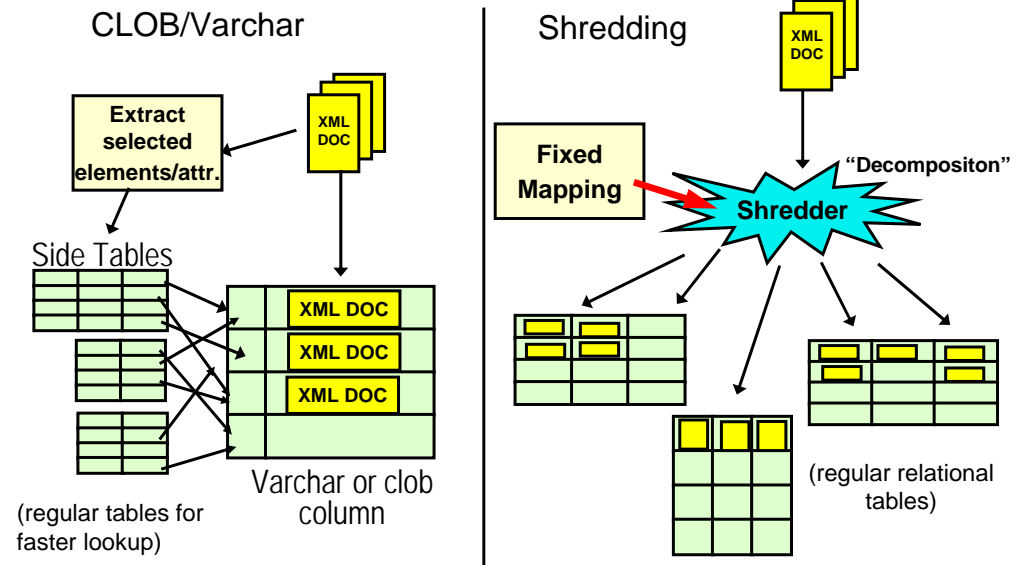


What You Can Do with pureXML

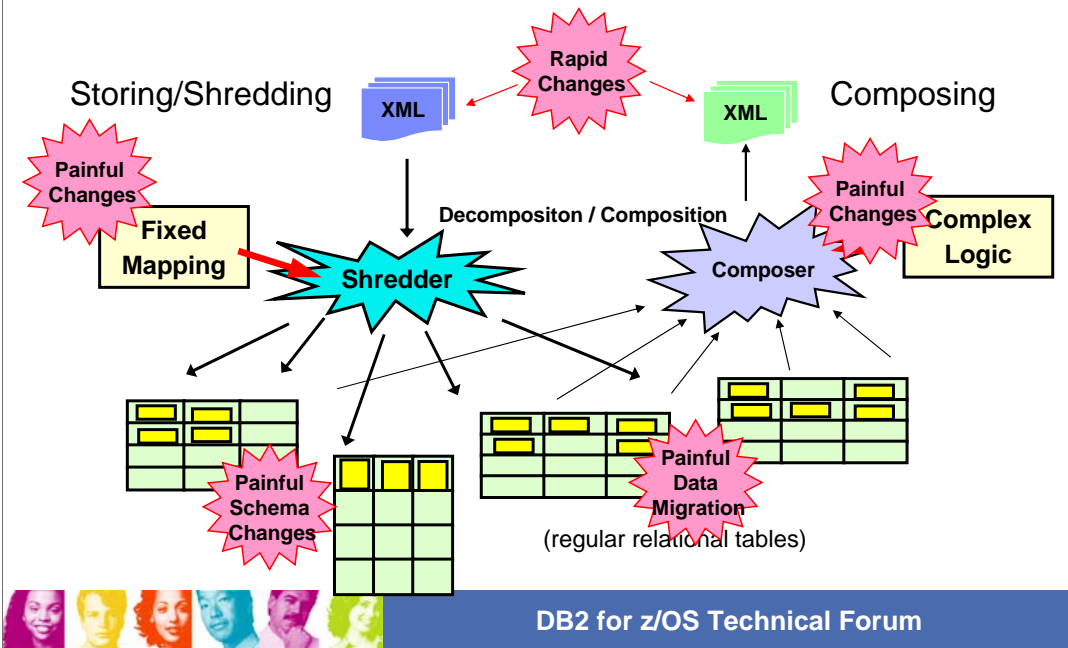
- Create tables with XML columns
- Insert XML data, optionally validated against schemas
- Create indexes on XML data
- Efficiently search XML data
- Extract XML data
- Decompose XML data into relational data
- Construct XML documents from relational and XML data
- All the utilities and tools support for XML



XML-Enabled Databases: Two Main Options

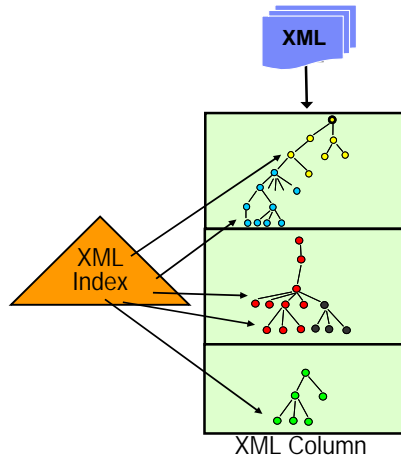


XML Data Processing before pureXML



DB2 pureXML Advantages

DB2's hierarchical storage:
XML type as XML



- Directly store XML, no decomp/comp, normalize/denormalize
- Eliminates database schema evolution bottleneck
- Declarative language, reduce complexity, dramatically improve application development productivity
- Native processing, high performance
- Unparalleled reliability, availability, scalability

Up to 10 times



DB2 for z/OS Technical Forum

Usage Scenarios

- Directly processing XML (data in motion)
 - UNIFI, ACORD, FIXML, FpML, MIMSO, XBRL,
 - DJXDM, HR-XML, HL7, ARTS, HIPAA, NewsML, XForms
 - Insurance policy, contract, purchase order, emails etc
- Versatile schemas
- Sparse attribute values (null v.s. absence)
- Object persistence (single column v.s. many tables)
- Migration from legacy data model (network, hierarchical, relational)
- Web page: XHTML
- Web Services (SOAP), support SOA
- ...



Business Value of pureXML

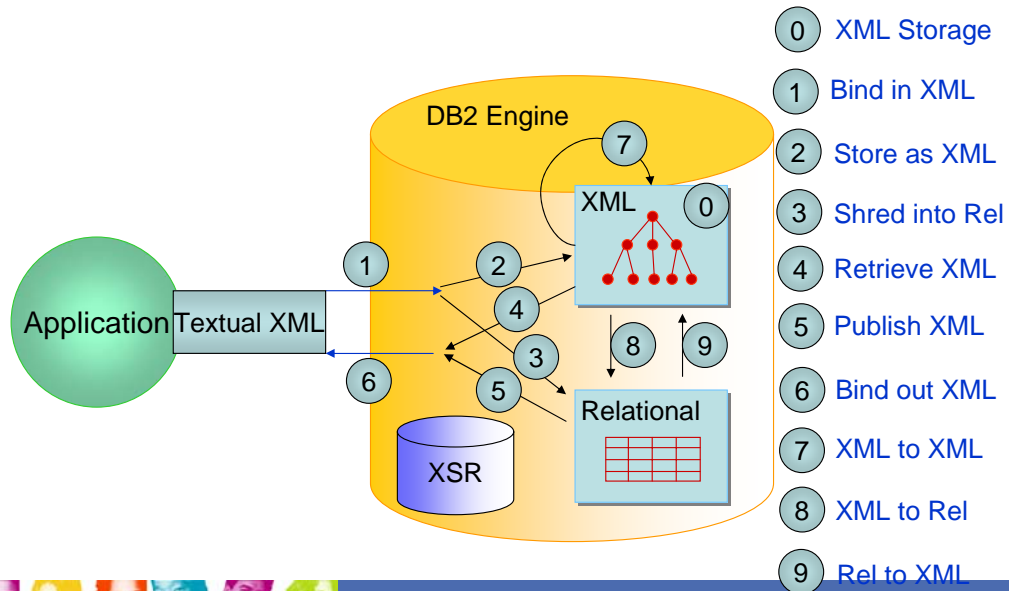
- **Lower Development Costs**
 - Reduced system and development complexity
 - Improved developer productivity

- **Greater Business Agility**
 - Easily accommodate changes to data and schemas
 - Update applications rapidly and reduce maintenance costs

- **Improved Business Insight**
 - Access to “hidden gems” (data) in otherwise unexploited documents
 - Unprecedented application performance



Summary of SQL/XML Features



DB2 for z/OS Technical Forum

Something we are working on.

XML Type and DDL

```
CREATE TABLE PurchaseOrders (  
  ponumber      varchar(10) not null,  
  podate        date not null,  
  status        char(1),  
  XMLpo xml);  
[or: IN MYDB.MYTS; ]  
[or: IN DATABASE MYDB; ]  
[or: IN MYTS; ]
```

- Hidden DocID column
- One DocID index
- Internal XML table (16K BP) for each XML column
- NodeID index
- No associated schema

```
CREATE TABLE PO LIKE PurchaseOrders;
```

```
CREATE VIEW ValidPurchaseOrders as  
  SELECT ponumber, podate, XMLpo  
  FROM PurchaseOrders  
  WHERE status = 'A';
```

```
ALTER TABLE PurchaseOrders  
  ADD revisedXMLpo xml;
```

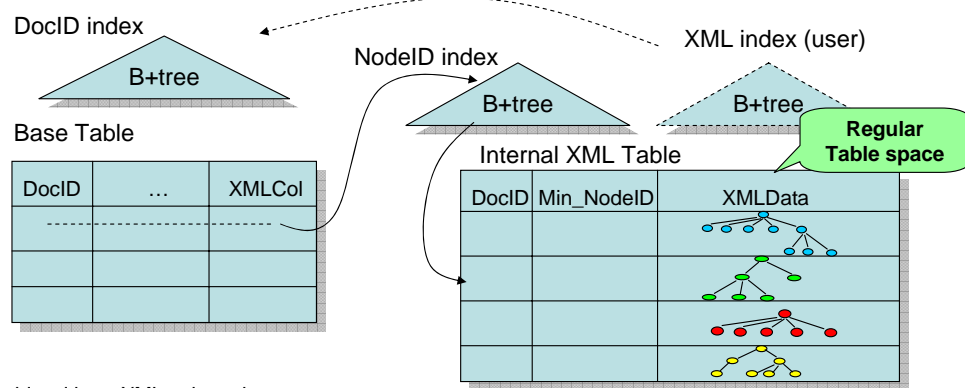


Go through the statement first, then explain the 5th point, leave the rest of the following with the next page.

There are a few things happening when a table includes an XML column, or it's altered to include an XML column.

1. A hidden identity DocID column is added into the table (DB2_GENERATED_DOCID_FOR_XML), it's a BIGINT type.
2. A DocID index is created on the base table, which provides association from DocID from XML index to base rows.
3. An internal XML table (16K page size) is created for each XML column. It is a partition by growth table space (for single or segmented or partition by growth base table) or partitioned table space (for partitioned base table).
4. A NodeID index is created on the internal XML table. It can have multiple entries for each record, but since each record typically contains a big sub-tree, it is a sparse index at node level.
5. No XML schema is associated with an XML column. The same column can contain any well-formed documents and schema validated XML documents with different schemas.
6. The implicitly created XML table inherits some attributes from the base table.

XML Storage on Mature Infrastructure



A table with an XML column has a DocID column, used to link from the base table to the XML table. A DocID index is used for getting to base table rows from XPath value indexes.

Each XMLData column is a VARBINARY, containing a subtree or a sequence of subtrees, with context path. Rows in XML table are freely movable, linked with a NodeID index.



Manipulating XML Data

```
EXEC SQL BEGIN DECLARE SECTION;  
    SQL TYPE IS XML AS CLOB(1M) xmlPo;  
EXEC SQL END DECLARE SECTION;
```

Host var of XML type

```
INSERT INTO PurchaseOrders VALUES ('200300001',  
    CURRENT DATE, 'A', :xmlPo);
```

String literal is OK

```
INSERT INTO PurchaseOrders VALUES ('200300001',  
    CURRENT DATE, 'A', CAST(? AS XML));
```

```
INSERT INTO PurchaseOrders VALUES('200300003', CURRENT DATE, 'A',  
    XMLPARSE(DOCUMENT :vchar PRESERVE WHITESPACE) );
```

```
INSERT into PurchaseOrders VALUES( '200300004', CURRENT DATE, 'A',  
    DSN_XMLValidate(:lobPo, 'SYSXSR.myPOSchema'));
```

```
UPDATE PurchaseOrders SET XMLpo = :XMLpo_revised  
    WHERE ponumber = '12345';
```

Whole document
replacement

```
DELETE FROM PurchaseOrders WHERE ponumber = '12345';
```



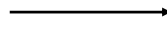
LOAD has a limit of 32K size as LOB, but file references support allows any size documents to be loaded (up to 2GB).

No cross load support for XML yet.

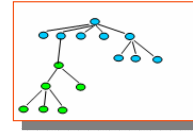
XMLParse and XMLSerialize

```
<?xml version="1.0"?>  
<purchaseOrder orderDate="1999-10-  
  <shipTo country="US">  
    <name>Alice Smith</name>
```

XMLParse



XMLSerialize



XMLParse

- Allows strip whitespace or preserve whitespace
- Implicit XMLParse applies for bind-in XML hostvar or inserting hostvar or string literal.

XMLSerialize

- With or without XML declaration
- Implicit XMLSerialize applies for bind-out XML type



On the client, XML data is still a string. DB2 uses XMLParse (and calls XMLSS XML parser) to convert it into internal XDM hierarchical format during bind-in/insert. On the other direction, DB2 uses XMLSerialize to serialize the hierarchical data into a string. XMLParse and XMLSerialize can be implicitly performed.

Retrieving XML Data

- **Simple select:**

```
SELECT XMLpo INTO :xmlPo  
FROM PurchaseOrders  
WHERE ponumber = '200300001';
```

- **Select with condition:**

```
SELECT XMLPO  
FROM PurchaseOrders  
WHERE XMLEXISTS('//items/item[desc = "Shoe"]' PASSING  
XMLpo);
```

- **Extract from a document:**

```
SELECT XMLQUERY('//items/item/quantity' PASSING XMLpo)  
FROM PurchaseOrders WHERE ...;
```



Can use variable in XPath expressions, the value is passed in from the SQL functions.

Join Queries

- PO's containing a product with name from PRODUCT.NAME
SELECT XMLPO
FROM PurchaseOrders, **Product**
WHERE XMLEXISTS('/items/item[desc = \$n]' PASSING XMLpo,
Product.name as "n");
- PO's containing a product with price > :price
SELECT XMLPO
FROM PurchaseOrders, **Catalog**
WHERE XMLEXISTS('/items/item[desc=\$x]' PASSING XMLpo,
XMLQUERY('/category/product[price > \$y]' PASSING
XCategory, :price as "y") as "x") AND
XMLEXISTS('/category/product[price > \$y]' PASSING
XCategory, :price as "y")



XMLTable – Processing XML with SQL Power

```
SELECT TX.*
FROM PurchaseOrders PO,
  XMLTable ('/purchaseorder/items/item'
    PASSING PO.XMLpo
    COLUMNS
      "Part #"          CHAR(6)          PATH '@partnum',
      "Product Name"   CHAR(20)         PATH 'productName',
      "Quantity"       INTEGER          PATH 'quantity',
      "US Price"       DECIMAL(9,2)     PATH 'USPrice',
      "Ship Date"      DATE              PATH 'shipDate',
      "Comment"        CHAR(80)         PATH 'comment'
    WITH ORDINALITY "Seqno") AS TX
WHERE PO.ponumber = '200300001';
```

XMLTable function will be delivered after V9 GA



DB2 for z/OS Technical Forum

SQL/XML Constructors

- Construct XML from relational data
 - XMLElement, XMLAttributes, XMLNamespaces, XMLForest, XMLConcat, XMLAGG (V8)
 - Support Binary data types and null handling options(V9)
 - XMLText, XMLPI, XMLComment, XMLDocument (V9)
- Construct new document by extracting parts from an existing document (XMLQuery) and other data.



Constructor Example

```
SELECT XMLDOCUMENT(  
  XMLELEMENT(NAME "hr:Department",  
    XMLNAMESPACES('http://example.com/hr' as "hr"),  
    XMLATTRIBUTES (e.dept AS "name" ),  
    XMLCOMMENT('names in alphabetical order'),  
    XMLAGG(XMLELEMENT(NAME "hr:emp", e.lname  
      ORDER BY e.lname )  
    ) AS "dept_list"  
FROM employees e  
GROUP BY dept;
```

Can construct XHTML
for web pages

```
<?xml version="1.0" encoding="UTF-8">  
<hr:Department xmlns:hr="http://example.com/hr"  
  name="Shipping">  
  <!-- names in alphabetical order -->  
  <hr:emp>Lee</hr:emp>  
  <hr:emp>Martin</hr:emp>  
  <hr:emp>Oppenheimer</hr:emp>  
</hr:Department>
```



This example is to illustrate generating department with nested employee list (in lastname order) from employees table, grouped by DEPT.

Use SQL/XML to Achieve XQuery Functionality

- Use XMLEXISTS with XPath to find documents.
- Use XMLQuery and XMLTable with XPath to extract parts of documents.
- XPath cannot be used to construct new document.
- SQL/XML has complete constructor functions to make up missing functionality in XPath.
- Use SQL/XML constructor functions and XMLQuery (and XMLTable) to construct new documents from existing documents.



The major differences between XQuery and XPath are:

1. No document constructors in XPath;
2. No complex join, grouping, etc.

But we can use SQL to make up some significant part of missing features.

Example: Construct Invoice from Purchase Order

```
SELECT XMLDocument(  
  XMLElement(NAME "invoice",  
    XMLAttributes( '12345' as "invoiceNo"),  
    XMLQuery ('/purchaseOrder/billTo' PASSING xmlpo),  
    XMLElement(NAME "purchaseOrderNo",  
      PO.ponumber)  
    XMLElement(NAME "amount",  
      XMLQuery  
        ('fn:sum(/purchaseOrder/items/item/xs:decimal(USPrice))'  
        PASSING xmlpo) )  
  ) )  
FROM PurchaseOrders PO  
WHERE PO.ponumber = '200300001';
```

```
<?xml version="1.0" encoding="utf-8" ?>  
<invoice invoiceNo = "12345">  
  <billTo country="US">  
    <name>Robert Smith</name>  
    . . .  
  </billTo>  
  <purchaseOrderNo>200300001</purchaseOrderNo>  
  <amount>188.93</amount>  
</invoice>
```



DB2 for z/OS Technical Forum

For illustration purpose.

'12345' invoiceNo can be result of XMLQuery – extract from existing purchase order.

You may also want to include all the line items from the purchase order.

XPath Support

- Used in XMLEXISTS, XMLQUERY, XMLTABLE and XML indexing
- XPath 1.0 + - (subset of XPath 2.0/XQuery 1.0)
 - XPath 1.0 constructs in XPath 2.0 semantics
 - + more data types: xs:boolean, xs:integer, xs:decimal, xs:double, xs:string
 - + namespace declaration from XQuery prolog
 - - Axes: only forward axes (child, attribute, descendant, descendant-or-self, self, ., //, @) & parent axis (..) are supported.
- All stored XML data are untyped in V9.
 - Explicit type casting may be needed in some cases



The 5 axes are: child, descendant, self, descendant-or-self, and attribute.
No positional predicate support.

Application Interfaces

- XML type is supported in
 - Java (JDBC, SQLJ), ODBC,
 - C/C++, COBOL, PL/I, Assembly
 - .NET
- Applications use:
 - XML as CLOB(n), XML as CLOB_FILE
 - XML as DBCLOB(n), XML as DBCLOB_FILE
 - XML as BLOB(n), XML as BLOB_FILE
 - All character or binary string types are supported
- XMLParse and XMLSerialize apply (implicitly or explicitly)



DRDA supports internally encoded or externally encoded XML type (in serialized string format).

Host Interface Examples

```
CREATE TABLE T1(ID INT, XMLCOL XML);

INSERT INTO T1 VALUES(100, :XMLHV);
INSERT INTO T1 VALUES(150, :VBHV);
INSERT INTO T1 VALUES(200, XMLPARSE(DOCUMENT :VBHV) );
INSERT INTO T1 VALUES(210, XMLPARSE(DOCUMENT :VBHV
PRESERVE WHITESPACE) );

SELECT XMLCOL INTO :XMLHV
FROM T1
WHERE T1.ID = 100;

SELECT XMLCOL INTO :VBHV
FROM T1
WHERE T1.ID = 100;

SELECT XMLSERIALIZE(XMLCOL AS BLOB(100K))
      INTO :BLOBHV
FROM T1
WHERE T1.ID = 200;
```



Java JDBC Example

Use standard interface:

```
PreparedStatement pstmt = connection.prepareStatement("INSERT INTO
PurchaseOrders VALUES(?, ?); // second column: XML type
...
InputStream fin = new FileInputStream(file);
pstmt.setBinaryStream( 2, fin, flen );
pstmt.execute();

Statement s = connection.createStatement();
ResultSet rs = s.executeQuery ("select ponumber, xmlpo from purchaseOrders");
while (rs.next()) {
    int po_no = rs.getInt ("ponumber");
    String spo= rs.getString(2);
    System.out.println (spo); // uninterpreted flat xml text
}
```

Or use com.ibm.db2.jcc.DB2Xml interface



DB2 for z/OS Technical Forum

Just to illustrate bind-in XML and get XML string value.
com.ibm.db2.jcc.DB2Xml is the Java class for XML interface (before
JDBC standard).

FETCH CONTINUE for XML and LOB

- No size associated with XML values
- Hard to allocate large memory
- Shortcomings with LOB Locator
- New FETCH CONTINUE statements: (one of two ways)
 - DECLARE CURSOR1 CURSOR FOR SELECT C2 FROM T1;
 - OPEN CURSOR1;
 - **FETCH WITH CONTINUE** CURSOR1 into :clobhv;
 - if (sqlcode >= 0) & sqlcode <> 100
 - Loop if truncation occurs until lob/xml complete (total length)
 - **FETCH CURRENT CONTINUE** CURSOR1 into :clobhv;
 - Consume :clobhv content
 - end loop
- Another way is to use FETCH ... INTO DESCRIPTOR :SQLDA



XML Indexes

- XPath value index: index values of elements or attributes inside a document.
- Index entries include: (key value, DocID, NodeID, RIDx)
- Support string (VARCHAR) or numeric (DECFLOAT) key type

**CREATE INDEX ON PurchaseOrders(XMLPO)
Generate Keys Using XMLPATTERN
'/purchaseOrder/items/item/desc'
as SQL VARCHAR(100);**

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    ...
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    ...
  </billTo>
  <comment>Hurry, my lawn is going wild!</comment>
  <items>
    <item partNum="872-AA">
      <desc>Lawnmower</desc>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <comment>Confirm this is electric</comment>
    </item>
    <item partNum="926-AA">
      <desc>Baby Monitor</desc>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>2003-05-21</shipDate>
    </item>
  </items>
</purchaseOrder>
```

This index can be used for predicate:
XMLEXISTS('/purchaseOrder/items/item[desc = "Baby Monitor"]' passing XMLPO)



XML related indexes are DocID index on the base table, NodeID index on the XML table, and XML index (a.k.a. XPath value index) on a base table XML column (externally) but actually on XML table (internally). It is defined on a base table XML column.

Something Special for XML Index

- The number of keys for each document (each base row) depends on the document and XMLPattern.
- For a numeric index, if a string from a document cannot be converted into a number, it is ignored.
→ <a>X5, XMLPattern '/a/b' as SQL Decfloat. Only one entry '5' in the index.
- For a string (VARCHAR(n)) index, if a key value is longer than the limit, INSERT or CREATE INDEX will fail.



1. m:1 for a row
2. Type mismatch – ignored.
3. String index enforced – otherwise a query cannot use the index – or return wrong result.
4. Restriction: spanning records not supported due to utilities infrastructure (generate key values from a single record).

We had restriction with create index on atomic values only – the above restriction is better than that early restriction.

New Access Methods

Access Methods	Description
DocScan "R" (QuickXScan)	Base algorithm: given a document, scan and evaluate XPath
DocID list access "DX" unique DocID list from an XML index, then access the base table and XML table.	XMExists('/Catalog/Categories/Product[RegPrice > 100]' passing catalog) with index on '/Catalog/Categories/Product/RegPrice' as SQL DECFLOAT
DocID ANDing/ORing "DX/DI/DU" union or intersect (unique) DocID lists from XML indexes, then access the base table and XML table.	XMExists('/Catalog/Categories/Product[RegPrice > 100 and Discount > 0.1]' ...) With indexes on: '//RegPrice' as SQL DECFLOAT and '//Discount' as SQL DECFLOAT



Another Example for Indexes and Query

```
CREATE TABLE ACORD.REQUEST (  
  ID      BIGINT NOT NULL PRIMARY KEY,  
  REQUESTXML XML,  
  RESPONSEXML XML  
) IN DATABASE DBACORD
```

```
CREATE INDEX ACORD.ACORDINDEX1 ON  
  ACORD.REQUEST(REQUESTXML)  
GENERATE KEYS USING XMLPATTERN  
'declare default element namespace "http://ACORD.org/Standards/Life/2";  
/TXLife/TXLifeRequest/TransRefGUID' as SQL VARCHAR(24)
```

```
CREATE INDEX ACORD.ACORDINDEX2 ON  
  ACORD.REQUEST(REQUESTXML)  
GENERATE KEYS USING XMLPATTERN  
'declare default element namespace "http://ACORD.org/Standards/Life/2";  
/TXLife/TXLifeRequest/OLifE/Holding/Policy/@id' AS SQL VARCHAR(9)
```



Another Example for Indexes and Query (cont'ed)

```

SELECT
XMLQuery('declare default element namespace
"http://ACORD.org/Standards/Life/2";
/TXLife/TXLifeRequest/OLifE/
Holding/Policy/Life/Coverage/LifeParticipant'
PASSING R.REQUESTXML),
XMLQuery('declare default element namespace
"http://ACORD.org/Standards/Life/2";
/TXLife/TXLifeRequest/OLifE/Party
[@id =
/TXLife/TXLifeRequest/OLifE/
Holding/Policy/Life/Coverage/
LifeParticipant/@PartyID ]'
PASSING R.REQUESTXML)
FROM ACORD.REQUEST R
WHERE XMLeXists('declare default element namespace
"http://ACORD.org/Standards/Life/2";
/TXLife/TXLifeRequest[TransRefGUID="2004-1217-141016-000012"]/
OLifE[Holding/Policy/@id="POLICY12"]'
PASSING R.REQUESTXML)
    
```

Find participant information about a policy.

	PLANNO	ACCESSTYPE	MATCHCOLS	ACCESSCREATOR	ACCESSNAME	MIXOPSEQ
1_	1	M	0			0
2_	1	DX	1	ACORD	ACORDINDEX2	1
3_	1	DX	1	ACORD	ACORDINDEX1	2
4_	1	DI	0			3

XML Schema Support

- XML Schema adds constraints on XML data.
- Register a schema in XML Schema Repository (XSR)
- 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.ORDERSCHEMA
- Where are schemas used?
 - SYSFUN.DSN_XMLValidate in SQL (UDF for XMLValidate)
 - Decomposition



DB2 provides an XML schema repository – DB2 cannot go out and find the Schema based on target namespaces or schema location. The W3C standard does not require XML schema is available on the Web. Its target namespace only provides a unique identifier.

DB2 requires a SQL identifier for identification.

Registering an XML Schema (Stored Procedures)

- XSR_REGISTER (rschema, name, schemalocation, xsd, docproperty)
- XSR_ADDSCHEMADOC (rschema, name, schemalocation, xsd, docproperty)
- XSR_COMPLETE (rschema, name, schemaproperties, isUsedForDecomp)
- XSR_REMOVE(rschema, name)

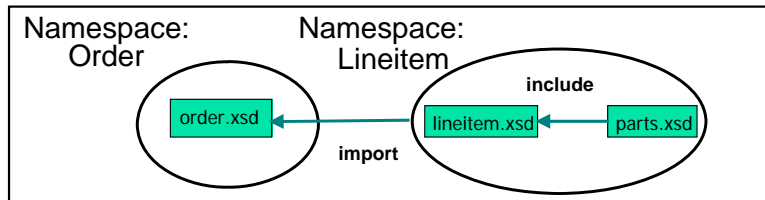
- Parameters:
 - rschema – null or 'SYSXSR';
 - identifier – SQL name (VARCHAR(128));
 - schemalocation – VARCHAR(1000);
 - xsd – XML schema document (BLOB(30M));
 - docproperty – BLOB(5M), may be used by tools;
 - schemaproperties – same as docproperties
 - isUsedForDecomp – INTEGER, 1 yes, 0 no.

- Java Driver (JCC) provides a set of APIs for schema registration



Example: Registering an XML Schema

Orderschema



- XSR_REGISTER('SYSXSR', 'ORDERSHEMA',
'http://www.n1.com/**order.xsd**', :xsd, :docproperty)
- XSR_ADDSCHEMADOC('SYSXSR', 'ORDERSHEMA',
'http://www.n1.com/**lineitem.xsd**', :xsd, :docproperty)
- XSR_ADDSCHEMADOC('SYSXSR', 'ORDERSHEMA',
'http://www.n1.com/**parts.xsd**', :xsd, :docproperty)
- XSR_COMPLETE ('SYSXSR', 'ORDERSHEMA', :schemaproperty, 0)



Example: Use CLP to Register XML Schema

- Register schema

```
REGISTER XMLSCHEMA
http://www.test.com/order.xsd
FROM file://C:/xmlschema/order.xsd
AS ORDERSHEMA
ADD http://www.test.com/lineitem.xsd
FROM file://C:/xmlschema/lineitem.xsd
ADD http://www.test.com/parts.xsd
FROM file://C:/xmlschema/parts.xsd
COMPLETE [ENABLE DECOMPOSITION];
```

- Remove schema

```
REMOVE XMLSCHEMA ORDERSHEMA;
```

SchemaLocation

SQL Identifier

SchemaLocation

Recommended
for DBAs



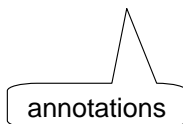
Using XML Schema

- Schema validation (type annotation not kept)

```
INSERT into PurchaseOrders  
VALUES( '200300001', CURRENT DATE, 'A',  
SYSFUN.DSN_XMLValidate(:lobPO,'SYSXSR.ORDERSCHEMA'));
```

- Annotated schema-based decomposition – store using tables. (XDBDECOMPXML stored proc)
E.g. orderID -> PORDER.ORDERID

```
<attribute name="orderID" type="xs:string"  
db2-xdb:rowSet = "PORDER"  
db2-xdb:column = "ORDERID" />
```



PORDER	
ORDERID	ORDE
19991201-ZFG	1999-



Because DSN_XMLValidate is a UDF, not as efficient as XMLParse.

Decomposition is really not a native XML feature.

Utilities

- Enhanced to handle new XML type, XML tablespaces, and XML indexes
- CHECK DATA
- CHECK INDEX
- COPY INDEX
- COPY TABLESPACE
- COPYTOCOPY
- LISTDEF
- LOAD
- MERGECOPY
- QUIESCE TABLESPACESET
- REAL TIME STATISTICS
- REBUILD INDEX
- RECOVER INDEX
- RECOVER TABLESPACE
- REORG INDEX
- REORG TABLESPACE
- REPORT TABLESPACESET
- UNLOAD
- Basic RUNSTATS



And relationship between base table and internal XML tables.

Some of the utilities do not do lot of work, but to tolerate and recognize XML data.

LOAD file references for large XML documents.

Believe or not ONLINE REORG is supported for XML tablespaces.

Performance and Scalability

- Very compact storage format.
- XML storage leverages mature optimized storage infrastructure (regular table spaces).
- Next generation parsers: z/OS XML System Services and XLXP.
 - Validation is much more expensive than well-formedness checking only
- Highly efficient XPath streaming algorithm
- Support partitioned table spaces and data sharing.
- Initial sweet spot: a large number of small documents.
- Storage, Insert, and Fetch performance follows. Stay tuned for more performance numbers



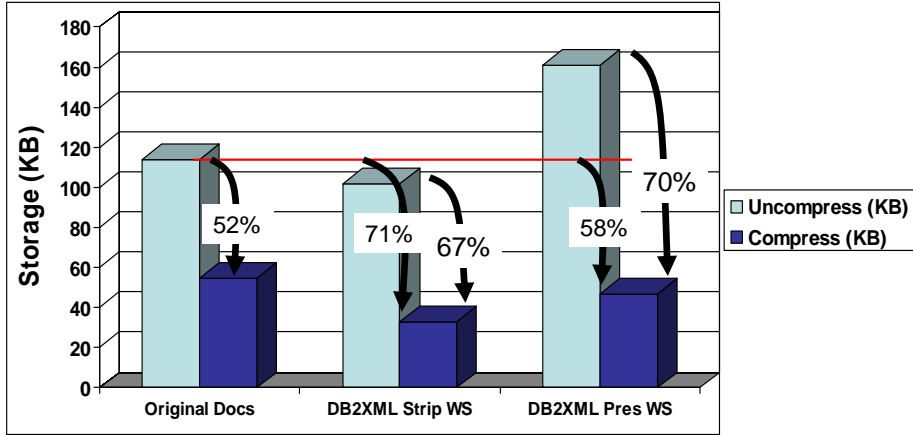
DB2 for z/OS Technical Forum

XMLSS is z/OS XML System Services, a non-validating parser.

XLXP is a validating parser for XML Schema.

XML parsing (XMLSS) will be zAAP-able.

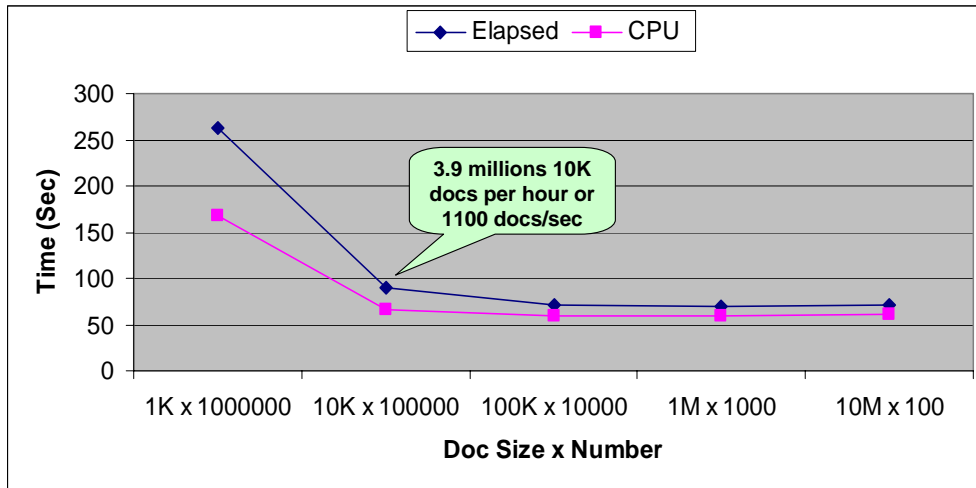
Storage for UNIFI Messages



96 sample documents
Strip WS: Strip Whitespaces
Pres WS: Preserve Whitespaces



Insert Performance (Batch)



Measurement in March 2007, z9 DS8300, Single thread, Docs in EBCDIC



DB2 for z/OS Technical Forum

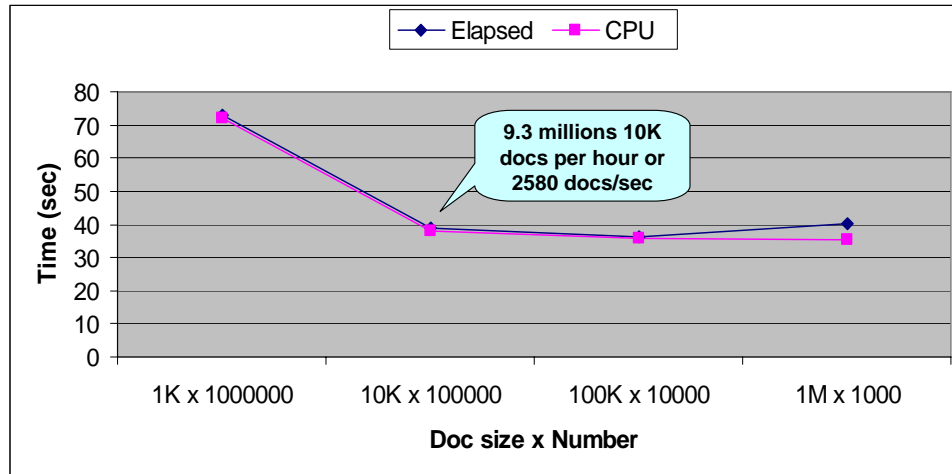
This is scalability testing with different size of Document. When 1K document, you will see more overhead but in general we have accomplished linear scalability.

Note:

Insert chart 1K doc 100,000 insert, 10K doc 10,000 insert, etc

Serialize chart 1K doc 600,000 select, 10K doc 60,000 select, etc.

Fetch Performance



Measurement in March 2007, z9 DS8300, Single thread, Docs in EBCDIC



DB2 for z/OS Technical Forum

This is scalability testing with different size of Document. When 1K document, you will see more overhead but in general we have accomplished linear scalability.

Note:

Insert chart 1K doc 100,000 insert, 10K doc 10,000 insert, etc

Serialize chart 1K doc 600,000 select, 10K doc 60,000 select, etc.

Tools

- Tool choices:
 - Rational Data Architect
 - Rational Application Developer
 - Developer Workbench (DWB)
 - .NET
 - QMF
 - SPUFI
 - CLP
- Schema registration, validation
- Annotation for decomposition
- Mapping relational to XML schema for XML generation



XML Features in V9 - Summary

- Complete SQL/XML constructor functions
- First-class XML type, native storage of XQuery Data Model
- XMLPARSE and XMLSERIALIZE
- XML indexes
- Other SQL/XML functions with XPath
 - XMLEXISTS, XMLQUERY, XMLTABLE
- XML Schema repository, Validation UDF, and decomposition
- DRDA (distributed support) and application interfaces
- Utilities and tools

