IBM Information ON Demand 2010

Create Insight. Transform. Go Beyond.

TSB-3010A

New pureXML® Features in DB2® for z/OS: Breaking Relational Limit

Guogen (Gene) Zhang, Senior Technical Staff Member, DB2 for z/OS development

Housekeeping

- We value your feedback - don't forget to complete your evaluation for each session you attend and hand it to the room monitors at the end of each session
- Overall Conference Evaluation will be provided at the General Session on Friday
- Visit the Expo Solutions Centre
- Please remember this is a 'non-smoking' venue!
- Please switch off your mobile phones
- Please remember to wear your badge at all times

IBM Disclaimer

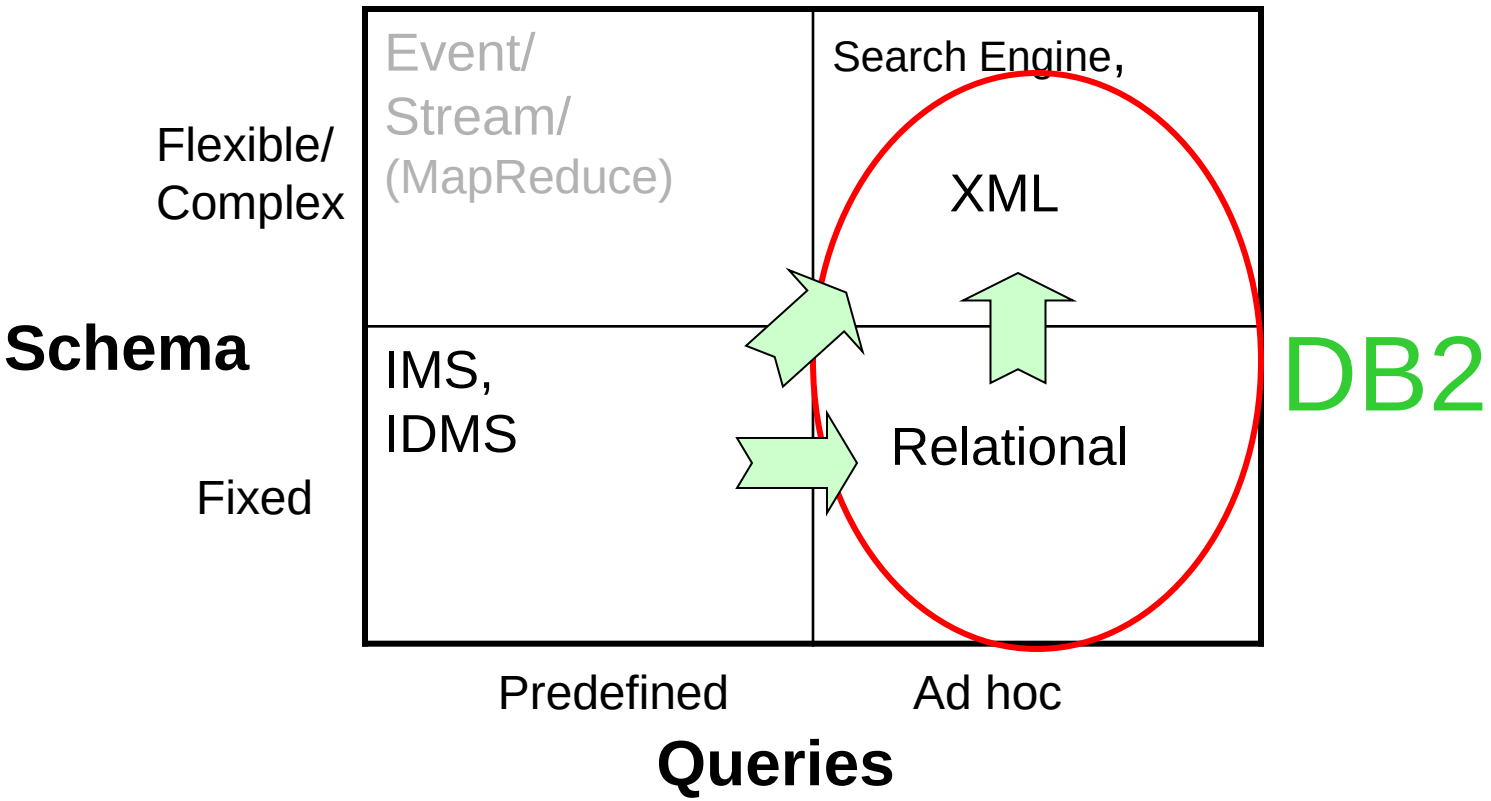
The Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Examples contained herein are for illustration purpose only.

Agenda

- Review of pureXML in DB2 9 for z/OS
- Recent enhancements in DB2 9 for z/OS
- Preview of new XML features in DB2 10
- X*-programming paradigm and architecture pattern

DB2 XML Positioning



Highlights of DB2 for z/OS pureXML

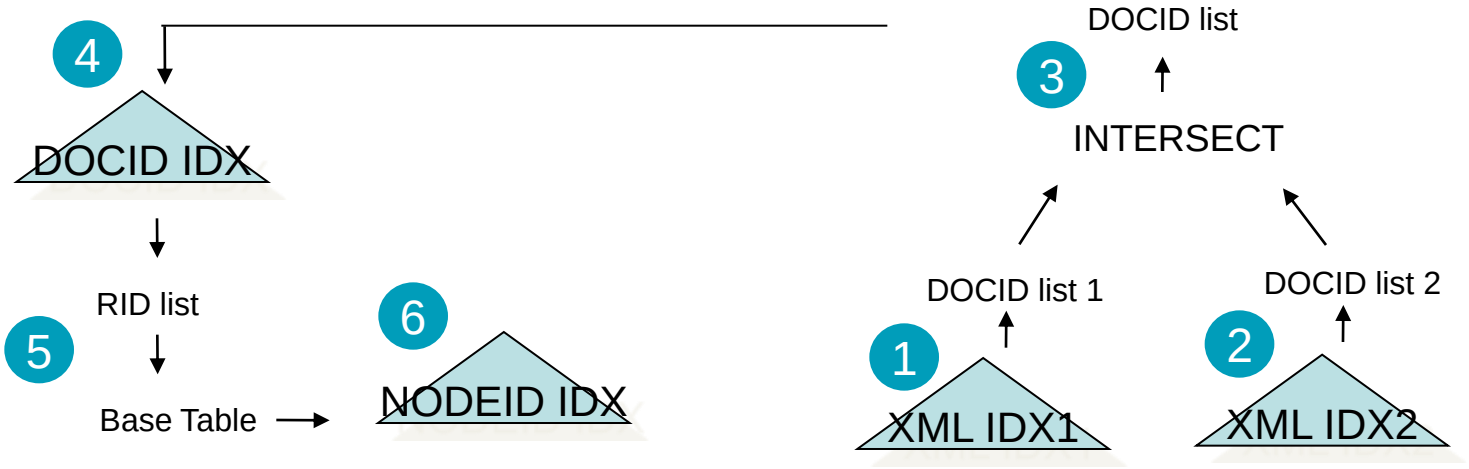
- Store XML data and complex data without mapping
- Efficient binary hierarchical storage format.
- XML indexes for query performance
- SQL/XML standard query language
- Same utilities and familiar DBA tools for XML objects.
- zIIP/zAAP redirection for XML processing, and 100% zIIP/zAAP for XML parsing.

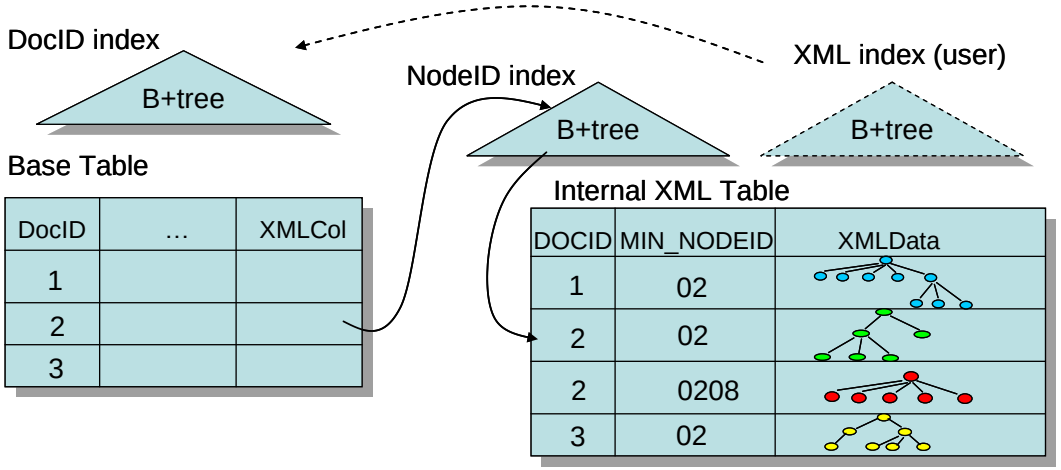
Benefits

- For application developers, new flexible data model and powerful language to deal with new requirements.
  - Higher productivity
  - Shorter time to market
- For DBAs, use the familiar tools managing XML objects.
  - Manageability for XML data
- z platform: unparalleled reliability, availability, scalability, serviceability, and security.

Query processing using index ANDing in a picture

XMLEXISTS('/a/b[c<5 and d="good"]' …) or XMLEXISTS('/a/[…]/d[…]…' …)
or XMLEXISTS(…) AND XMLEXISTS(…) <= same column or different columns





Some recent performance enhancements

- XML insert lock reduction (PK68265)
  - Up to 10% improvements for INSERT throughput
- XML indexes for XML joins (PK55783/PK81260)
- XML index enhancements (PK80732/PK80735)
- XPath scan improvement (PK80732/PK80735)
- Skip XPath scan after XML indexes when possible (PK82631)
  - 4.7% CPU reduction on TPoX benchmark
- Improving traversal in XMLAGG (PK88034)

- Some highlight follows.
  For further info: XML Info APAR: II14426

XML insert/update lock reduction

- XML locking:
  - XML lock: Document level lock. S-lock, X-lock.
  - XML TS: page level locking
- Before PK68265:
  - Non-UR readers: XML lock only when materializing to workfile (order by/group by)
  - UR readers: XML lock to prevent partial document read
  - Readers do not hold/use XML TS page locks.
  - Updaters: XML lock + XML TS page locks
- After PK68265:
  - No change for the readers.
  - Updaters: XML lock + XML TS enhanced row lock
  - Non data sharing: page latch only
  - Data sharing: P-Lock

XML index for XML joins

- TPoX Q7: Find the most expensive order of a customer
- SELECT …
  FROM custacc, order
  WHERE … AND XMLEXISTS(
  'declare default element namespace
  "http://www.fixprotocol.org/FIXML-4-4";
  declare namespace c="http://tpox-benchmark.com/custacc";
  $odoc/FIXML/Order[@Acct=
  $cadoc/c:Customer/c:Accounts/c:Account/@id] '
  PASSING cadoc AS "cadoc", odoc AS "odoc")
- Join between orders and account
- XML index XMLPATTERN: '/FIXML/Order/@Acct'
- PK55783/PK81260 enhanced to use XML index for the joins

Index enhancements in PK80732/80735

- How to represent NULL in XML? Usually with absence
- How to search for it? Use fn:exists() or fn:not()
- Index support for fn:exists() and fn:not()
  - CREATE INDEX … XMLPATTERN '/a/b/fn:exists(c)' AS SQL VARCHAR(1)
  - Query: XMLEXISTS('/a/b[fn:not(c)]'…) or XMLEXISTS('/a/b[c]'…) or XMLEXISTS('/a/b[fn:exists(c)]'…)

- Case-insensitive search using fn:upper-case()
- Index support for fn:upper-case():
  - CREATE INDEX … XMLPATTERN '/a/b/c/fn:upper-case(.)' AS SQL VARCHAR(20)
  - Query: XMLEXISTS('/a/b[fn:upper-case(c)="IBM"]'…)

- Index support for fn:starts-with(), substring(s,1, n)
  - Query: XMLEXISTS('/a/b[starts-with(c, "AC")]' …) or XMLEXISTS('/a/b[substring(c,1,2)="AC"]'…)

- Index support for "!=": XMLEXISTS('/a/b[c!="CLEARED"]' …)

XPath scan improvement in PK80732/80735

- QuickXScan is the algorithm for XPath evaluation by scan
- Optimization is being done to improve the performance
  - Make simple XPath, such as /a/b/c extremely fast
  - Make simple XPath predicate such as /a/b[c=5] extremely fast
  - Do not accumulate sequences but predicate results /a/b[c="abc" and d>5]
  - Pushdown the search during tree traversal: /a/b[.//c=5]
  - …

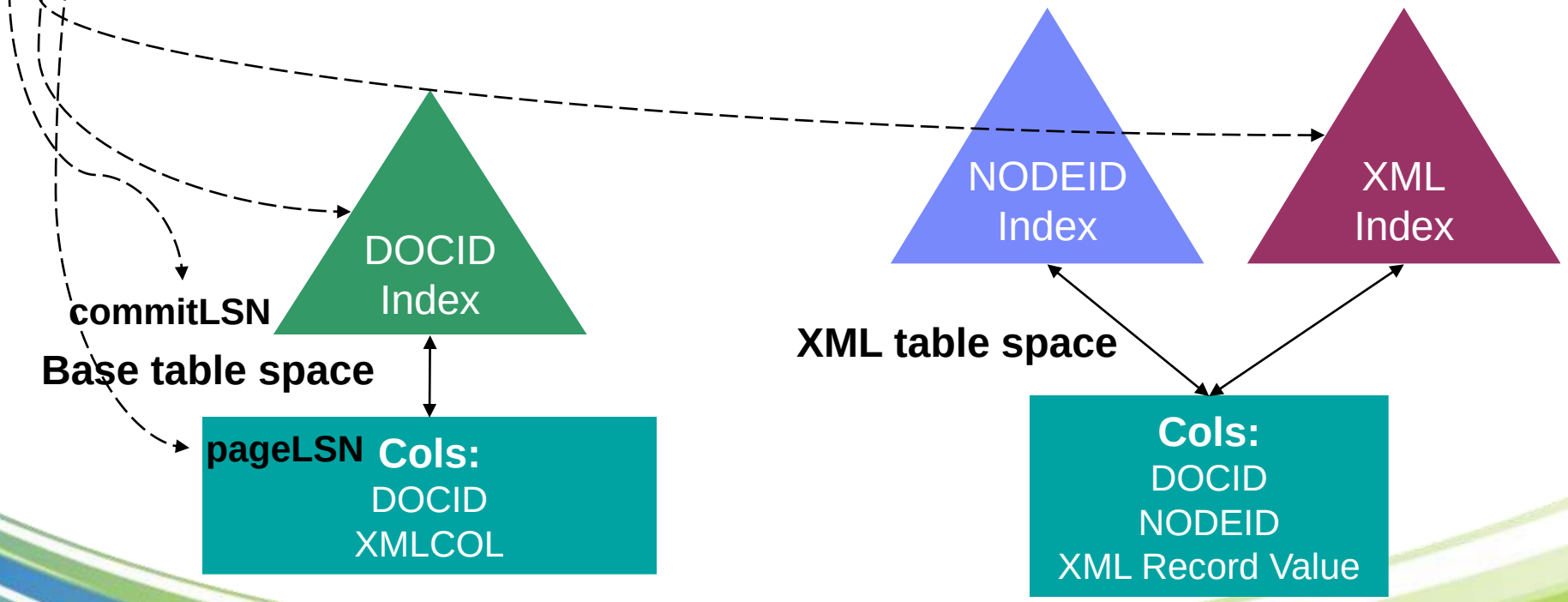
Avoid XMLEXISTS XPath scan after XML indexes PK82631

- XMLEXISTS predicates are stage 2 predicates.
- The XPath in XMLEXISTS is re-evaluated by QSCAN after XML index matches.
  - For two reasons: not exact match, and possible update
- Evaluating XPath using scan is quite CPU intensive.
- Under certain condition, re-evaluation can be avoid and performance improved:
  1. The XPath in the predicates matches exactly with the XPath for the index. (prepare)
  2. No update to the XML document since the start of the index scan. (runtime)

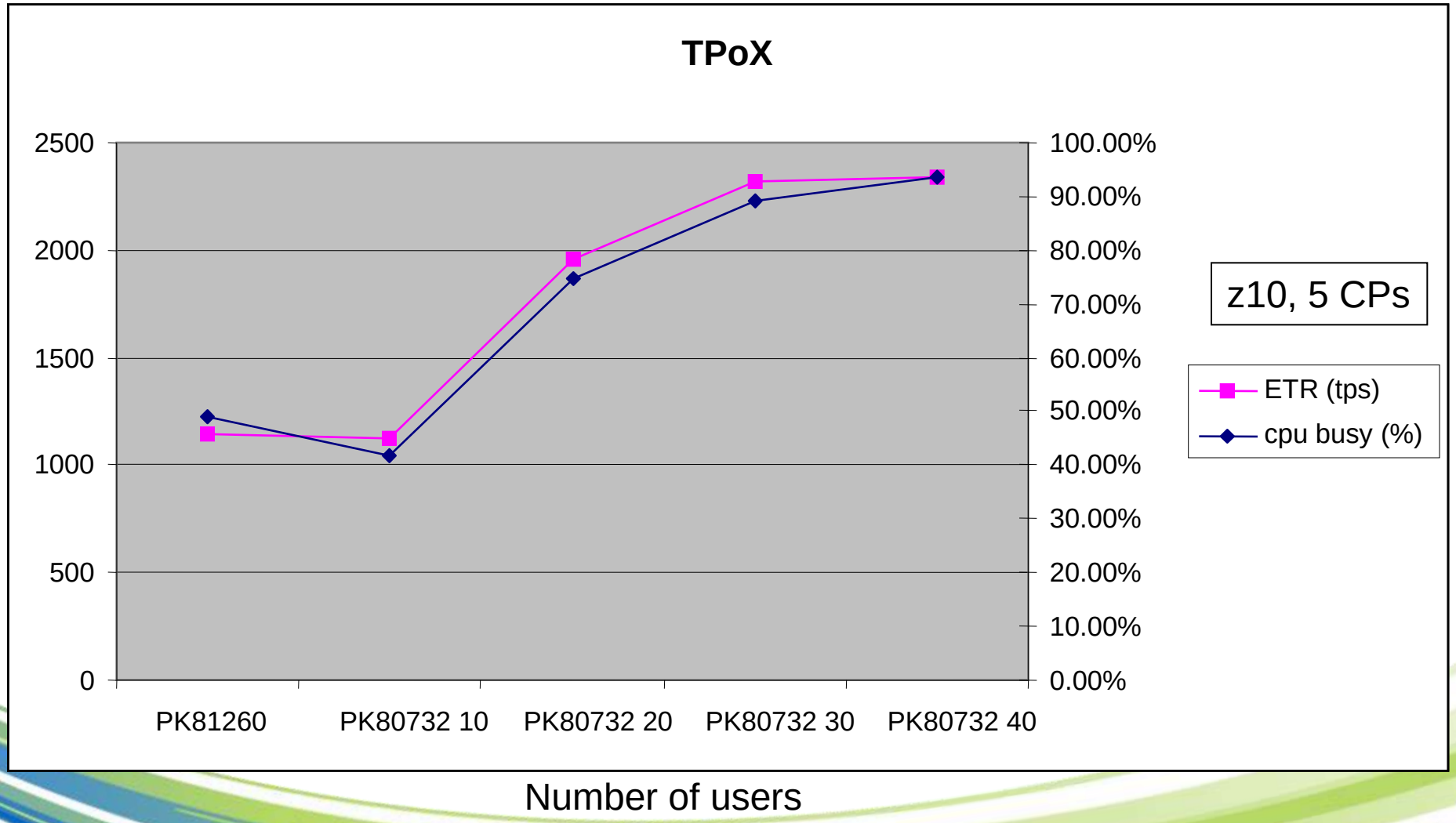
Avoid re-evaluation of XMLEXISTS

Runtime check the flag and do following to check for condition #2

1. Retrieve system's commitLSN.
2. Do XML index scan and retrieve the DocID List.
3. Retrieve RID list from DOCID index.
4. Retrieve pageLSN for each row in the RID list.
5. Compare pageLSN and commitLSN to decide if the re-evaluation can be avoided.



TPoX Benchmark



Key XML Features in DB2 10

- To ensure the quality & consistency of XML data inserted into a column
  ➔ XML schema association with XML columns (a.k.a. XML column type modifier) and automatic schema validation for insert/update

- XML schema validation is CPU intensive, and with 50MB size limit.
  ➔ Using z/OS XML System Services, 100% zIIP/zAAP redirectable

- Native XML Date and Time important in business processing
  ➔ xs:date, xs:dateTime, and xs:dateTime support and XML index support

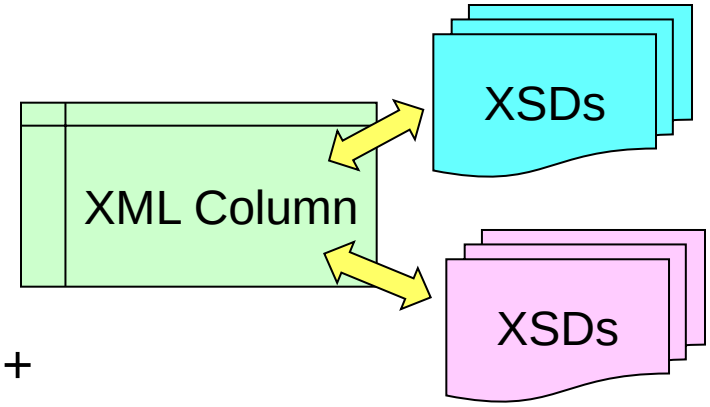
- Update a piece of information in an XML document is common
  ➔ Basic XML sub-document update

- Business logic standardization/performance with native SQL PL stored procedures and user defined functions
  ➔ XML support in SQL PL STP/UDF

- Performance enhancements, including binary XML between client and server.

XML schemas as XML column type modifier

- Add XML schemas as column type modifier, DB2 enforces the conformance.
- Two ways to identify an XML schema (existing)
  - Schema name, or target namespace + optional schema location



Example

- CREATE TABLE T1 (X1 XML(XMLSCHEMA ID SYSXSR.schema1, URI 'uri' LOCATION 'location'), X2 XML);
- ALTER TABLE T1 ALTER X2 SET DATA TYPE XML(XMLSCHEMA URI 'http://www.example.com/po' ELEMENT "purchaseorder")

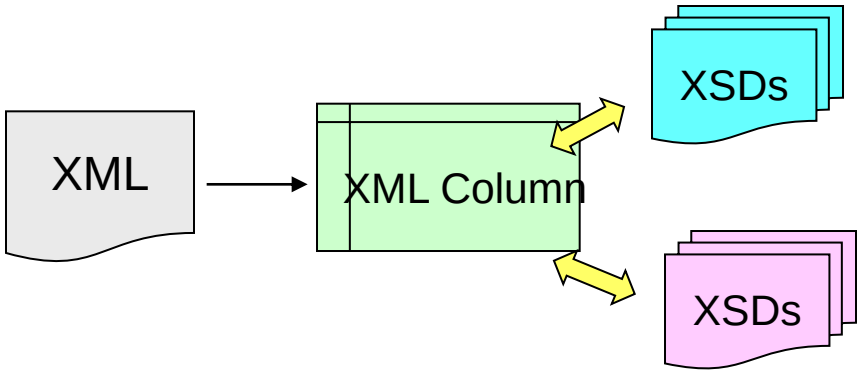
Automatic Validation

- Insert

```
INSERT INTO purchase_orders
VALUES (:hv);
```

- Update

```
UPDATE purchase_orders
    SET content =:hv;
```



- Load performing validation
- If the source is already validated according to the same XML schema in the type modifier, it won't be revalidated again.

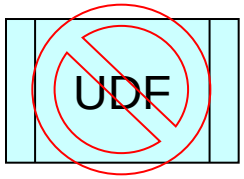
- Schemas evolve, multiple versions coexist
  - At insert/update time, choose one of them based on information from instance doc (target namespace, schema location).
  - If no schema location, with multiple schema matches, the latest will be chosen.

Alter XML type modifier

- Always use ALTER TABLE T1 ALTER X2 SET DATA TYPE XML(…): whole set of schemas – DB2 figures out the delta
- CHECK DATA – validate docs or put invalid ones in exception table

Change from Type Modifier	To Type Modifier	XML Table Space Impact
No modifier	PO	Check pending
PO1	PO1, PO2	No
PO1	PO2	Check pending
PO1, PO2	PO1	Check pending
PO1	No modifier	No

Schema Validation inside Engine



- Invoke z/OS XML System Services and make it 100% zIIP/zAAP redirectable
- No need to specify schema in validation function.
  - Latest registered one will be chosen if there are multiple matches (better use schema location).
- Old syntax still works

Examples

```
INSERT INTO T1 VALUES (:hv)
INSERT INTO T1 VALUES (DSN_XMLVALIDATE(:hv) )
```

You don’t need to call DSN_XMLVALIDATE if the column has schema modifier

More Usage Examples

- New Syntax:
  - INSERT INTO T1(X1) VALUES (
    DSN_XMLVALIDATE(:xmlInHV, :xmlSchemaNameHV));
  - UPDATE T1 SET X1= DSN_XMLVALIDATE(:xmlInHV,:xmlSchemaNameHV)
    WHERE NAME='Dorothy';
  - SELECT DSN_XMLVALIDATE (XMLCol1)
    FROM T1
    WHERE NAME='Dorothy‘
  - INSERT INTO T1(X1) VALUES (DSN_XMLVALIDATE(:xmlInHV));
  - UPDATE T1 SET X1= DSN_XMLVALIDATE(:xmlInHV)
    WHERE NAME='Dorothy';

Schema Validation Impact

- XSR setup is changed to call z/OS schema compiler (same Java based) in NFM.
- The compiled schemas in V9 still work for V10.
- UDF invocation will be converted to BIF starting V10 CM mode.
- Streaming parsing v.s. whole doc in UDF
- Performance improved for small documents
- Some degradation for large documents (> 10MB) in z/OS V1.9, Expect to improve in V1.10 and later.
- 100% zIIP/zAAP redirectable if available.

Schema determination (1 of 4)

XML schema name	Target Namespace	Schema Location	Registration Timestamp
PO1	http://www.example.com/PO1	http://www.example.com/PO1.xsd	2009-01-01 10:00:00.0000
PO2	http://www.example.com/PO2	http://www.example.com/PO2.xsd	2010-01-01 10:00:00.0000
PO3	NO NAMESPACE	http://www.example.com/PO3.xsd	2010-01-30 10:00:00.0000
PO4	http://www.example.com/PO2	http://www.example.com/PO4.xsd	2010-02-23 08:00:00.000

Example 1:

```
INSERT INTO purchase_orders VALUES(1,
'<po:purchaseOrder xmlns:po="http://www.example.com/PO1">
...
</po:purchaseOrder>
);
```

Schema determination (2 of 4)

XML schema name	Target Namespace	Schema Location	Registration Timestamp
PO1	http://www.example.com/PO1	http://www.example.com/PO1.xsd	2009-01-01 10:00:00.0000
PO2	http://www.example.com/PO2	http://www.example.com/PO2.xsd	2010-01-01 10:00:00.0000
PO3	NO NAMESPACE	http://www.example.com/PO3.xsd	2010-01-30 10:00:00.0000
PO4	http://www.example.com/PO2	http://www.example.com/PO4.xsd	2010-02-23 08:00:00.000

Example 2:

```
INSERT INTO purchase_orders VALUES(2,
'<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO2.xsd">
...
</po:purchaseOrder>');
```

Schema determination (3 of 4)

XML schema name	Target Namespace	Schema Location	Registration Timestamp
PO1	http://www.example.com/PO1	http://www.example.com/PO1.xsd	2009-01-01 10:00:00.0000
PO2	http://www.example.com/PO2	http://www.example.com/PO2.xsd	2010-01-01 10:00:00.0000
PO3	NO NAMESPACE	http://www.example.com/PO3.xsd	2010-01-30 10:00:00.0000
PO4	http://www.example.com/PO2	http://www.example.com/PO4.xsd	2010-02-23 08:00:00.000

Example 3:

```
INSERT INTO purchase_orders VALUES(2,
'<po:purchaseOrder xmlns:po="http://www.example.com/PO2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.example.com/PO2
http://www.example.com/PO4.xsd'>
...
</po:purchaseOrder>');
```

Schema determination (4 of 4)

XML schema name	Target Namespace	Schema Location	Registration Timestamp
PO1	http://www.example.com/PO1	http://www.example.com/PO1.xsd	2009-01-01 10:00:00.0000
PO2	http://www.example.com/PO2	http://www.example.com/PO2.xsd	2010-01-01 10:00:00.0000
PO3	NO NAMESPACE	http://www.example.com/PO3.xsd	2010-01-30 10:00:00.0000
PO4	http://www.example.com/PO4	http://www.example.com/PO4.xsd	2010-02-23 08:00:00.000

Example 4:

```
INSERT INTO purchase_orders VALUES(3,
'<purchaseOrder xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://www.example.com/PO3.xsd">
...
</purchaseOrder>');
```

XML Date and Time Support

- Date and time are not supported in XPath in V9
  - Workaround: XMLTABLE and SQL date and time
- Add support xs:date, xs:time, and xs:dateTime and indexes for DATE and TIMESTAMP
  - Implicit timezone will be UTC if a date or time does not have a timezone
  - You get back the original or use fn:adjust-…-to-timezone() for a local timezone
- Supported types
  - xs:dateTime
  - xs:time
  - xs:date
  - xs:duration
  - xs:yearMonthDuration
  - xs:dayTimeDuration
- And a bunch of functions and operators: with time zone support, surpass SQL.

Examples

- XMLQUERY(‘/purchaseorder/items/item[shipdate > xs:date(“2007-01-31”)]’ PASSING X1)
- CREATE INDEX XIX1 ON T1(X1) GENERATE KEYS USING XMLPATTERN ‘/purchaseorder/items/item/shipdate’ as SQL DATE
- XMLQUERY(‘/purchaseorder [ordertime > xs:dateTime(“2007-01-31T10:20:30-08:00”)]’ PASSING X1)
  - Note: xs:dateTime(“2007-01-31T10:20:30-8:00”) does not work
- CREATE INDEX XIX1 ON T1(X1) GENERATE KEYS USING XMLPATTERN ‘/purchaseorder/ordertime’ as SQL TIMESTAMP
- XMLQUERY('fn:adjust-dateTime-to-timezone(xs:dateTime("2007-01-31T10:20:30-05:00"), xs:dayTimeDuration("-PT8H"))')
  - Result: 2007-01-31T07:20:30-08:00

Sub-document Update

- No easy way to update parts of a document in V9
- Sub-document update with multi-versioning in DB2 10
- Only simple update: insert, replace, delete from XQuery update facility

Example

Increase premium by 10%

```
UPDATE POLICYTAB SET POLICY =
XMLMODIFY
( 'replace value of node /policy/premium with /policy/premium * 1.1')
WHERE POLICYID = '12345';
```

Add a new payment into payment record

```
UPDATE POLICYTAB SET PAYMENTS = XMLMODIFY
('insert node $n as last into /payments', :newpayment as "n")
WHERE POLICYID = :policyid;
```

More on sub-document update

- XMLMODIFY can only be used in RHS of UPDATE SET.
- One updater at a time for a document, concurrency control by the base table – row level locking, page level locking etc.
  - Document level lock to prevent UR reader
- Only changed rows in XML table are updated
- If there is a schema type modifier on the updated XML column, partial revalidation occurs.
  - Global constraints are not validated.

Insert expression (1 of 3)

Sample XML document:

```
<person>
  <firstName>Joe</firstName>
  <lastName>Smith</lastName>
  <nickName>Joey</nickName>
</person>
```

Sample insert statement:

```
update personinfo set info = xmlmodify('
  insert node $ins/ename
  after  /person/nickName', xmlparse(document '
      <ename>Joe.Smith@de.ibm.com</ename>‘) as  "ins") ;
```

Insert Operation	Resulting XML document
insert node $ins/ename **into** $ins/person , XMLPARSE(document ' <ename>Joe.Smith@de.ibm.com</ename>‘) as "ins") (nondeterministic position)	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename>Joe.Smith@de.ibm.com</ename>** </person>
insert node $ins/ename **as last into** $ins/person, XMLPARSE(document ' <ename>Joe.Smith@de.ibm.com</ename>‘) as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename>Joe.Smith@de.ibm.com</ename>** </person>

Insert expression (2 of 3)

```
<person>
   <firstName>Joe</firstName>
   <lastName>Smith</lastName>
   <nickName>Joey</nickName>
</person>
```

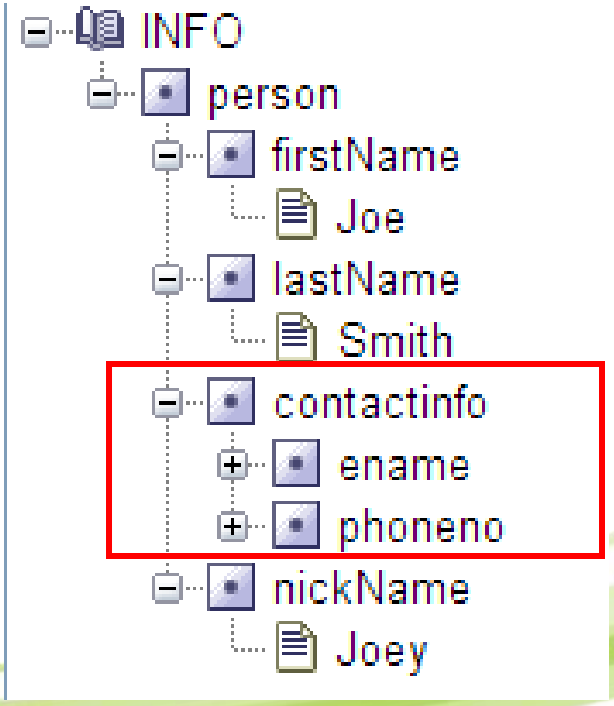
Insert Operation	Resulting XML document
Insert node $ins/ename **as first into** $ins/person, XMLPARSE(document <ename>Joe.Smith@de.ibm.com</ename>‘) as "ins")	<person> **<ename>Joe.Smith@de.ibm.com</ename>** <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> </person>
Insert node $ins/ename **after** $ins/person/nickName , XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>‘) as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename>Joe.Smith@de.ibm.com</ename>** </person>
Insert node $ins/ename **before** $ins/person/nickName , XMLPARSE(document ' <ename>Joe.Smith@de.ibm.com</ename>‘) as "ins")	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> **<ename>Joe.Smith@de.ibm.com</ename>** <nickName>Joey</nickName> </person>

Insert expression (3 of 3)

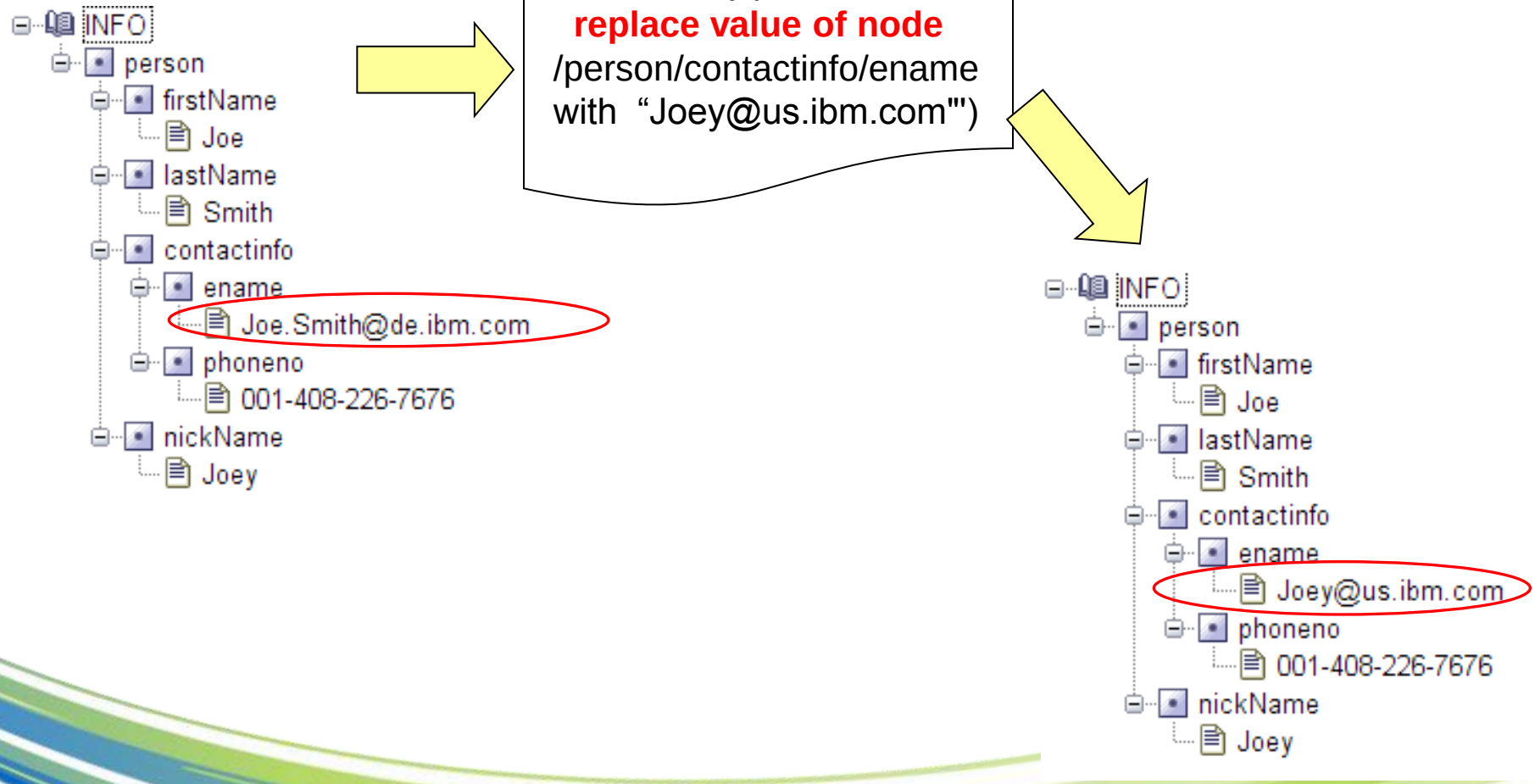
- Insertion of sequence of elements also possible
- Use same keywords as introduced before

```
<person>
    <firstName>Joe</firstName>
    <lastName>Smith</lastName>
    <nickName>Joey</nickName>
</person>
```

```
update personinfo set info = xmlmodify('
  insert node $ins/contactinfo
before /person/nickName',
xmlparse(document '
<contactinfo>
<ename>Joe.Smith@de.ibm.com</ename>
<phoneno>001-408-226-7676</phoneno>
</contactinfo>
') as "ins") ;
```

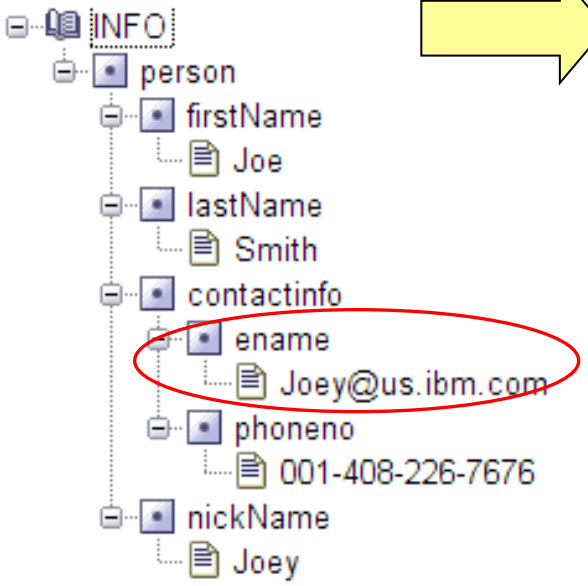


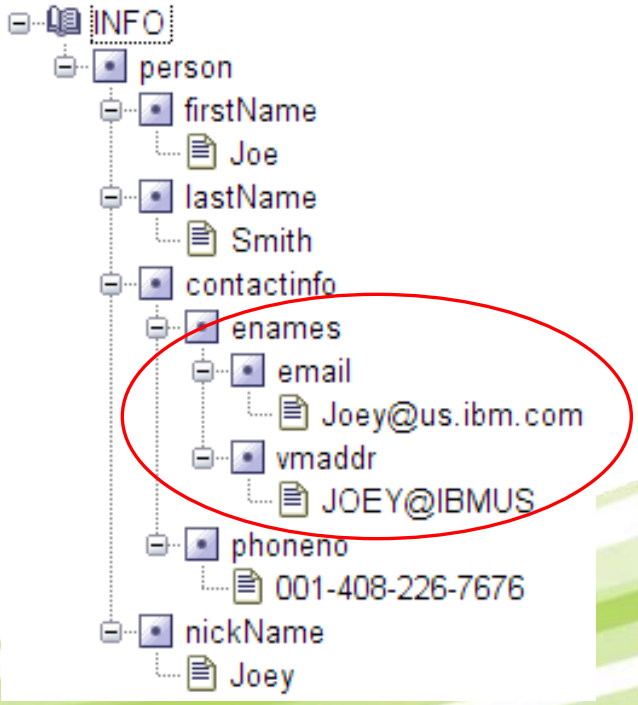
Replace expression (1 of 2)



Replace expression (3 of 3)

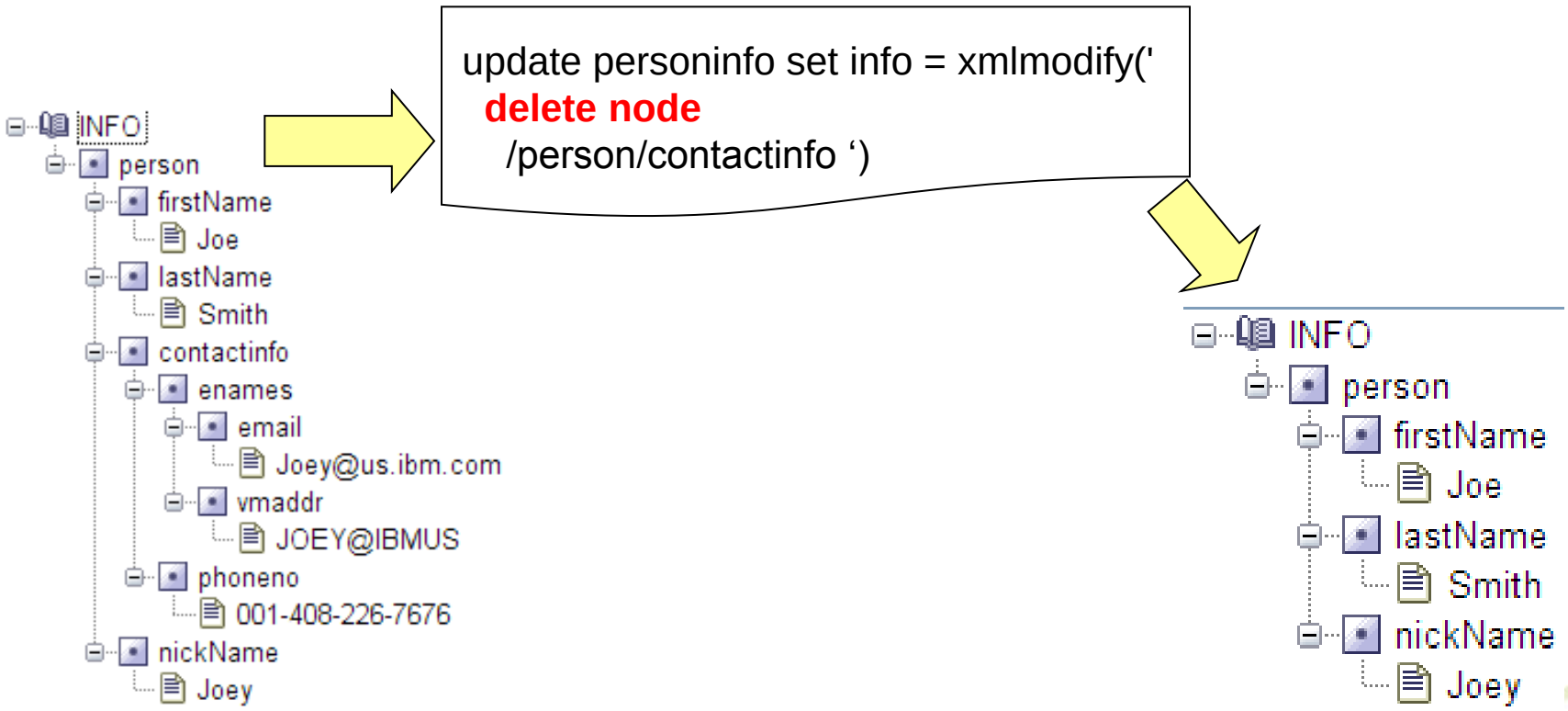
```
update personinfo set info = xmlmodify('
  replace node /person/contactinfo/ename
  with  $x', XMLPARSE(document '
  <enames>
    <email>Joey@us.ibm.com</email>
     <vmaddr>JOEY@IBMUS</vmaddr>
  </enames>') as "x");
```





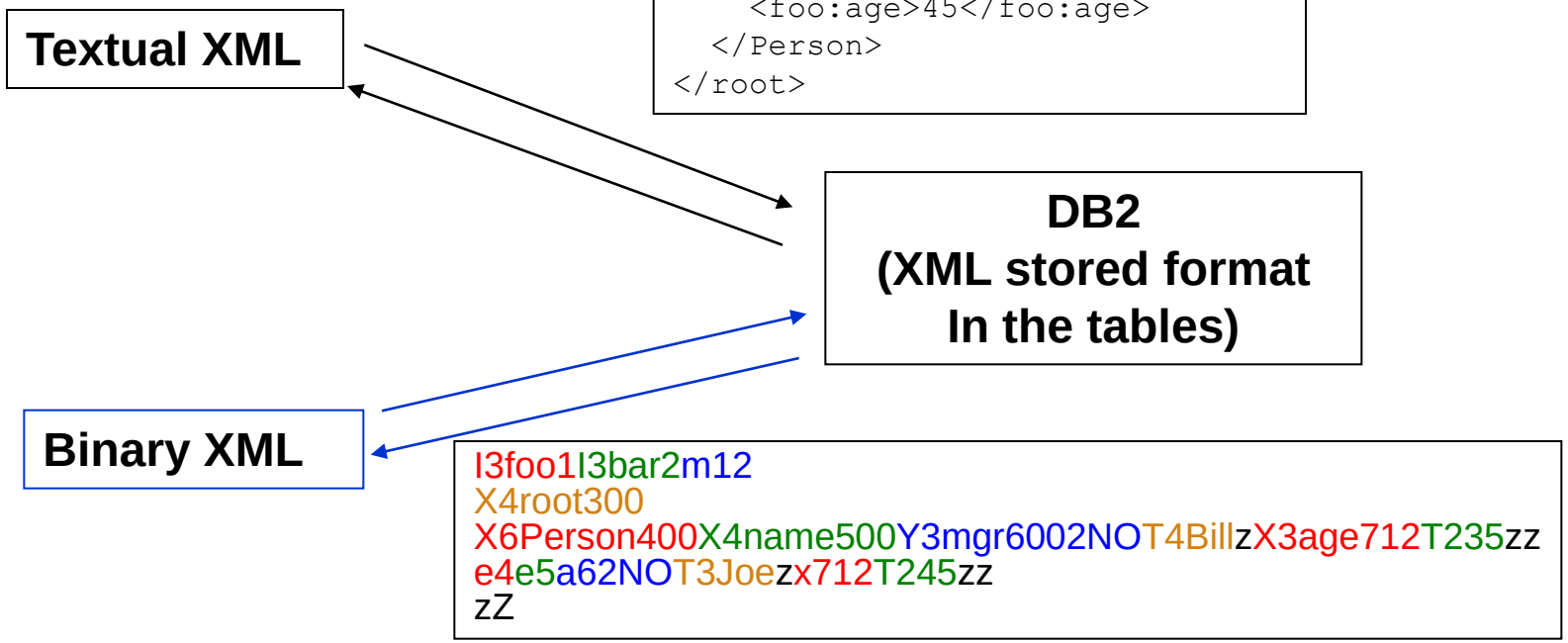
Delete expression

– Delete can be used to delete nodes from a node sequence



Binary XML support

```
<root xmlns:foo = “bar”>
  <Person>
    <name mgr = “NO”>Bill</name>
    <foo:age>35</foo:age>
  </Person>
  <Person>
    <name mgr = “NO”>Joe</name>
    <foo:age>45</foo:age>
  </Person>
</root>
```



-- Binary XML is about 17%-46% smaller in size
-- Save DB2 over 9%-30% CPU during insert,
-- End to end time saving 8%-50% for insert

Usage: Application using JDBC 4.0(JSR-221) -1/2

Fetch XML as SQLXML type:

```
String sql = "SELECT xml_col from T1";
PreparedStatement pstmt = con.prepareStatement(sql);
ResultSet resultSet = pstmt.executeQuery();

// get the result XML as SQLXML
SQLXML sqlxml = resultSet.getSQLXML(column);

// get a DOMSource from SQLXML object
DOMSource domSource = sqlxml.getSource(DOMSource.class);
Document document = (Document) domSource.getNode();

// or: get a SAXSource from SQLXML object
SAXSource saxSource = sqlxml.getSource(SAXSource.class);
XMLReader xmlReader = saxSource.getXMLReader();
xmlReader.setContentHandler(myHandler);
xmlReader.parse(saxSource.getInputSource());

// or: get binaryStream or string from SQLXML object
InputStream binaryStream = sqlxml.getBinaryStream(); // or:
String xmlString = sqlxml.getString();
```

New SQLXML type

Retrieve DOM tree

Or get SAX events

SQLXML object can only be read once

Usage: Application using JDBC 4.0(JSR-221) -2/2

Insert and update XML using SQLXML

```
String sql = "insert into T1 values(?)";
PreparedStatement pstmt = con.prepareStatement(sql);

SQLXML sqlxml = con.createSQLXML();

// create a SQLXML object from the DOM object
DOMResult domResult = sqlxml.setResult(DOMResult.class);
domResult.setNode(myNode);

// create a SQLXML object from a string
sqlxml.setString(xmlString);

// set that xml doucment as the input to parameter marker 1
pstmt.setSQLXML(1, sqlxml);

pstmt.executeUpdate();

Sqlxml.free();
```

Support XML in SQL PL STP/UDF

- In V9 XML data type cannot be used as parameters and return type – workaround: use LOBs
- Add support for SQL PL stored procedures, scalar UDFs, and table UDFs
  - XML type: SQL variables, arguments, and return type
- Used for business logic close to data storage, encapsulate/abstract complex functions, etc.
- Alternative to host language programming (COBOL, C, Java etc.)

XML in SQL PL native procedures

```
CREATE TABLE tab1(C1 XML)

CREATE PROCEDURE proc1(IN parm1 XML,
                       IN parm2 VARCHAR(32000))
LANGUAGE SQL
  BEGIN
        DECLARE var1 XML;
        SET var1 = XMLPARSE(document parm2);
        /*INS parm1 if it contains an item with value<200 */
        IF(XMLEXISTS('$x/ITEM[value < 200]' passing parm1 as "x"))
         THEN
              SET var1 = XMLDOCUMENT(XMLElement(NAME "doc", parm1, var1));
        END IF;
        /* parse parm2 and insert it */
        INSERT INTO tab1 VALUES(var1);
  END
```

Decomp to multiple tables w/ SQL PL proc

```
CREATE PROCEDURE DECOMP1(IN DOC XML)  /* or IN DOC BLOB */
LANGUAGE SQL
 BEGIN
       /* DECLARE xdoc XML;
       SET xdoc = XMLPARSE(document doc); */
       INSERT INTO tab1 SELECT *
         FROM XMLTABLE('/doc/head/rows' PASSING DOC /* or xdoc */
              COLUMNS C1 INT PATH 'C1',
                    C2 VARCHAR(10) PATH 'C2') AS X;

       INSERT INTO tab2 SELECT *
         FROM XMLTABLE('/doc/body/rows' PASSING DOC /* or xdoc */
              COLUMNS C3 INT PATH 'C3',
                    C4 VARCHAR(10) PATH 'C4') AS X;
 END
```

Parse once and decompose into multiple tables

If using Java caller, document could be parsed into binary XML in the client

Encapsulate a web service – stock quote before

```
SELECT *
FROM XMLTABLE('/StockQuotes/Stock' PASSING
 XMLPARSE(DOCUMENT XMLCAST(
  XMLQUERY('declare namespace soap="http://schemas.xmlsoap.org/soap/envelope/";
 declare default element namespace "http://www.webserviceX.NET/";
/soap:Envelope/soap:Body/GetQuoteResponse/GetQuoteResult'
passing XMLPARSE(DOCUMENT
DB2XML.SOAPHTTPNC('http://www.webservicex.net/stockquote.asmx',
  'http://www.webserviceX.NET/GetQuote',
  '<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetQuote xmlns="http://www.webserviceX.NET/">
      <symbol>IBM</symbol>
    </GetQuote>
  </soap:Body>
</soap:Envelope>') )  ) as VARCHAR(2000)) )
COLUMNS
 "Symbol"  VARCHAR(4),
 "Last"    DECIMAL(6,2),
 "Date"    VARCHAR(10),
 "Time"    VARCHAR(8),
 "Change"  VARCHAR(8),
 "Open"    VARCHAR(8),
 "High"    VARCHAR(8),
 "Low"     VARCHAR(8),
 "Volume"  VARCHAR(12) ) XT
```

Hiding details and parameterizing it - after

```
CREATE FUNCTION STOCKQUOTE(IN SYMBOL VARCHAR(4))
 RETURNS TABLE (Symbol VARCHAR(4), LAST DECIMAL(6,2), …)
 LANGUAGE SQL
 NOT DETERMINISTIC
RETURN
SELECT *
FROM XMLTABLE('/StockQuotes/Stock' PASSING
 XMLPARSE(DOCUMENT XMLCAST(
  XMLQUERY('declare namespace soap="http://schemas.xmlsoap.org/soap/envelope/";
 declare default element namespace "http://www.webserviceX.NET/";
/soap:Envelope/soap:Body/GetQuoteResponse/GetQuoteResult'
passing XMLPARSE(DOCUMENT
DB2XML.SOAPHTTPNC('http://www.webservicex.net/stockquote.asmx',
  'http://www.webserviceX.NET/GetQuote',
  '<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetQuote xmlns="http://www.webserviceX.NET/">
      <symbol>' || SYMBOL ||'</symbol>
    </GetQuote>
  </soap:Body>
</soap:Envelope>') )  ) as VARCHAR(2000)) )
COLUMNS
 "Symbol"  VARCHAR(4),
 "Last"    DECIMAL(6,2),
 "Date"    VARCHAR(10),
 "Time"    VARCHAR(8),
 "Change"  VARCHAR(8),
 "Open"    VARCHAR(8),
 "High"    VARCHAR(8),
 "Low"     VARCHAR(8),
 "Volume"  VARCHAR(12) ) XT
```

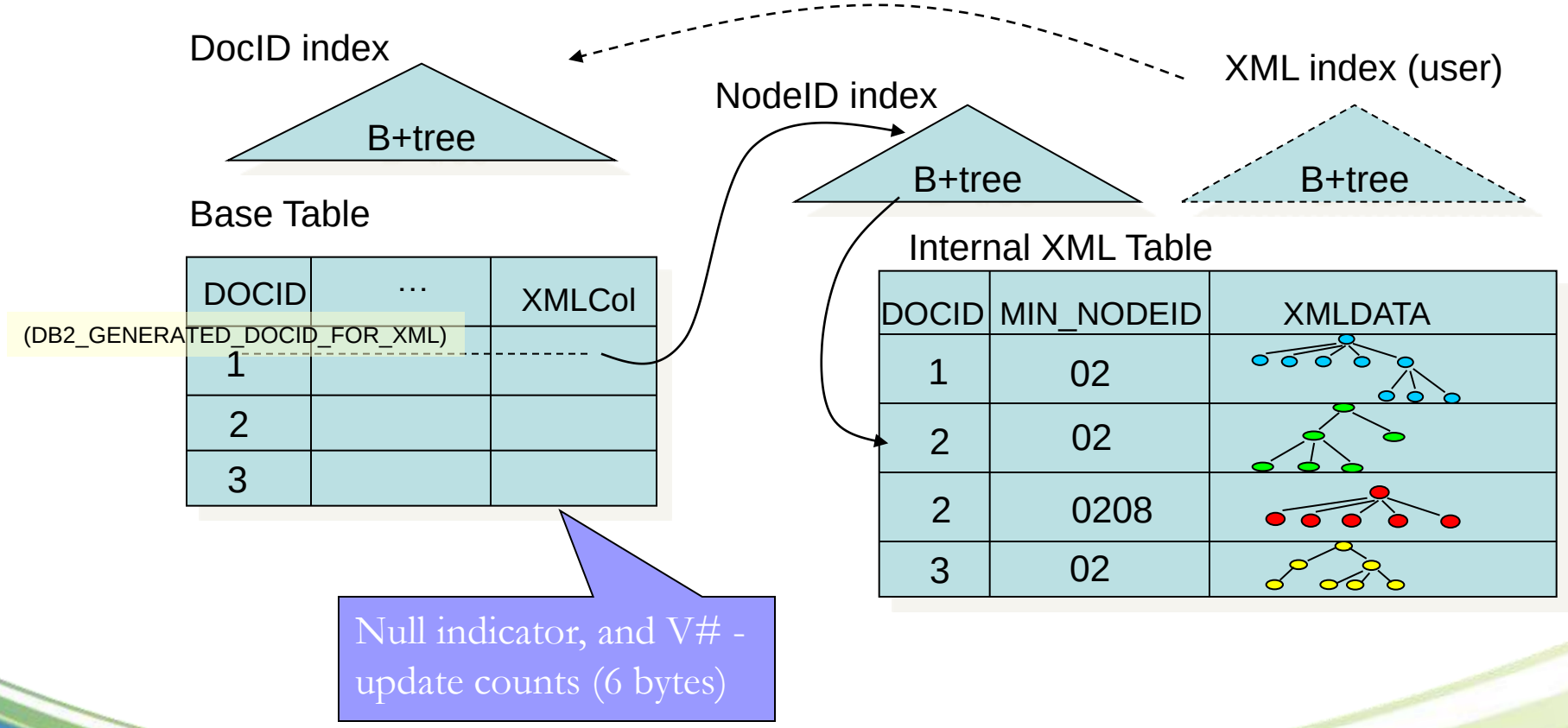
Invocation:

```
SELECT *
FROM TABLE( STOCKQUOTE('IBM')) as X;
```

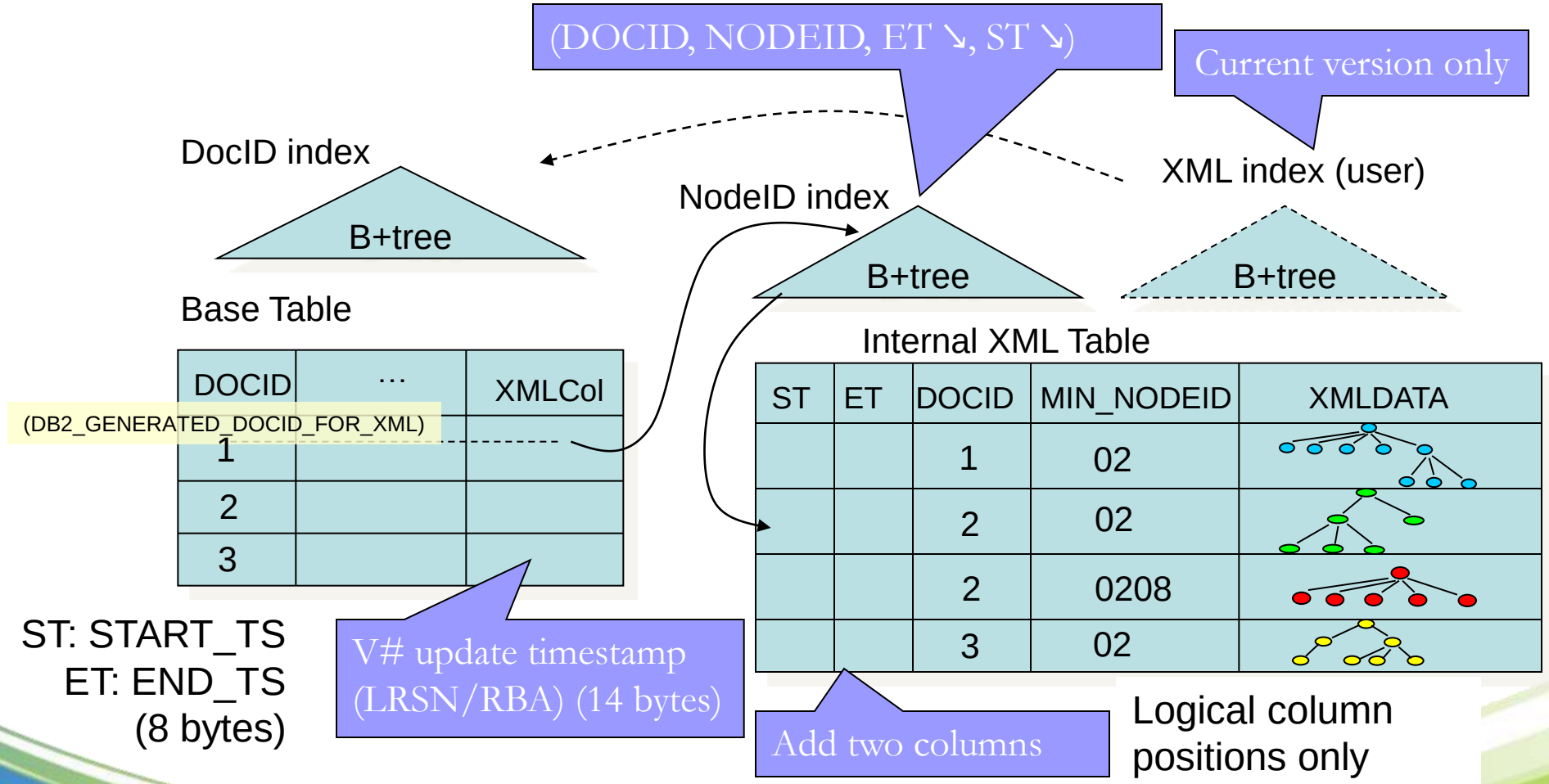
Multi-versioning for XML data (MVCC)

- In V9, deferred fetch of XML column data requires locks to be held.
- Cannot support OLD DATA in SELECT FROM UPDATE, DELETE (and trigger) easily
- Multi-versioning allows no locks for readers (not for UR), and no copy into memory:
  - Materialization of XML column references into workfile
  - Streaming support for XML bind-out
  - Access currently committed data for XML
  - Temporal data for XML
  - XML update
  - Select from OLD DATA: support queues

Basic Storage Scheme in V9



Multi-versioning Scheme



MV Format Storage Impact

- MV format only applies to base tables of UTS in NFM only.
- The XML indicator column in the base table is increased with additional 8 bytes for RBA/LRSN (total 14 bytes).
- XML table rows have two additional BINARY(8) columns for START_TS and END_TS. Due to RRF, these columns are moved to after DOCID column.
- END_TS contains all “FF”s initially. To avoid compression causing update overflow, columns up to END_TS are not compressed.
- A whole document may consist of rows with different START_TS due to update.
- Rows for the current version always contain all “FF” in END_TS.
- NodeID indexes have the two columns added.
- A daemon thread cleans up the expired rows (physical deletion).

XML Locking Scheme with Multi-Versioning

Readers do not need to lock XML !

SQL	Base Page/Row Lock (Business as usual)	XML Lock	XML Table space Lock
INSERT	x page/row lock	x lock, release at commit	P-lock, release at commit
UPDATE/DELETE	u->x, s->x, x stays x	x lock, release at commit	P-lock, release at commit
SELECT UR, CS-CDN	None	s lock, release at next row fetch	
SELECT CS-CDY no workfile	s page/row lock, release on next row fetch	s lock, release at next row fetch	
SELECT CS-CDY workfile	s page/row lock, release on next row fetch	s lock, release at close cursor	
SELECT UR, CS-CDN, CS-CDY with Multirow fetch and dynamic scrolling	s page/row lock on rowset, release on next fetch	s lock, release on next fetch	
SELECT RR, RS	s page/row lock	s lock, release at commit	

CHECK DATA for XML

- **DB2 9**
  - CHECK DATA utility does not cover inconsistency inside XML data
  - NODEID index consistency with XML TS (CHECK INDEX)
  - Base table consistency with NODEID index (CHECK DATA)
- **DB2 10 CHECK DATA** does:
  - Base table consistency with NODEID index (CHECK DATA)
  - NODEID index consistency with XML TS (V9 CHECK INDEX) 
  - Check document structure consistency for each document 
  - Schema validation if column(s) have a type modifier 

 New in V10 CHECK DATA !!

CHECK DATA for XML (cont'ed)

Example

```
CHECK DATA DBTEST.TBT0000 INCLUDE XML TABLESPACES ALL SHRLEVEL
    CHANGE

REPAIR
LOCATE TABLESPACE "DBTEST "."XTBT0000 " DOCID 100
DELETE SHRLEVEL CHANGE

REPAIR
LOCATE TABLESPACE "DBTEST "."TSTEST "
TABLEID 205 COLID 5 DOCID 100
INVALIDATE SHRLEVEL CHANGE
```

- Can specify specific XML tablespace or XML column:
  - XML TABLESPACES xtablespace
  - TABLE POTABLE XMLCOLUMN POXML
  - Can specify NOSCHEMA option to skip schema validation.
- SCOPE XMLSCHEMAONLY option to perform schema validation only.

- Structurally corrupted XML documents will be deleted.
- XML columns with schema type modifier will be updated with validated documents.
- Schema-invalid documents will be moved to an exception table, containing columns of original DOCID, and XML.
- For MV format, only the latest version is checked.

CHECK DATA on XML table spaces

- CHECK DATA utility check validity of XML documents against exisiting type modifiers.
- Use:
  - CHECK DATA ….. INCLUDE XML TABLESPACES ALL to check all related XML table spaces
  - CHECK DATA …... INCLUDE XML TABLESPACES (db.XMLTS, db.XMLTS2) to check specific XML table spaces

Sample CHECK DATA .... INCLUDE XML TABLESPACES output:

```
DSNU849I    007 00:48:58.86 DSNUKERK - XML DOCUMENT WITH DOCID X'0080000000000000016' IS NOT VALID AGAINST
  THE XML TYPE MODIFIER OF XML COLUMN C FOR TABLE ADMF006.EMPLOYEE
 DSNU849I    007 00:48:58.86 DSNUKERK - XML DOCUMENT WITH DOCID X'0080000000000000017' IS NOT VALID AGAINST
  THE XML TYPE MODIFIER OF XML COLUMN C FOR TABLE ADMF006.EMPLOYEE
 DSNU849I    007 00:48:58.86 DSNUKERK - XML DOCUMENT WITH DOCID X'0080000000000000018' IS NOT VALID AGAINST
  THE XML TYPE MODIFIER OF XML COLUMN C FOR TABLE ADMF006.EMPLOYEE
```

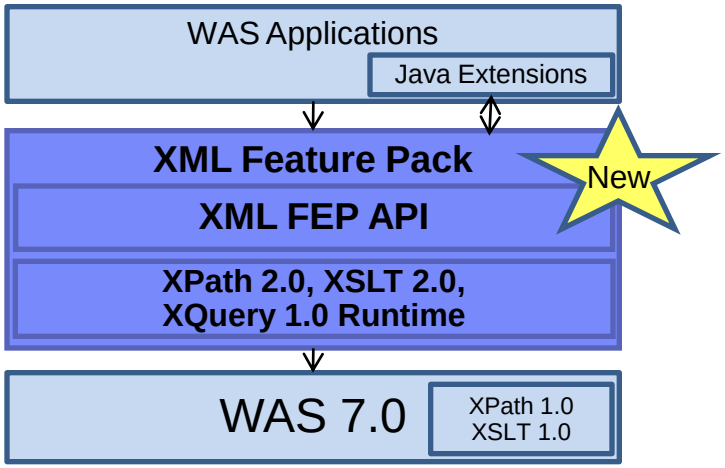
Basic XQuery Embedded in SQL

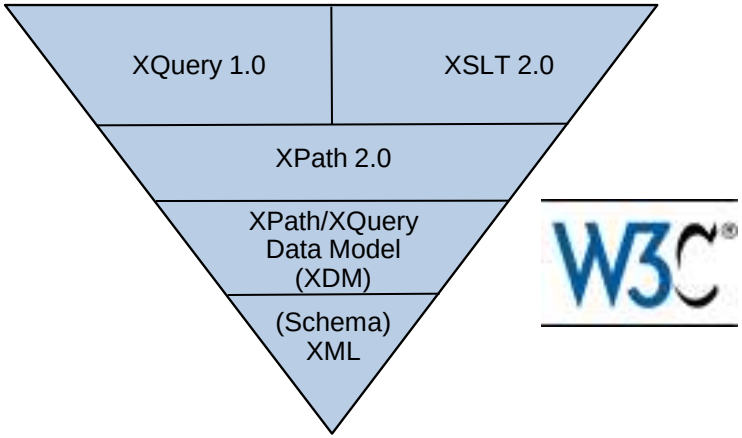
- XPath cannot construct new documents, need to use tedious SQL/XML constructors, also need XMLTABLE to iterate through sequence items

Example

```
SELECT XMLQUERY(
    'let $x := /purchaseOrder
     return
     <invoice invoiceNo= “12345”>
       {$x/billTo}
       <purchaseOrderNo> { $y } </purchaseOrderNo>
       <amount> { fn:sum($x/items/item/xs:decimal(USPrice)) } </amount>
     </invoice>' PASSING xmlpo, PO.ponumber as “y”)
FROM PurchaseOrders PO
WHERE PO.ponumber = ‘200300001’;
```

WebSphere XML Feature Pack Overview





- Values
  - Help improve developer productivity and application performance
  - Unlock enterprise XML data and documents to new application development scenarios

- Highlights / Key Features
  - Full support for the XPath 2.0, XSLT 2.0, and XQuery 1.0 W3C standards
    - Languages for access, query and mining, transformation, styled presentation of XML data sources
  - XML runtime API offering consistent execution/invocation and data navigation while allowing access to existing Java logic

- Additional value
  - Enterprise class multi-threaded scalability, serviceability with IBM support in the application server or thin clients
  - For WAS applications that work with XML through object binding programming models (DOM, JAXB, etc.)
    - Improve performance, fidelity and ease of use by natively working with XML data
  - Over 40 samples of the new standards features
    - End to end samples showing typical web applications as well as integration with XML databases such as DB2 pureXML

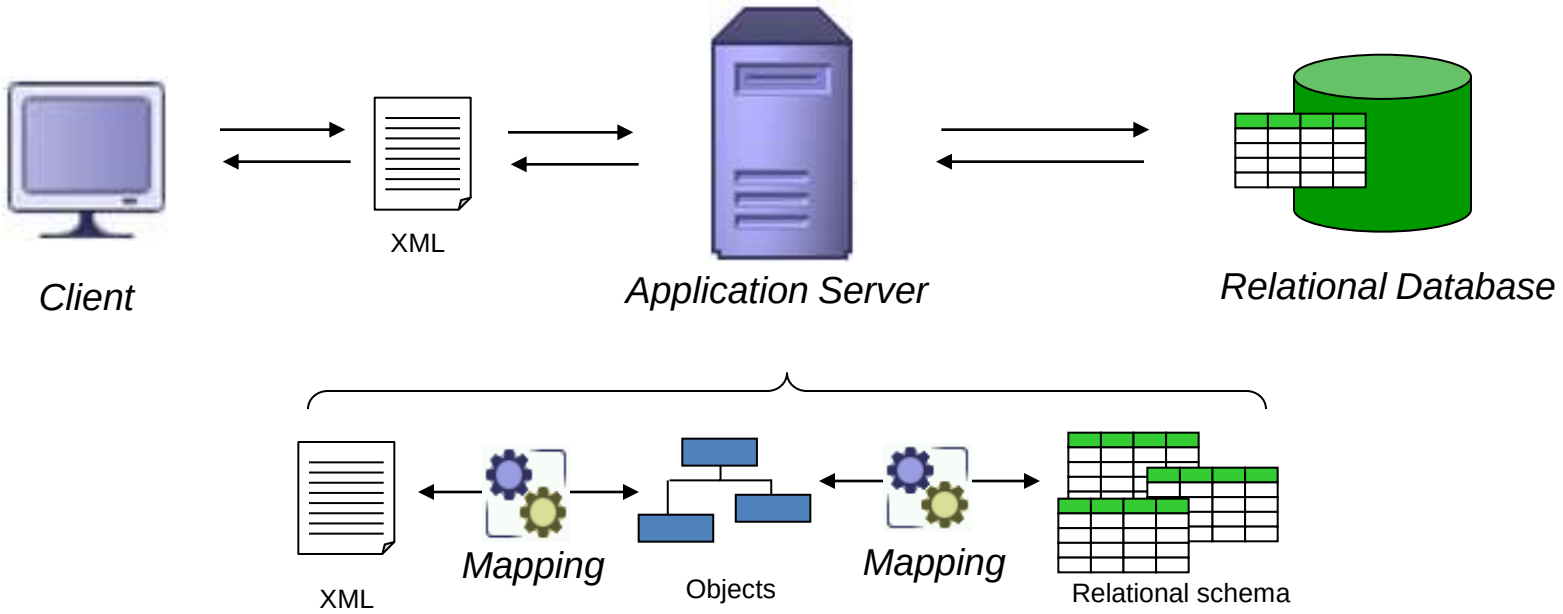
Named query for SQL/XML in Java:

```
dbStatements = new HashMap(String, String>();
dbStatements.put("getCreditDefaultSwapsByEntityName",
       "select xmldocument( "+
              "xmlquery(" +
              " 'declare default element namespace \"http://www.fpml.org/2009/FpML-4-7\"; " +
              " $DOCUMENT/FpML/trade/creditDefaultSwap') " +
              " ) " +
       "from fpmladmin.fpml43 " +
       "where comment like ? and " +
             "xmlexists(" +
                "'declare default element namespace \"http://www.fpml.org/2009/FpML-4-7\"; " +
                "$fpml/FpML/trade/creditDefaultSwap/generalTerms/referenceInformation/
                        referenceEntity[entityName = $name]' " +
               " passing docuemtn as \"fpml\", cast(? as varchar(100)) as \"name\" "
   );
JDBCCollectionResolver inputResolver = new JDBCCollectionResolver(conn, dbStatements);
dc.setCollectionResolver(inputResolver);
```

XQuery (inside Java):

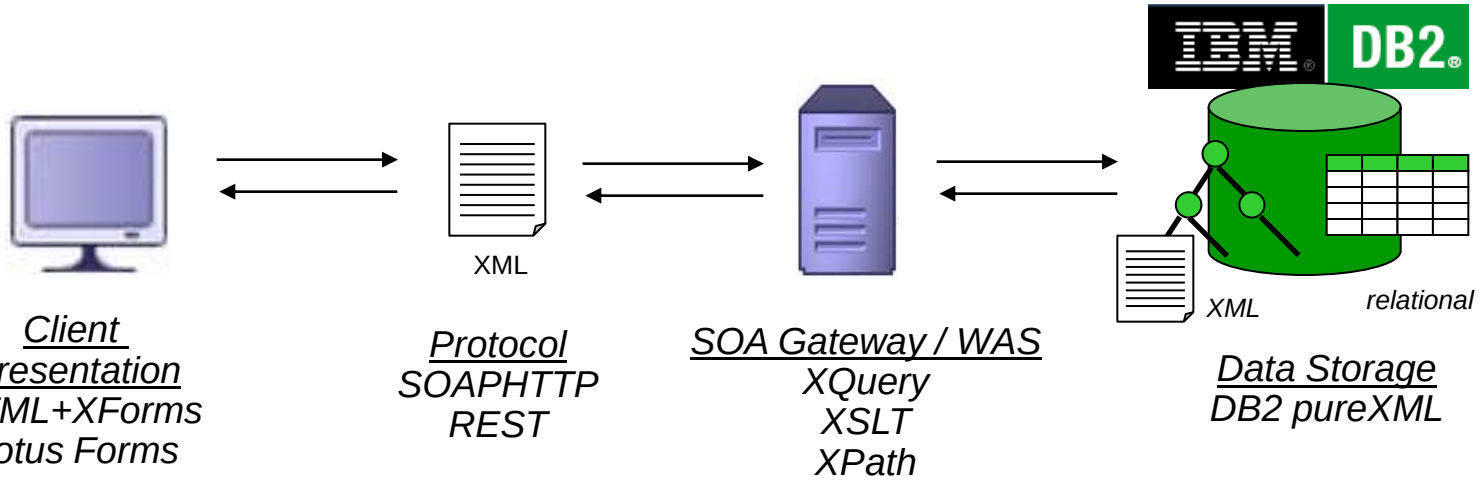
```
declare variable $entityName as xs:string external;
declare variable $databaseURI := concat('jdbc:db2://getCreditDefaultSwapsByEntityName?cd%&',
                                        $entityName);
declare variable $creditDefaultSwaps := collection($databaseURI);

for $bond in $creditDefaultSwaps//fmpl:bond
  return
<tr>
    <td>{ $bond/fpml:instrumentId }</td>
    <td>{ $bond/fpml:couponRate }</td>
    <td>{ $bond/fpml:maturity }</td>
</tr>
```

Web applications before DB2 pureXML



- XML to object mapping and object to relational mapping
  - Very costly
  - Complex
  - Inflexible
- Java + XML?

An End-to-end XML Paradigm



- End-to-End Straight Through Processing using XML
- XML programming paradigm and architecture pattern
  - XForms
  - REST/SOAP web services
  - XQuery suite: XQuery, XQuery update facility, XQuery scripting extension, etc.

Summary

- XML to reduce complexity and cost, time to market
- Flexibility is the key benefit, paradigm shift
- DB2 9 pureXML has matured.
- DB2 10 improves pureXML features further.
- XML programming paradigm: "*Simple, Elegant, and Disruptive."*

Data Management Communities for DB2

- IDUG – the worldwide community of DB2 users
  - Membership is FREE – join today! www.idug.org
- Data Management Community – share and interact with peers around the world
  - www.ibm.com/software/data/management/community.html
- developerWorks – IBM's resource for developers & IT professionals
  - www.ibm.com/developerworks/data/products/db2
- Information Champions – recognizes individuals who have made the most outstanding contributions to the Information Management community
  - www.ibm.com/software/data/champion