#SHAREorg

SHARE
Technology · Connections · Results

Breaking the Relational Limit with pureXML in DB2 for z/OS

Guogen Zhang
IBM

August 10, 2012
Session 11787

SHARE in Anaheim 2012

Agenda

- Introduction and Overview of new XML features in DB2 10
- Tips and hints of using XML in DB2
  - Generating XML test data
  - Enforcing XML schema conformance automatically
  - Sub-document Update
  - CHECK DATA for XML
  - Binary XML format for performance
  - SQL PL Stored Procs and UDFs
  - Data movement, distribution and gathering
  - Simulating a queue
  - Search for documents without a certain element or attribute with indexes
  - Case-insensitive search with XML indexes
  - String pattern matching using regular expressions
  - Flexible parameter passing, simulating arrays and structures
  - Design decision making for XML storage and hybrid storage using triggers
  - Consuming Web services
  - End-user customizable application framework
- Summary

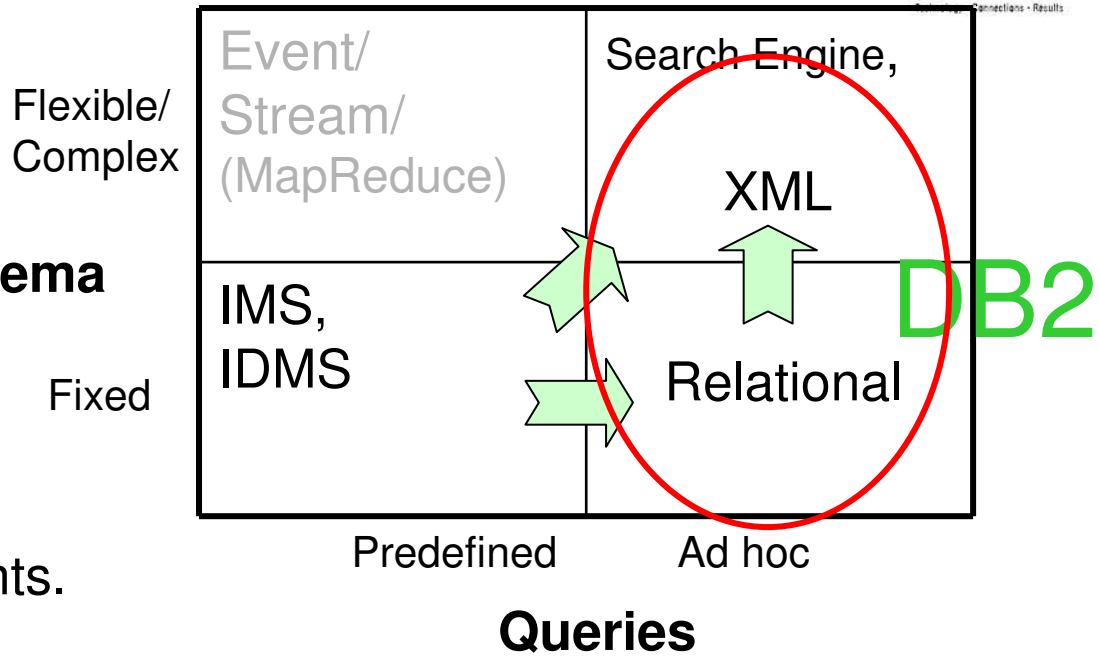
Relational Limit and Pain points

- Static schema
- Add or drop columns
- Normal form: no arrays, nested structures
- Schema evolution: from a single value to multiple values

- New demand:
- Schema less: free to change
- Flexible structures
- XML data

Complete your sessions evaluation online at SHARE.org/AnaheimEval

DB2 XML Positioning



- Breaking relational limit
- 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.

Rule of thumb:
No need to query – use LOB/VARCHAR/VARBINARY;
Extensive queries over regular data -- use relational;
Some queries on complex or flexible data – use XML

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Common XML Usage Scenarios

- Managing XML documents
- Object persistence
- Sparse attributes
- Front-end – bridge to back-end when full normalization is overkill
- Flexible structures – many scenarios
- Flexible parameter passing
- Web services

- Rapid prototyping
- Event logging
- End-user customizable applications/generators

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Key XML Features in DB2 10

- XML schema association with XML columns (a.k.a. XML column type modifier) and automatic schema validation for insert/update
- XML schema validation using z/OS XML System Services, 100% zIIP/zAAP redirectable (retrofit to V9)
- Native XML Date and Time support and XML index support
- Basic XML sub-document update
- XML support in SQL PL STP/UDF
- Performance enhancements, including binary XML between client and server.
- CHECK DATAT for XML, LOAD/UNLOAD
- Multi-versioning for concurrency control
- Post-GA: XQuery support (PM47617, PM47618)

Complete your sessions evaluation online at SHARE.org/AnaheimEval

A Few Words about Multi-versioning format

- DB2 10 NFM, base table in UTS, XML will be in multi-versioning (MV) format
- MV format is required to support many new features for XML:
  - XMLMODIFY
  - Currently Committed Read
  - SELECT FROM OLD TABLE for XML
  - No XML locking for readers
  - Temporal data (“AS OF”)

- No automatic conversion from non-MV format to MV format (unfortunately)

Complete your sessions evaluation online at SHARE.org/AnaheimEval

XQuery Support

- Basic XQuery (PM47617, PM47618)
  - FLWOR expression
  - XQuery constructors
  - If-then-else expression
  - More XQuery functions and operators
  - Control of namespaces and whitespace in constructed XML
  - Best used in XMLQuery, replacing complex SQL/XML XMLTABLE-Constructors-XMLAGG in DB2 9
  - No top-level XQuery

Complete your sessions evaluation online at SHARE.org/AnaheimEval

FLWOR expression

- FLWOR

for $x in …, $y in …
let $z := …
where $x/c = $y/d
order by $x/c2
return …

- Select stmt

WITH y(d1, d2) AS (…)
Select …
From T1 x, y
Where x.c1 = y.d1
Group by …
Having …
Order By x.c2

Complete your sessions evaluation online at SHARE.org/AnaheimEval

XQuery Showcases – Basic FLWOR Constructs

- FLWOR - ***For, Let, Where, Order By, Return***
  - Query all items purchased in a purchase order, and order the items ascendingly on the total cost

```
SELECT XMLQUERY(
  'for $i at $pos in $po/purchaseOrder/items/item
   let $p := $i/USPrice,
       $q := $i/quantity,
       $r := $i/productName
   where xs:decimal($p) > 0 and xs:integer($q) > 0
   order by $p * $q
   return fn:concat("Item ", $pos, " productName: ", $r,
                    " price: US$", $p, " quantity: ", $q)
 '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

xquery

```
<purchaseOrder orderDate="2011-05-18">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>..</street><city>..</city><state>..</state><zip>..</zip>
  </shipTo>
  <billTo country="US"> ..</billTo>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>149.99</USPrice>
      <shipDate>2011-05-20</shipDate>
    </item>
    <item partNum="926-AA">..</item>
    <item partNum="945-ZG">..</item>
  </items>
</purchaseOrder>
```

1 2 3

Input xml doc

```
<?xml version="1.0" encoding="IBM037"?>
Item 2 productName: Baby Monitor price: US$39.98 quantity: 2
Item 1 productName: Lawnmower price: US$149.99 quantity: 1
Item 3 productName: Sapphire Bracelet price: US$178.99 quantity: 2
```

Output xml doc

Conditional expression

- ***If*** (<TestExpression>) ***Then*** (<Expression>) ***Else*** (<Expression>)
  - Query all items purchased in a purchase order, and calculate the shipping cost for each item

```
SELECT XMLQUERY(
  'let $s := $po/purchaseOrder/shipTo
   let $b := $s/@country="US" and $s/state="California"
   for $i in $po/purchaseOrder/items/item
   return (
     if ($b)
     then fn:concat($i/productName, " : shipping cost US$", 8)
     else fn:concat($i/productName, " : shipping cost US$", 12)
       )
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

XQuery

```
<purchaseOrder orderDate="2011-05-18">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>..</street>…<state>California</state><zip>..</zip>
  </shipTo>
  <billTo country="US"> ..</billTo>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName> ...
    </item>
    <item partNum=..><productName>Baby Monitor</productName>..
    </item>
    <item partNum=..><productName>Sapphire Bracelet</productName>..
    </item>
  </items>
</purchaseOrder>
```

Input xml doc

```
<?xml version="1.0" encoding="IBM037"?>
Lawnmower : shipping cost US$8
Baby Monitor : shipping cost US$8
Sapphire Bracelet : shipping cost US$8
```

Output xml doc

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Value Comparison and Node Comparison

- Value comparisons compare two atomic values - ***eq, ne, lt, le, gt, ge***
- Node comparisons compare two nodes - ***is, <<, >>***
  - Check if the product Lawnmower has the part number 872-AA

```
SELECT XMLQUERY(
  'if ($po/purchaseOrder/items/item[@partNum eq "872-AA"]
       is
     $po/purchaseOrder/items/item[productName eq "Lawnmower"])
   then "partNum 872-AA matches the product Lawnmower"
   else "partNum 872-AA DOES NOT match the product Lawnmower"
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

XQuery

```
<purchaseOrder orderDate="2011-05-18">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>..</street>…<state>California</state><zip>..</zip>
  </shipTo>
  <billTo country="US"> ..</billTo>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName> ...
    </item>
    ..
  </items>
</purchaseOrder>
```

Input xml doc

```
<?xml version="1.0" encoding="IBM037"?>
partNum 872-AA matches the product Lawnmower
```

Output xml doc

Castable and fn:avg()

- *Castable* expressions test whether a value can be cast to a specific data type – Expression ***castable as*** TargetType?
- ***fn:avg***(sequence-expression)
  - Calculate the average cost of each item

```
SELECT XMLQUERY(
  'let $cost := (
     for $i in $po/purchaseOrder/items/item
     let $p := if ($i/USPrice castable as xs:decimal)
                 then xs:decimal($i/USPrice)
                 else 0.0
     let $q := if ($i/quantity castable as xs:integer)
                 then xs:integer($i/quantity)
                 else 0
     return $p * $q)
   return fn:round(fn:avg($cost))
  '
  PASSING PODOC AS "po")
FROM PURCHASE_ORDER
WHERE POID = 1;
```

XQuery

```
<purchaseOrder orderDate="2011-05-18">
  <shipTo country="US"> .. </shipTo>
  <billTo country="US"> ..</billTo>
  <items>
    <item partNum="872-AA"> ..<quantity>1</quantity>
      <USPrice>149.99</USPrice>
    </item>
    <item partNum="926-AA"> ..<quantity>2</quantity>
      <USPrice>39.98</USPrice>
    </item>
    <item partNum="945-ZG"> ..<quantity>2</quantity>
      <USPrice>178.99</USPrice>
    </item>
  </items>
</purchaseOrder>
```

Input xml doc

(149.99*1 + 39.98*2 + 178.99*2)/3 = 195.97666667

```
<?xml version="1.0" encoding="IBM037"?>196
```

Output xml doc

Complete your sessions evaluation online at SHARE.org/AnaheimEval

XQuery Constructors

- Reconstruct all items NOT shipped yet in a purchase order

```
declare boundary-space strip;
declare copy-namespaces no-preserve, inherit;
<shipping orderDate="{$po/purchaseOrder/@orderDate}">
  <shipTo>{ let $s := $po/purchaseOrder/shipTo
              return fn:concat($s/state, " ", $s/zip, " ", $s/@country)
            }
  </shipTo>
  {for $i in $po/purchaseOrder/items/item
   where fn:not($i/shipDate castable as xs:date) or
         xs:date($i/shipDate) > fn:current-date()
   return <item partNum="{$i/@partNum}">
            <partName> {$i/productName/text()} </partName>
            { $i/quantity , $i/USPrice }
          </item>
  }
</shipping>
```

XQuery

```
<purchaseOrder orderDate="2011-05-18">
  <shipTo country="US"> ..
    <street>..</street><city>..</city><state>..</state><zip>..</zip>
  </shipTo>
  <billTo country="US"> ..</billTo>
  <items>
    <item partNum="872-AA"> ..
      <shipDate>2011-05-20</shipDate>
    </item>
    <item partNum="926-AA">..
      <shipDate>2011-05-22</shipDate>
    </item>
    <item partNum="945-ZG">
      <productName>Sapphire Bracelet</productName>
      <quantity>2</quantity>
      <USPrice>178.99</USPrice>
      <comment>Not shipped</comment>
    </item>
  </items>
</purchaseOrder>
```

Input xml doc

```
SELECT XMLQUERY('…' PASSING PO as "po")
FROM PURCHASEORDERS
WHERE …
```

```
<shipping orderDate="2011-05-18">
  <shipTo>California 95123 US</shipTo>
  <item partNum="945-ZG">
    <partName>Sapphire Bracelet</partName>
    <quantity>2</quantity>
    <USPrice>178.99</USPrice>
  </item>
</shipping>
```

Output xml doc

Tricks in Generating XML test data

- Generate from relational data using constructors

  INSERT INTO MYTABLE SELECT XMLDOCUMENT(XMLELEMENT(…. XMLAGG(…)))
  FROM …

- Repeat a document 1000 times just to create volume (applies to any other data)

```
INSERT INTO MYTABLE
WITH T(n) AS (
SELECT 1 FROM SYSIBM.SYSDUMMYU
UNION ALL
SELECT n+1 FROM T WHERE n <1000)
SELECT n, '<doc>xml testing document </doc>' FROM T;
```

1	<doc>xml testing document </doc>
2	<doc>xml testing document </doc>
3	<doc>xml testing document </doc>
4	<doc>xml testing document </doc>
5	<doc>xml testing document </doc>
..	…
1000	<doc>xml testing document </doc>

- Stuff a document with long values

```
INSERT INTO MYTABLE VALUES(1,
  XMLPARSE(DOCUMENT CLOB('<A> testing doc')
          || CLOB(REPEAT('1',32000))
          || CLOB(REPEAT('2',32000))
          || CLOB(REPEAT('3',10000))
          || CLOB('</A>')
    ) );
```

```
<A> testing doc
11111111111……..11111111
22222222222……..222222222
33333333333……333333</A>
```

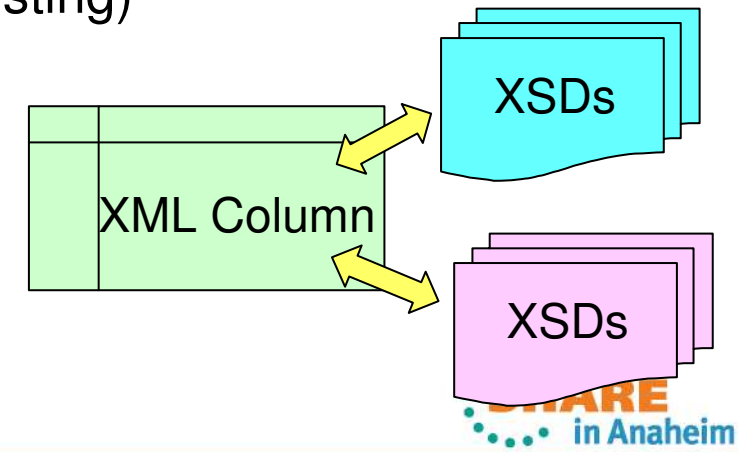
Enforcing schema conformance automatically

- XML schemas as XML column type modifier, DB2 enforces the conformance.

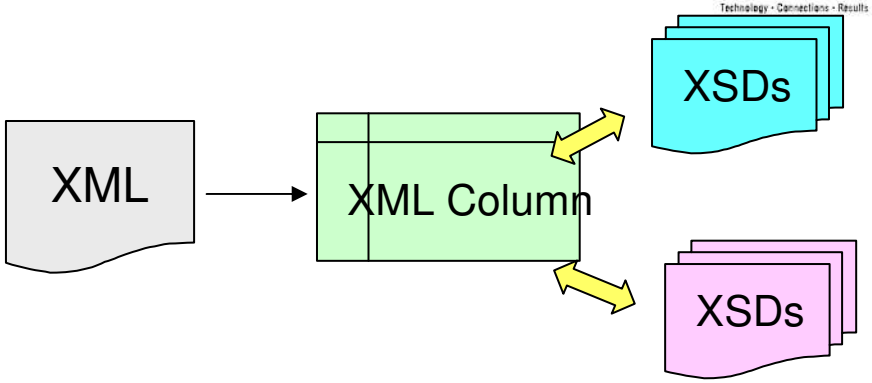
Example

- CREATE TABLE POTABLE (ID INT, ORDER XML(XMLSCHEMA ID SYSXSR.PO1, URI 'uri' LOCATION 'location'), INVOICE XML);
- ALTER TABLE POTABLE ALTER INVOICE SET DATA TYPE XML(XMLSCHEMA URI 'http://www.example.com/po' ELEMENT "invoice")

- Two ways to identify an XML schema (existing)
  - Schema name, or
  - target namespace + optional schema location



Automatic Validation



- Insert

```
INSERT INTO purchase_orders
VALUES (:id, :hv, NULL);
```

- Update

```
UPDATE purchase_orders
    SET invoice =: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.

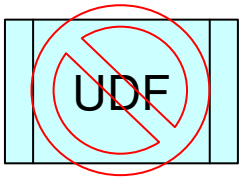
Alter XML type modifier

- Always use ALTER TABLE T1 ALTER X2 SET DATA TYPE XML(…): whole set of schemas – DB2 figures out the delta

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

- CHECK DATA – validate docs and put invalid ones in exception table

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.
You could specify preferred schema in the function.

PK90032 and PK90040 retrofit SYSIBM.DSN_XMLVALIDATE to DB2 9.

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>');
```


Complete your session evaluation online at SHAREpro.org/AnaheimEval

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 Schema Evolution Considerations

- Compatible changes: e.g. additional optional elements and attributes
  - Use new schema or schema versions (latest will be picked)
- Incompatible changes: mandatory new elements or attributes, different data types, different schema target namespace
  - DB2 can accommodate multiple schema versions or schemas
  - DSN_XMLVALIDATE can find the right schema versions given target namespace and schema location in instance documents
  - Application has to deal with document differences
    - Cardinality
    - Data types
    - Schemas
  - Conversions of old data to new data

Sub-document Update

- SQL UPDATE statement with XMLMODIFY() function
- Only simple update: insert, replace, delete from XQuery update facility

Example

Insert a ship date into purchase order

```
UPDATE POTABLE SET ORDER = XMLMODIFY
('insert node $n after /purchaseorder/orderdate', :shipdate as “n”)
WHERE POID = :poid;
```

Change to a new address for bill to:

```
UPDATE POTABLE SET INVOICE = XMLMODIFY
( 'replace node /invoice/billto/address with $newaddr',
    XMLPARSE(DOCUMENT :newaddress) as "newaddr")
WHERE POID = '12345';
```

More on sub-document update

- XMLMODIFY can only be used in RHS (Right-Hand Side) 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 records 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.
- Requires multi-versioning (MV) format for XML table space.

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 $n
  after  /person/nickName', xmlparse(document
 '<ename>Joe.Smith@de.ibm.com</ename>') as  "n") ;
```

Insert Operation	Resulting XML document
'insert node $n **into** /person', XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "n") (nondeterministic position, DB2 treats it as last)	<person> <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> **<ename>Joe.Smith@de.ibm.com</ename>** </person>
'insert node $n **as last into** /person', XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "n")	<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>
```

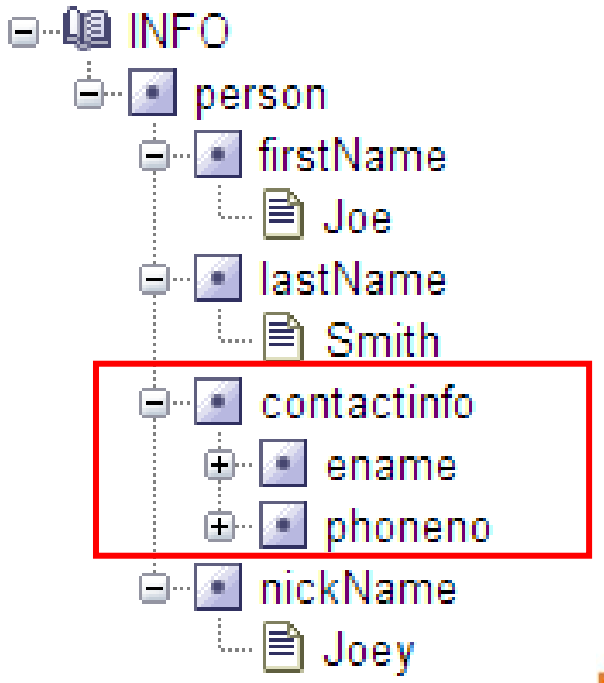
Insert Operation	Resulting XML document
'insert node $n **as first into** /person', XMLPARSE(document '<ename>Joe.Smith@de.ibm.com</ename>') as "n")	<person> **<ename>Joe.Smith@de.ibm.com</ename>** <firstName>Joe</firstName> <lastName>Smith</lastName> <nickName>Joey</nickName> </person>
'insert node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

Insert expression (3 of 3)

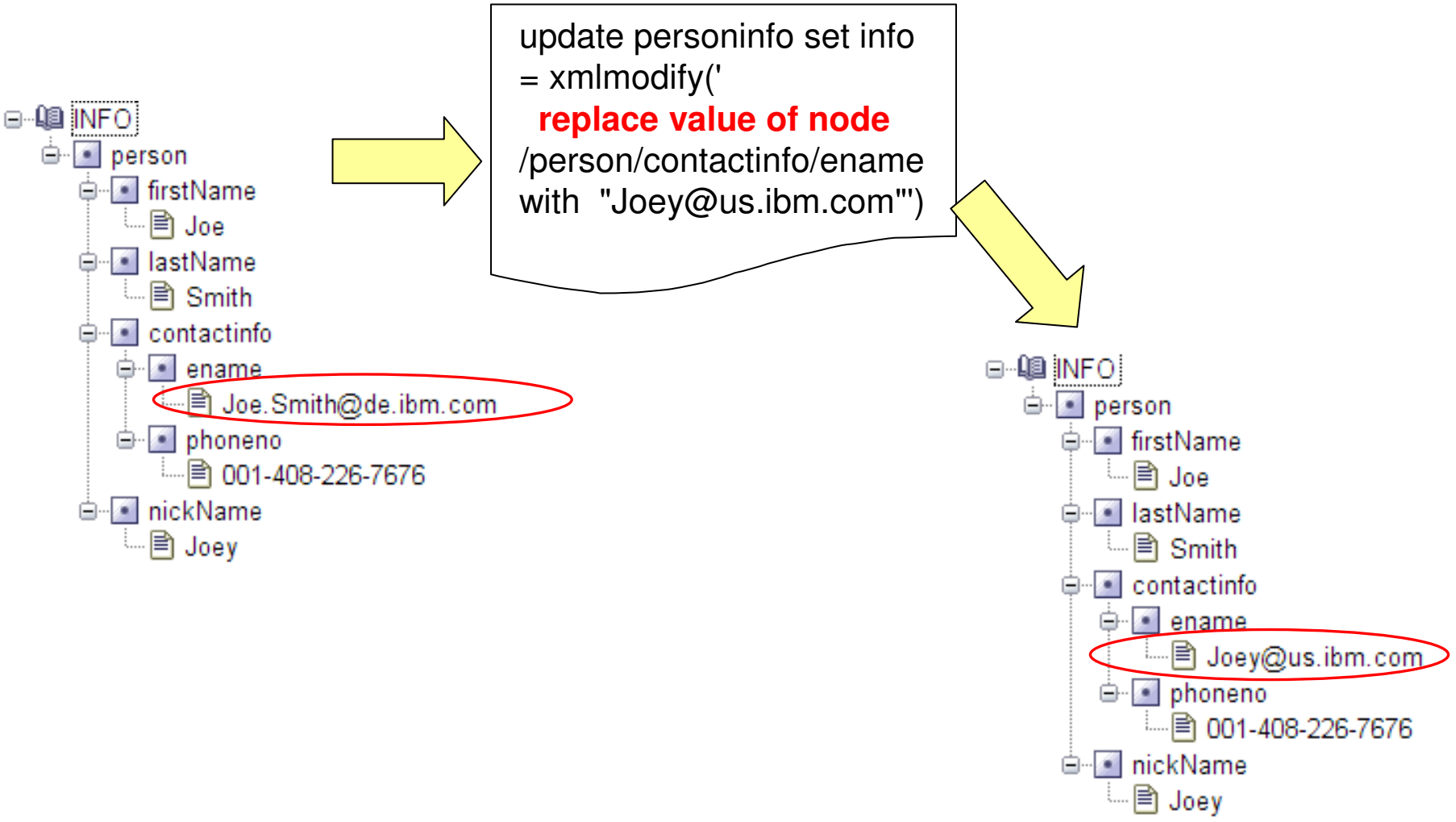
- Insertion of sequence of elements also possible
- Use same keywords as introduced before

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

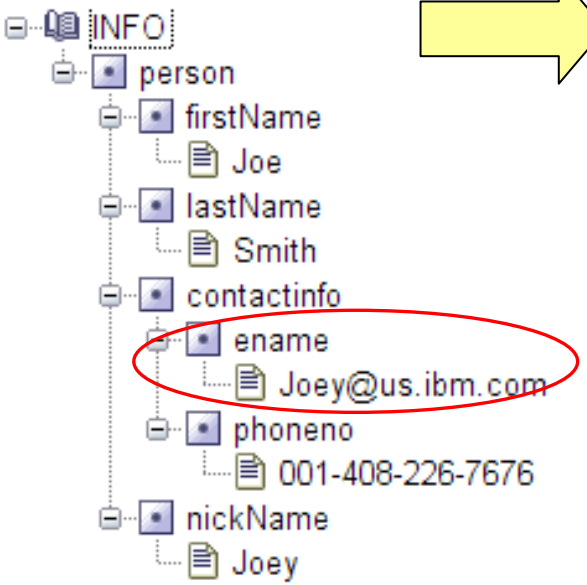
```
update personinfo set info = xmlmodify('
  insert node $n
before /person/nickName',
xmlparse(document
'<contactinfo>
<ename>Joe.Smith@de.ibm.com</ename>
<phoneno>001-408-226-7676</phoneno>
</contactinfo>') as "n") ;
```



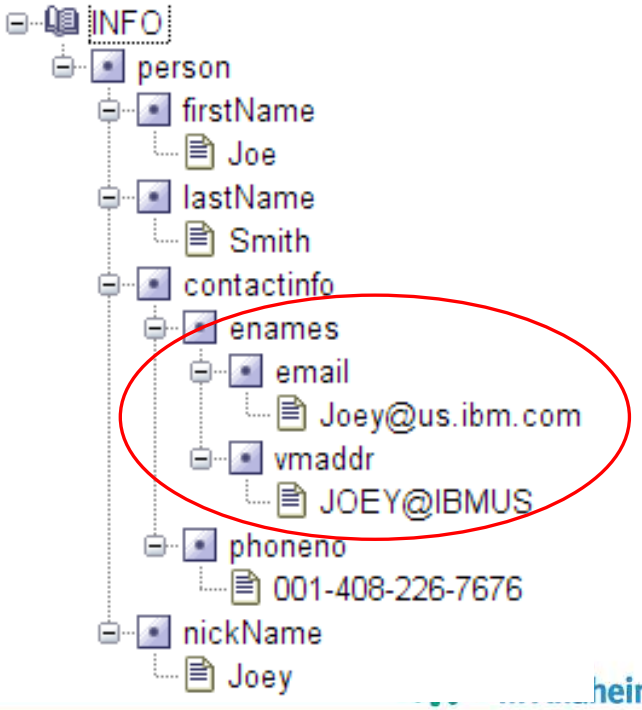
Replace expression (1 of 2)



Replace expression (2 of 2)

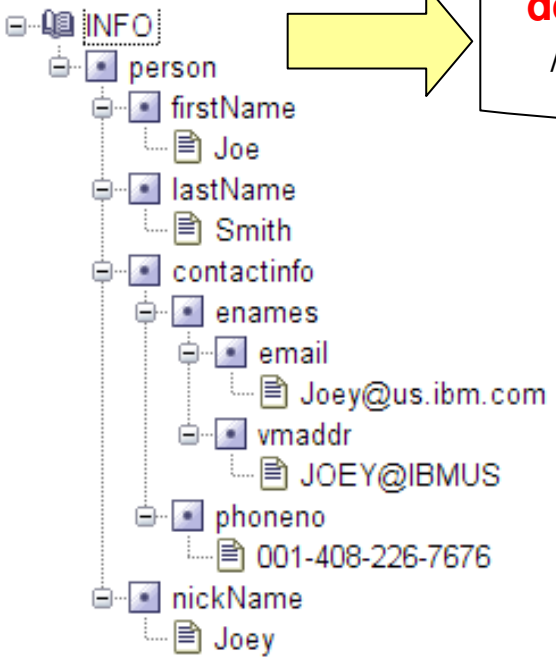



```
update personinfo set info = xmlmodify('
  replace node /person/contactinfo/ename
  with  $n', XMLPARSE(document
  '<enames>
    <email>Joey@us.ibm.com</email>
     <vmaddr>JOEY@IBMUS</vmaddr>
  </enames>') as "n");
```

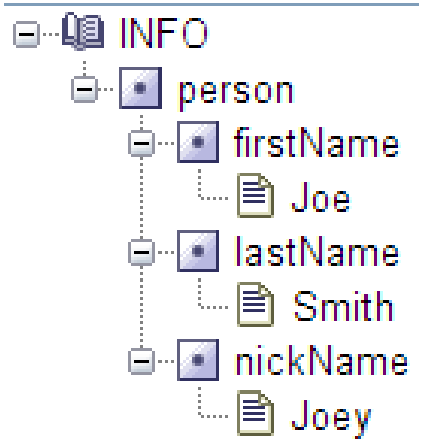


Delete expression

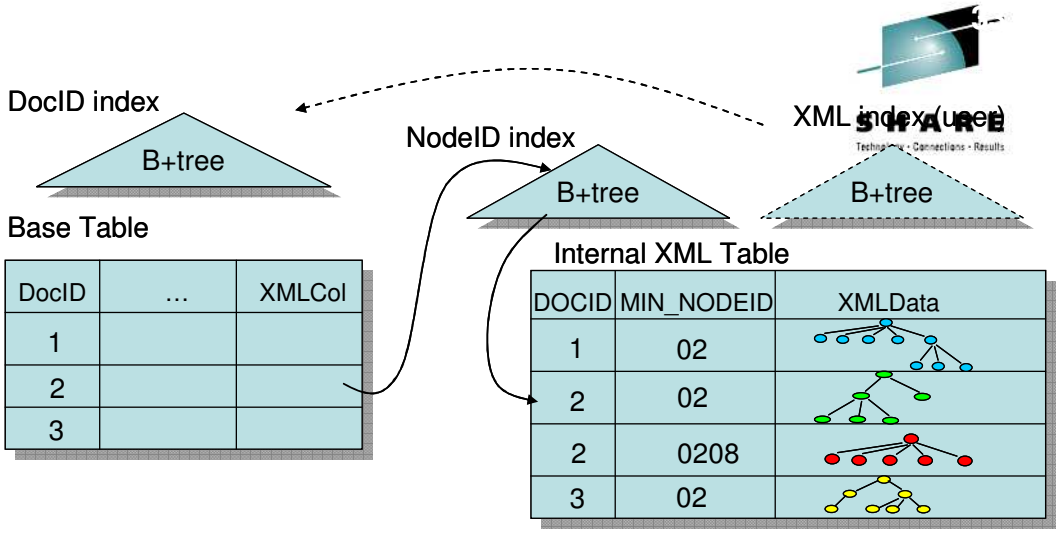
- Delete can be used to delete nodes from a node sequence




```
update personinfo set info = xmlmodify('
  delete node
    /person/contactinfo')
```



CHECK DATA for XML

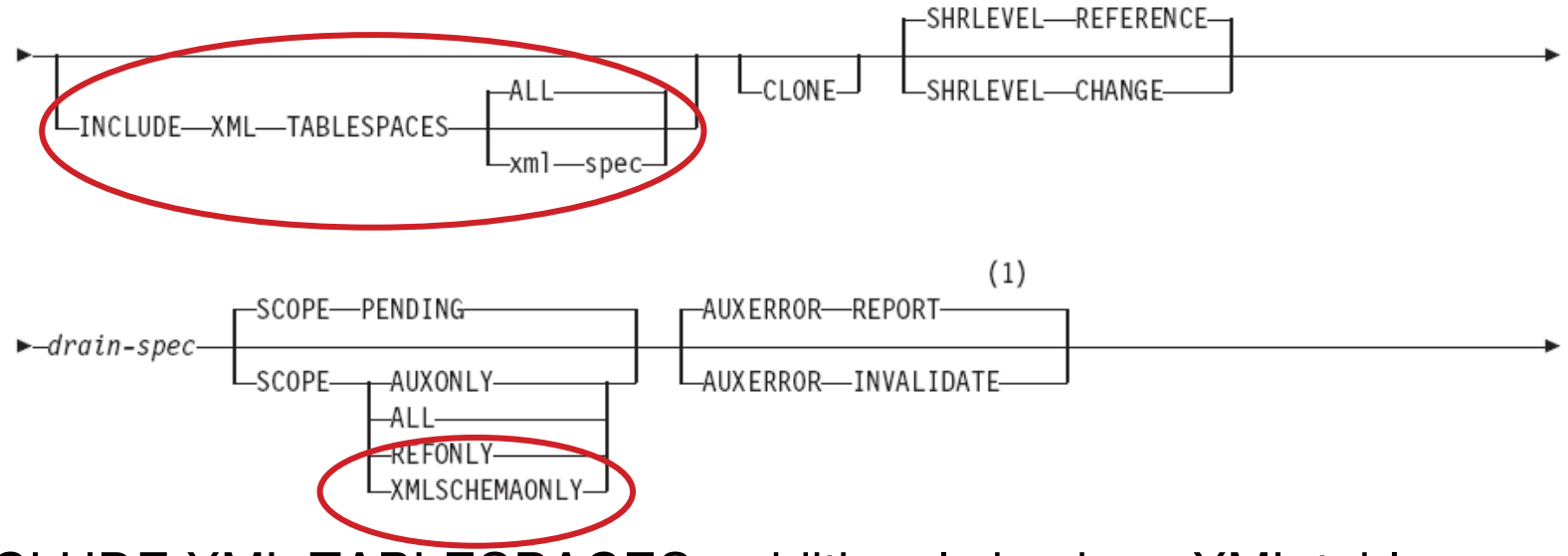


- **DB2 9**
  - CHECK DATA utility does not cover inconsistency inside XML data
  - NODEID index consistency with XML TS (CHECK INDEX)
  - Base table consistency with NODEID index (CHECK DATA)
- **DB2 10 CHECK DATA for XML** does:
  - Base table (referential) consistency with NODEID index (CHECK DATA)
  - 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 !!

Complete your sessions evaluation online at SHARE.org/AnaheimEval

CHECK DATA: new for XML



- INCLUDE XML TABLESPACES: additional check on XML table spaces, including
  - (1) structural consistency (including NODEID Index checking);
  - (2) schema validation.
- SCOPE XMLSCHEMAONLY: schema validation only
- Must specify AUXERROR INVALIDATE or XMLERROR INVALIDATE to replace existing docs with validated docs

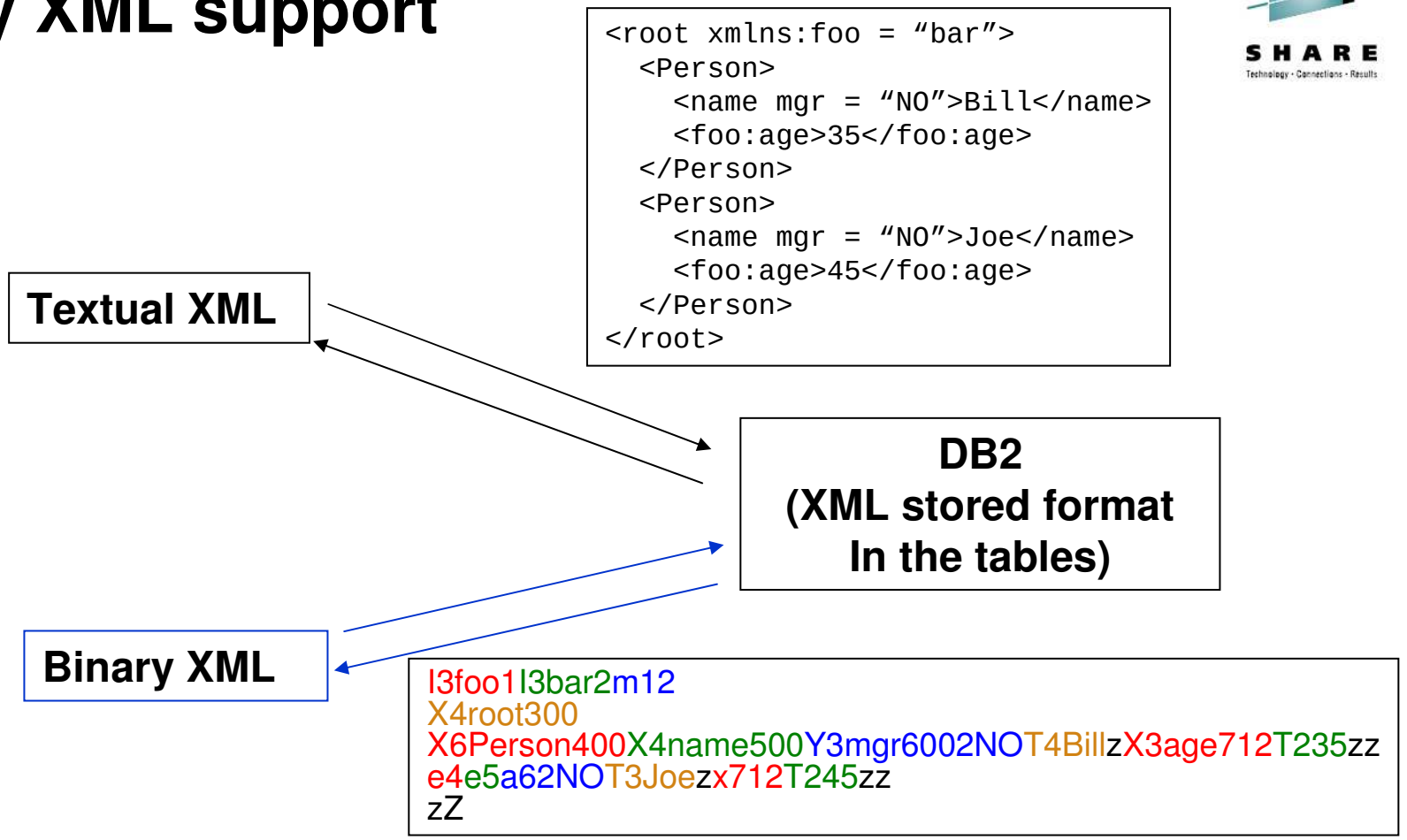
CHECK DATA for XML: Common scenarios

Options	Referential consistency	Structural consistency	Schema validation	To change docs ✸
SHRLEVEL CHANGE SCOPE ALL	Yes	-	-	
SHRLEVEL CHANGE SCOPE ALL INCLUDE XML TABLESPACES ALL	Yes	Yes	-	REPAIR
SHRLEVEL REFERENCE SCOPE PENDING INCLUDE XML TABLESPACES ALL	Yes	-	Yes if pending	AUXERROR INVALIDATE or XMLERROR INVALIDATE
SHRLEVEL REFERENCE SCOPE ALL INCLUDE XML TABLESPACES (…) XMLSCHEMA	Yes	Yes	Yes	AUXERROR INVALIDATE or XMLERROR INVALIDATE
SHRLEVEL REFERENCE SCOPE XMLSCHEMAONLY	-	-	Yes	XMLERROR INVALIDATE or AUXERROR INVALIDATE

✸ To change docs: delete corrupted, invalidate base table rows, replace with validated docs, or move to exception table.

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Binary XML support



-- 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

Binary XML 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

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Binary XML 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);

// or: 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();
```

Binary XML Usage: UNLOAD XML data in SYSREC

- To unload XML data directly to output record, specify XML as the field type and use BINARYXML for better performance
- No 32KB size limit to unload to SYSREC (VBS data set)
- Specify NOPAD SPANNED YES, and put XML and LOB at the end of field spec

```
UNLOAD TABLESPACE DSN00009.MYTABLE
    PUNCHDDN SYSPUNCH UNLDDN SYSREC NOPAD SPANNED YES
FROM TABLE ADMF001.MYTABLE
          (
           I          POSITION(*) INT
          ,X          POSITION(*) XML BINARYXML
          )
```


Complete your sessions evaluation online at SHARE.org/AnaheimEval

XML Support in SQL PL STP/UDF

- In V9 XML data type cannot be used as parameters, SQL variables, and return type – workaround: use LOBs
- XML 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.)

Decomposition into multiple tables w/ SQL PL proc

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

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

```
Tables: TAB1( C1, C2)
        TAB2(C3, C4)
Document:
 <doc>
   <head>
    <row>
      <C1>1</C1>
      <C2>AAA</C2>
    </row>
   </head>
   <body>
   <row>
     <C3>10</C3>
     <C4>XXXX</C4>
   </row>
   <row>
     <C3>20</C3>
     <C4>YYYYY</C4>
   </row>
  </body>
</doc>
```

Parse once and decompose into multiple tables
If using Java caller, document could be parsed into binary XML in the client

Merging XML into relational data using SQL PL (Gathering Data Changes)

- Use XML for data update, such as code, not just for simple decomposition with insert
- MERGE does not take table source yet. Use SQL PL and cursor for source.

```
CREATE PROCEDURE APPLYDIFF(IN CHANGE XML)
LANGUAGE SQL
 BEGIN
    DECLARE XID INT;
    DECLARE XNAME VARCHAR(20);
    DECLARE XPRICE DECIMAL(10,2);
    DECLARE SQLCODE INT;
    DECLARE C1 CURSOR FOR
     SELECT ID, NAME, PRICE FROM XMLTABLE('/ROWS/ROW' PASSING CHANGE
          COLUMNS    ID  INT, NAME  VARCHAR(20), PRICE DECIMAL(10,2)) X;
     OPEN C1;
     LOOP1: LOOP
          FETCH C1 INTO XID, XNAME, XPRICE;
          IF SQLCODE <> 0 THEN LEAVE LOOP1; END IF;
          MERGE INTO MYCODETABLE USING (VALUES(XID, XNAME, XPRICE))
                AS N(XID, XNAME, XPRICE)
               ON (MYCODETABLE.ID = N.XID)
          WHEN MATCHED THEN UPDATE SET NAME=XNAME, PRICE=XPRICE
          WHEN NOT MATCHED THEN INSERT VALUES(XID, XNAME, XPRICE);
        END LOOP LOOP1;
        CLOSE C1;
END
```

Table: MYCODETABLE(ID, NAME, PRICE)
CHANGE Document:

```
<ROWS>
 <ROW>
  <ID>1003</ID>
  <NAME>GALAXY</NAME>
  <PRICE>388.00</PRICE>
 </ROW>
 <ROW>
  <ID>1004</ID>
  <NAME>IdeaPad</NAME>
  <PRICE>300.00</PRICE>
 </ROW>
</ROWS>
```

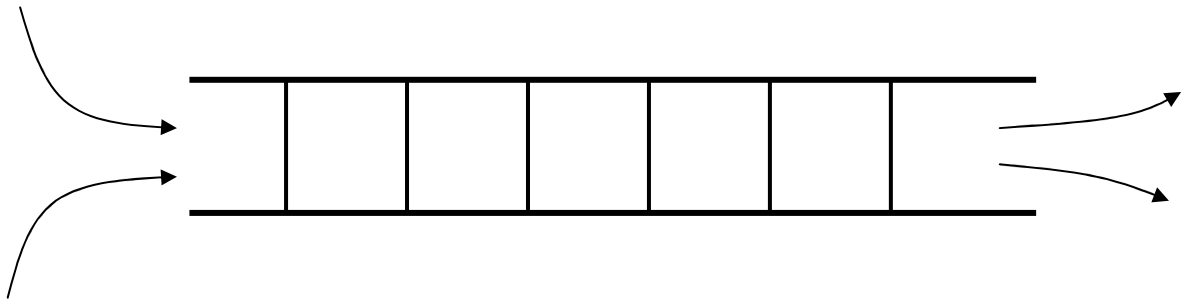
ID	Name	Price
1001	iPad – 16GB	499.99
1002	iPhone 4-16GB	199.99
1003	Galaxy	580.00

ID	Name	Price
1001	iPad – 16GB	499.99
1002	iPhone 4-16GB	199.99
1003	Galaxy	388.00
1004	IdeaPad	300.00

Simulating a Queue using SELECT FROM DELETE

- Using certain criteria to delete from a table, such as minimum sequence number. And select the deleted rows back.
- SELECT ID, XDOC, …
  FROM OLD TABLE(
    DELETE FROM MYTABLE
    WHERE ID = (SELECT MIN(ID) FROM MYTABLE
                WHERE …))

- This requires new multi-versioning (MV) format for XML data.



XML Index Flexibility

- Multiple indexes on one XML column
- Multiple indexes on the same data with different data types
- 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.
- Restriction: Index key value cannot span multiple rows. Always safe to index leaf nodes with short values.

Dynamic Typing

- -- The same value can be used as numeric or string
- SELECT XMLQUERY('/catalog/category/product[size >= 9]/name' PASSING XCAT)
- FROM MYCATALOG
- WHERE N=1#

- SELECT XMLQUERY('/catalog/category/product[size = "L" or size = "XL"]/name' PASSING XCAT)
- FROM MYCATALOG
- WHERE N=1#

- -- Or they can be used as both numeric or string in the same query
- SELECT XMLQUERY('/catalog/category/product[size >= 9 or size = "L" or size = "XL"]/name' PASSING XCAT)
- FROM MYCATALOG
- WHERE N=1#

XML Index for Non-Existential Predicate (missing an element or attribute) (PK80732/PK80735)

```
XMLEXISTS('/purchaseOrder/items/item[fn:not(shipDate)]'
PASSING XMLPO);
```

Internal implementation

```
XMLEXISTS('/purchaseOrder/items/item[fn:exists(shipDate)="F"]'
PASSING XMLPO);
```

XML Index scan returns DocID = 1002

```
CREATE INDEX IDX1 ON PurchaseOrders(XMLPO)
 GENERATE KEY USING XMLPATTERN
 '/purchaseOrder/items/item/fn:exists(shipDate)'
 AS SQL VARCHAR(1);
```

Key	DocID	…
F	1002	
T	1001	
T	1003	

DOCID	XMLPO	
1001	<purchaseOrder orderDate="2009-10-05"> . . . <items> <item partNum="872-AA"> <desc>Lawnmower</desc> <quantity>1</quantity> <shipDate>2010-05-01</shipDate> </item> </items> </purchaseOrder>	. . .
1002	<purchaseOrder orderDate="2009-10-05"> . . . <items> <item partNum="926-AA"> <desc>Baby Monitor</desc> <quantity>1</quantity> </item> <items> </purchaseOrder>	. . .
1003	<purchaseOrder orderDate="2009-10-04"> . . . <items> <item partNum="957-BB"> <desc>Air Conditioner</desc> <quantity>2</quantity> <shipDate>2010-01-01</shipDate> </item> <items> </purchaseOrder>	. . .

XML Index for Existential Predicate (containing an element or attribute)

```
XMLEXISTS('/purchaseOrder/items/item[fn:exists(shipDate)]'
PASSING XMLPO);
```

Internal Implementation

```
XMLEXISTS('/purchaseOrder/items/item[fn:exists(shipDate)="T"]'
PASSING XMLPO);
```

XML Index scan returns DocID = 1001, 1003

```
CREATE INDEX IDX1 ON PurchaseOrders(XMLPO)
 GENERATE KEY USING XMLPATTERN
 '/purchaseOrder/items/item/fn:exists(shipDate)'
 AS SQL VARCHAR(1);
```

Key	DocID	…
F	1002	
T	1001	
T	1003	

DOCID	XMLPO
1001	<purchaseOrder orderDate="2009-10-05"> . . . <items> <item partNum="872-AA"> <desc>Lawnmower</desc> <quantity>1</quantity> <shipDate>2010-05-01</shipDate> </item> </items> </purchaseOrder>
1002	<purchaseOrder orderDate="2009-10-05"> . . . <items> <item partNum="926-AA"> <desc>Baby Monitor</desc> <quantity>1</quantity> </item> <items> </purchaseOrder>
1003	<purchaseOrder orderDate="2009-10-04"> . . . <items> <item partNum="957-BB"> <desc>Air Conditioner</desc> <quantity>2</quantity> <shipDate>2010-01-01</shipDate> </item> <items> </purchaseOrder>

Case-insensitive Search with XML indexes

- Use fn:upper-case() in the XPath predicate for Case-insensitive comparison.
- Use fn:upper-case() in the XMLPattern to create eligible XML index.

```
SELECT ORD_NO
FROM PurchaseOrders
WHERE XMLEXISTS('/purchaseOrder/items/item[fn:upper-case(desc)=‘BABY
MONITOR’]‘
PASSING XMLPO);
```

```
CREATE INDEX IDX2 ON PurchaseOrders(XMLPO) Generate Keys
Using XMLpattern '/purchaseOrder/items/item/desc/fn:upper-case(.)'
AS SQL VARCHAR(20);
```

Pattern matching for strings using regular expressions

- Three important functions in XPath using regular expressions:
  fn:matches, fn:replace, fn:tokenize
- fn:replace(): to replace some patterns with a replacement string.
  For example, strip spaces and dashes: fn:replace($x, "[\-]", "")
  Remove non-digit characters: fn:replace($x, "[^0-9]+", "")

- fn:matches(): check if a string matches a pattern
  For example: all digits starting with "4"
  fn:matches($y, "^4[0-9]+$")

- fn:tokenize(): break a string into a sequence
  For example, split a sequence of keywords separated by a space:
  fn:tokenize($x, " ")

Returning Credit Card Type according to the number

- This example contains a preview of XQuery FLWOR expressions and if-then-else

```
SELECT XMLQUERY('let $y := fn:replace($x, "[ \-]", "") return
if (fn:matches($y, "^4[0-9]{12}([0-9]{3})?$")) then "Visa"
else if (fn:matches($y, "^5[1-5][0-9]{14}$")) then "Master"
else if (fn:matches($y,"^3[47][0-9]{13}$")) then "AmEx"
else if (fn:matches($y, "^6(011|5[0-9]{2})[0-9]{12}$")) then "Discover"
else  "Unknown"'
   PASSING '4123 456 789012' as "x")
FROM sysibm.sysdummyu #
```

This returns “Visa".

Returning items from a string

- Items separated by space or comma, can be used to pass an array (use XMLAGG to construct it)
- Using space: SELECT * FROM XMLTABLE('fn:tokenize($x, " ")' PASSING '12 34 56 78' as "x" COLUMNS A INT PAHT '.') as XT

```
A
12
34
56
78
```

- Using comma: SELECT * FROM XMLTABLE('fn:tokenize($x, ",")' PASSING '12 AB, 34 CD, 56, 78' as "x" COLUMNS B VARCHAR(10) PATH '.') as XT

```
B
12 AB
 34 CD
 56
 78
```

Flexible Parameter Passing using XML

- Passing flexible structures, or arrays

```
CALL  MyProc1(XMLELEMENT(NAME "Array",
    XMLFOREST('1001' as "prodID",
                '1003' as "prodID",
                '1080' as "prodID",
                '1088' as "prodID")) );
In MyProc1(x):
SELECT …
FROM XMLTABLE('Array' passing x
    column prodID int PATH 'prodID'), …
```

1001
1003
1080
1088

- Passing a flexible structure

```
CALL myProc2(XMLELEMENT(NAME "Struct",
      XMLFOREST('1001' as "id", 'prod_name' as "pname", 123.45 as
   "qty"));

In myProc2(x):
SELECT …
FROM XMLTABLE('Struct' passing x
    column id int PATH 'id', pname varchar(20) PATH 'pname', qty
   decfloat PATH 'qty'), …
```

1001	'prod_name'	123.45

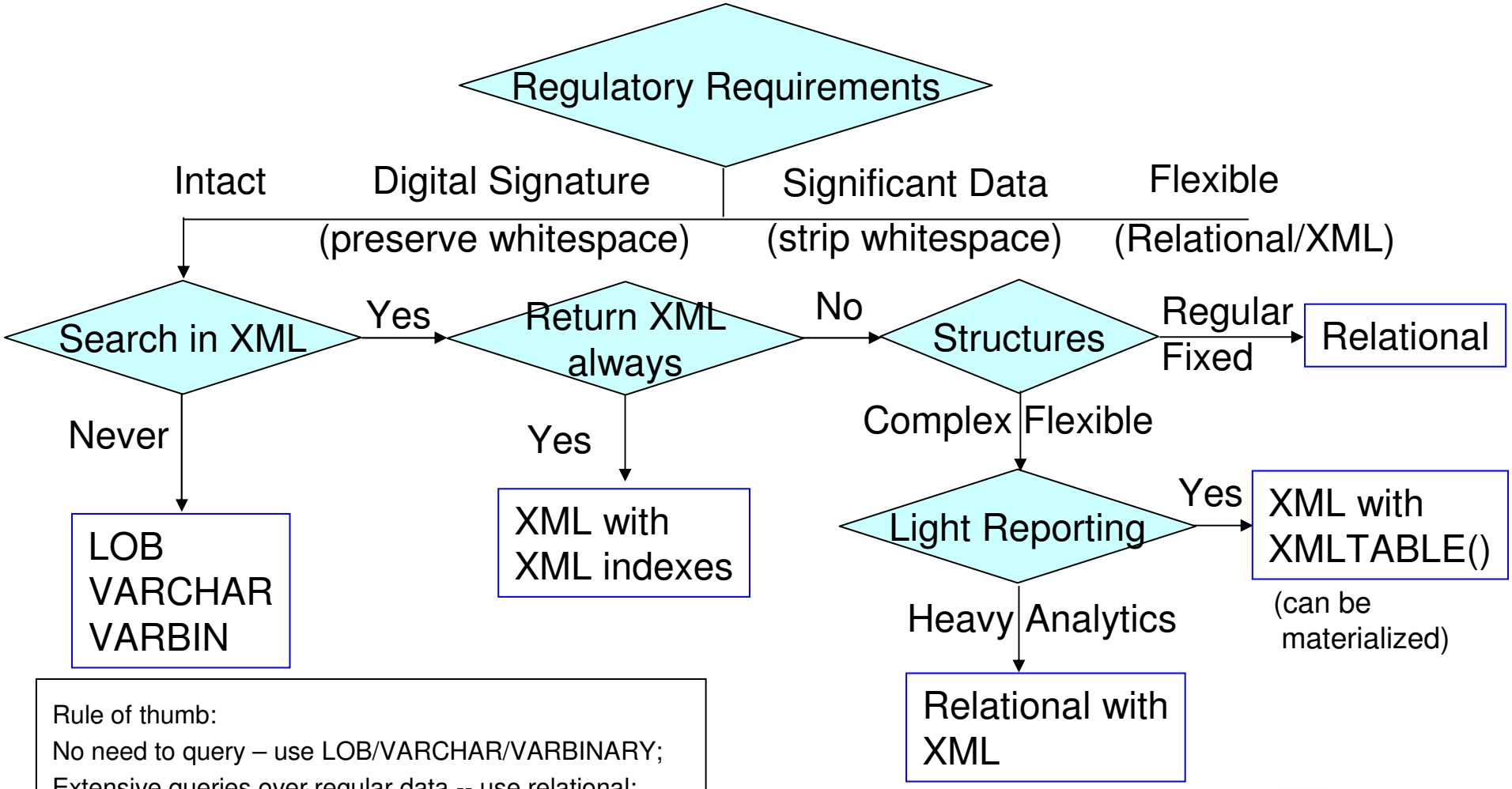
Complete your sessions evaluation online at SHARE.org/AnaheimEval

Simulating Arrays and Structures

- Flexibility of XML allows for simulation of arrays and structures.
- Regular structures can be represented as a table using XMLTABLE function.

User View	XML Representation	XPath to refer to the values	XMLTable
Products ID: 196 NAME: z QTY: 100 --- ID: P7 NAME: P QTY: 10000 --- ID: I5 NAME: i QTY: 1000	<products> <produc> <id>196</id> <name>z</name> <qty>100</qty> </product> <product> <id>P7</id> <name>P</name> <qty>10000</qty> </product> … </products>	/products/product[1]/id /products/product[1]/name /products/product[1]/qty /products/product[2]/id /products/product[2]/name /products/product[2]/qty …	XMLTABLE('/products/product' PASSING XDATA COLUMNS "id" VARCHAR(10), "name" VARCHAR(20), "qty" DECIMAL(10,2))

Design Decision making: XML input => storage



Rule of thumb:
No need to query – use LOB/VARCHAR/VARBINARY;
Extensive queries over regular data -- use relational;
Some queries on complex or flexible data – use XML

Complete your sessions evaluation online at SHARE.org/AnaheimEval

Extracting values from XML for Hybrid Store using Triggers

- CUST(ID, NAME, CITY, ZIP, INFO): extract NAME, CITY, ZIP from INFO (XML)
-
```
CREATE TRIGGER ins_cust
 AFTER INSERT ON cust
 REFERENCING NEW AS newrow
 FOR EACH ROW MODE DB2SQL
 BEGIN ATOMIC
     update cust
     set (name, city, zip) =
         (select X.name, X.city, X.zip
          from cust, XMLTABLE('customerinfo' PASSING CUST.INFO
          COLUMNS
             name varchar(30)  PATH 'name',
             city varchar(20)  PATH 'addr/city',
             zip  varchar(12)  PATH 'addr/pcode-zip') as X
           where cust.id = newrow.id
           )
     where cust.id = newrow.id;
 END #
```

Usage Scenario: DB2 as Web Services Consumer

- Invoke light-weight SOAPHTTP UDF from SQL, consume response in the same query conveniently.

- Tooling support for DB2 in RAD
  - Create a SQL UDF specifically for a WS operation from WSDL
  - Creates 'wrapper' SQL UDF to hide XML/relational mapping.

How much is EUR1000 worth in USD?

```
SELECT  1000 *
XMLCAST( XMLQUERY('$d//*:ConversionRateResult' PASSING
 XMLPARSE (DOCUMENT
  DB2XML.SOAPHTTPNV(
  'http://www.webservicex.net/CurrencyConvertor.asmx',
  'http://www.webserviceX.NET/ConversionRate',
  '<?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>
    <ConversionRate xmlns="http://www.webserviceX.NET/">
      <FromCurrency>EUR</FromCurrency>
      <ToCurrency>USD</ToCurrency>
    </ConversionRate>
  </soap:Body>
</soap:Envelope>')
) AS "d") AS DECIMAL(10,5))
FROM SYSIBM.SYSDUMMYU#
```


Complete your sessions evaluation online at SHARE.org/AnaheimEval

Encapsulating a Web Service

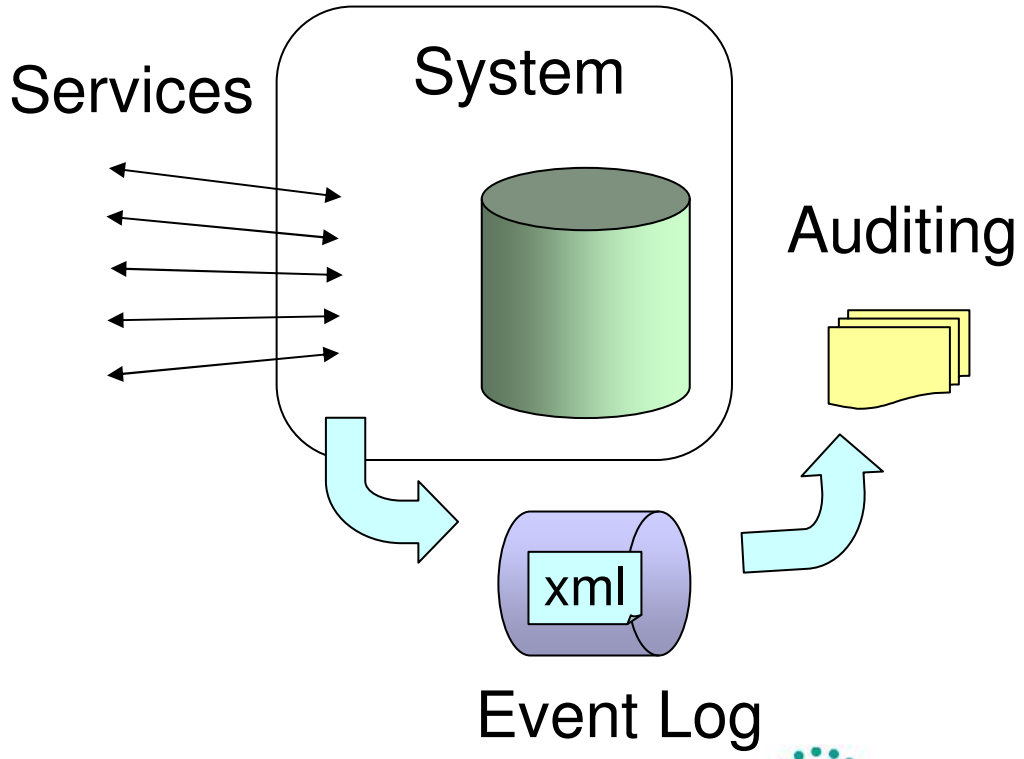
```
CREATE FUNCTION STOCKQUOTE(INSYMBOL VARCHAR(4))
 RETURNS TABLE (Symbol VARCHAR(4), LAST DECIMAL(6,2), …)
 LANGUAGE SQL
 NOT DETERMINISTIC
RETURN
SELECT *
FROM XMLTABLE('/StockQuotes/Stock' PASSING
 XMLPARSE(DOCUMENT XMLCAST(
  XMLQUERY('declare namespace soap="http://schemas.xmlsoap.org/soap/envelope/";
 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>' || INSYMBOL ||'</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;
```

Scenario: XML Event Log for Auditing

- Event log keeps records of diverse events, including:
  - TIMESTAMP, SESSIONID, USERID, REQUEST, RESULTSTATUS, SOURCE, EVENTTYPE, ACTION, RESPONSE etc.
- An XML column can contain the diverse data specific to each event type.
- Many XML indexes can be created to serve diverse query needs with good performance.

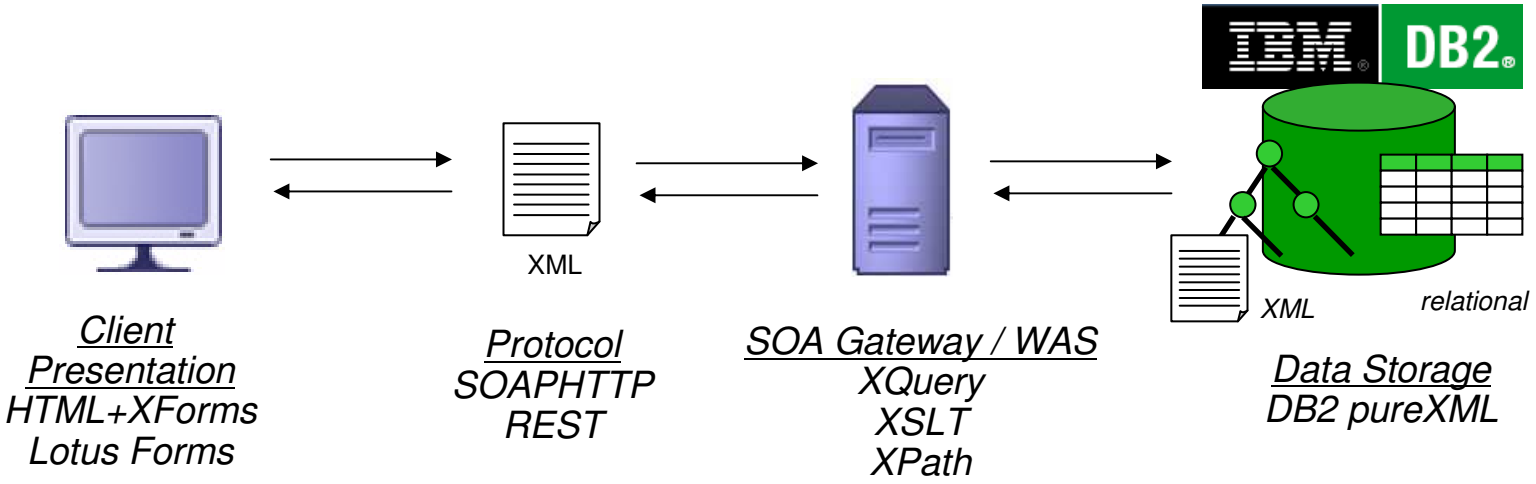


Scenario: End-user Customizable Apps

- ISVs or large enterprise ITs need to serve diverse needs.
- Information such as product spec, customer info varies drastically.
- Use one XML column to contain the customizable data.

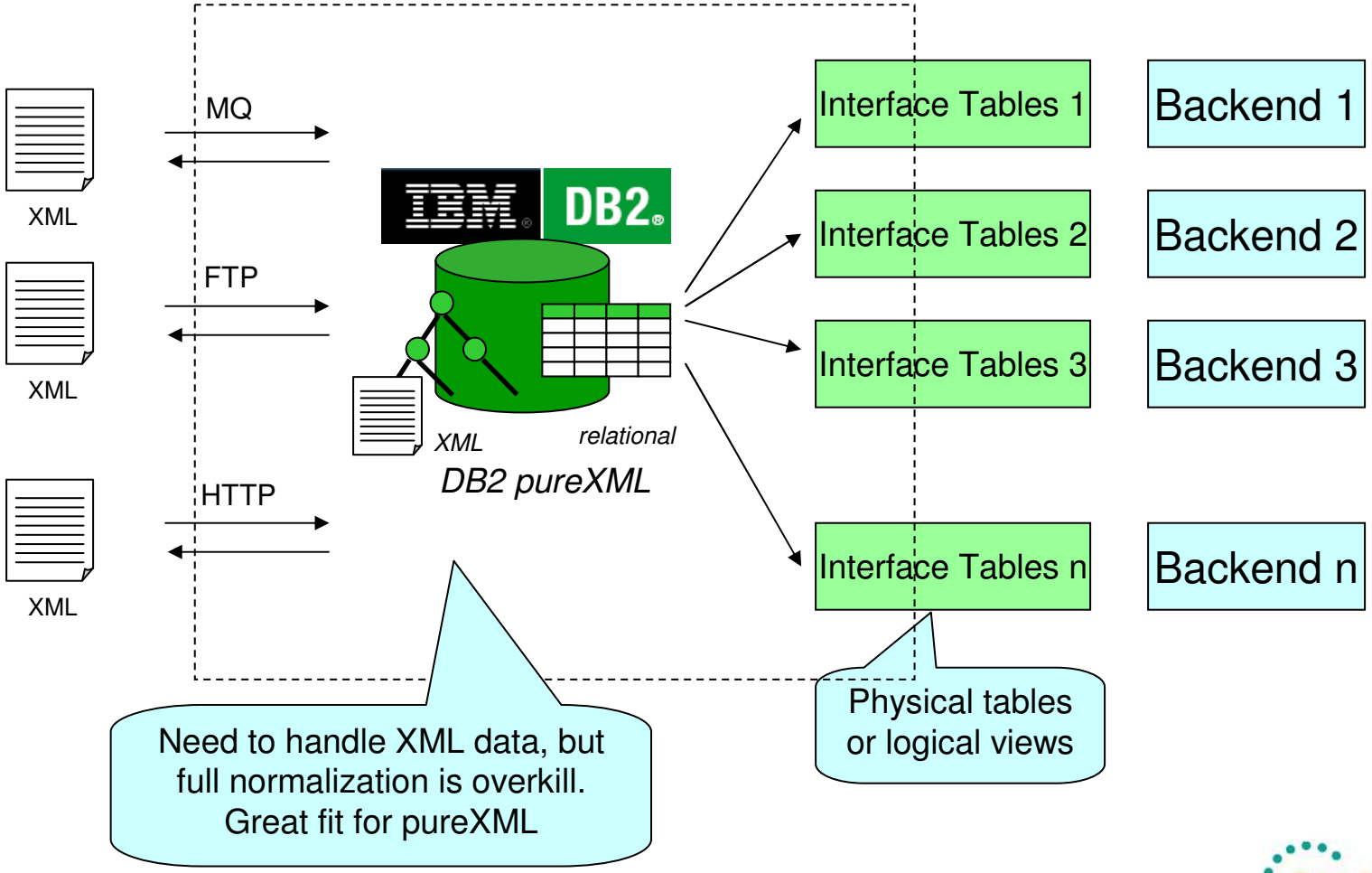
User View	XML Representation	XPath to refer to the values	Display
product spec unit unit price Color Size Battery Voltage Standards	<product> <spec> <unit>box</unit> <unitprice>8.5</unitprice> <color>black</color> <size>8 x 10</size> <standards> <std>ISO910001</std> </standards> </spec> </product>	/product/spec/unit /product/spec/unitprice /product/spec/color /product/spec/size /product/spec/battery /product/spec/voltage /product/spec/standards/std	Using XSLT or XForms or your own framework based on XHTML

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.

XML as Front to Backend/Core Systems



Summary

- XML breaks relational limit, and is surprisingly versatile.
- DB2 10 improves pureXML features significantly
- Flexibility is the key benefit, a paradigm shift
- A set of great XML features to help reduce complexity and cost, time to market
- XML programming paradigm: productivity and developer experiences.

Thank you!