IBM Software Group

Leveraging pureXML in DB2 9 for z/OS and LUW

Guogen (Gene) Zhang, gzhang@us.ibm.com
GSE, May 29, 2007

DB2 Information Management Software

ON DEMAND BUSINESS

Agenda

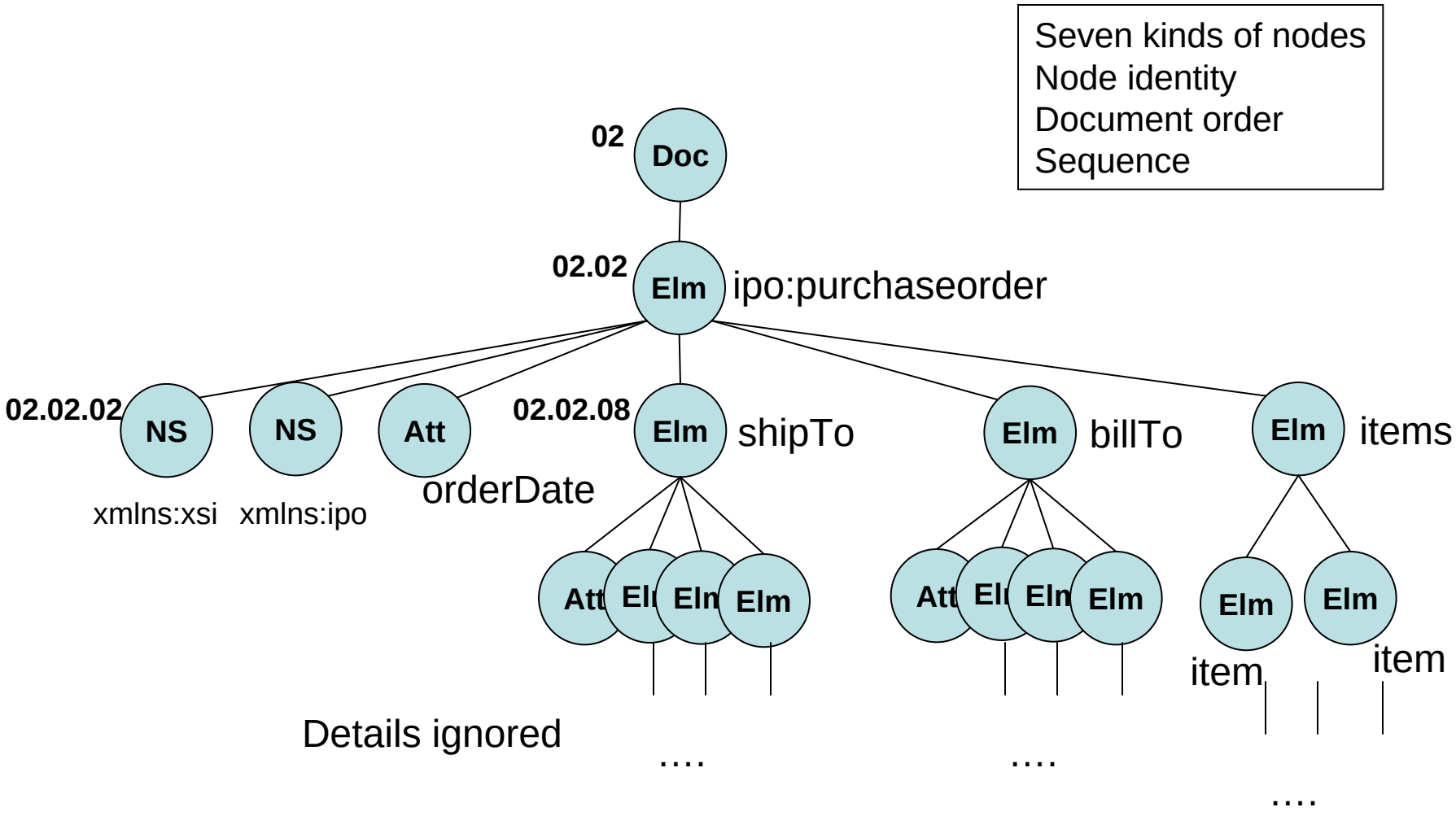
- Overview: what you can do with pureXML in DB2 9
- Some usage scenarios
- Business value: comparison with existing approaches
- pureXML Common Features
- Platform Similarities and Differences
- Future Directions

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 (XQuery Data Model, XDM)



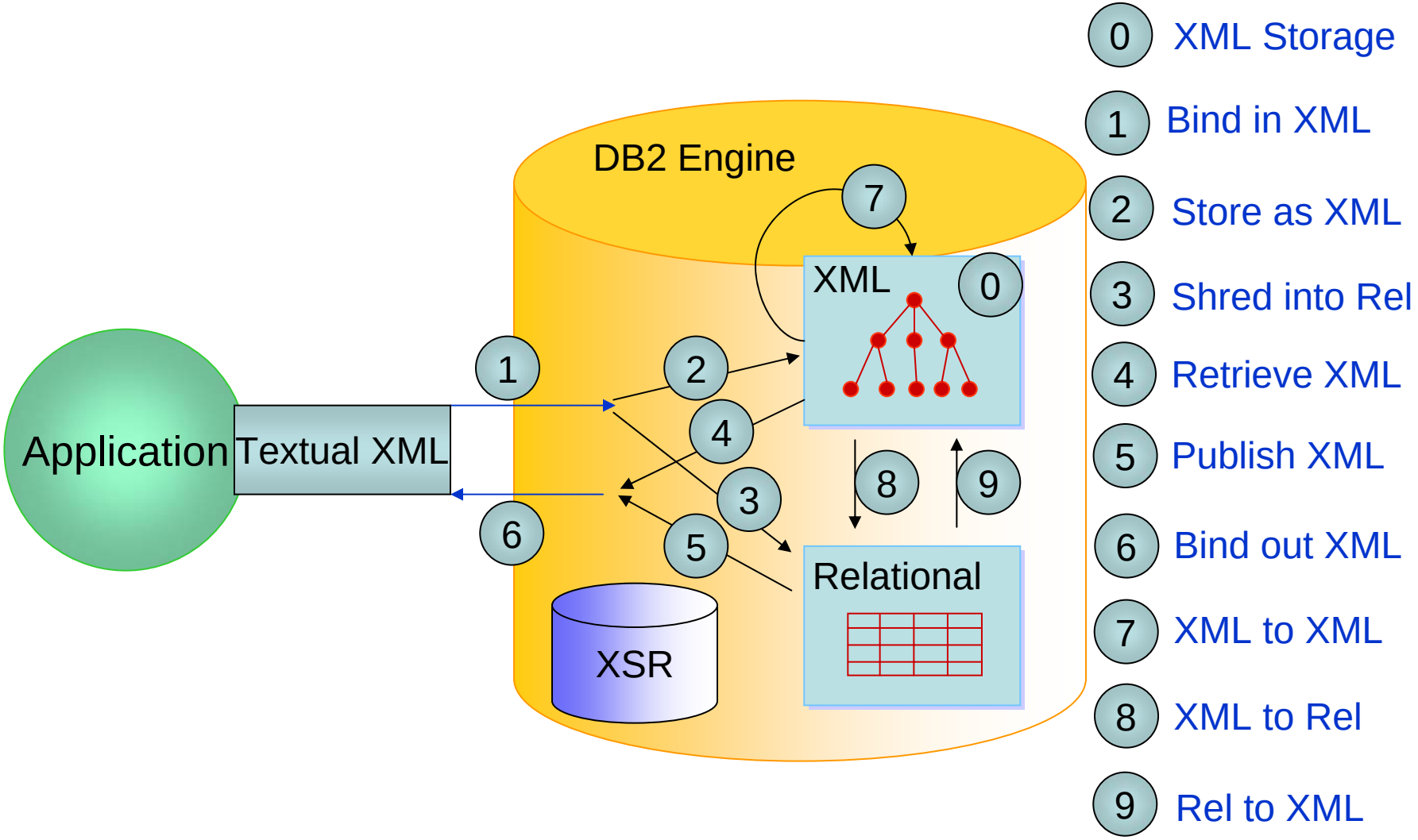
XML Characteristics

- XML represents 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)

pureXML in DB2 9

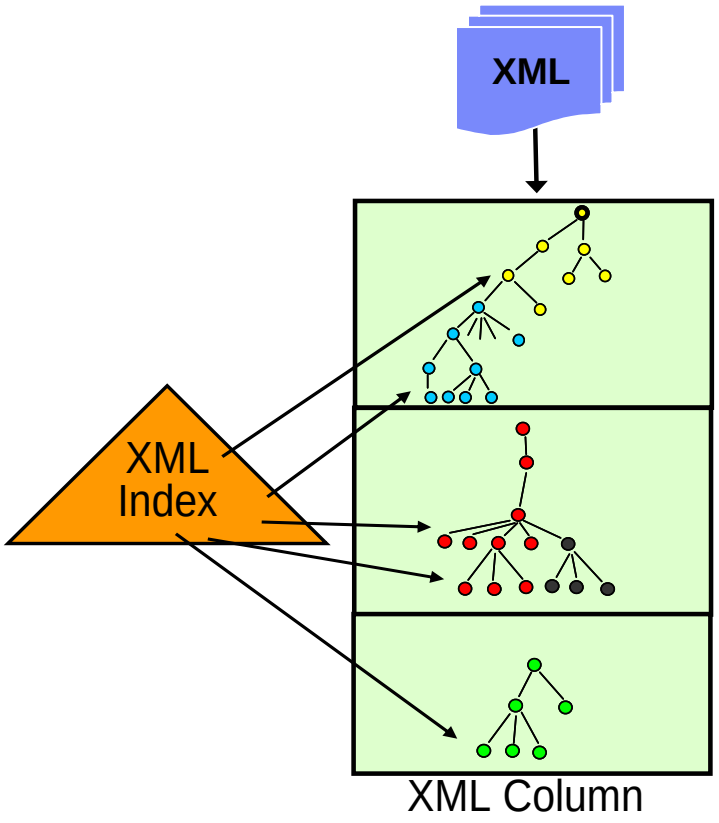
- SQL XML data type and native storage
- Designed specifically for XML from the ground up
  - Supports XML hierarchical structure storage
  - Native operations and languages: XPath, SQL/XML, XQuery (LUW only, z/OS in the future)
- Not transformating into relational
- Not using objects or nested tables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

Summary of XML Features



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



What you can do with pureXML (1)

```
CREATE TABLE PurchaseOrders (
  ponumber      varchar(10) not null,
  podate        date not null,
  status        char(1),
  XMLpo         xml);

CREATE TABLE PO LIKE PurchaseOrders;

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

ALTER TABLE PurchaseOrders
    ADD revisedXMLpo xml;
```

What you can do with pureXML (2)

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

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

INSERT INTO PurchaseOrders VALUES('200300002', CURRENT DATE, 'A',
    XMLPARSE(DOCUMENT :vchar PRESERVE WHITESPACE) );

z/OS: INSERT into PurchaseOrders VALUES( '200300001', CURRENT DATE,
    'A', DSN_XMLValidate(:xmlPo,'SYSXSR.myPOSchema'));

LUW: INSERT into PurchaseOrders VALUES( '200300001', CURRENT DATE,
    'A', XMLValidate(:xmlPo according to XMLSCHEMA ID
    "SYSXSR.myPOSchema"));

UPDATE PurchaseOrders SET XMLpo = :XMPpo_new
          WHERE ponumber = '12345';

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

What you can do with pureXML (3)

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

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

SELECT XMLQUERY('$po//items/item/quantity'   PASSING XMLpo AS
    "po") FROM PurchaseOrders;

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

SELECT XMLPO
FROM PurchaseOrders, Product
WHERE XMLEXISTS('$po//items/item[desc = $n]'   PASSING XMLpo
    AS "po", Product.name AS "n");
```

What you can do with pureXML (4)

```
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>
```

What you can do with pureXML (5)

```
SELECT XMLDocument(
   XMLElement(NAME "invoice",
    XMLAttributes( '12345' as "invoiceNo'),
      XMLQuery ('$po/purchaseOrder/billTo' PASSING xmlpo AS
   "po"),
      XMLElement(NAME "purchaseOrderNo",
          PO.ponumber)
      XMLElement(NAME "amount",
       XMLQuery
        ('fn:sum($po/purchaseOrder/items/item/xs:decimal(USPrice))'
          PASSING xmlpo AS "po") )
      )    )
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>
```

Leveraging the Power of SQL and Connecting to Relational

```
CREATE VIEW ORDER_VIEW AS
SELECT PO.POID, X.*
FROM PurchaseOrders PO,
      XMLTABLE( '//item' PASSING PO.XMLPO
        COLUMNS  "orderDate"       DATE PATH '../../@orderDate',
                 "shipTo City"     VARCHAR(20) PATH '../../shipTo/city',
                 "shipTo State"    CHAR(2) PATH '../../shipTo/state',
                 "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"         VARCHAR(60) PATH 'comment' ) AS X;
```

```
SELECT "Product Name", "shipTo State",
         SUM("US Price" * "Quantity") AS TOTAL_SALE
FROM ORDER_VIEW
GROUP BY "Product Name", "shipTo State";
```

```
SELECT "shipTo City", "shipTo State",
          RANK() OVER(ORDER BY SUM("Quantity")) AS SALES_RANK
FROM ORDER_VIEW
WHERE "Product Name" = 'Baby Monitor'
GROUP BY "shipTo State", "shipTo City"
ORDER BY SALES_RANK;
```

What you can do with pureXML (6)

- XML Schema Repository (XSR), register schemas for
  - Schema validation
  - Annotated schema decomposition
- Remote DRDA support
- Host language support: C/C++, COBOL, (PL/I, Assembler – z/OS only), Java, .Net
- Utilities and tool support for XML

Agenda

- Overview: what you can do with pureXML in DB2 9
- Some usage scenarios
- Business value: comparison with existing approaches
- pureXML Common Features
- Platform Similarities and Differences
- Future Directions

Usage Scenarios

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

When to use XML columns?

- Flexibility is more important than performance?
  - Schema is volatile? Yes – XML
  - In many complex schema cases, XML improves performance.
- Will data be processed heavily as relational later? No - XML
- Data components have meaning outside the hierarchy? No - XML
- Data attributes apply to all data or a small subset? Latter - XML
- Referential integrity is required? Yes - Relational
- Data needs to be updated often? Yes - Relational

Tedious normalization and frustrated changes of schema are an indicator for using native XML.

Example 1. Trading Exceptions

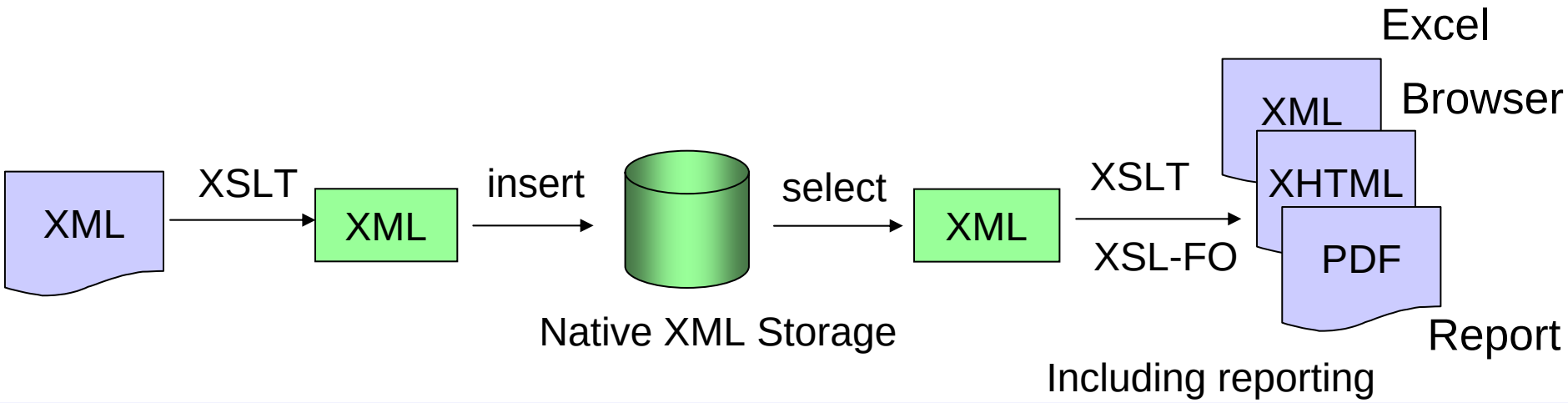
- Trading exception handling
  - Exceptions come in as XML documents
  - Exceptions from the different systems have different "attributes”
- Today’s approach - shredded into 5 tables
  - 100 common fields into one table
  - Exception attributes into 4 type-based tables: 200 integer columns, 200 varchar columns, 200 date columns, and 200 float columns, all with generic names
  - A view joining the 5 tables – too many columns, not scalable
- Solution using XML
  - 100 common columns + an XML column in one table

Example 2. Auto Insurance Policy Variations

- Vehicles have many different features, but each vehicle has only a few features, and insured may choose different policy variations
- New features may come up each model year, and new policy variations can come up too.
- It's hard to design a set of columns to cover all possible features and variations
- Some of the features and variations need to be searched upon
- Solution: use XML column

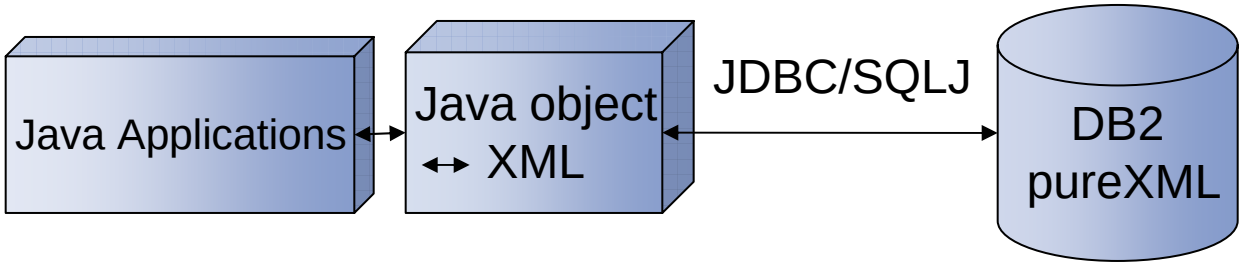
Scenario 1. Processing XML Data

- Store and process XML data directly (use XML as the data model)
- Insert/Update/Delete/Select/Extract/Construct
- Indexing/Search
- All XML solutions
  - From one angle: trade-off - speed of development v.s. storage space



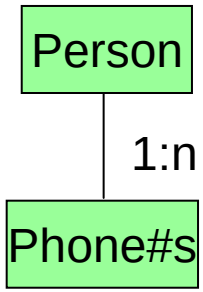
Scenario 2. Object Persistence

- Convert objects into XML and store them (SDO, JDO).
- Select objects using XMLEXISTS with XPath, and convert them back to objects.
- This approach is more efficient than object persistence with relational tables due to no need of normalization and joins.



Scenario 3. Migration from Old Data Model

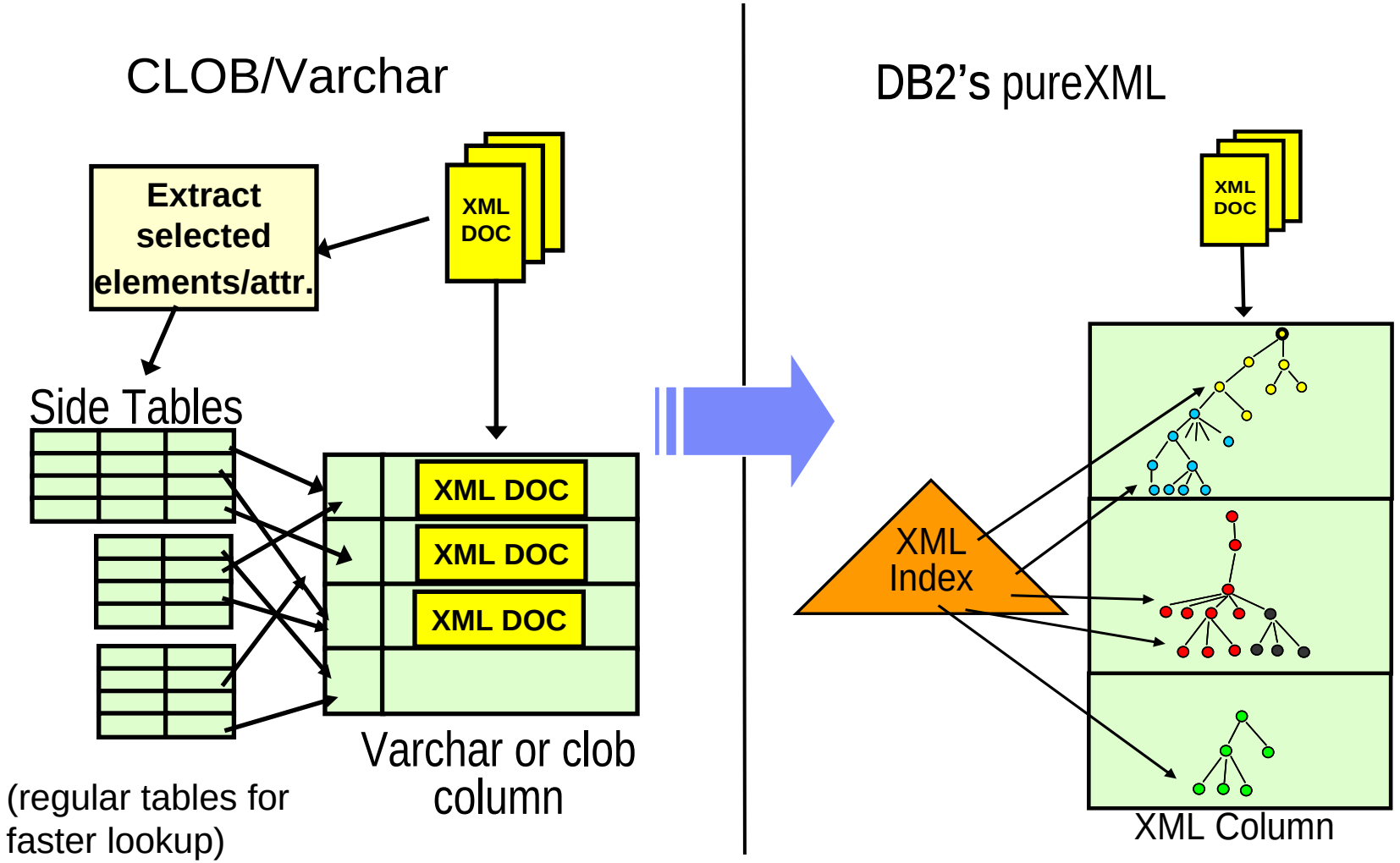
- Such as IDMS (network data model). XML matches better than relational.
- XML can represent 1:n relationships without introducing artificial keys.



Scenario 4. Migration from XML Extender

- Migrate from character columns + side tables to XML columns + XML indexes, use SQL/XML for composition.
- Migrate from decomposition
  - Application is XML-oriented → use XML columns + XML indexes
  - Application is relational -> Store as XML but use XMLTABLE function to provide relational views
  - Application is relational → use new XDBDECOMPXML
  - Use SQL/XML for composition

Migration from LOB/VARCHAR Storage

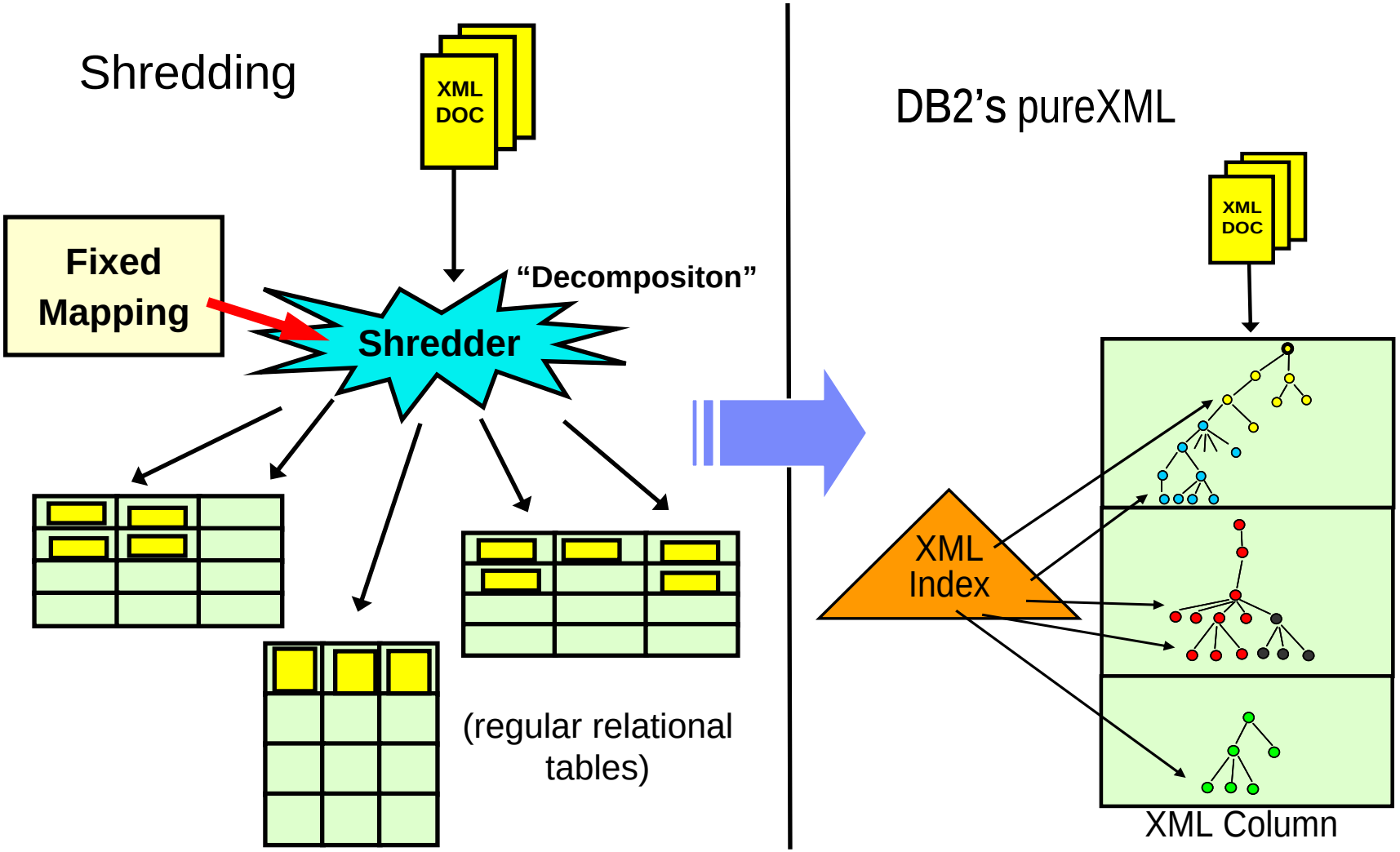


Migration from LOB/VARCHAR Storage (Cont'ed)

- LOB/VARCHAR columns => XML columns
  - UNLOAD/LOAD (export/import) to transfer data (file reference supported)
- Side tables + indexes => XML indexes
- Search => XMLEXISTS
- Extraction => XMLQuery, XMLTable
- Composition => SQL/XML constructor/publishing func
- Sub-document update => XMLUPDATE (stored proc)

Look near the bottom of this web page about migration from XML Extender to native XML (LUW):
http://www-03.ibm.com/developerworks/wikis/display/db2xml/Technical+Papers+and+Articles

Migration from Decomposed Relational Storage

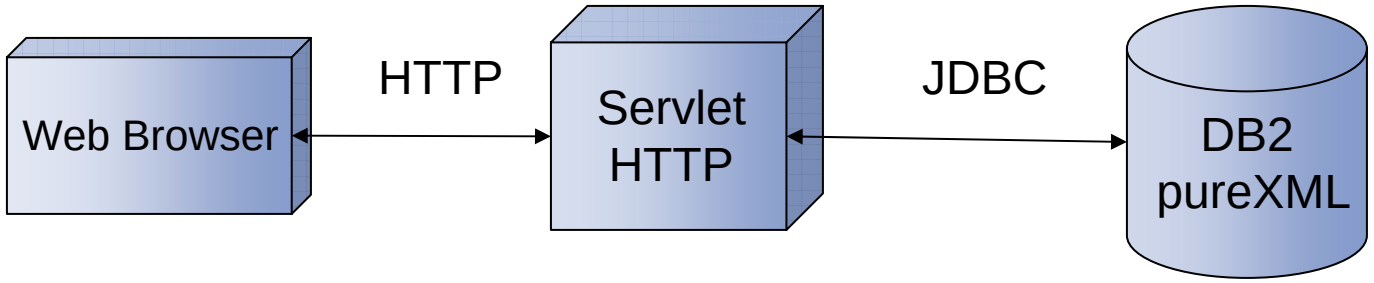


Migration from Decomposed Relational Storage (Cont'ed)

- If application is XML-oriented or to be upgraded from relational to XML
  - Tables => XML columns
  - Indexes => XML indexes
  - Decomposition/Composition => GONE
  - Search => XMLEXISTS
  - Extraction => XMLQuery, XMLTable
  - Other composition => SQL/XML constructor/publishing functions
- If application is relational-oriented and to maintain compatibility:
  - Decomposition => new XDBDECOMPXML
  - Composition => SQL/XML constructor/publishing functions
  - Relational tools => XMLTABLE view

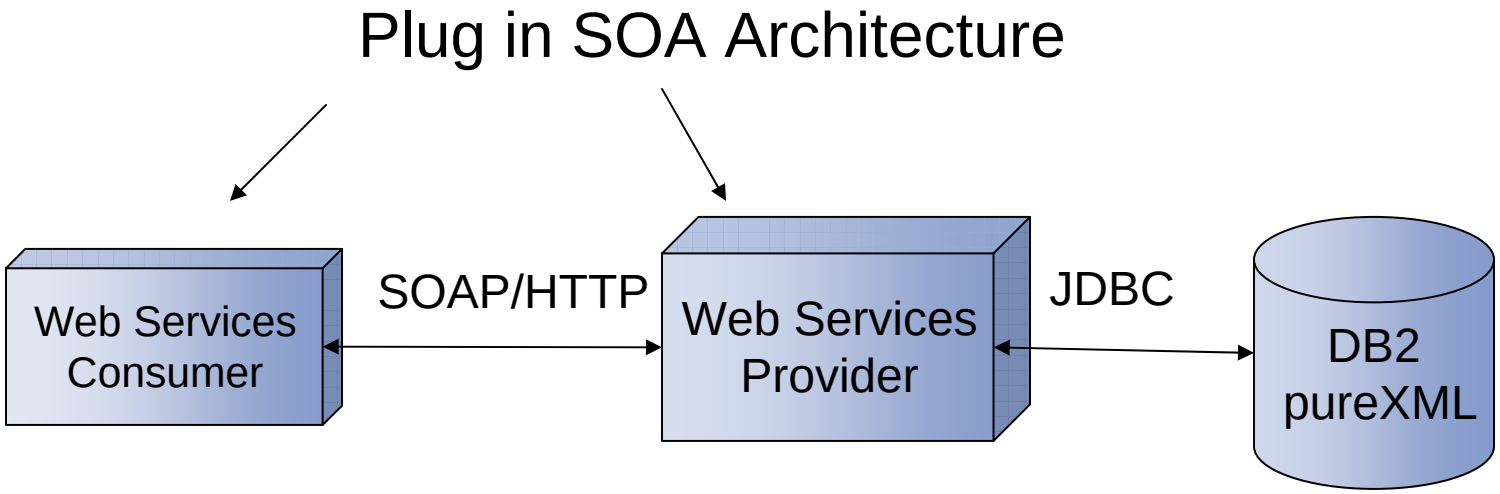
Scenario 5. Generate XHTML for Web

- Instead of using Java or other languages to generate XHTML, use SQL directly to generate dynamic web pages



Scenario 6. Support Web Services/SOAP

- Instead of using Java or other languages to handle SOAP, use more SQL/XML to handle SOAP



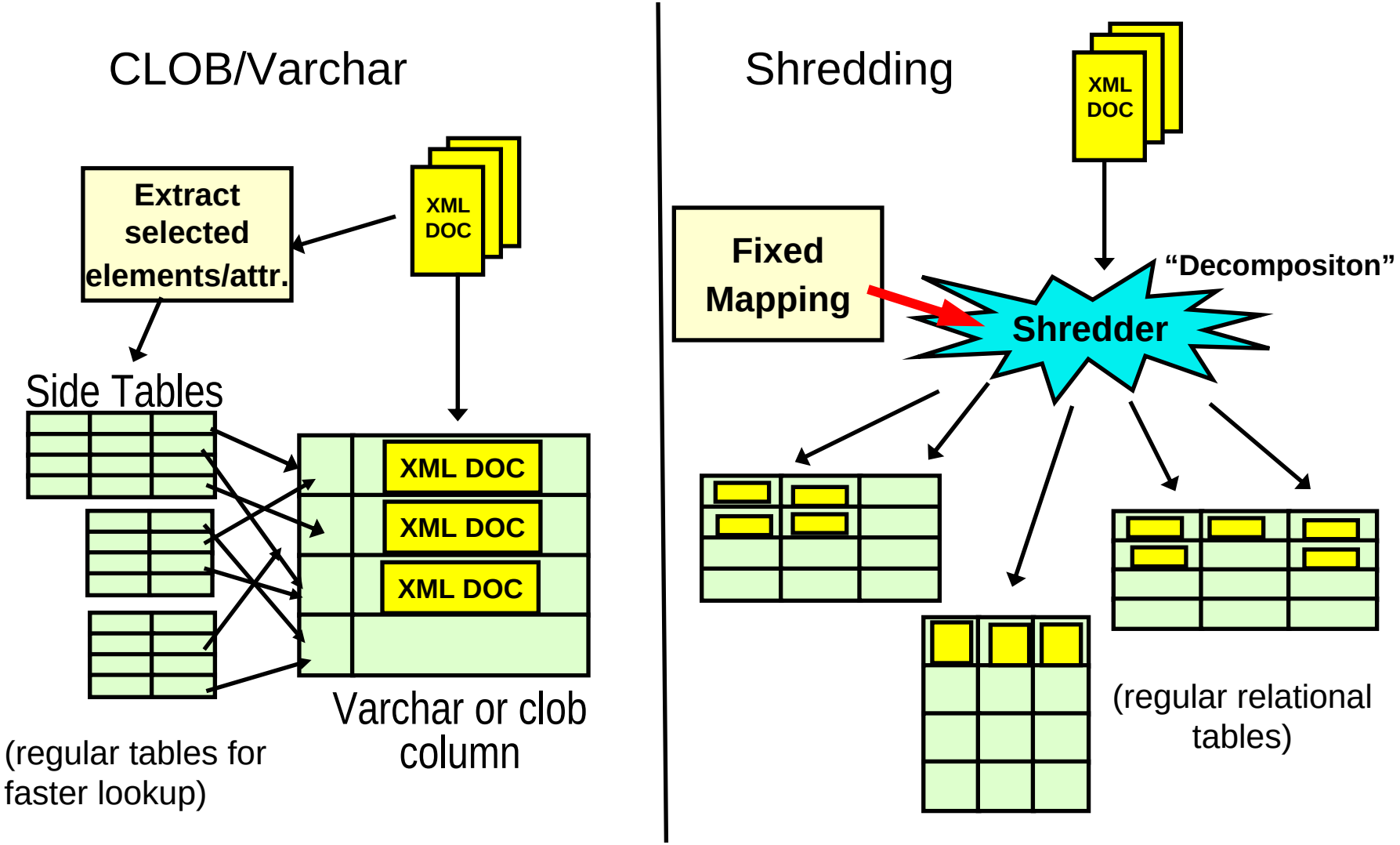
Agenda

- Overview: what you can do with pureXML in DB2 9
- Some usage scenarios
- Business value: comparison with existing approaches
- pureXML common features
- Platform similarities and differences
- Future directions

Why (Hybrid) XML Databases?

- Businesses need to manage XML data with ACID properties, auditing and regulatory compliance.
- 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)

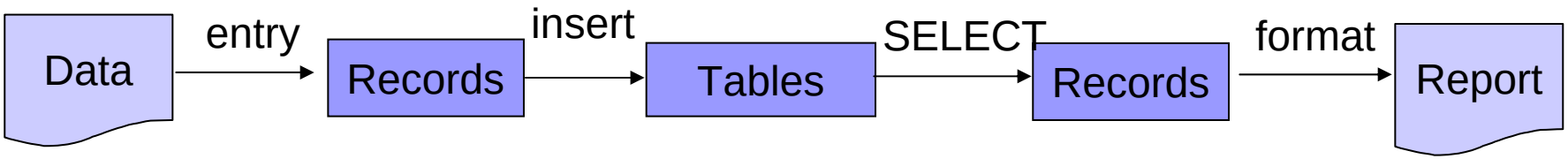
XML-Enabled Databases: Two Main Options



Problems of XML-enabled Databases

- CLOB storage:
  - Query evaluation & sub-document level access requires costly XML Parsing – too slow !
- Shredding:
  - Mapping from XML to relational often too complex
  - Often requires dozens or hundreds of tables
  - Complex multi-way joins to reconstruct documents
  - XML schema changes break the mapping
    - no schema flexibility !
    - For example: Change element from single- to multi-occurrence requires normalization of relational schema & data

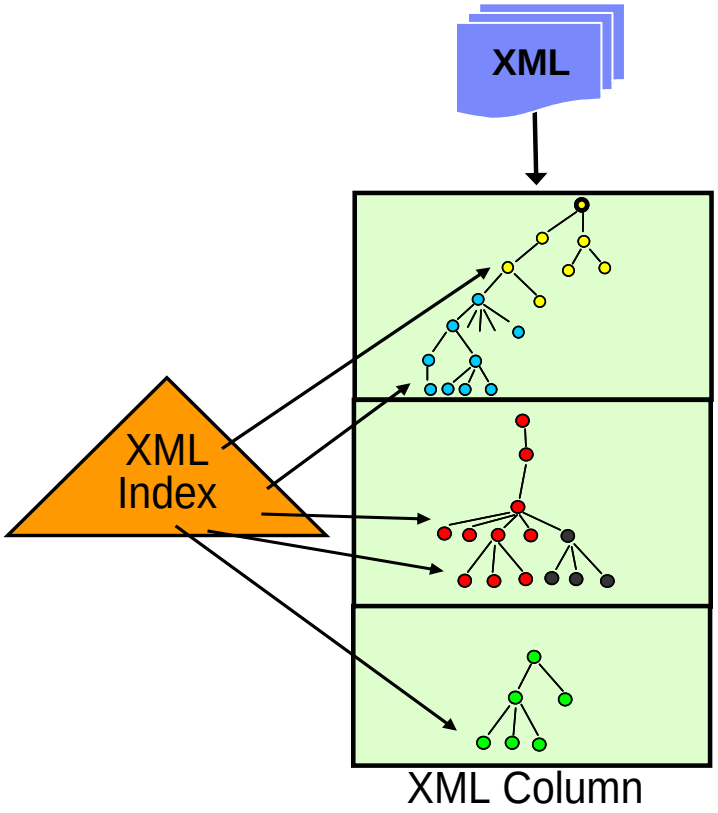
Normalization and Schema Evolution



- Normalization to eliminate redundancy to:
  - Avoid update anomaly.
  - Avoid inconsistency.
  - Reduce storage requirements
- Fact: rigid table structures and 3NF also cause:
  - Programming overhead in normalization/denormalization.
  - Performance problems due to joins.
  - Difficulties in schema evolution.
- Fact: people de-normalize to avoid some problems.
  - MVDB has a niche (and growing) market.

DB2 pureXML Advantages

DB2's hierarchical storage:
XML type as XML



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

Up to 10 times

Business Value of pureXML

- ***Lower Development Costs***
  - Reduced system and development complexity
  - Improved development productivity
- ***Greater Business Agility***
  - Easily accommodate changes to data and schemas
  - Update applications rapidly and reduce maintenance costs
- ***Improved Business Insight***
  - Access to information in otherwise unexploited documents
  - Unprecedented application performance

Agenda

- Overview: what you can do with pureXML in DB2 9
- Some usage scenarios
- Business value: comparison with existing approaches
- pureXML common features
- Platform similarities and differences
- Future directions

Common XML Features in DB2 9

- First-class XML type, native storage of XQuery Data Model (XDM)
- Complete SQL/XML constructor functions
- XMLPARSE and XMLSERIALIZE
- XML indexes
- Other SQL/XML functions with XPath
  - XMLEXISTS, XMLQUERY, XMLTABLE (z/OS post GA)
- XML Schema repository, validation, and decomposition stored proc
- DRDA (distributed support) and application interfaces
- Utilities and tools

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.

XML Type and DDL

```
CREATE TABLE PurchaseOrders (
  ponumber       varchar(10) not null,
  podate                     date not null,
  status         char(1),
  XMLpo          xml)
    ;

CREATE TABLE PO LIKE PurchaseOrders;

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

ALTER TABLE PurchaseOrders
    ADD revisedXMLpo xml;
```

- No length limit
- No associated schema

Manipulating XML Data

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

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

UPDATE PurchaseOrders SET XMLpo = :XMLpo_backup
       WHERE ponumber = '12345';

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

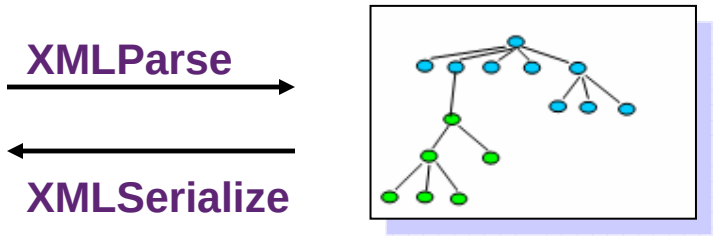
Host var of XML type

String literal is OK

Whole document replacement

XMLParse and XMLSerialize

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



- 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

Examples of XPath – Typing (z/OS on all documents and LUW on non-validated documents)

- No cast is needed: “Find all the products in the Catalog with RegPrice > 100”
  XMLQUERY(‘$x/Catalog/Categories/Product[RegPrice > 100]’ PASSING XCatalog as “x”)

- Cast is needed: “Find all the products on sale in the Catalog”
  XMLQUERY(‘$x/Catalog/Categories/Product[RegPrice > xs:double(SalePrice)]’ PASSING XCatalog as “x”)

- No cast is needed: “Find all the products with more than 10% discount in the Catalog”
  XMLQUERY(‘$x/Catalog/Categories/Product[RegPrice * 0.9 > SalePrice]’ PASSING XCatalog as “x”)

Examples of XPath - Cardinality

- No cardinality problem: “Find all the products in the Catalog with RegPrice > $price”
  XMLQUERY(‘$x/Catalog/Categories/Product[RegPrice > $price]’ PASSING XCatalog as “x”, 200 as “price”)
- To avoid cardinality violation: “Find all the products on sale in the Catalog”
  XMLQUERY(‘$x/Catalog/Categories/Product[RegPrice > SalePrice/xs:double(.))]’ PASSING XCatalog as “x”)
- To avoid cardinality violation: “Find all the products with more than 10% discount in the Catalog”
  XMLQUERY(‘$x/Catalog/Categories/Product[RegPrice/(. * 0.9) > SalePrice]’ PASSING XCatalog as “x”)

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 Index Usage

- Support for **XMLEXISTS** predicate only, not XMLQuery
- XMLExists is a stage 2 predicate (z/OS), but can use XML value indexes
- XPath index exploitation is much like summary table matching:
  - Need to use matching logic for exploitation
- Criteria:
  - Index pattern is equal to or less restrictive than the query predicate:
    index: //product/regprice v.s.
    query: /catalog//product[regprice > 10]
  - Data types have to match.
- Use internal “between” for better performance.
  - //item[@size > 5 and @size < 10]
  - //product[wt > 10 and wt < 20] =>
    //product[wt[. > 10 and . <20]]

Application Interfaces

- XML type is supported in
  - Java (JDBC, SQLJ), ODBC,
  - C/C++, COBOL, (PL/I, Assembly – z/OS only)
  - .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)

Java JDBC Example

Use standard interface:

```
PreparedStatement pstmt = connection.prepareStatement("INSERT INTO
     PurchaseOders 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

XML Schema Repository

- 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

Registering an XML Schema (Procedure)

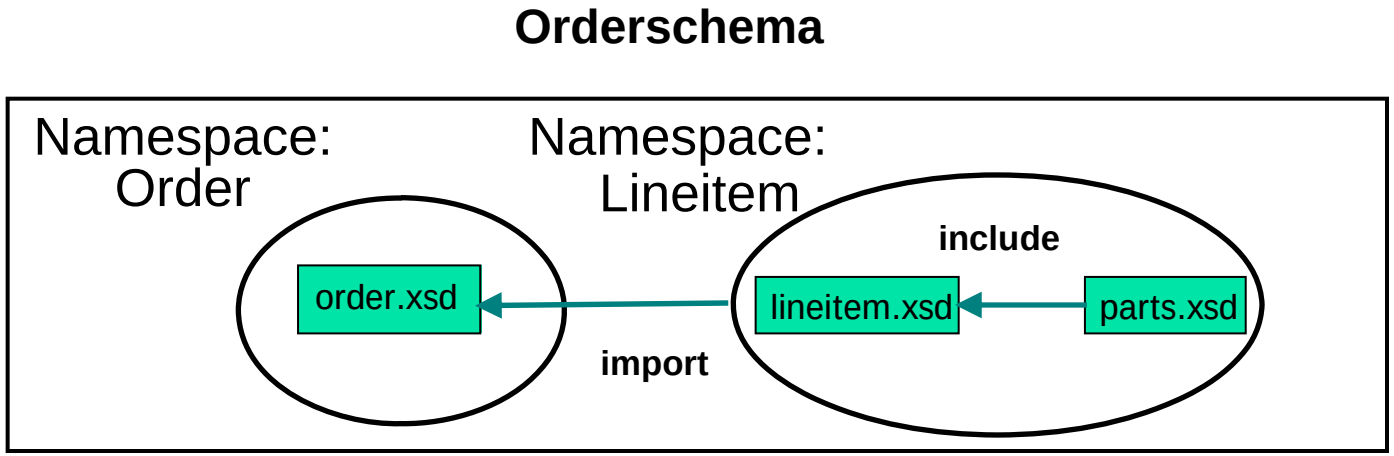
- XSR_REGISTER (rschema, name, schemalocation, xsd, docproperty)
- XSR_ADDSCHEMADOC (rschema, name, schemalocation, xsd, docproperty)
- XSR_COMPLETE (rschema, name, schemaproperties, isUsedForDecomp)
- XSR_REMOVE(rschema, name)

- 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



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

Using XML Schema

- Schema validation (type annotation not kept)
  INSERT into PurchaseOrders
  VALUES('200300001', CURRENT DATE, 'A',
  SYSFUN.DSN_XMLValidate(:xmlPo,'SYSXSR.ORDERSchema'))
  ; (z/OS)

LUW: XMLVALIDATE(:xmlPo according to XMLSCHEMA ID "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" />

annotations

PORDER

ORDERID	ORDE
19991201-ZFG	1999-

Best Practices

- Tip 1: Choose your XML document granularity wisely
- Tip 2: Be aware of XML schema validation overhead
- Tip 3: Avoid code page conversion during XML insert and retrieval
- Tip 4: In XPath expressions, use fully specified paths as much as possible
- Tip 5: Define lean XML indexes, and avoid indexing everything
- Tip 6: Put document filtering predicates in XMLEXISTS instead of XMLQUERY
- Tip 7: Use square brackets [ ] to avoid Boolean predicates in XMLEXISTS
- Tip 8: Use RUNSTATS to collects statistics for XML data and indexes
- Tip 9: Use SQL/XML publishing views to expose relational data as XML
- Tip 10: For short queries or OLTP applications, use SQL/XML statements with parameter markers

Adapted from LUW's Top 15 best practices – 10 applies to z/OS
http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0610nicola/

Tools

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

Agenda

- Overview: what you can do with pureXML in DB2 9
- Some usage scenarios
- Business value: comparison with existing approaches
- pureXML Common Features
- Platform Similarities and Differences
- Future Directions

Color Code

- Some charts are platform-specific
- Black – both
- Red – z/OS
- Blue – LUW

SQL/XML Publishing Functions in V8

- Scalar functions
  - Formally called XML constructors
  - XMLELEMENT, XMLATTRIBUTES
  - XMLNAMESPACES
  - XMLFOREST
  - XMLCONCAT
- Aggregate function: XMLAGG
- Transient XML data type – internal use only
- Cast function
  - Cast internal XML values to CLOB: XML2CLOB, XMLSERIALIZE
- Conformant to 2003 ANSI SQL/XML standard

SQL/XML in V9

- XMLCAST
- XMLCOMMENT
- XMLDOCUMENT
- XMLPI
- XMLTEXT
- XMLPARSE

- XMLVALIDATE and DSN_XMLVALIDATE
- XMLTABLE
- XMLQUERY – full support in LUW, XPATH only in z/OS
- XMLEXISTS – full support in LUW, XPATH only in z/OS

- XMLSERIALIZE (was added in z/OS)

Validation in V9

- Both systems use schema repository and can only validate against schemas in repository
  - Schema can include other schema documents
  - Schema ID is preserved in the document (no function yet to check it in z/OS)
  - Support for annotated schema for decomposition

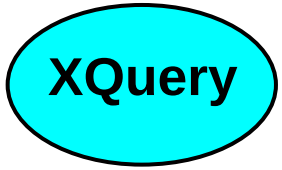
- Differences:
  - LUW - result of validation is annotated document
  - z/OS - result is a binary object no type annotation information preserved

  - LUW example:
    - xmlvalidate(xmlparse (document '<…….>') according to xmlschema id foo)
    - xmlvalidate(? According to xmlschema uri 'http://my.dept.com')
    - xmlvalidate(?)
  - z/OS example:
    - xmlparse (document SYSFUN.DSN_XMLValidate (:xxx, 'SYSXSR.foo'))

XPath/XQuery in V9

- LUW has more extensive support for XQuery, built-in types, and functions
  - 48 built in data types and about 150 functions
  - XQuery is one of the primary languages.
  - z/OS supports 6 **most common** data types:
    - xs:string, xs:boolean, xs:decimal, xs:double, xs:integer, xs:untypedAtomic
  - and 14 **most common** functions: fn:abs, fn:boolean, fn:compare, fn:concat, fn:contains, fn:count, fn:data, fn:length, fn:normalize-space, fn:not, fn:round, fn:string, fn:substring, fn:sum
  - All relevant navigation axis are supported: child, descendant, descendant-or-self, self, attribute, parent axis, are supported: //, .., /, @
  - LUW – full XQuery language support
  - z/OS – XPath Axes navigation only
    - No FLWOR expressions
    - Explicit type casting maybe needed since all data is untyped
    - No XQuery constructors

LUW: XQuery: The FLWOR Expression



- **F**OR: iterates through a sequence, bind variable to items
- **L**ET: binds a variable to a sequence
- **W**HERE: eliminates items of the iteration
- **O**RDER: reorders items of the iteration
- **R**ETURN: constructs query results

```
XQUERY for $movie in xmlcolumn('movies.doc')
let $actors := $movie//actor
where $movie/duration > 90
order by $movie/@year
return   <movie>
              {$movie/title, $actors}
          </movie>
```

```
<movie>
  <title>Chicago</title>
  <actor>Renee  Zellweger</actor>
  <actor>Richard  Gere</actor>
  <actor>Catherine  Zeta-Jones</actor>
</movie>
```

Common queries: same for LUW and zOS

Common queries are identical across DB2 LUW and DB2 z/OS.
Example:

"*Extract the order date of the order with ID 123456.*"

```
SELECT XMLQUERY ('$d/Order/Date' passing doc AS "d")
FROM order
WHERE XMLEXISTS('$d/Order[@ID = 123456]' passing doc AS "d")
```

DB2 LUW can express the same query in XQuery without SQL.

Use SQL/XML to Achieve XQuery Functionality

- Use XMLEXISTS with XPath to find documents.
- Use XMLQuery 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.

Differences for XML construction

DB2 LUW only

```
SELECT
  XMLQUERY('<Customer_Profile CUSTOMERID="{$c/Customer/@id}">
                          {$c/Customer/Name}
                          {$c/Customer/DateOfBirth}
                    </Customer_Profile>'
    PASSING doc AS "c")
FROM custacc
WHERE XMLEXISTS('$c/Customer[@id=1002]' PASSING doc AS "c")
```

```
<Customer_Profile CUSTOMERID="1002">
  <Name>John Doe</Name>
  <DateOfBirth>11-02-1964</DateOfBirth>
</Customer_Profile>
```

DB2 zOS, DB2 LUW:

```
SELECT
   XMLELEMENT ( Name "Customer_Profile",
     XMLATTRIBUTES(
       VARCHAR(XMLSerialize(
                XMLQUERY (fn:string($c/Customer/@id)' PASSING  doc AS "c")  as CLOB(100)))
            AS "CUSTOMERID" ),
      XMLQUERY ($c/Customer/Name'  PASSING doc AS "c"),
      XMLQUERY ($c/Customer/DateOfBirth' PASSING  doc AS "c")   ) AS CustomerProfile
FROM custacc
WHERE XMLEXISTS('$c/Customer[@id=1002]' PASSING doc AS "c")
```

Differences for XML construction

DB2 LUW only

```
SELECT
  XMLQUERY('<Customer_Profile CUSTOMERID="{$c/Customer/@id}">
                          {$c/Customer/Name}
                          {$c/Customer/DateOfBirth}
                  </Customer_Profile>'
    PASSING doc AS "c")
FROM custacc
WHERE XMLEXISTS('$c/Customer[@id=1002]' PASSING doc AS "c")
```

```
<Customer_Profile CUSTOMERID="1002">
  <Name>John Doe</Name>
  <DateOfBirth>11-02-1964</DateOfBirth>
</Customer_Profile>
```

DB2 zOS, DB2 LUW:

```
SELECT
   XMLELEMENT ( Name "Customer_Profile",
    XMLATTRIBUTES(
     VARCHAR(XMLSerialize(
              XMLQUERY (fn:string($c/Customer/@id)' PASSING  doc AS "c")  as CLOB(100)))
          AS "CUSTOMERID" ),
     XMLQUERY ($c/Customer/Name'  PASSING doc AS "c"),
     XMLQUERY ($c/Customer/DateOfBirth' PASSING  doc AS "c")   ) AS CustomerProfile
FROM custacc
WHERE XMLEXISTS('$c/Customer[@id=1002]' PASSING doc AS "c")
```

Differences for XML construction

DB2 LUW only

```
SELECT
  XMLQUERY('<Customer_Profile CUSTOMERID="{$c/Customer/@id}">
                          {$c/Customer/Name}
                          {$c/Customer/DateOfBirth}
                  </Customer_Profile>'
    PASSING doc AS "c")
FROM custacc
WHERE XMLEXISTS('$c/Customer[@id=1002]' PASSING doc AS "c")
```

```
<Customer_Profile CUSTOMERID="1002">
  <Name>John Doe</Name>
  <DateOfBirth>11-02-1964</DateOfBirth>
</Customer_Profile>
```

DB2 zOS, DB2 LUW:

```
SELECT
   XMLELEMENT ( Name "Customer_Profile",
     XMLATTRIBUTES(
       VARCHAR(XMLSerialize(
                XMLQUERY (fn:string($c/Customer/@id)' PASSING  doc AS "c")  as CLOB(100)))
           AS "CUSTOMERID" ),
      XMLQUERY ($c/Customer/Name'  PASSING doc AS "c"),
      XMLQUERY ($c/Customer/DateOfBirth' PASSING  doc AS "c")   ) AS CustomerProfile
FROM custacc
WHERE XMLEXISTS('$c/Customer[@id=1002]' PASSING doc AS "c")
```

Index differences in V9

- DDL statement for XML CREATE INDEX is the same in both system
- Types supported in indexes:
  - z/OS has support only for two types: VARCHAR and DECFLOAT
  - LUW supports 5 types: VARCHAR, VARCHAR HASHED, DOUBLE, DATE, TIMESTAMP
- Example for both systems:

```
CREATE INDEX EMPINDEX ON DEPARTMENT (DEPTDOCS)
GENERATE KEYS USING XMLPATTERN
'/department/emp/name/last'
AS SQL VARCHAR(20)
```

Utilities and language bindings in V9

- z/OS has more extensive support for XML in utilities:
  - LOAD
  - ONLINE REORG
- JDBC, SQLJ, ODBC, C/C++, COBOL, .NET, PHP, PL/I, ASSEMBLER
- XDBDECOMPXML
- LUW XQuery supported by all APIs
  - *Result sequence will be treated as a resultset*
  - *Each item will be treated as a row.*

Utilities on z/OS

- 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

LUW: Utilities & Tools for XML

- XML Import & Export
- Runstats collects stats for XML data
- XML type support in SQL stored procedures
- XML columns supported by HADR
- XML columns supported by backup/restore
- XQuery Builder GUI
- GUI for XML Schema annotations for shredding
- XML Index Definition GUI
- Control Center extensions for XML

z/OS: Performance and Scalability

- XML storage leverages mature optimized storage infrastructure.
- Next generation parsers: z/OS XMLSS and XLXP-C.
- Highly efficient XPath streaming algorithm
- Support partitioned table spaces and data sharing.
- Initial sweet spot: a large number of small documents.

z/OS: 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

z/OS: 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.	XMLExists('/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.	XMLExists('/Catalog/Categories/Product[RegPrice > 100 and Discount > 0.1]' …) With indexes on: '//RegPrice' as SQL DECFLOAT and '//Discount' as SQL DECFLOAT

LUW: New Query Operators

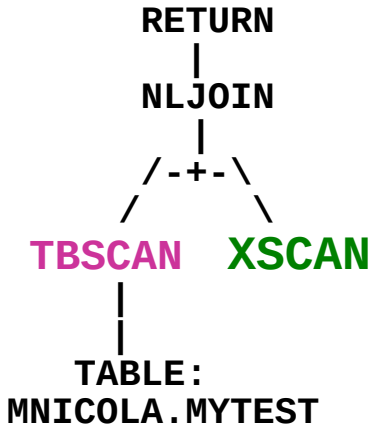
- **XML Scan** and Navigation: **XSCAN**
- **XML Index** access: **XISCAN**
- **XML Index** Joins: **XANDOR**

LUW Example: XML Operators & Execution Plans

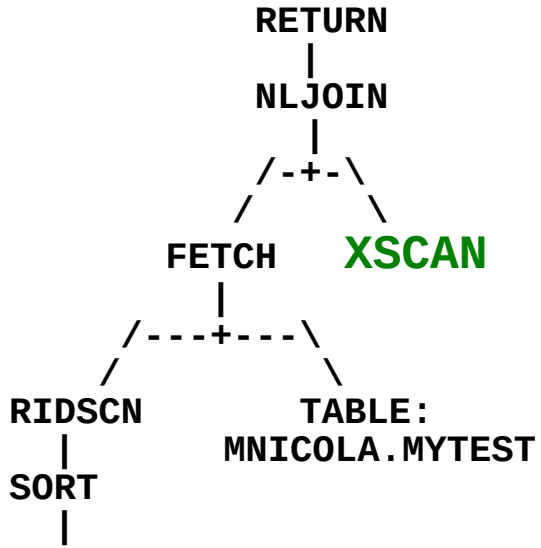
Query: /customerinfo[name="Matt Foreman" AND phone="905-555-4789"]

```
<customerinfo>
   <name>Matt Foreman</name>
   <phone>905-555-4789</phone>
</customerinfo>
```

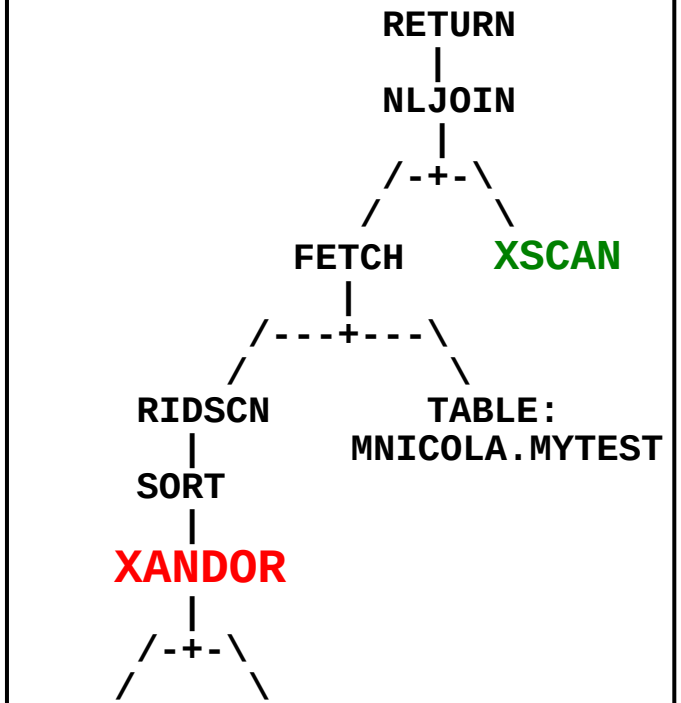
No index 😐



1 index, on name or phone ☺



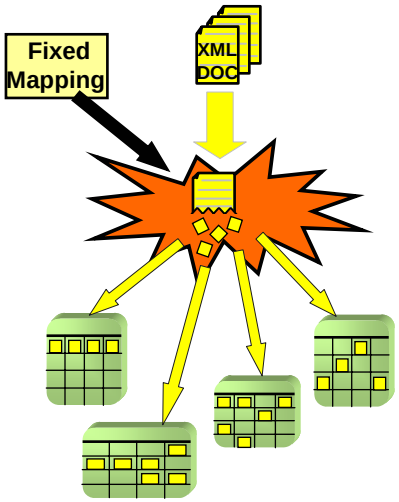
2 indexes, on name and phone ☺



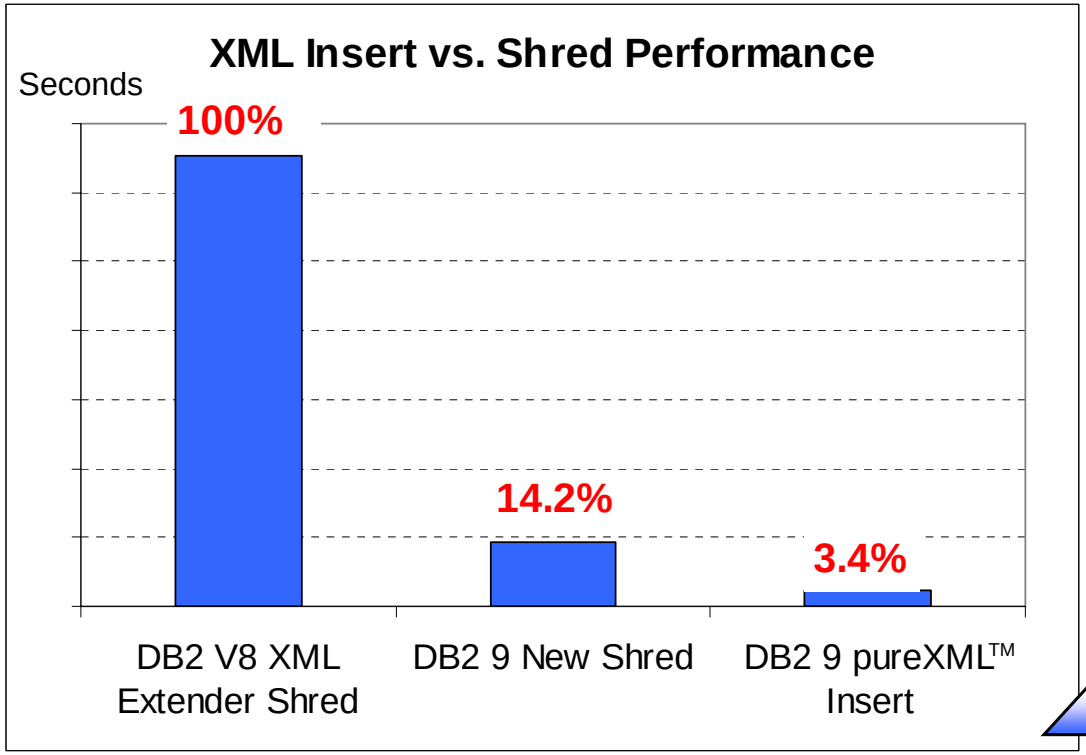
LUW: pureXML™ Insert vs. Shredding Performance

10,000 XML Documents, 4Kb to 20Kb…

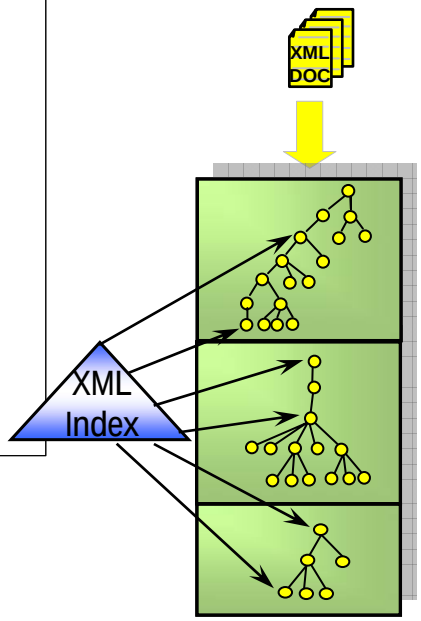
…shredded to 87 columns in 12 tables.



(regular relational tables)

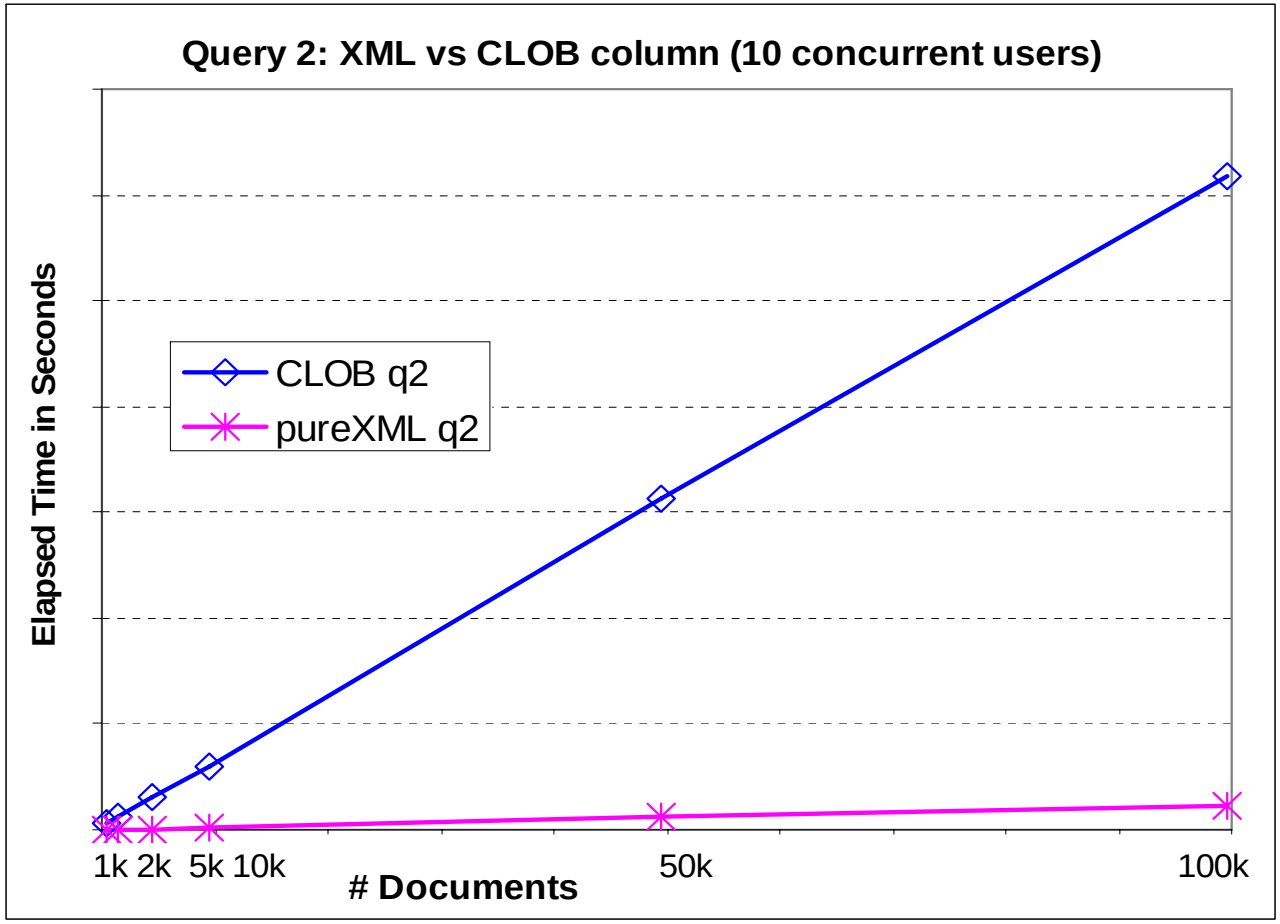


…pureXML™ insert, 1 XML column, 1 table.



XML Column

LUW: pureXML™ vs CLOB Query Performance



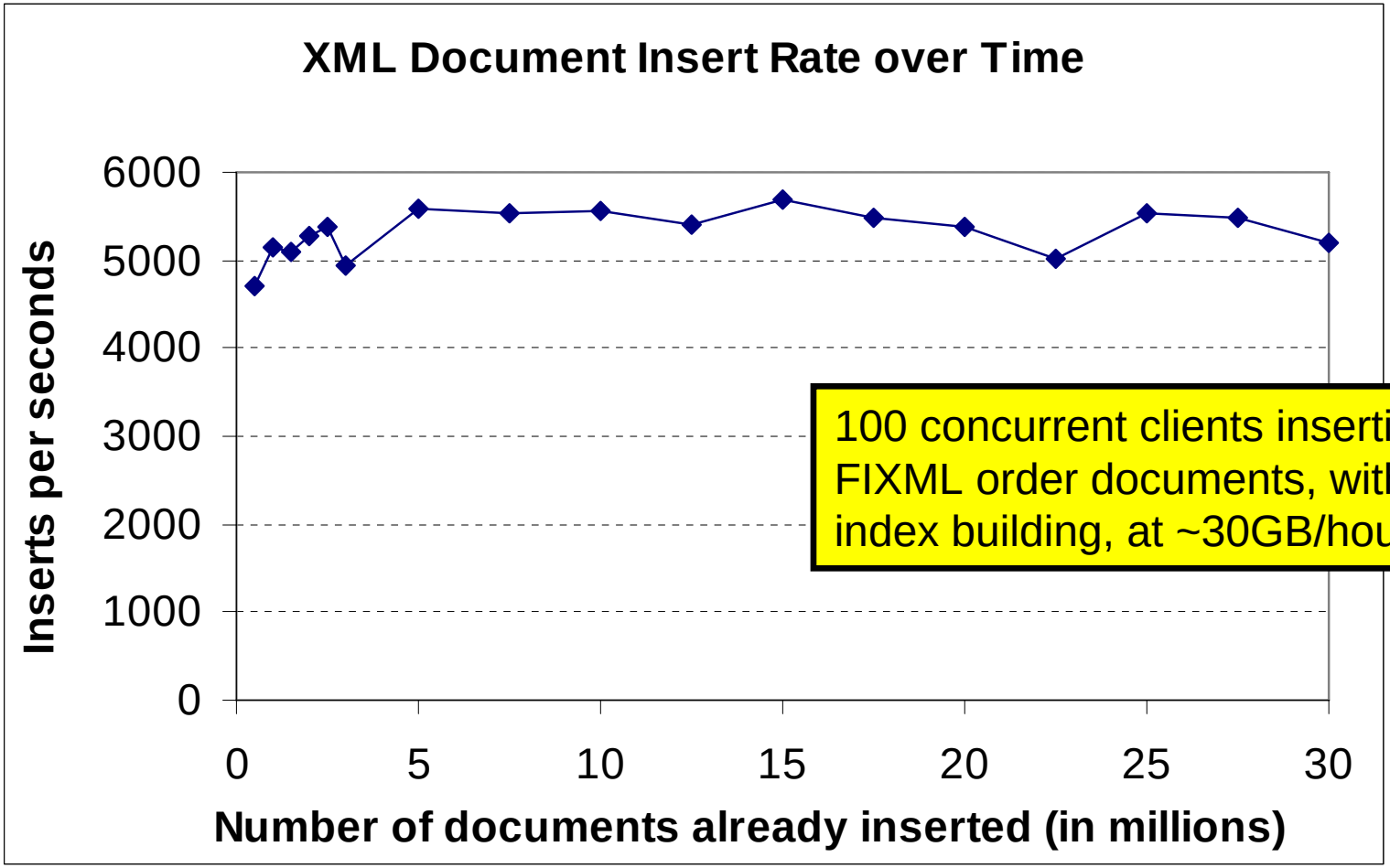
Query 2: Retrieves one document based on a single search condition.

No index is used.

The larger the table (1,000 to 100,000 documents) the bigger the performance benefit of pureXML™ !

Query Response Time **lower is better**

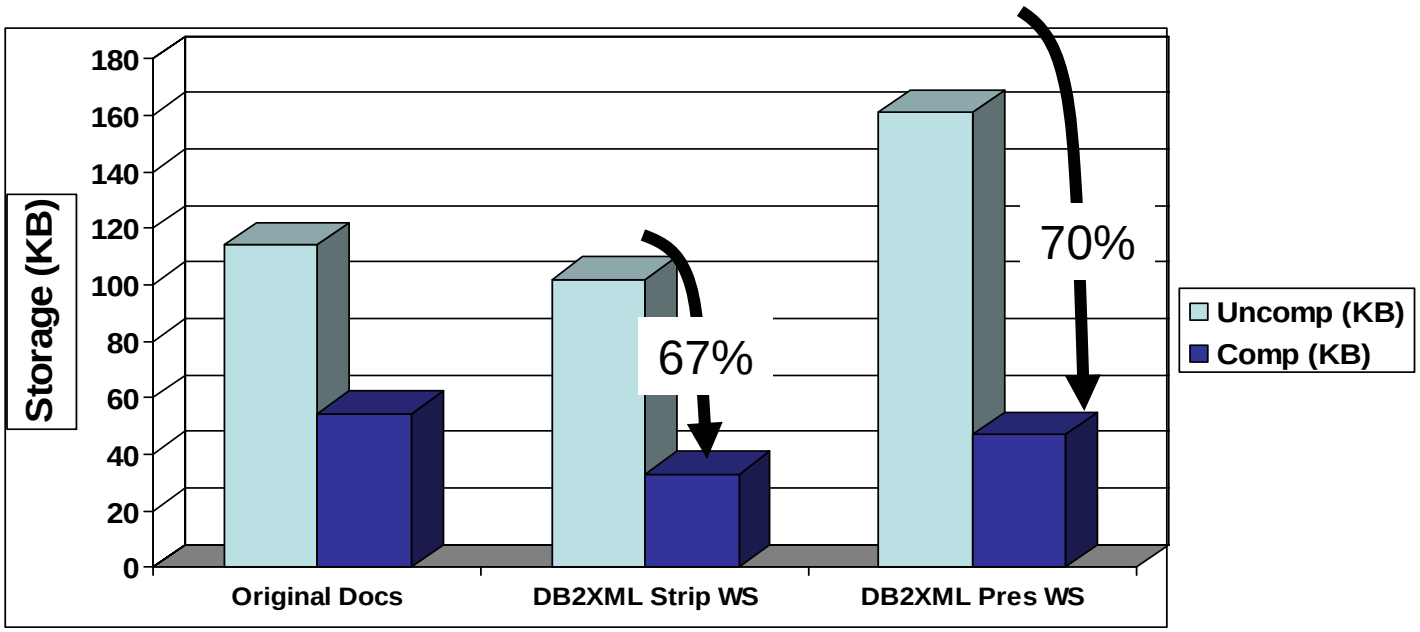
LUW: Constantly High XML Insert Throughput



AIX 5.3, P-Series P5-560Q, 8 CPUs, TotalStorage DS8100,

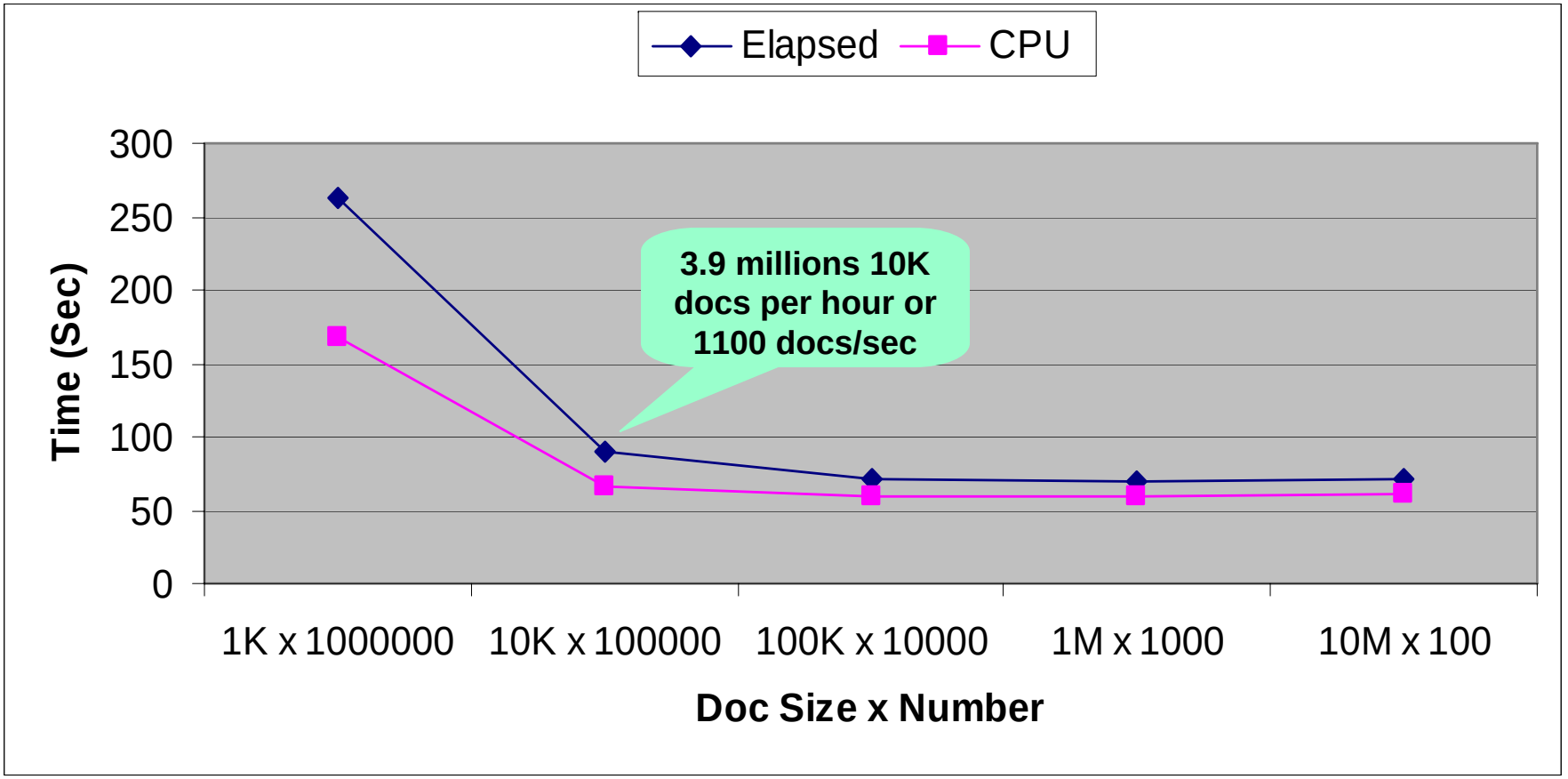
http://www.ibm.com/developerworks/db2/library/techarticle/dm-0606schiefer

z/OS: Storage for UNIFI XML Messages



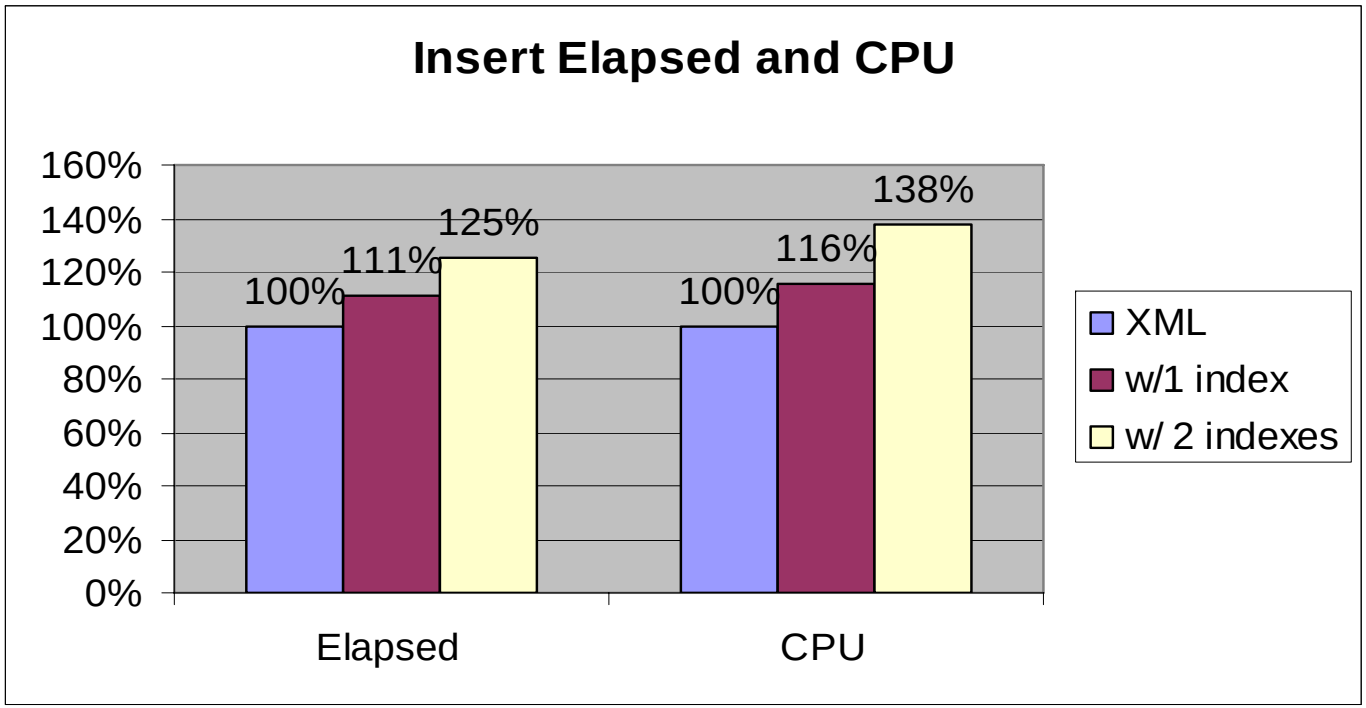
96 sample documents
Strip WS: Strip Whitespaces
Pres WS: Preserve Whitespaces

z/OS: Insert Performance (Batch)

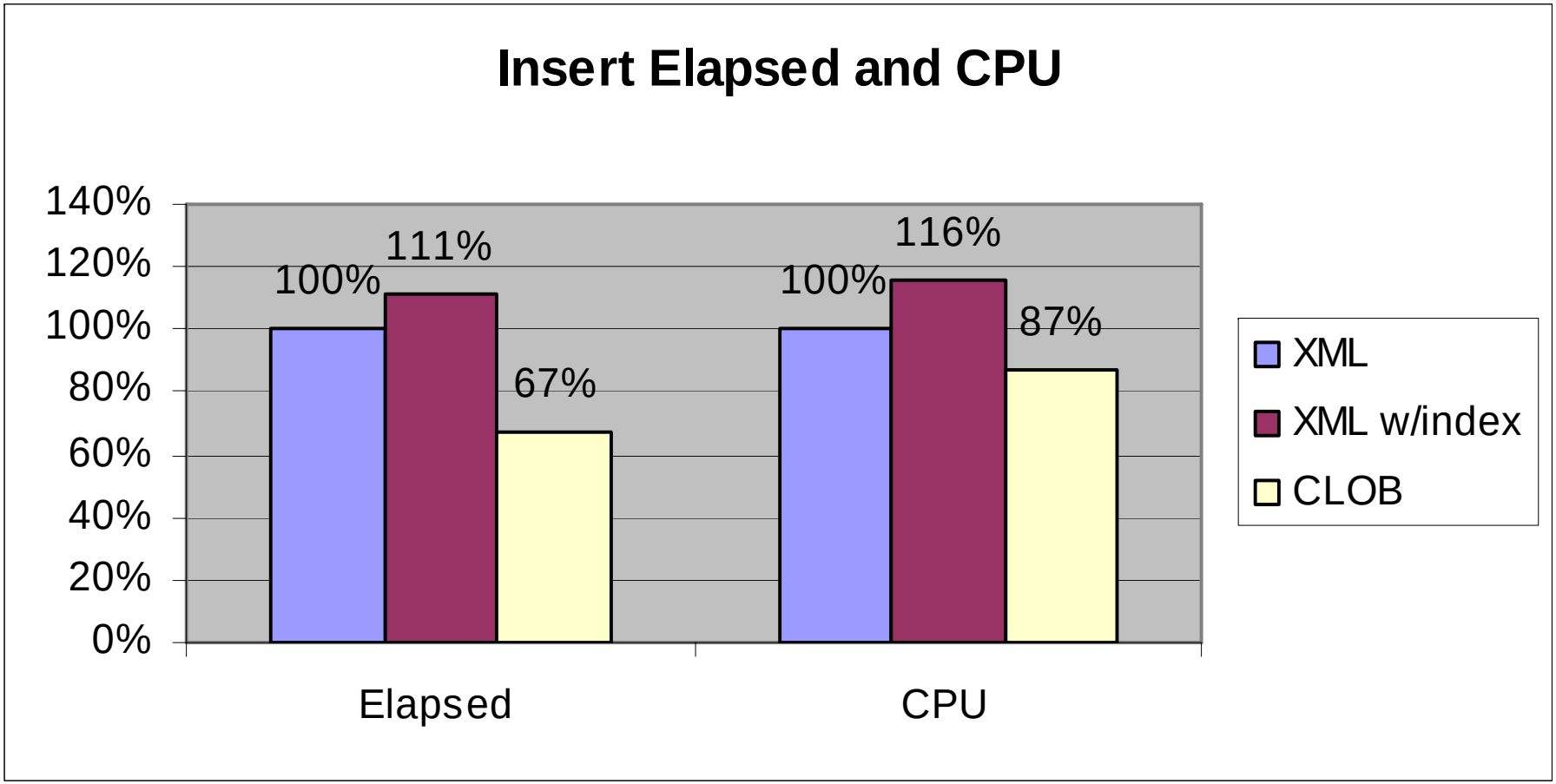


Measurement in March 2007, z9 DS8300, Single thread, Docs in EBCDIC

z/OS: Insert XML – with indexes

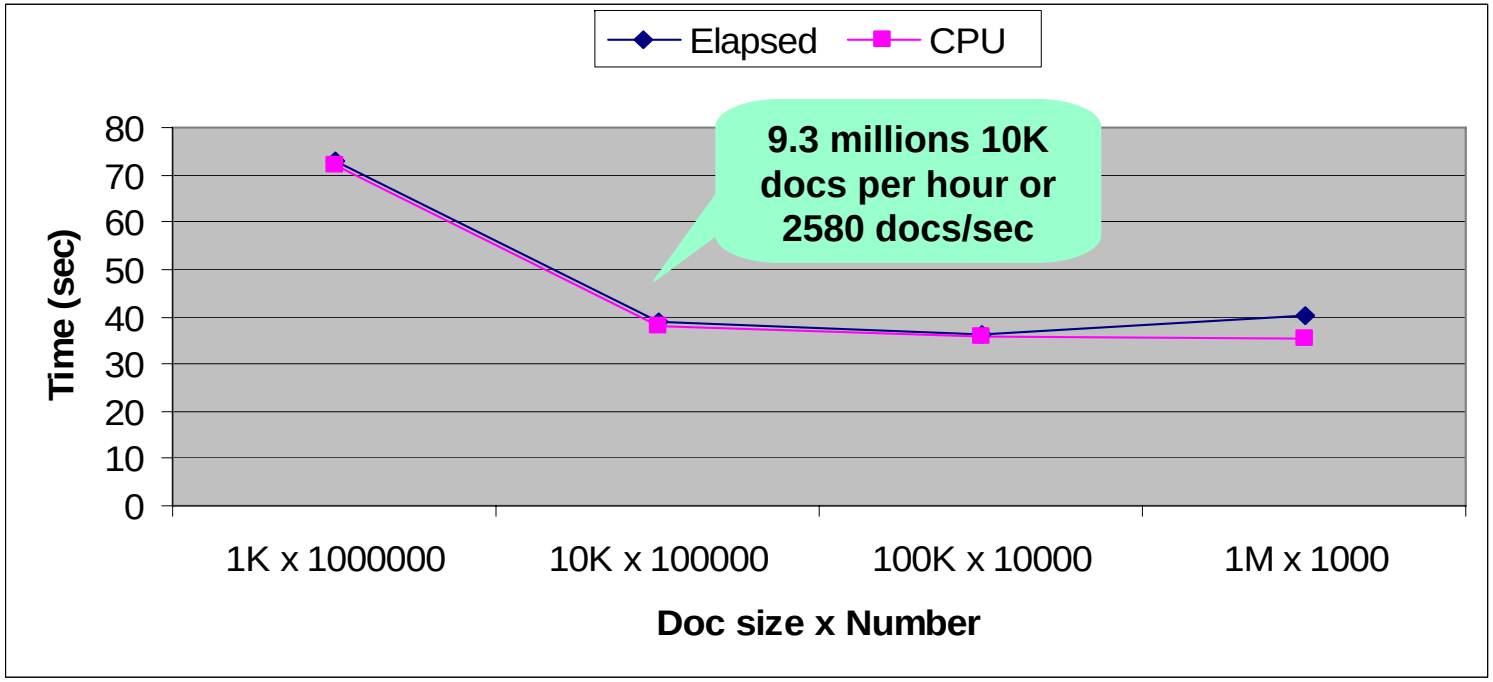


z/OS: Insert Performance



(average of 1K to 10M document insert performance)

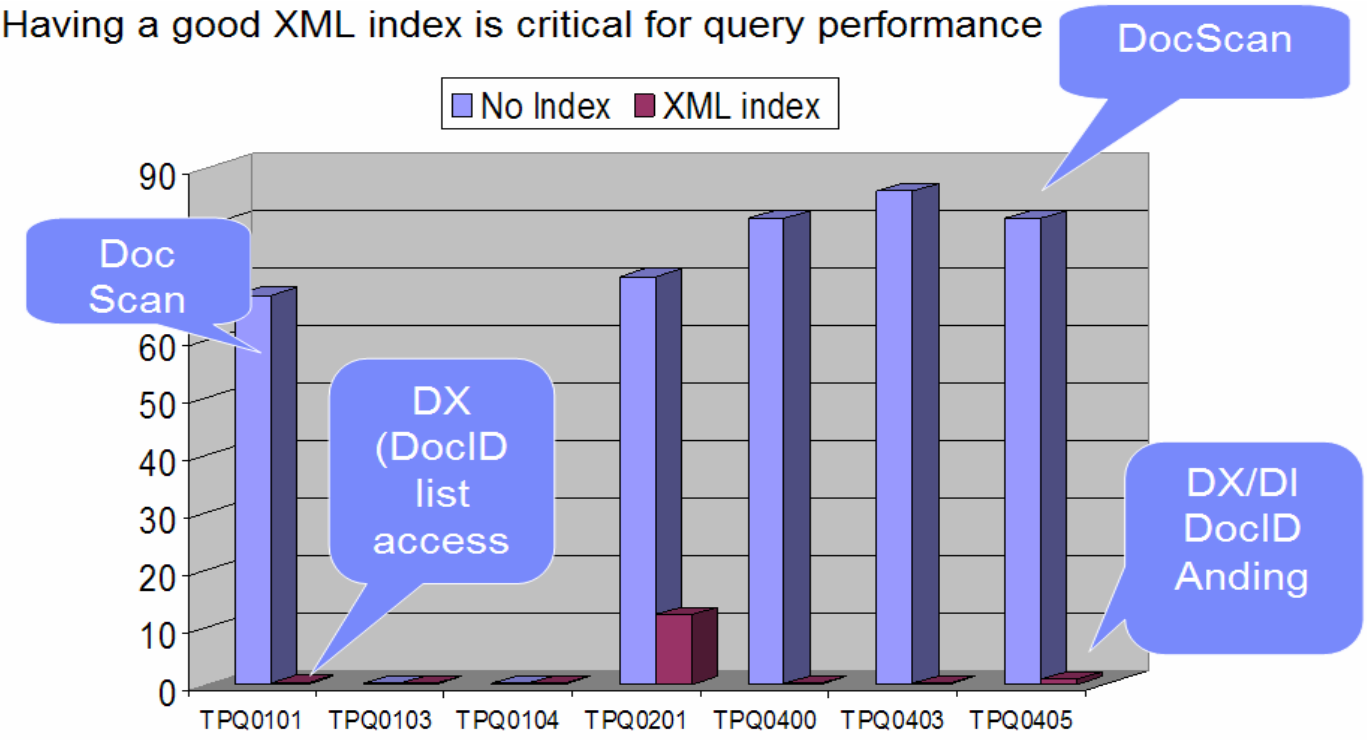
z/OS: Fetch Performance (Batch)



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

z/OS: XML Index Exploitation

- Having a good XML index is critical for query performance

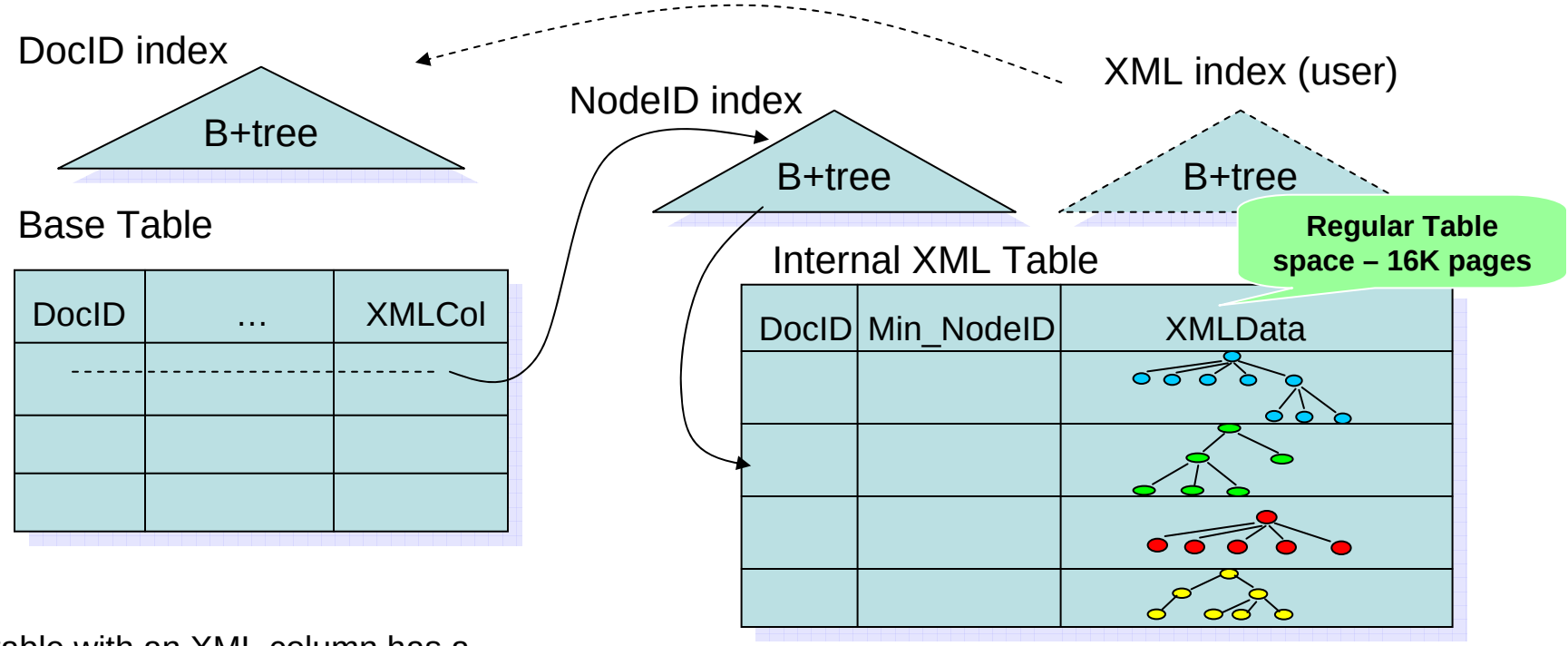


z/OS: Insert v.s. Validation v.s. Decomposition

Testcases	Insert elapsed	Valid. elapsed	Insert CPU	Valid. CPU	Decomp elapsed	Decomp CPU
Custacc - 10,000 docs 4-19K	35.90s	43.51s	8.37s	13.98s	43.51s	13.98s
Qcapture – 4k doc x 3000 times	1.89s	3.37s	0.97s	2.21s	10.33s	5.6s
Shred23 – three 2k docs x 1000	0.55s	2.02s	0.26s	0.62s		

z9-109, 3 x 1.7GHz, 12GB CS, ESS M800

z/OS: 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.

z/OS DBA Considerations

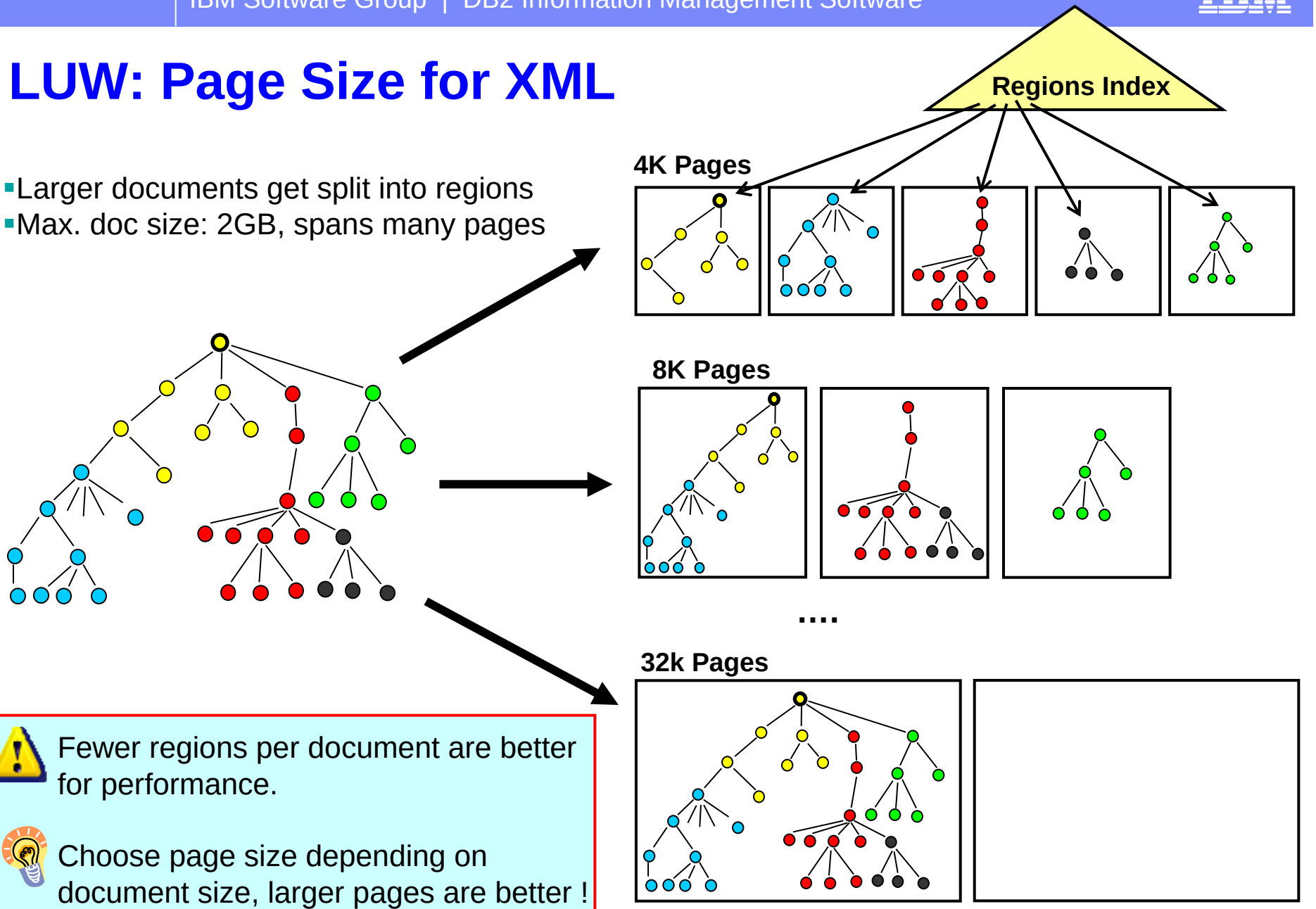
- Database administrators should treat XML database objects as they do in LOB database objects.
- Like LOB objects, XML objects contain data stored outside the base table space.
  - XML table spaces and index spaces must be consistent also with their related base table
- All the z/OS utilities either support or tolerate XML, with some specific XML keyword support.
- In addition, XML has more considerations, such as table space size limit, compression, and indexes.

z/OS Operation and Recovery

- To recover base table space, take image copies of all related objects
  - Use REPORT TABLESPACESET to obtain a list of related objects
  - Use QUIESCE TABLESPACESET to quiesce all objects in the related set
- Use SQL SELECT to query the SYSIBM.SYSXMLRELS table for relationships between base table spaces and XML table spaces
- COPYTOCOPY may be used to replicate image copies of XML objects.
- MERGECOPY may be used to merge incremental copies of XML table spaces.
- Point in Time recovery (RECOVER TOCOPY, TORBA, TOLOGPOINT)
  - All related objects, including XML objects must be recovered to a consistent point in time
- CHECK utilities to validate base table spaces with XML columns, XML indexes and related XML table spaces.
- If there is an availability issue with one object in the related set, availability of the others may be impacted.

LUW: Page Size for XML

- Larger documents get split into regions
- Max. doc size: 2GB, spans many pages



Fewer regions per document are better for performance.

Choose page size depending on document size, larger pages are better !

LUW: TableSpaces for XML

- Use **DMS** tablespaces for best performance !
- Choose page size large enough !
  - Need smaller pages for the relational part of the table?
  - Need smaller pages for the indexes?

```
CREATE TABLE mytable(c1 integer, c2 char(8),…,c9 double,  c10 XML)
IN mytspace1
INDEX IN mytspace2
LONG IN mytspace3
```

XDA (XML Data Area) will go to LONG if specified

Place XML data and/or indexes in separate table spaces to use different page sizes and separate tuning (prefetch size, etc.)…

…but only if *really* needed ! Otherwise keep it simple !

Platform Similarities for DB2 9 pureXML

	z/OS	LUW
XML Data Type & DDL (same syntax)	Yes	Yes
Query language	SQL/XML with XPath	XQuery & SQL/XML with XQuery
Indexing (same syntax)	XPath value index string and numeric (DECFLOAT). No hashed.	XPath value index string, numeric (double), date, timestamp
Validation	XML Schema, DSN_XMLVALIDATE() UDF, type annotation not kept	XML Schema, XMLVALIDATE() BIF, type annotation kept
I/U/D/M (same syntax)	Whole doc, lock on read	Whole doc w/ versioning during update
Host languages and APIs	JDBC, ODBC, C/C++, Java, COBOL, PL/I, Assembly, .Net	JDBC, ODBC, C/C++, Java, Cobol, .Net
Decomposition	Annotated schema-based	Annotated schema-based

Platform Differences for DB2 9 pureXML

	z/OS	LUW
Query language	XPath	XQuery
FETCH CONTINUE	Yes	No
XMLTable/XMLCast	No	Yes
Encoding	Table of any Encoding, UTF-8 internal for XML	Unicode database only
Stored proc	Use xLOB or string for XML	+XML as CLOB in parms
Partitioned tables	Yes	No DPF
Compression	Yes	No
Data sharing	Yes	No MPP
Utilities	Almost all	Fewer utilities

Agenda

- Overview: what you can do with pureXML in DB2 9
- Some usage scenarios
- Business value: comparison with existing approaches
- pureXML Common Features
- Platform Similarities and Differences
- Future Directions

LUW Viper 2 additions

- XQuery update facility – language for document updates
- Support for non-Unicode databases
- Automatic validation through use of “before” triggers
- Association of columns with specific set of schemas through use of “check” constraints.
- Performance improvements
- XSLT function
- Improvements in decomposition
- XML Replication
- Load
- Compatible schema evolution
- Passing parameters to Sqlquery()
- User-friendly publishing functions
- SQL/XML function simplification
- Federation
- Inlining for small documents (with compression support)

Future Directions in DB2 LUW

- DPF
- Range partitioning
- Compression
- View pushdown
- XML Index adviser
- Optimizer profiles (hints) for XML queries
- Run time performance improvements
- More utilities
- View / update statistics
- XQuery enhancements
- …

DB2 z/OS V9 Post-GA Improvements

- XMLTABLE
- XMLCAST

- More XPath functions:
  - fn:distinct-values
  - fn:max
  - fn:min
  - fn:upper-case
  - fn:lower-case
  - fn:translate
  - fn:tokenize
  - fn:matches
  - fn:replace

- Performance enhancement

Future Directions in DB2 z/OS

- Basic XQuery support in XMLQuery, XMLTable, XMLExists
  - FLWOR and constructors
  - More XML built-in types, XQuery functions
- Column association with schemas
- Trigger support
- XML in arguments for stored procedures and UDFs
- Sub-document update
- Schema validation inside the engine
- XSLT support
- …

V9 Manuals and more information

- LUW: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp
- z/OS: http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp
- XML WIKI:
  http://www-03.ibm.com/developerworks/wikis/display/db2xml/Home