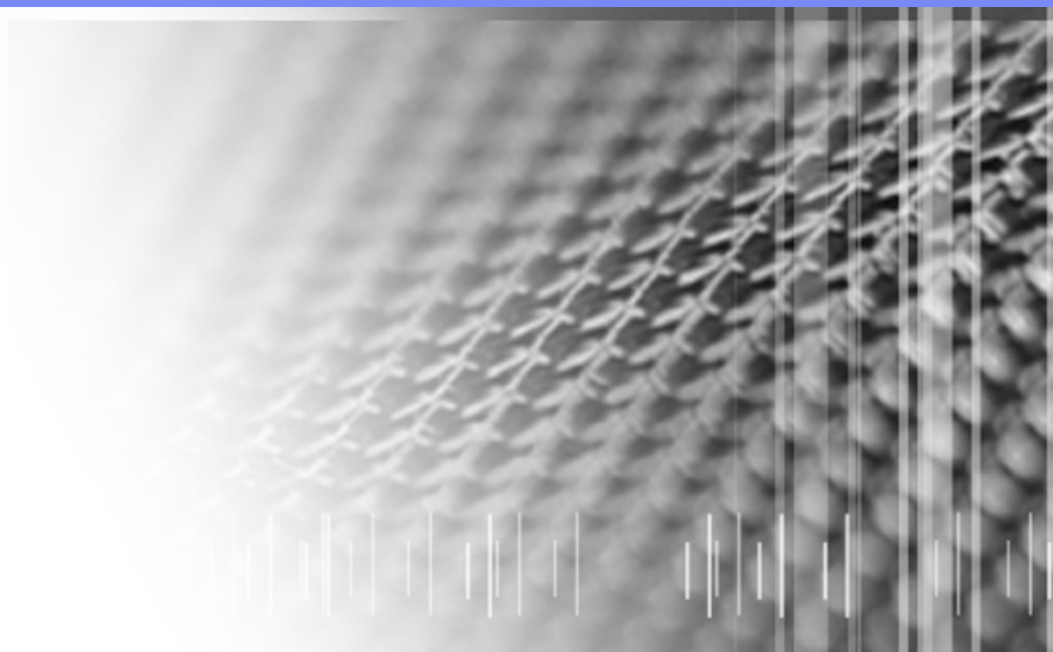




IBM Software Group

May 22, 2008

# DB2 pureXML™



@business on demand software

Jan-Eike Michels – IBM  
Information Management Standards  
[janeike@us.ibm.com](mailto:janeike@us.ibm.com)

© 2008 IBM Corporation



## Important Disclaimer

**THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.**

**WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.**

**IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.**

**IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.**

**NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:**

**CREATING ANY WARRANTY OR REPRESENTATION FROM IBM (OR ITS AFFILIATES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS); OR**

**ALTERING THE TERMS AND CONDITIONS OF THE APPLICABLE LICENSE AGREEMENT GOVERNING THE USE OF IBM SOFTWARE.**



# Agenda

- Introduction
  - ▶ What is XML?
  - ▶ XML storage options:  
XML-enabled databases vs. hybrid database  
(DB2 9)
- Processing XML with DB2 pureXML
  - ▶ Publishing
  - ▶ Storing
  - ▶ Querying
  - ▶ Indexing
  - ▶ Updating



# What is XML?

- XML – Extensible Markup Language
  - ▶ Relational data for structured data, XML for (semi-)structured data
  - ▶ Standardized, simple, self-describing markup language
    - <http://www.w3.org/TR/REC-xml/>
  - ▶ XML Schema can be used to define document structure, constraints, *etc.*
  - ▶ XQuery can be used to query XML
  - ▶ Today XML is used
    - for exchanging data, between applications/business partners
    - as storage format, e.g., for word processing
    - as intermediate format, then transformed into HTML or something else
    - by more and more businesses
    - by you...?



## XML Characteristics

- XML can represent 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)



# XML vs. Relational

```
<DEPARTMENT deptid="15" deptname="Sales">
  <EMPLOYEE>
    <EMPNO>10</EMPNO>
    <FIRSTNAME>CHRISTINE</FIRSTNAME>
    <LASTNAME>SMITH</LASTNAME>
    <PHONE>408-463-4963</PHONE>
    <SALARY>52750.00</SALARY>
  </EMPLOYEE>
  <EMPLOYEE>
    <EMPNO>27</EMPNO>
    <FIRSTNAME>MICHAEL</FIRSTNAME>
    <LASTNAME>THOMPSON</LASTNAME>
    <SALARY>41250.00</SALARY>
  </EMPLOYEE>
</DEPARTMENT>
```

## Relational

Set oriented

Structure

Strong schema

Strongly typed

Tabular data model

Flat

"Null"

ANSI/ISO

## XML

Sequences (*ordered!*)

Semi-structured

*Schema-flexibility*

Optionally typed

XML data model

Nested, hierarchical

Not there at all

W3C

### Department

DEPTID	DEPTNAME
15	Sales

### Employee

DEPTID	EMPNO	FIRSTNAME	LASTNAME	PHONE	SALARY
15	27	MICHAEL	THOMPSON	NULL	41250
15	10	CHRISTINE	SMITH	408-463-4963	52750



# Schema Evolution: Easy in XML, not in Relational

*“Employees are now allowed to have multiple phone numbers...”*

```
<DEPARTMENT deptid="15" deptname="Sales">
  <EMPLOYEE>
    <EMPNO>10</EMPNO>
    <FIRSTNAME>CHRISTINE</FIRSTNAME>
    <LASTNAME>SMITH</LASTNAME>
    <PHONE>408-463-4963</PHONE>
    <PHONE>415-010-1234</PHONE>
    <SALARY>52750.00</SALARY>
  </EMPLOYEE>
  <EMPLOYEE>
    <EMPNO>27</EMPNO>
    <FIRSTNAME>MICHAEL</FIRSTNAME>
    <LASTNAME>THOMPSON</LASTNAME>
    <PHONE>406-463-1234</PHONE>
    <SALARY>41250.00</SALARY>
  </EMPLOYEE>
</DEPARTMENT>
```

## Requires:

- Normalization of existing data !
- Change of applications

## Phone

EMPNO	PHONE
27	406-463-1234
10	415-010-1234
10	408-463-4963

## Department

DEPTID	DEPTNAME
15	Sales

**Costly!**

## Employee

DEPTID	EMPNO	FIRSTNAME	LASTNAME	PHONE	SALARY
15	27	MICHAEL	THOMPSON	406-463-1234	41250
15	10	CHRISTINE	SMITH	408-463-4963	52750



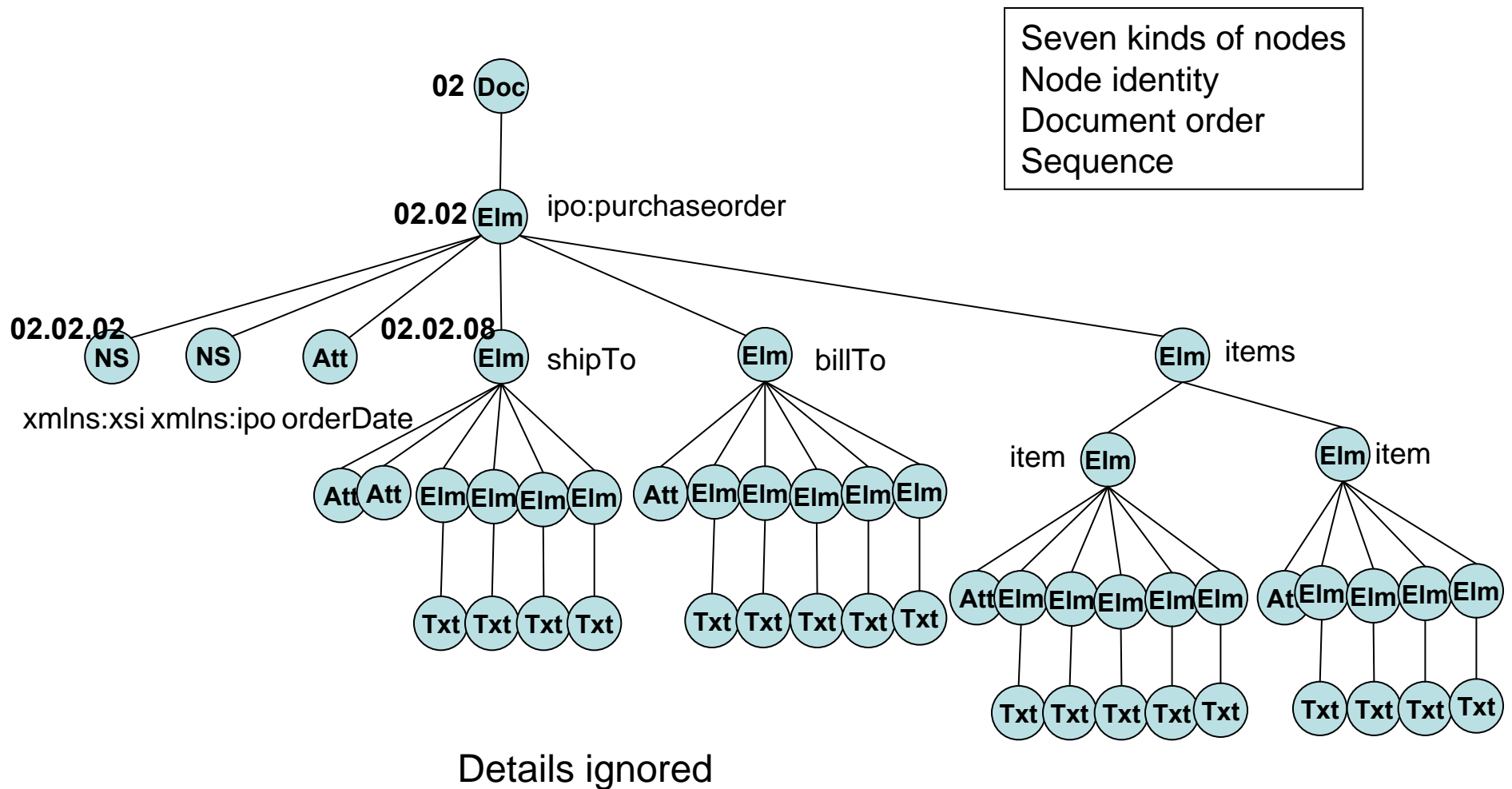
## An XML Purchase Order (serialized)

```
<?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/XQuery Data Model (XDM)





## A Data Record View and XPath Examples

```

- ipo:purchaseOrder orderDate="1999-12-01"
  - shipTo exportCode="1" xsi:type="ipo:UKAddress"
    -name: Helen Zoe
    -street: 47 Eden Street
    -city: Cambridge
    -postcode: CB1 1JR
  - billTo xsi:type="ipo:USAddress"
    -name: Robert Smith
    -street: 8 Oak Avenue
    -city: Old Town
    -state: PA
    -zip: 95819
  - items
    -item partNum="833-AA"
      -productName: Lapis necklace
      -quantity: 1
      -USPrice: 99.95
      -comment: Want this for the holidays!
      -shipDate: 1999-12-05
    -item partNum="926-AA"
      -productName: Baby Monitor
      -quantity: 1
      -USPrice: 39.98
      -shipDate: 1999-12-21

```

ShipTo name: /ipo:purchaseOrder/shipTo/name

BillTo: /ipo:purchaseOrder/billTo

Total amount:

fn:sum(/ipo:purchaseOrder/items/item/xs:decimal(USPrice))

Product names:

/ipo:purchaseOrder/items/item/productName

Ship date of the first item:

/ipo:purchaseOrder/items/item[1]/shipDate

Item info with product "Baby Monitor":

/ipo:purchaseOrder/items/item[productName = "Baby Monitor"]

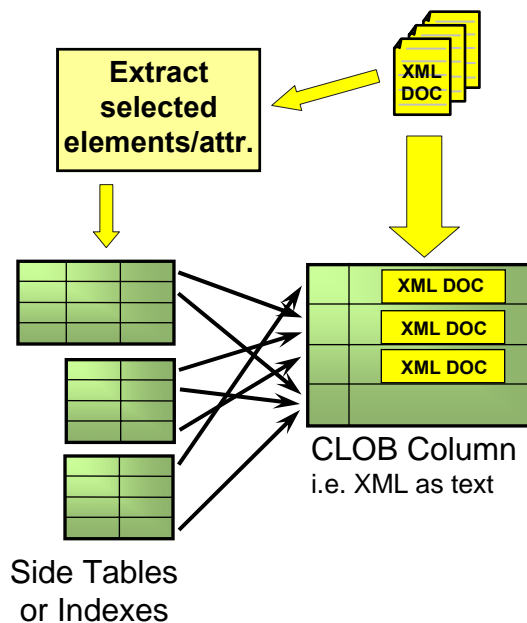
Sold price of product "Baby Monitor":

/ipo:purchaseOrder/items/item[productName = "Baby Monitor"]/USPrice



# XML Storage Options

Unstructured XML  
storage: XML as text

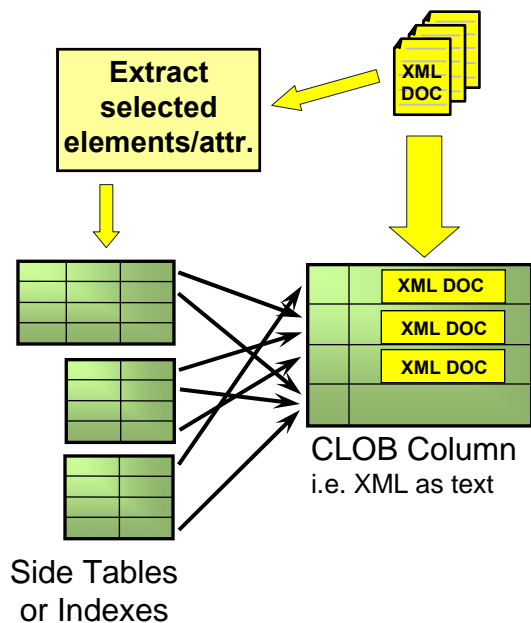


Any sub-document level access  
requires XML parsing – *slow*.



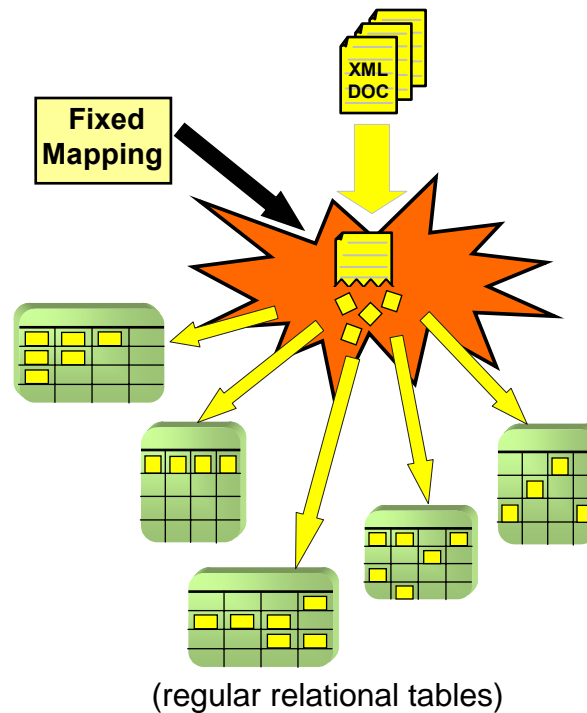
## XML Storage Options

### Unstructured XML storage: XML as text



Any sub-document level access requires XML parsing – **slow**.

### Shredding: XML → Relational

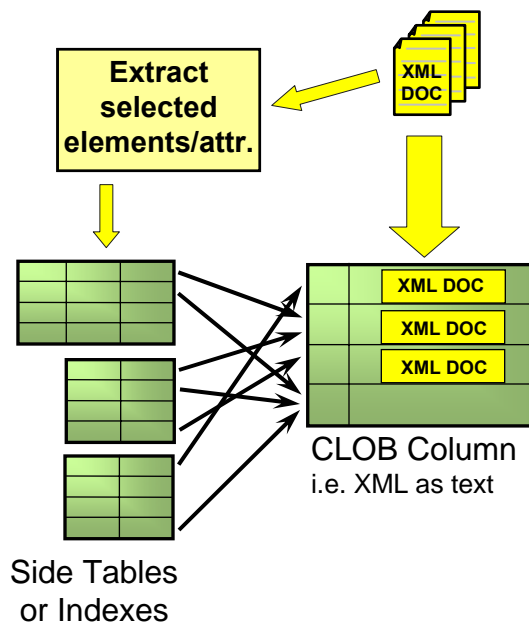


Mapping prevents XML schema changes, and is often **too complex**. XML reconstruction is **slow**.



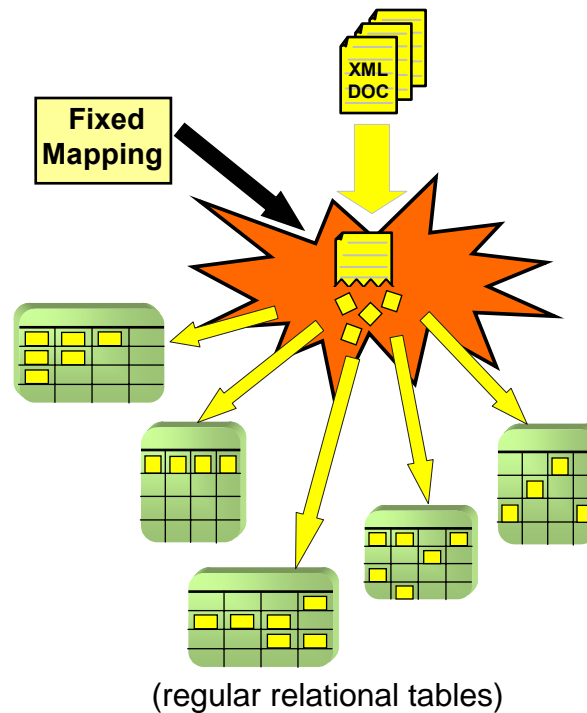
# XML Storage Options

## Unstructured XML storage: XML as text



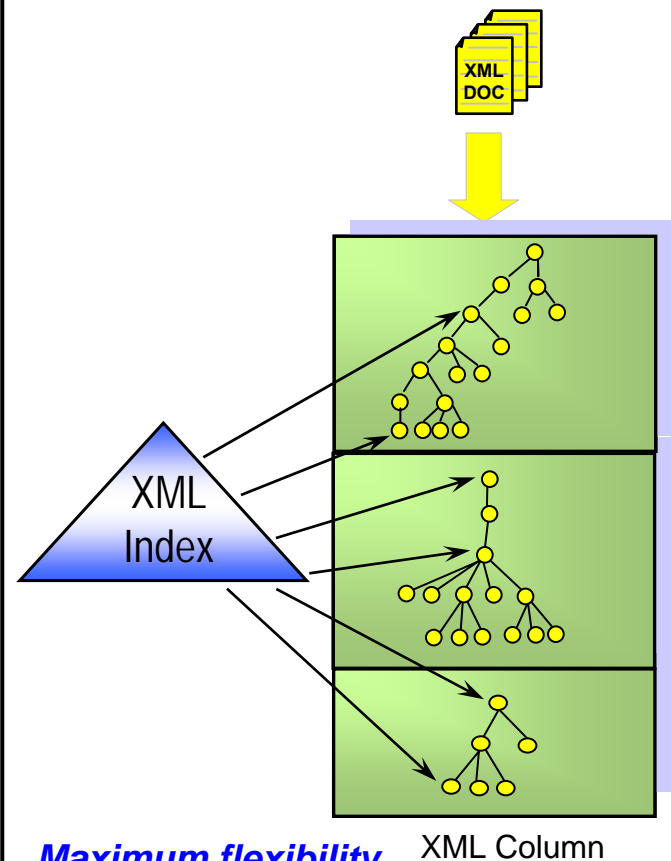
Any sub-document level access requires XML parsing – **slow**.

## Shredding: XML → Relational



Mapping prevents XML schema changes, and is often **too complex**. XML reconstruction is **slow**.

## DB2 9 pureXML: XML as XML



**Maximum flexibility and performance**



## Why Hybrid XML Databases?

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



## pureXML in DB2 9

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



## DB2 pureXML Advantages

- 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
- Unparalleled reliability, availability, scalability

Up to 10  
times



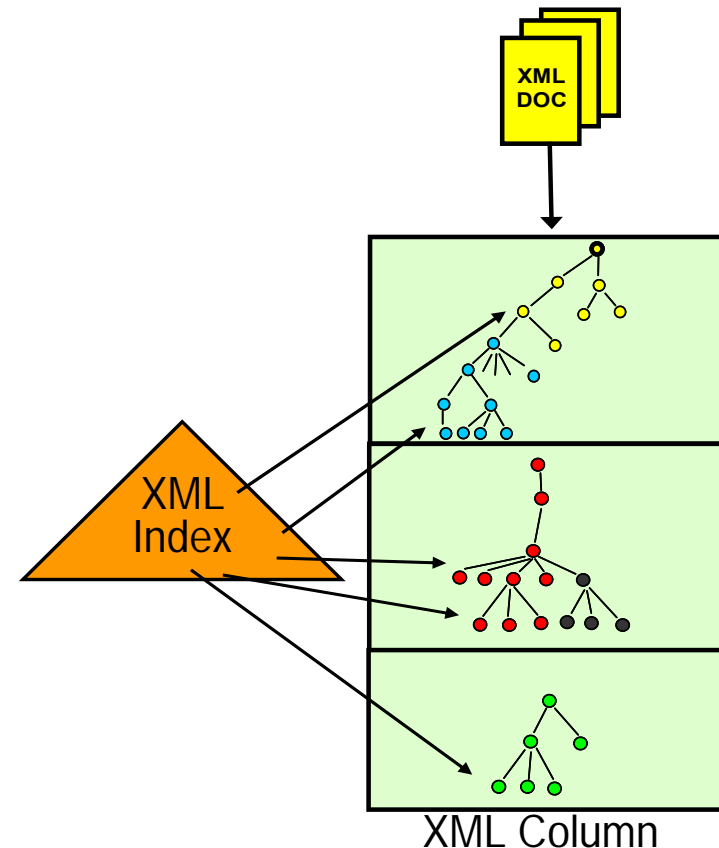
# Business Value of pureXML

- ***Speed up Application Development***
  - ▶ Reduced system and development complexity
  - ▶ Improved developer productivity
- ***Greater Business Agility***
  - ▶ Easily accommodate changes to data and schemas
  - ▶ Update applications rapidly and reduce maintenance costs
- ***Improved Business Insight***
  - ▶ Access to information in otherwise unexploited documents
  - ▶ Unprecedented application performance
- ***Consolidating converged information on System z***
  - ▶ Reduce floor space, power consumption, cooling cost
  - ▶ Consolidate information resources and reduce admin cost



## What You Can Do with pureXML

- Create tables with XML columns or add 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 or provide relational views
- Construct XML documents from relational and XML data
- All the utilities and tools support for XML





# Agenda

- *Introduction*
  - ▶ *What is XML?*
  - ▶ *XML storage options:*  
*XML-enabled databases vs. hybrid database (DB2 9)*
- **Processing XML with DB2 pureXML**
  - ▶ Publishing
  - ▶ Storing
  - ▶ Querying
  - ▶ Indexing
  - ▶ Updating
- *Specifics for DB2 for z/OS*
- *Specifics for DB2 for LUW*



## SQL/XML publishing/constructor functions

- XMLELEMENT
  - ▶ generates an XML element
  - ▶ defines attributes and namespaces using XMLATTRIBUTES and XMLNAMESPACES, respectively
- XMLFOREST
  - ▶ generates multiple XML elements
- XMLCOMMENT
  - ▶ generates an XQuery comment node
- XMLPI
  - ▶ generates an XQuery processing instruction node
- XMLTEXT
  - ▶ generates an XQuery text node
- XMLCONCAT
  - ▶ concatenates XML values
- XMLAGG
  - ▶ aggregates XML values across multiple relational tuples
- XMLDOCUMENT
  - ▶ wraps an XQuery document node around an XML value



## XMLEMENT

- XMLEMENT can produce simple element structures:

```
SELECT e.id, XMLEMENT (NAME "Emp", e.fname || ' ' || e.lname) AS "result"  
FROM employees e  
WHERE ... ;
```



ID	result
1001	<Emp>John Smith</Emp>
1006	<Emp>Mary Martin</Emp>



## XMLEMENT (continued)

- XMLEMENT can produce nested element structures:

```
SELECT e.id, XMLEMENT (NAME "Emp",  
                      XMLEMENT (NAME "name", e.lname ),  
                      XMLEMENT (NAME "hiredate", e.hire )  
          ) AS "result"
```

```
FROM employees e  
WHERE ... ;
```



ID	result
1001	<pre>&lt;Emp&gt;   &lt;name&gt;Smith&lt;/name&gt;   &lt;hiredate&gt;2000-05-24&lt;/hiredate&gt; &lt;/Emp&gt;</pre>
1006	<pre>&lt;Emp&gt;   &lt;name&gt;Martin&lt;/name&gt;   &lt;hiredate&gt;1996-02-01&lt;/hiredate&gt; &lt;/Emp&gt;</pre>



## XMLELEMENT (continued)

- XMLELEMENT can produce elements with mixed content:

```
SELECT e.id, XMLELEMENT (NAME "Emp",  
                        'Employee ',  
                        XMLELEMENT (NAME "name", e.lname ),  
                        ' was hired on ',  
                        XMLELEMENT (NAME "hiredate", e.hire )  
                        ) AS "result"
```

FROM employees e  
WHERE ... ;



ID	result
1001	<Emp> Employee <name>Smith</name> was hired on <hiredate>2000-05-24</hiredate> </Emp>
1006	<Emp> Employee <name>Martin</name> was hired on <hiredate>1996-02-01</hiredate> </Emp>



## XMLELEMENT (continued)

- XMLELEMENT can take subqueries as arguments:

```
SELECT e.id, XMLELEMENT (NAME "Emp",  
                        XMLELEMENT (NAME "name", e.lname ),  
                        XMLELEMENT (NAME "dependants",  
                                (SELECT COUNT (*)  
                                FROM dependants d  
                                WHERE d.parent = e.id))  
                        ) AS "result"  
FROM employees e  
WHERE ... ;
```



ID	result
1001	<Emp> <name>Smith</name> <dependants>3</dependants> </Emp>



## XMLATTRIBUTES (within XMLELEMENT)

- Attribute specifications must be bracketed by XMLATTRIBUTES keyword and must appear directly after element name and optional namespace declaration.
- Each attribute can be named implicitly or explicitly.

```
SELECT e.id, XMLELEMENT (NAME "Emp",  
                        XMLATTRIBUTES (e.id, e.lname AS "name")) AS "result"  
FROM employees e  
WHERE ... ;
```



ID	result
1001	<Emp ID="1001" name="Smith" />
1006	<Emp ID="1206" name="Martin" />



## XMLNAMESPACES (within XMLEMENT)

- Namespace declarations are bracketed by XMLNAMESPACES keyword and must appear directly after element name.

```
SELECT empno,  
       XMLEMENT(NAME "ex:employee",  
                 XMLNAMESPACES('http://www.example.com' AS "ex"),  
                 XMLATTRIBUTES(e.workdept AS "ex:department"),  
                 e.lastname  
       ) AS "result"  
FROM employees e  
WHERE e. job = 'ANALYST';
```



ID	result
1130	<ex:employee xmlns:ex="http://www.example.com" ex:department="C01">QUINTANA</ex:employee>
1140	<ex:employee xmlns:ex="http://www.example.com" ex:department="C01">NICHOLLS</ex:employee>



## XMLCONCAT

- Produces an XML value given two or more expressions of XML type.
- If any of the arguments evaluate to the null value, it is ignored.

```
SELECT e.id, XMLCONCAT (XMLELEMENT ( NAME "first", e.fname),  
                        XMLELEMENT ( NAME "last", e.lname)  
                        ) AS "result"  
FROM employees e ;
```



ID	result
1001	<first>John</first> <last>Smith</last>
1006	<first>Mary</first> <last>Martin</last>



# XMLFOREST

- Produces a sequence of XML elements given named expressions as arguments. Arguments can also contain a list of namespace declarations
- Element can have an explicit name:
  - ▶ `e.salary AS "empSalary"`
- Element can have an implicit name, if the expression is a column reference:
  - ▶ `e.salary`



## XMLFOREST – Example

```
SELECT e.id, XMLFOREST (e.hire, e.dept AS "department") AS "result"  
FROM employees e  
WHERE ... ;
```



ID	result
1001	<HIRE>2000-05-24</HIRE> <department>Accounting</department>
1006	<HIRE>1996-02-01</HIRE> <department>Shipping</department>



# XMLAGG

- An aggregate function, similar to SUM, AVG, etc.
  - ▶ The argument for XMLAGG must be an expression of XML type.
- Semantics
  - ▶ For each row in a group G, the expression is evaluated and the resulting XML values are concatenated to produce a single XML value as the result for G.
  - ▶ An ORDER BY clause can be specified to order the results of the argument expression before concatenating.
  - ▶ All null values are dropped before concatenating.
  - ▶ If all inputs to concatenation are null or if the group is empty, the result is the null value.



## XMLAGG - Example

```
SELECT XMLELEMENT ( NAME "Department",  
                  XMLATTRIBUTES ( e.dept AS "name" ),  
                  XMLAGG (XMLELEMENT (NAME "emp", e.lname))  
                  ) AS "dept_list",  
      COUNT(*) AS "dept_count"  
FROM employees e  
GROUP BY dept ;
```



dept_list	dept_count
<Department name="Accounting"> <emp>Yates</emp> <emp>Smith</emp> </Department>	2
<Department name="Shipping"> <emp>Oppenheimer</emp> <emp>Martin</emp> </Department>	2



## XMLAGG and ORDER BY - Example

```
SELECT XMLELEMENT ( NAME "Department",  
                  XMLATTRIBUTES ( e.dept AS "name" ),  
                  XMLAGG (XMLELEMENT (NAME "emp", e.lname)  
                        ORDER BY e.lname)  
                  ) AS "dept_list", COUNT(*) AS "dept_count"  
FROM employees e  
GROUP BY dept ;
```



dept_list	dept_count
<Department name="Accounting"> <emp>Smith</emp> <emp>Yates</emp> </Department>	2
<Department name="Shipping"> <emp>Martin</emp> <emp>Oppenheimer</emp> </Department>	2



# XMLCOMMENT & XMLPI

## ■ XMLCOMMENT

- ▶ Creates an XQuery comment node (XML comment)
- ▶ One mandatory character string input argument

`XMLCOMMENT('This is a comment')`

→ `<!--This is a comment-->`

## ■ XMLPI

- ▶ Creates an XQuery processing instruction node
- ▶ Two input arguments:
  - One mandatory name for the PI (an SQL identifier); a.k.a. the PI target
  - One optional character string input argument; defines the content of the PI target

`XMLPI(NAME "includeFile", '/POs/template.hls')`

→ `<?includeFile /POs/template.hls?>`



## XMLTEXT

- Creates an XQuery text node
- One mandatory character string input argument

Seqno	Text	Emphasized_Text
1	We need an	XMLText()
2	constructor for	mixed content
3	to be generated by	XMLAgg
4	. This example	illustrates
5	the point. You	cannot
6	generate the same result using	XMLElement.

- Example:

```
SELECT XMLELEMENT (NAME "p",
  XMLAGG (XMLCONCAT
    (XMLText (T.Text),
    XMLELEMENT (NAME "em", T.Emphasized_text) )
  ORDER BY T.Seqno ) ) AS "result"
```

FROM T

=>

result

```
<p>We need an <em>XMLText()</em> constructor for <em>mixed
content</em> to be generated by <em>XMLAgg</em>. This example
<em>illustrates</em> the point. You <em>cannot</em> generate
the same result using <em>XMLElement.</em></p>
```



## SQL/XML Publishing: *Turn relational data into XML*

```
SELECT XMLELEMENT(NAME "Department",
      XMLATTRIBUTES (e.dept AS "name" ),
      XMLAGG( XMLELEMENT(NAME "emp", e.firstname) )
      ) AS "dept_list"

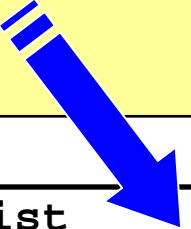
FROM employee e
WHERE .....
GROUP BY e.dept;
```

SQL/XML



firstname	lastname	dept
SEAN	LEE	A00
MICHAEL	JOHNSON	B01
VINCENZO	BARELLI	A00
CHRISTINE	SMITH	A00

dept\_list



```
<Department name="A00">
  <emp>CHRISTINE</emp>
  <emp>VINCENZO </emp>
  <emp>SEAN</emp>
</Department>

<Department name="B01">
  <emp>MICHAEL</emp>
</Department>
```

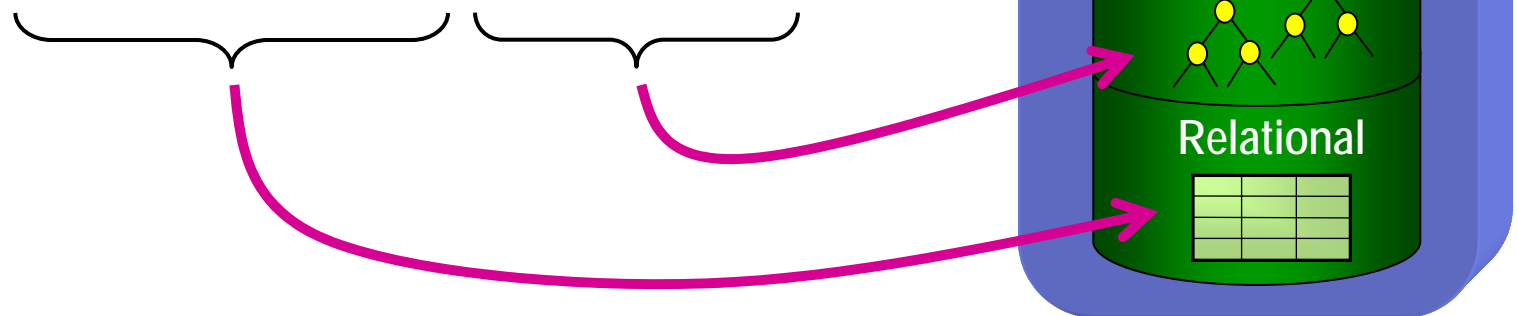


## The XML data type & pureXML storage

- Tables can contain relational and/or XML columns
- Relational columns stored in tabular format
- XML columns stored in a **parsed** hierarchical format
- No XML parsing for query evaluation → **High Performance**
- Specialized indexing for XML & relational → **High Performance**

create table **dept** (deptID char(8),...,deptdoc **xml**);

deptID	...	deptdoc
ABCD1234	...	<dept> ... <emp>...</emp> </dept>
...	...	...





## XML Data Type (*continued*)

### ■ Examples:

```
EXEC SQL BEGIN DECLARE SECTION;
```

```
  CHAR hv_id(8);
```

```
    SQL TYPE IS XML AS CLOB (1M) hv_xr;
```

```
EXEC SQL END DECLARE SECTION;
```

```
EXEC SQL INSERT INTO dept (deptID, deptdoc)
```

```
VALUES (:hv_id, :hv_xr);
```

```
EXEC SQL SELECT d.deptID, d.deptdoc
```

```
INTO :hv_id, :hv_xr
```

```
FROM dept d
```

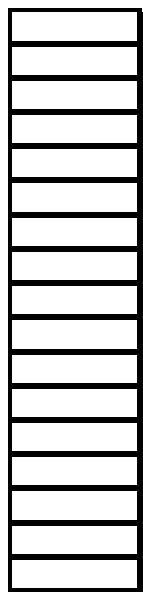
```
WHERE d.deptID = 'ABCD1234';
```

```
INSERT INTO dept VALUES('ABCD4321', XMLPARSE(DOCUMENT  
:hv_xr PRESERVE WHITESPACE) );
```

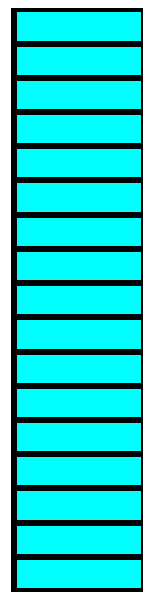


## XML Schema Support & Validation in DB2 9

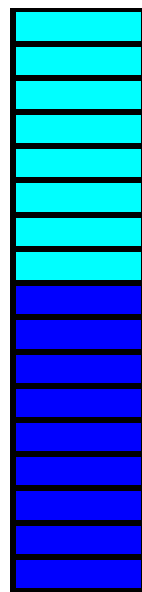
**Documents for zero, one, or many schemas per XML column:**



No Schema



One Schema



Schema V1  
& Schema V2



Documents  
w/ and w/o  
schema



Any mix you want!

**XML Schema Flexibility !**



## XML Schema Support & Validation in DB2 9

- Schemas are registered & stored in the DB2 Schema Repository (XSR) for fast and stable access.
- Use of XML Schemas is optional, on a per-document basis
- No need for a fixed schema per XML column
- Validation per document (i.e. per row)
- Zero, one, or many schemas per XML column
  - ▶ For example: different versions of the same schema, or schemas with conflicting definitions
- Mix validated & non-validated documents in 1 XML column



## XMLVALIDATE

- Ensure that XML values are valid according to a certain XML schema
  - ▶ XMLVALIDATE() validates and annotates XML values
  - ▶ Requires XML schema to be registered in the database
  - ▶ Multiple options to identify the XML schema to use
- Example:

```
INSERT INTO POrders VALUES ('WO20071204', CURRENT  
TIMESTAMP, 'R', 'W',  
XMLVALIDATE (XMLPARSE (DOCUMENT  
'<purchaseOrder>...</purchaseOrder>'  
PRESERVE WHITESPACE)  
ACCORDING TO XMLSCHEMA ID PORDER) );
```



## Document Validation with XML Schemas (LUW)

```
create table dept(deptID char(8), deptdoc xml);
```

**Validation is optional, and per document (per row):**

```
insert into dept values (?, ?)
insert into dept values (?, xmlvalidate(? according to
                                xmlschema id "dept.schema1" ) )
```

**Or, the DBA can force validation at the column level:**

```
CREATE TRIGGER TR1 NO CASCADE BEFORE INSERT ON dept
REFERENCING NEW AS n
FOR EACH ROW MODE DB2SQL
BEGIN ATOMIC
set n.deptdoc = xmlvalidate(n.deptdoc according to
                            xmlschema id "dept.schema1" );
END
```



## Document Validation with XML Schemas (z/OS)

```
create table dept(deptID char(8), deptdoc xml);
```

**Validation is optional, and per document (per row):**

```
insert into dept values (?, ?)
```

```
INSERT into dept VALUES( 'ABCD1234', XMLPARSE(DOCUMENT  
DSN_XMLValidate(:lobDe, 'SYSXSR.myDeptSchema' ) ) );
```



## Querying XML

- XMLQUERY
  - ▶ Evaluates an XQuery expression
- XMLTABLE
  - ▶ Converts XML into a table
- XMLEXISTS
  - ▶ Checks whether the result of an XQuery expression is not the empty sequence
- XMLCAST
  - ▶ Converts SQL to XML values and vice versa



# XMLQUERY

- Evaluates an XQuery or XPath expression
  - ▶ Provided as a character string literal
- Allows for optional arguments to be passed in
  - ▶ Zero or more named arguments
  - ▶ At most one unnamed argument can be passed in as the XQuery context item
  - ▶ Arguments can be of any predefined SQL data type incl. XML
  - ▶ Non-XML arguments will be implicitly converted using XMLCAST
- Returns a sequence of XQuery nodes



## Querying XML using SQL or SQL/XML

create table dept(deptID char(8), deptdoc xml)

```
select deptID, deptdoc  
from dept where deptID = "ABCD1234"
```

SQL

```
select deptID,  
       xmlquery('for $i in $d/dept  
                let $j := $i//name  
                return $j' passing deptdoc as "d")  
from dept  
where deptID LIKE "ABC%"  
and xmlexists('$d/dept[@bldg = 101]' passing deptdoc as "d")
```

SQL/XML  
XQuery



## XMLQUERY – Examples (LUW)

```
SELECT XMLQUERY('<e>hi</e>' RETURNING SEQUENCE BY REF)
      AS "result"
```

```
FROM T
```

```
=>
```

result
<e>hi</e>

```
SELECT XMLQUERY('for $i in $po/purchaseOrder/customer
                 where $i/zip[@type="US"]="95141" return $i/name'
                 PASSING BY REF po AS "po"
                 RETURNING SEQUENCE) AS "Name_elements"
```

```
FROM POrders
```

```
=>
```

Name_elements
<name>Miller</name>
<name>Smith</name>
<name>Johnson</name>
<name>Smith</name>



## XMLQUERY – Examples (z)

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

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



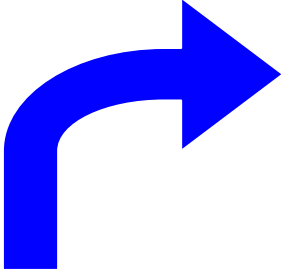
## XMLTABLE

- Transforming XML data into table format
- Evaluates an XQuery or XPath expression – the “row pattern”
  - ▶ Provided as a character string literal
- Allows for optional arguments to be passed in
  - ▶ Just like XMLQuery
- Element/attribute values are mapped to column values using path expressions (PATH) – the “path pattern”
  - ▶ Provided as a character string literal
- Names and SQL data types for extracted values/columns need to be specified
- Default values for “missing” columns can be provided
- ORDINALITY column can be generated
  - ▶ Contains a sequential number of the corresponding XQuery item in the XQuery sequence (result of the row pattern)



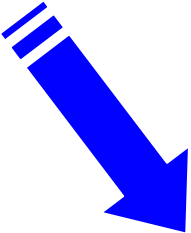
# XMLTABLE: Return XML in tabular format

SQL/XML  
XQuery



```
SELECT X.* FROM dept,
  XMLTABLE ('$d/dept/employee' passing deptdoc as "d")
  COLUMNS
    "emplID"      INTEGER          PATH '@id',
    "firstname"   VARCHAR(30)      PATH 'name/first',
    "lastname"    VARCHAR(30)      PATH 'name/last',
    "office"      INTEGER          PATH 'office') AS "X"
```

```
<dept bldg="101">
  <employee id="901">
    <name>
      <first>John</first>
      <last>Doe</last>
    </name>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>
      <first>Peter</first>
      <last>Pan</last>
    </name>
    <office>216</office>
  </employee>
</dept>
```



emplID	firstname	lastname	office
901	John	Doe	344
902	Peter	Pan	216



# XMLCAST

- Convert an SQL value into an XML value
  - ▶ Values of SQL predefined types are cast to XQuery atomic types using
    - The defined mapping of SQL types/values to XML Schema types/values
    - The semantics of XQuery's cast expression
- Convert an XML value into an SQL value
  - ▶ XML values are converted to values of SQL predefined types using a combination of
    - The defined mapping of SQL types to XML Schema types
    - XQuery's fn:data function
    - XQuery's cast expression
    - SQL's CAST specification
  - ▶ An XML value that is the empty sequence is converted to a NULL value of the specified SQL data type
- Note: XMLCAST to/from character strings is different from XMLSERIALIZE and XMLPARSE



## XMLCAST - Example

```
SELECT XMLCAST (  
    XMLQUERY ('<e>hi</e>' RETURNING SEQUENCE BY REF)  
    AS VARCHAR(20) ) AS "result"  
FROM T
```

=>

Result
hi



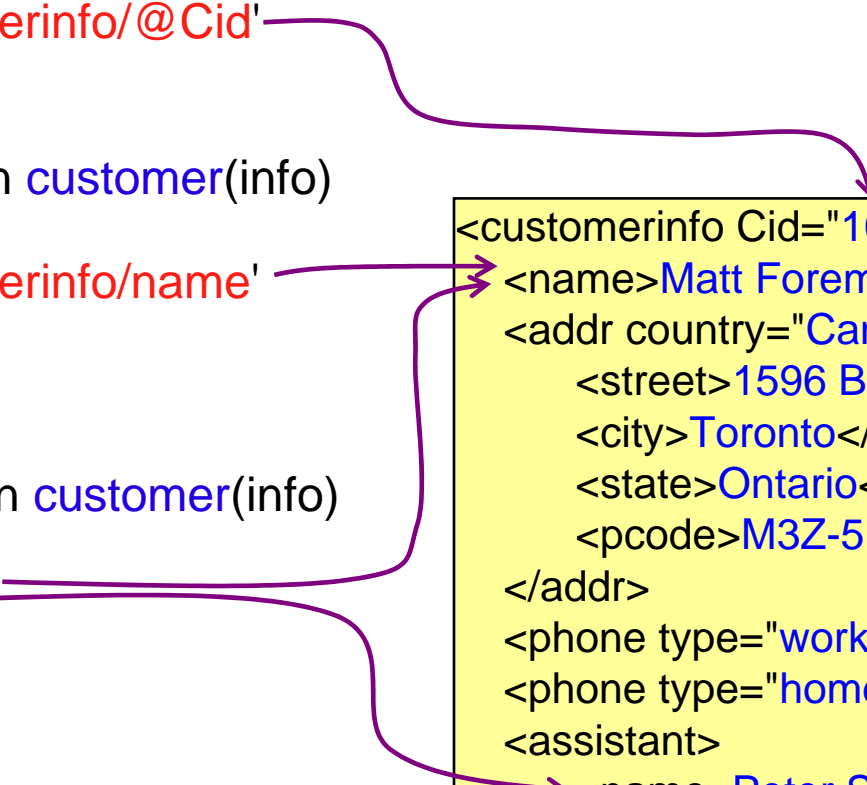
# XML Indexing Examples

```
create table customer( info XML);
```

create unique index **idx1** on **customer**(info)  
generate key using  
xpath '/customerinfo/@Cid'  
as sql double;

create index **idx2** on **customer**(info)  
generate key using  
xpath '/customerinfo/name'  
as sql varchar(40);

create index **idx3** on **customer**(info)  
generate key using  
xpath '//name'  
as sql varchar(40);



```
<customerinfo Cid="1004">  
  <name>Matt Foreman</name>  
  <addr country="Canada">  
    <street>1596 Baseline</street>  
    <city>Toronto</city>  
    <state>Ontario</state>  
    <pcode>M3Z-5H9</pcode>  
  </addr>  
  <phone type="work">905-555-4789</phone>  
  <phone type="home">416-555-3376</phone>  
  <assistant>  
    <name>Peter Smith</name>  
    <phone type="home">416-555-3426</phone>  
  </assistant>  
</customerinfo>
```



## XML Indexing Examples

```
create table customer( info XML);
```

create **unique** index **idx1** on **customer**(info)  
generate key using  
xmlpattern '/customerinfo/@Cid'  
as sql double;

create index **idx2** on **customer**(info)  
generate key using  
xmlpattern '/customerinfo/name'  
as sql varchar(40);

create index **idx3** on **customer**(info)  
generate key using  
xmlpattern '//name'  
as sql varchar(40);

create index **idx4** on **customer**(info)  
generate key using  
xmlpattern '/customerinfo/phone'  
as sql varchar(40);

```
<customerinfo Cid="1004">  
  <name>Matt Foreman</name>  
  <addr country="Canada">  
    <street>1596 Baseline</street>  
    <city>Toronto</city>  
    <state>Ontario</state>  
    <pcode>M3Z-5H9</pcode>  
  </addr>  
  <phone type="work">905-555-4789</phone>  
  <phone type="home">416-555-3376</phone>  
  <assistant>  
    <name>Peter Smith</name>  
    <phone type="home">416-555-3426</phone>  
  </assistant>  
</customerinfo>
```

The diagram illustrates the mapping of SQL XML indexes to specific XML elements. Arrows point from the following definitions to the XML structure:

- idx1** (unique index on `@Cid`) points to the `Cid="1004"` attribute of the root `customerinfo` element.
- idx2** (index on `/customerinfo/name`) points to the `<name>Matt Foreman</name>` element.
- idx3** (index on `//name`) points to the `<name>Peter Smith</name>` element inside the `<assistant>` sub-element.
- idx4** (index on `/customerinfo/phone`) points to the `<phone type="work">905-555-4789</phone>` element.

**Easy to index repeating elements**



# XML Indexes for High Query Performance

- Define 0, 1 or multiple XML Value Indexes per XML column
- Index **any** elements or attributes, incl. mixed content
- Index definition uses **XPath** to specify which elements/attributes to index (and which not to)
- Can index **all** elements/attributes, but not forced to do so
- Can index **repeating elements**  
⇒ 0 , 1 or multiple index entries per document
- New **XML-specific join and query evaluation methods**, evaluate multiple predicates concurrently with minimal index I/O

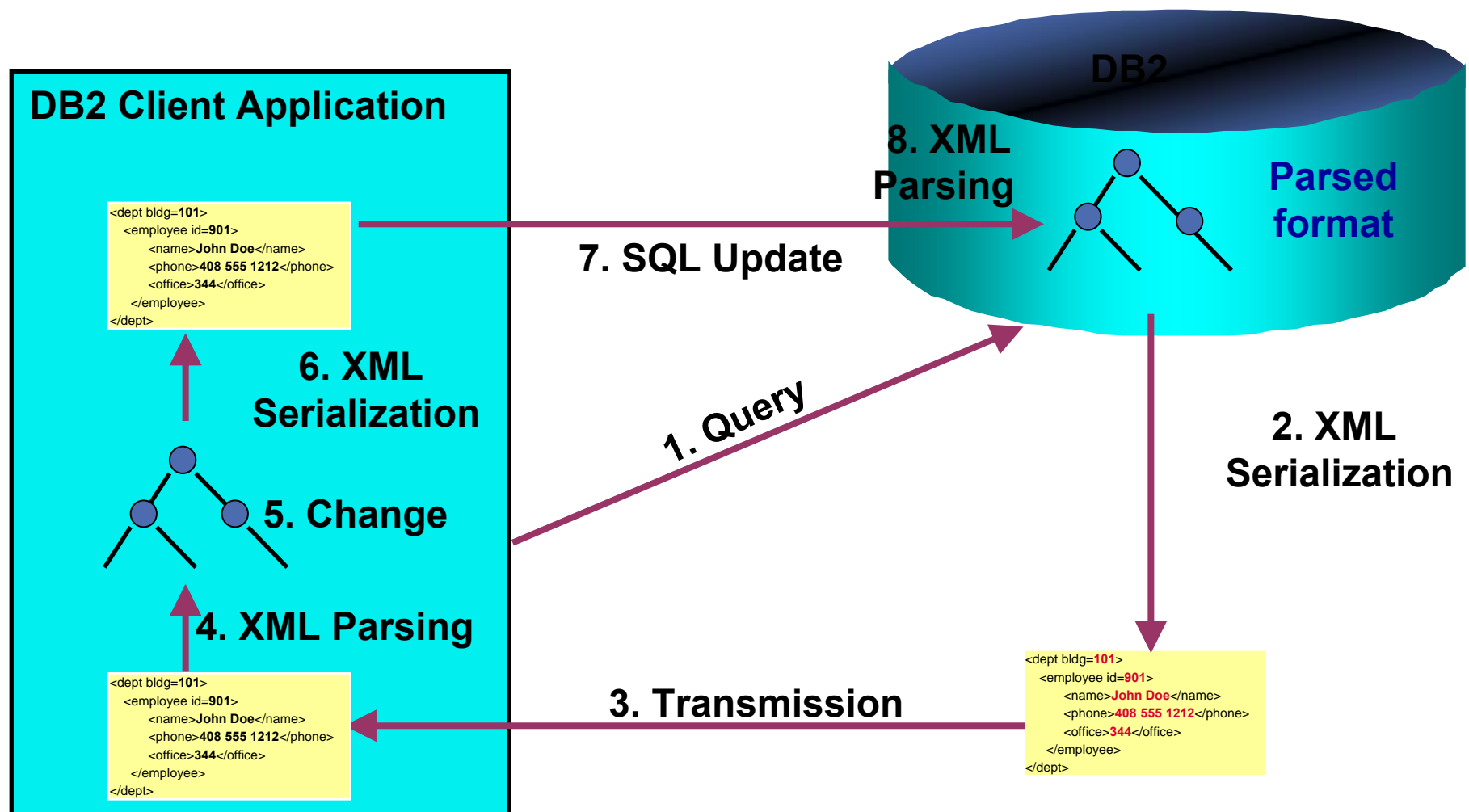


## Updating XML in DB2 (LUW 9.5)

- XQuery Update Facility, standardized by the W3C
- XQuery introduces TRANSFORM expressing, for node-level XML document manipulation:
  - ▶ replace value, replace node
  - ▶ insert node, insert after, insert before, insert as last, insert as first
  - ▶ delete node
  - ▶ rename node
- Let's take a look at why this is better than in DB2 9

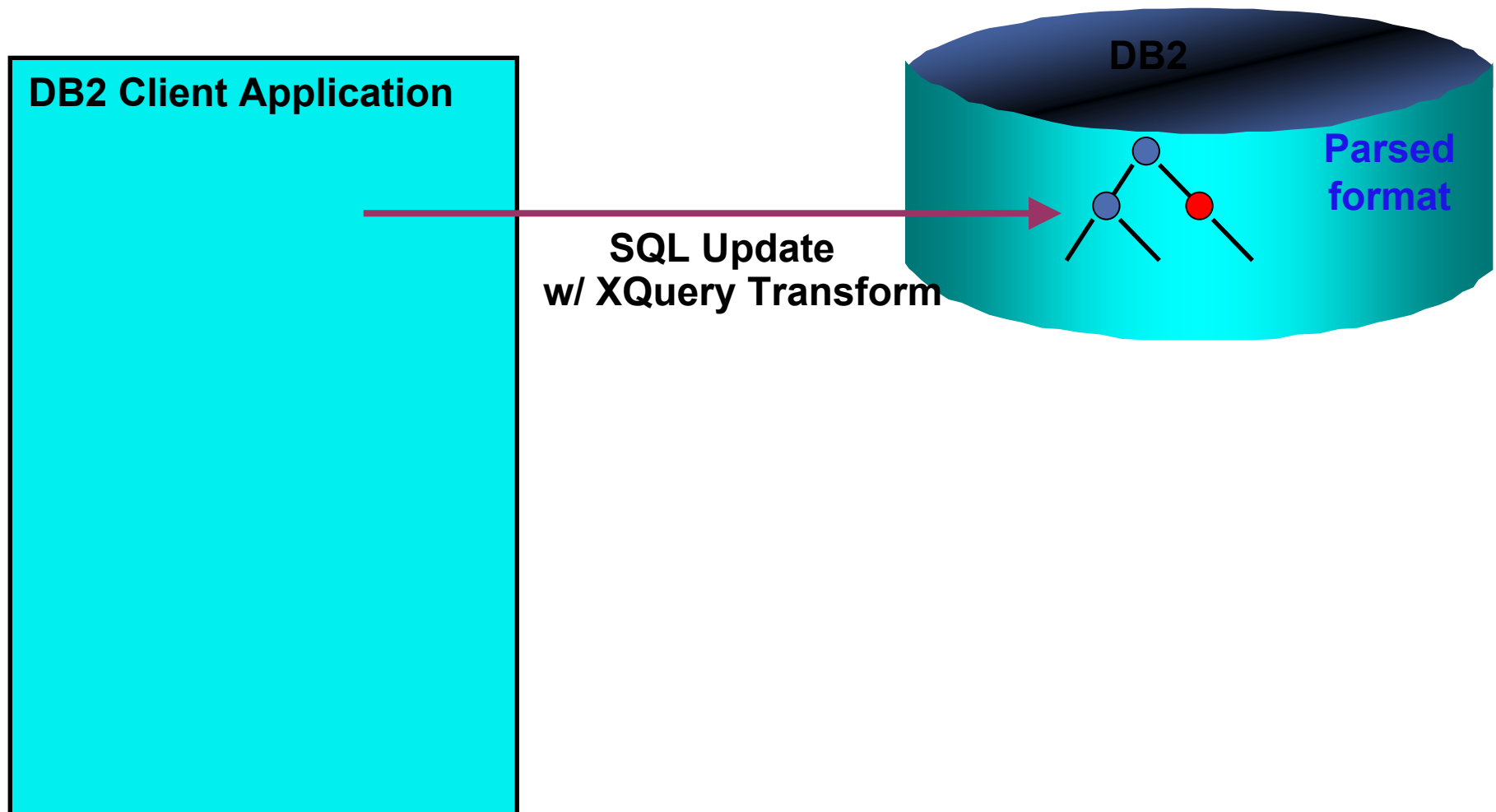


## Recap: Updating elements/attributes in DB2 9





## DB2 9.5: Updating elements/attributes





## Sub-document update

- In DB2 9:  
update T set doc = ?
- XQuery "transform" expression to update XML
  - ▶ Being standardized
- Example  
update T set doc = XMLQuery ('  
    transform  
    copy \$r := \$DOC  
    modify (  
        do delete {\$r/score},  
        do replace value of {\$r/salary} with \$r/salary \* \$raise )  
    return \$r'  
passing cast (? as double) as "raise")



## The XQuery TRANSFORM expression: semantics

- Creates modified copies of existing nodes, documents, or sequences.
- You typically use 4 clauses:

**transform**

Optional. Indicates that this is a transform expression

**copy**    \$new := ...

Copies the "input" XML data into a new variable

**modify**    ...\$new...

Specifies one or more updating expressions

**return**    \$new

Returns the modified copy of the input



## Sub-document update: example

- XQuery "transform" expression has many uses
- Example

```
xquery
let $env := <env> <header> dummy header </header>
                                     <payload/> </env>

let $msg := <msg> <empid> dummy empid </empid> </msg>
return
transform
copy $envelope := $env, $body := $msg
modify (
do replace value of $envelope/header with 23,
do replace value of $body/empid with 20485)
return (transform
copy $envelope2 := $envelope
modify ( do insert $body as last into $envelope2/payload)
return $envelope2)
```











## The XQuery Update Facility

- Standardized by the W3C
  - ▶ <http://www.w3.org/TR/xquery-update-10/>
- New expression to modify XML documents
  - ▶ In SQL update statements to update documents
  - ▶ In queries to modify retrieved XML data on the fly
  - ▶ Allows node-level document manipulation:
    - replace value, replace node
    - insert node, insert after, insert before, insert as last, insert as first
    - delete node
    - rename node



## XQuery Updates vs. XSLT

	XQuery Updates	XSLT
Produce custom XML formats for specific consumers		
Formatting how XML data is rendered in a browser		
Change, insert, delete specific elements/attributes (“point updates”)		
High performance database transactions		
etc.		



## The XQuery TRANSFORM expression

- Creates modified copies of existing nodes, documents, or sequences.
- You typically use 4 keywords

**Optional** keyword, indicates that this is a transform expression

**transform**

Copies the “input” XML data into a new variable

**copy** \$new := ...

**modify** ...\$new...

Specifies one or more updating expressions

**return** \$new

Returns the modified copy of the input



## Update the zip code

- Transform in an update (change data on disk):

update customer

```
set info = xmlquery( 'transform  
    copy $new := $INFO  
    modify do replace value of $new/customerinfo/addr/zipcode with 90111  
    return $new' )
```

where cid = 1000;

- Transform in an SQL/XML query (on the fly):

```
select xmlquery( ' transform  
    copy $new := $INFO  
    modify do replace value of $new/customerinfo/addr/zipcode with 90111  
    return $new' )
```

from customer

where cid = 1000;



## Update the zip code – with parameter markers

- Transform in an update (change data on disk):

```
update customer
set info = xmlquery( '
    copy $new := $INFO
    modify do replace value of $new/customerinfo/addr/zipcode with $z
    return $new'
    passing cast( ? as VARCHAR(10)) as "z")
where cid = ?;
```

- Transform in an SQL/XML query (on the fly):

```
select xmlquery( '
    copy $new := $INFO
    modify do replace value of $new/customerinfo/addr/zipcode with $z
    return $new'
    passing cast( ? as VARCHAR(10)) as "z")
from customer
where cid = ?;
```



## Delete the phone element

update customer

```
set info = xmlquery( 'copy $new := $INFO  
                    modify do delete $new/customerinfo/phone  
                    return $new')
```

where cid = 1000;

```
<customerinfo>  
  <name>John Smith</name>  
  <addr country="Canada">  
    <street>Fourth</street>  
    <city>Calgary</city>  
    <state>Alberta</state>  
    <zipcode>M1T 2A9</zipcode>  
  </addr>  
  <phone type="work">963-289-4136</phone>  
</customerinfo>
```



```
<customerinfo>  
  <name>John Smith</name>  
  <addr country="Canada">  
    <street>Fourth</street>  
    <city>Calgary</city>  
    <state>Alberta</state>  
    <zipcode>M1T 2A9</zipcode>  
  </addr>  
</customerinfo>
```



## Delete the phone element

update customer

```
set info = xmlquery( 'copy $new := $INFO  
                    modify do delete $new/customerinfo/phone  
                    return $new')
```

where cid = 1000;

```
<customerinfo>  
  <name>John Smith</name>  
  <addr country="Canada">  
    <street>Fourth</street>  
    <city>Calgary</city>  
    <state>Alberta</state>  
    <zipcode>M1T 2A9</zipcode>  
  </addr>  
  <phone type="work">963-289-4136</phone>  
</customerinfo>
```



```
<customerinfo>  
  <name>John Smith</name>  
  <addr country="Canada">  
    <street>Fourth</street>  
    <city>Calgary</city>  
    <state>Alberta</state>  
    <zipcode>M1T 2A9</zipcode>  
  </addr>  
</customerinfo>
```



## Delete the phone element

- In an update (change data on disk):  
update customer  
set info = `xmlquery( 'copy $new := $INFO  
                          modify do delete $new/customerinfo/phone  
                          return $new')`  
where cid = 1000;
- In an query (on the fly):  
select `xmlquery( 'copy $new := $INFO  
                          modify do delete $new/customerinfo/phone  
                          return $new')`  
from customer  
where cid = 1000;



## Rename the “addr” element to “address”

update customer

set info = **xmlquery**( 'copy \$new := \$INFO

**modify do rename \$new/customerinfo/addr as "address"**  
return \$new')

where cid = 1000;

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="work">963-289-4136</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <address country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </address>
  <phone type="work">963-289-4136</phone>
</customerinfo>
```



# Update phone number and “type” attribute

**update customer**

**set info =** **xmlquery( 'copy \$new := \$INFO**

**modify (**

**do replace value of \$new/customerinfo/phone with "416-123-4567",**

**do replace value of \$new/customerinfo/phone/@type with "home" )**

**return \$new')**

**where cid = 1000;**

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="work">963-289-4136</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="home">416-123-4567</phone>
</customerinfo>
```



# Replace the phone element incl. "type" attribute

update customer

```
set info = xmlquery( 'copy $new := $INFO  
                    modify do replace $new/customerinfo/phone with  
                      <phone type="home">416-123-4567</phone>  
                    return $new')
```

where cid = 1000;

```
<customerinfo>  
  <name>John Smith</name>  
  <addr country="Canada">  
    <street>Fourth</street>  
    <city>Calgary</city>  
    <state>Alberta</state>  
    <zipcode>M1T 2A9</zipcode>  
  </addr>  
  <phone type="work">963-289-4136</phone>  
</customerinfo>
```



```
<customerinfo>  
  <name>John Smith</name>  
  <addr country="Canada">  
    <street>Fourth</street>  
    <city>Calgary</city>  
    <state>Alberta</state>  
    <zipcode>M1T 2A9</zipcode>  
  </addr>  
  <phone type="home">416-123-4567</phone>  
</customerinfo>
```



## Let's add a phone number...

```
update customer
set info = xmlquery( '
  copy $new := $INFO
  modify do insert <phone type="cell">777-555-3333</phone> into $new
  return $new')
where cid = 1000;
```

**That's not what we wanted. The result is not a well-formed document. The update would fail.**

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="work">963-289-4136</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="home">416-123-4567</phone>
</customerinfo>
<phone type="cell">777-555-3333</phone>
```



## Let's add a phone number... (2<sup>nd</sup> try)

update customer

```
set info = xmlquery( '  
  copy $new := $INFO  
  modify do insert <phone type="cell">777-555-3333</phone>  
  as last into $new/customerinfo  
  return $new')  
where cid = 1000;
```

```
<customerinfo>  
  <name>John Smith</name>  
  <addr country="Canada">  
    <street>Fourth</street>  
    <city>Calgary</city>  
    <state>Alberta</state>  
    <zipcode>M1T 2A9</zipcode>  
  </addr>  
  <phone type="work">963-289-4136</phone>  
</customerinfo>
```



```
<customerinfo>  
  <name>John Smith</name>  
  <addr country="Canada">  
    <street>Fourth</street>  
    <city>Calgary</city>  
    <state>Alberta</state>  
    <zipcode>M1T 2A9</zipcode>  
  </addr>  
  <phone type="home">416-123-4567</phone>  
  <phone type="cell">777-555-3333</phone>  
</customerinfo>
```



## Let's add a phone number... (insert after)

```
update customer
set info = xmlquery( '
    copy $new := $INFO
    modify do insert <phone type="cell">777-555-3333</phone>
    after $new/customerinfo/addr
    return $new')
where cid = 1000;
```

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="work">963-289-4136</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="cell">777-555-3333</phone>
  <phone type="home">416-123-4567</phone>
</customerinfo>
```



## Let's add a phone number... (insert before)

```
update customer
set info = xmlquery( '
  copy $new := $INFO
  modify do insert <phone type="cell">777-555-3333</phone>
  before $new/customerinfo/addr
  return $new')
where cid = 1000;
```

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="work">963-289-4136</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <phone type="cell">777-555-3333</phone>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="home">416-123-4567</phone>
</customerinfo>
```



## Update with XML Schema Validation

```
update customer
set info = xmlvalidate( xmlquery('
    copy $new := $INFO
    modify do delete $new/customerinfo/phone
    return $new' ) according to schema "cust.custschema")
```

where cid = 1000;

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="work">963-289-4136</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
</customerinfo>
```



## Invalid Updates (will fail, as per spec):

- Update anything but a single node (such as an empty sequence or a sequence of multiple nodes).
- More than one **rename** for the same node.
- More than one **replace value of** for the same node.
- More than one **replace** operation for the same node.
- Update produces an invalid XML fragment or document (e.g. element with two attributes of the same name)
- Update produces inconsistent namespace bindings.



# Updating repeating elements (can be ambiguous)

```
update customer
set info = xmlquery( '
  copy $new := $INFO
  modify do replace value of $new/customerinfo/phone with "111-222-4444"
  return $new')
where cid = 1000;
```

**Not clear which phone element to modify. Update fails. Solutions:**

- (1) Use predicate to select single node.**
- (2) If you want to update all phones, use “for” to iterate.**

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="cell">777-555-3333</phone>
  <phone type="home">416-123-4567</phone>
</customerinfo>
```



**Error.**



# Updating repeating elements (pick one!)

**update customer**

**set info = `xmlquery(`**

`copy $new := $INFO`

**`modify do replace value of $new/customerinfo/phone[@type="cell"]`**  
**`with "111-222-4444"`**

`return $new')`

**where cid = 1000;**

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="cell">777-555-3333</phone>
  <phone type="home">416-123-4567</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="cell">111-222-4444</phone>
  <phone type="home">416-123-4567</phone>
</customerinfo>
```



# Updating repeating elements (update all)

update customer

```
set info = xmlquery('copy $new := $INFO
                    modify (  for $j in $new/customerinfo/phone
                               return do replace value of $j with "111-222-4444" )
                    return $new')
```

where cid = 1000;

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="cell">777-555-3333</phone>
  <phone type="home">416-123-4567</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="cell">111-222-4444</phone>
  <phone type="home">111-222-4444</phone>
</customerinfo>
```



# Trying to update an empty sequence

*We don't have a customer by the name of John Poodle*

update customer

set info = **xmlquery( '**

**copy \$new := \$INFO[customerinfo/name="John Poodle"]**

**modify do replace value of \$new/customerinfo/phone**

**with "111-222-4444" )**

**return \$new')**

where... ;

SQL16084N An assigned value in the copy clause of a transform expression is either **the empty sequence** or a sequence of more than one node.



## Multiple expressions on the same node

```
update customer
set info = xmlquery( '
    copy $new := $INFO
    modify (
        do replace value of $new/customerinfo/phone with "416-123-4567",
        .....,
        do replace value of $new/customerinfo/phone with "905-871-0024" )
    return $new')
where cid = 1000;
```

SQL16083N Incompatible "replace value of" expressions exist in the modify clause of a transform expression.  
Error QName=err:XUDY0017. SQLSTATE=10704



## But, no failure here !

```
update customer
set info = xmlquery( '
    copy $new := $INFO
    modify (
        if (fn:starts-with($new/customerinfo/phone,"416")) then
        do replace value of $new/customerinfo/phone with "416-123-4567"
        else
        do replace value of $new/customerinfo/phone with "905-871-0024" )
    return $new'
where cid = 1000;
```

Two “replace value of” expressions on the same node, but only one will apply. Hence, this is allowed and will work.



## How updating expressions are processed

- MODIFY clause can contain multiple changes to a document
- All changes are collected and applied **independently** from each other to a snapshot of the original document
- The changes don't see each others' effects ("snapshot semantics")
- The ordering of the changes in the modify clause does not matter



## Add and rename phone element(s)

update customer

set info = **xmlquery( '**

**copy \$new := \$INFO**

**modify ( do insert <phone type="cell">777-555-3333</phone>**

**after \$new/customerinfo/addr ,**

**for \$j in \$new/customerinfo/phone**

**return do rename \$j as "telephone" )**

**return \$new' )**

**where cid = 1000;**

The “*insert*” is not affected by the “*rename*”. Changes are applied independently.

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="work">963-289-4136</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="cell">777-555-3333</phone>
  <telephone type="work">963-289-4136</telephone>
</customerinfo>
```



## Conflict between delete and insert

update customer

set info = xmlquery( '

copy \$new := \$INFO

modify ( do delete \$new/customerinfo/addr,

do insert <POBox>15</POBOX> into \$new/customerinfo/addr

)

return \$new')

where cid = 1000;

**Delete wins.**

```
<customerinfo>
  <name>John Smith</name>
  <addr country="Canada">
    <street>Fourth</street>
    <city>Calgary</city>
    <state>Alberta</state>
    <zipcode>M1T 2A9</zipcode>
  </addr>
  <phone type="work">963-289-4136</phone>
</customerinfo>
```



```
<customerinfo>
  <name>John Smith</name>
  <phone type="cell">777-555-3333</phone>
</customerinfo>
```



## Summary

- Node-level updates
  - ▶ insert
  - ▶ replace
  - ▶ replace value of
  - ▶ rename
  - ▶ delete
- Less complexity in your application
- No XML parsing or serialization
- 2x to 5x faster than DB2 9



# Agenda

- *Introduction*
  - ▶ *What is XML?*
  - ▶ *XML storage options:*  
*XML-enabled databases vs. hybrid database (DB2 9)*
- *Processing XML with DB2 pureXML*
  - ▶ *Publishing*
  - ▶ *Storing*
  - ▶ *Querying*
  - ▶ *Indexing*
  - ▶ *Updating*
- **Specifics for DB2 for z/OS**
- *Specifics for DB2 for LUW*



## V9 XML Feature Summary

- First-class XML type, native storage of XQuery Data Model (XDM)
- Complete SQL/XML constructor functions
  - ▶ XMLDOCUMENT, XMLTEXT, XMLCOMMENT, XMLPI
  - ▶ From V8: XMLELEMENT, XMLATTRIBUTES, XMLNAMESPACES, XMLFOREST, XMLCONCAT, XMLAGG
- XMLPARSE, XMLSERIALIZE, XMLCAST
- XML indexes
  - ▶ use for XMLEXISTS and XMLTABLE
- Other SQL/XML functions with XPath
  - ▶ XMLEXISTS, XMLQUERY, XMLTABLE
- XML Schema repository, Validation UDF, and decomposition
- DRDA (distributed support) and application interfaces
- Utilities and tools

XMLTABLE and XMLCAST: available through PK51571, PK51572, PK51573



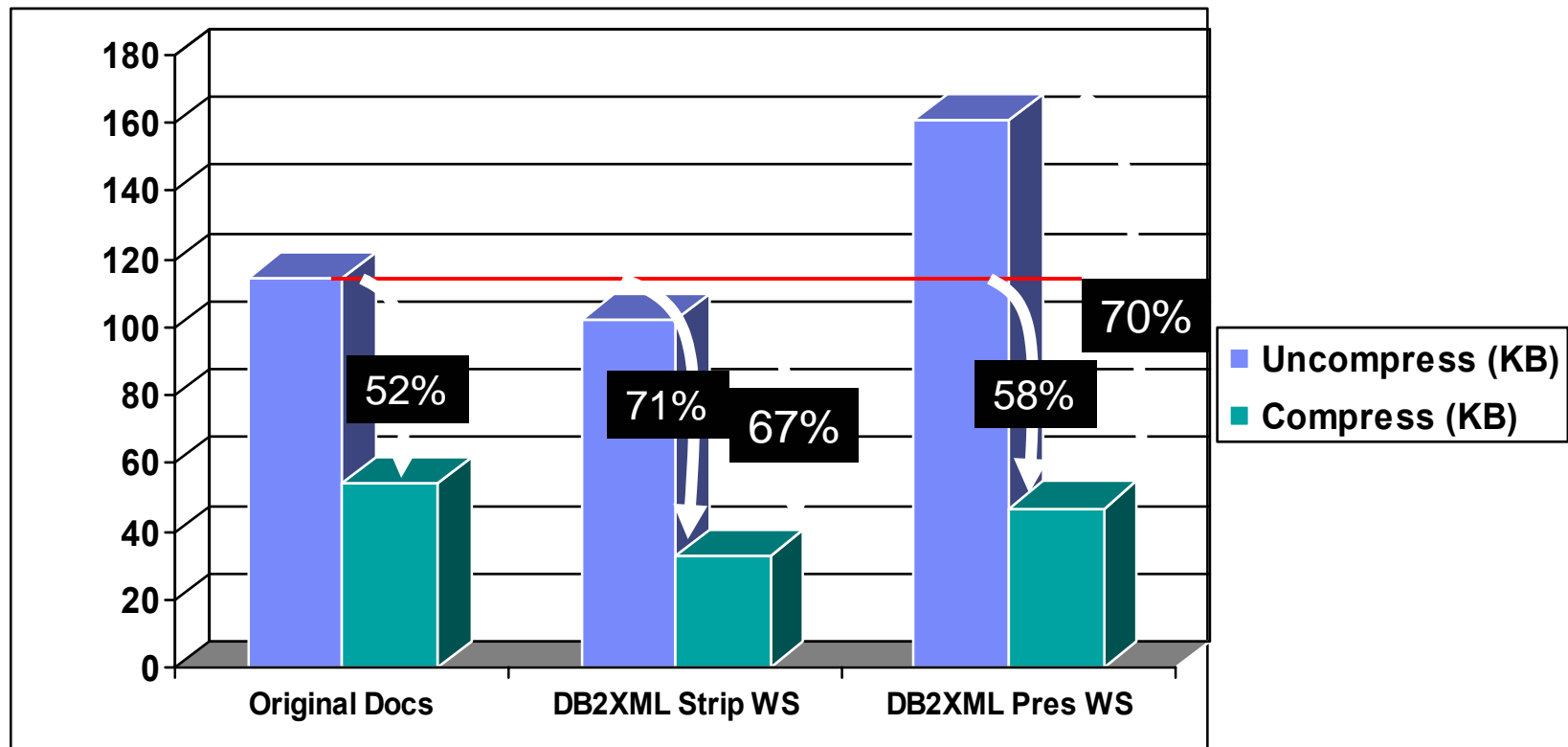


## Performance and Scalability

- XML storage leverages mature optimized storage infrastructure. Compact format, compressed well.
- Next generation parsers: z/OS XML System Services and XLXP-C.
  - ▶ Validation is more expensive than well-formedness checking only
- Highly efficient XPath streaming algorithm
- Support partitioned table spaces and data sharing.
- Initial sweet spot: a large number of small documents.



## Storage for UNIFI Messages



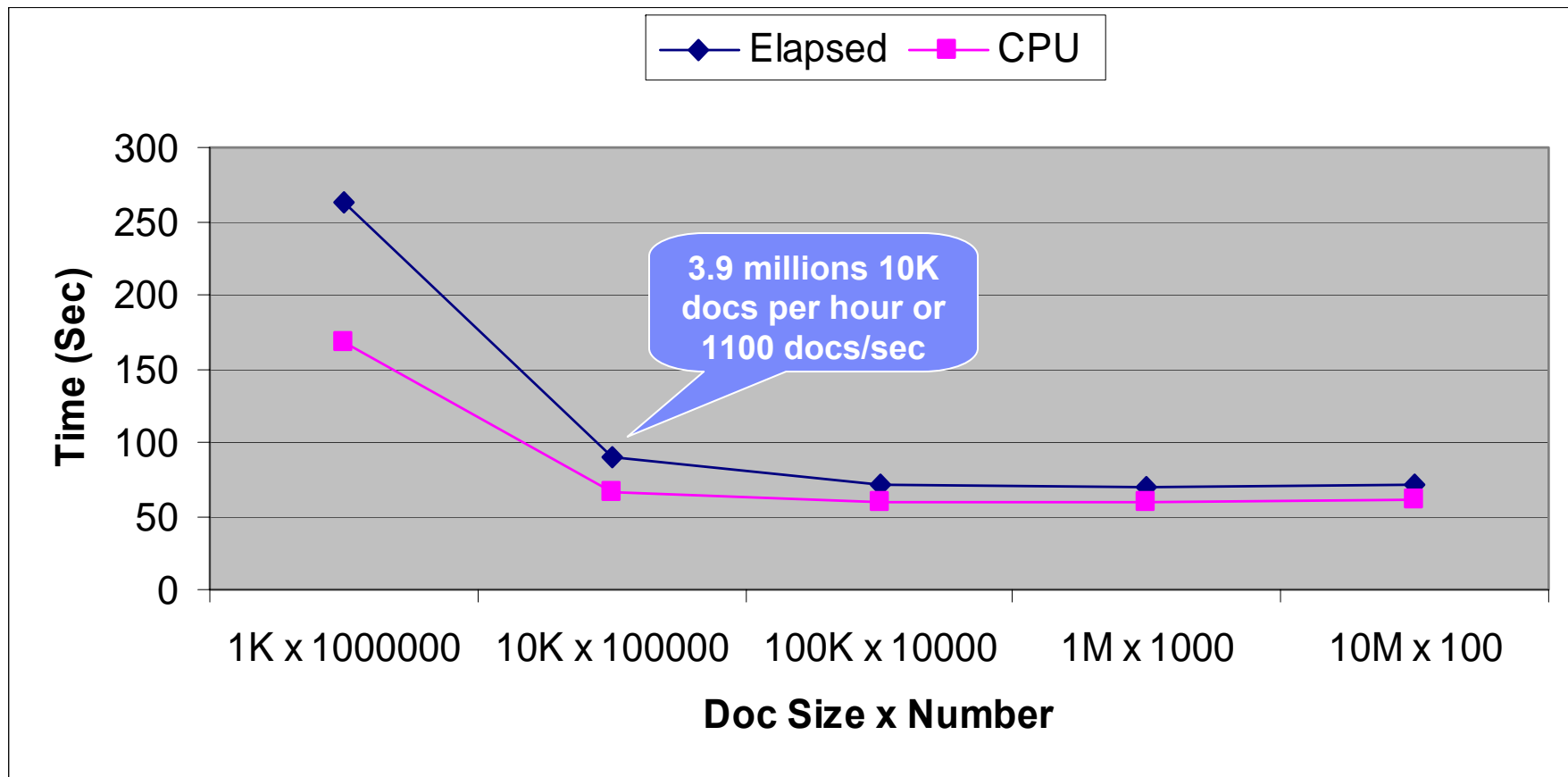
96 sample documents

Strip WS: Strip Whitespaces

Pres WS: Preserve Whitespaces



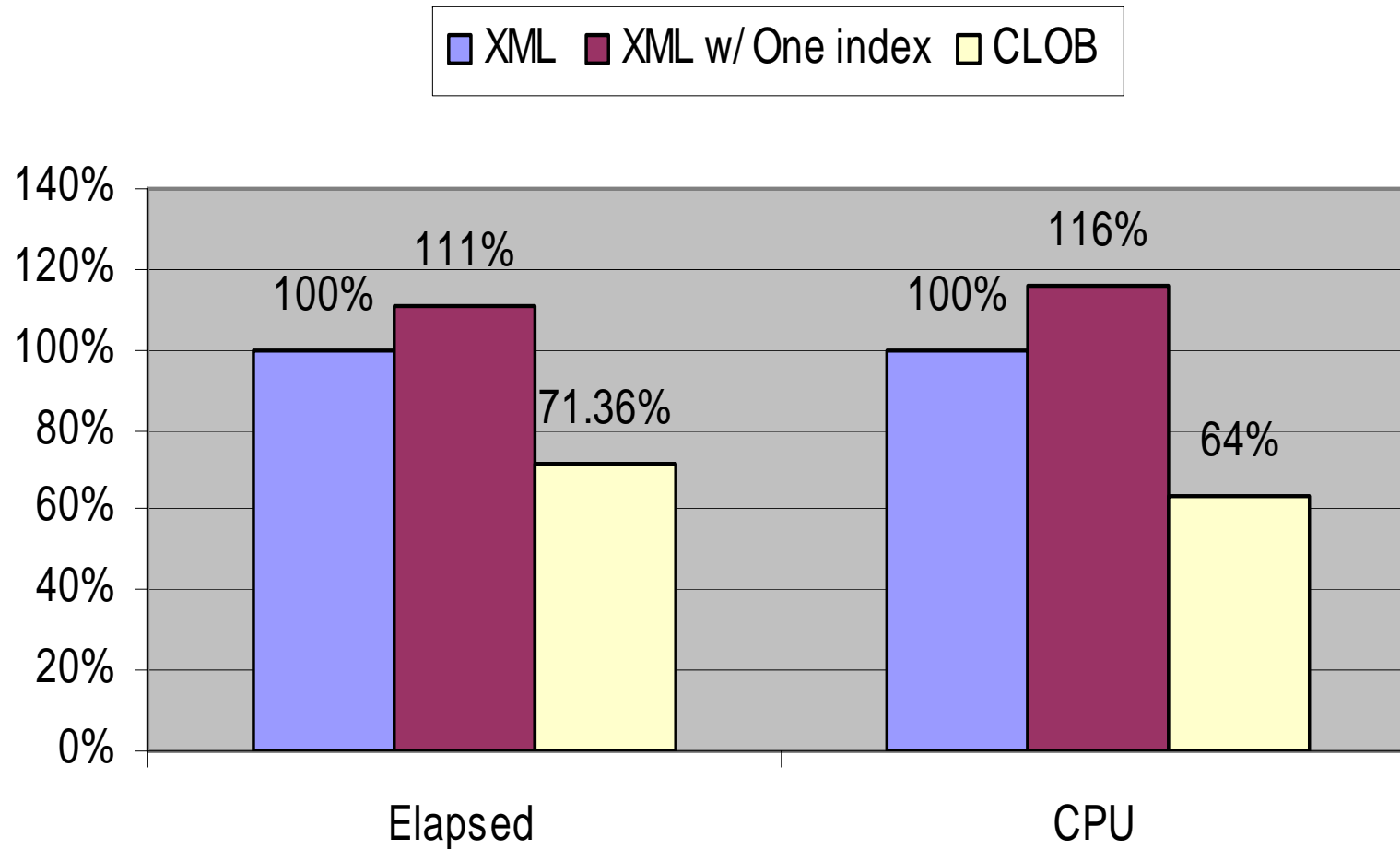
## Insert Performance (Batch)



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



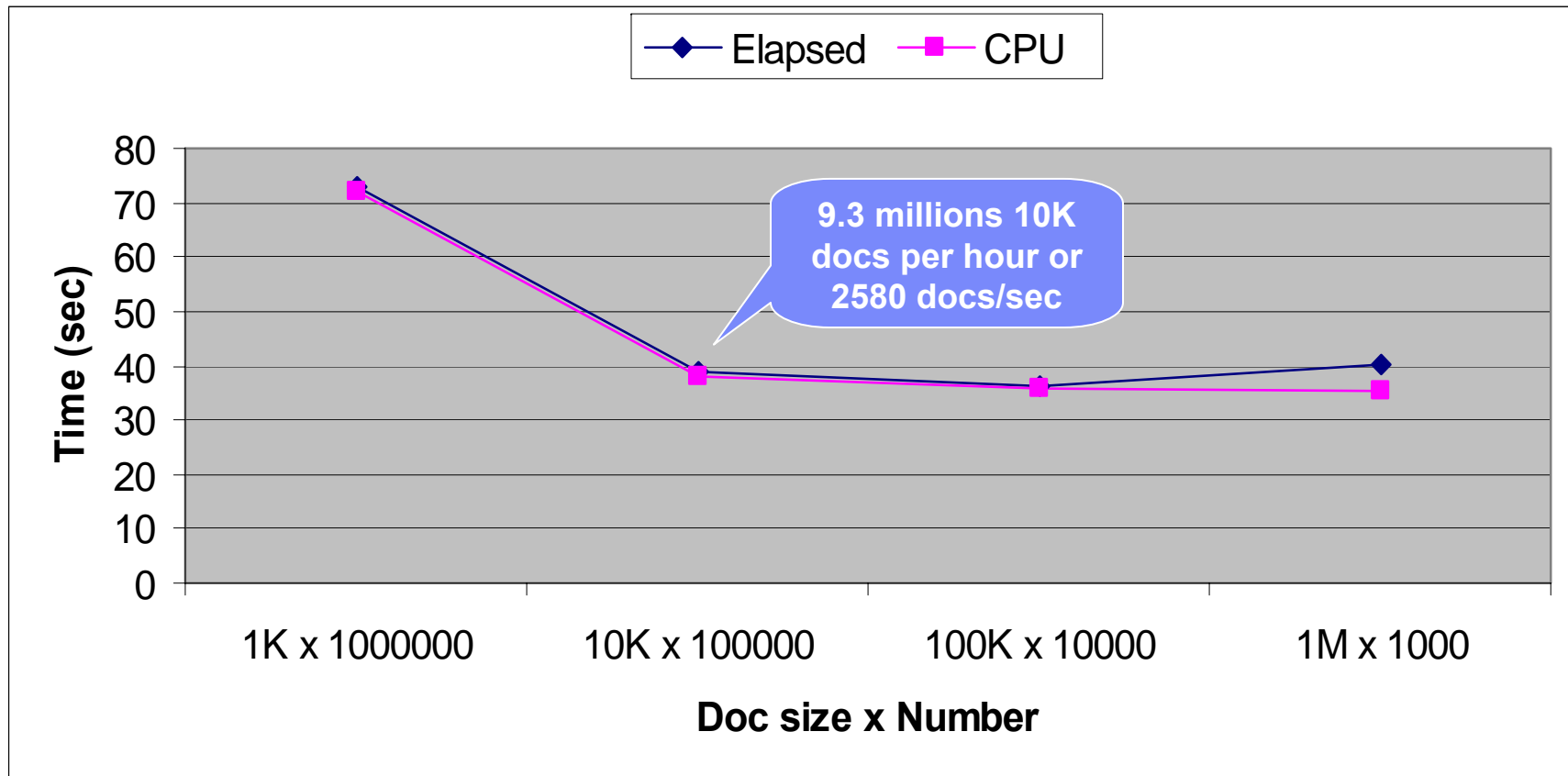
## Insert Performance – compare w/ CLOB



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



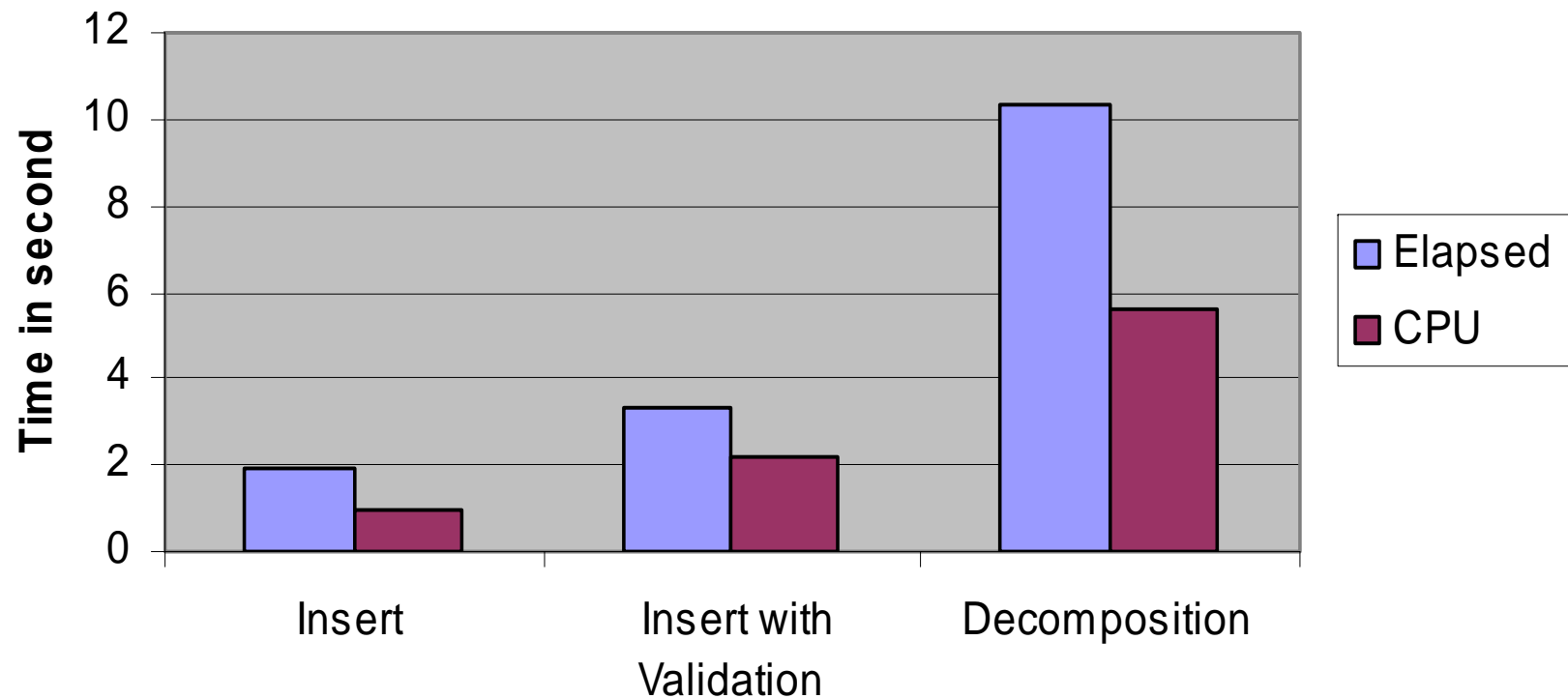
## Fetch Performance (Batch)



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



# XML Insert v.s. Validation v.s. Decomposition

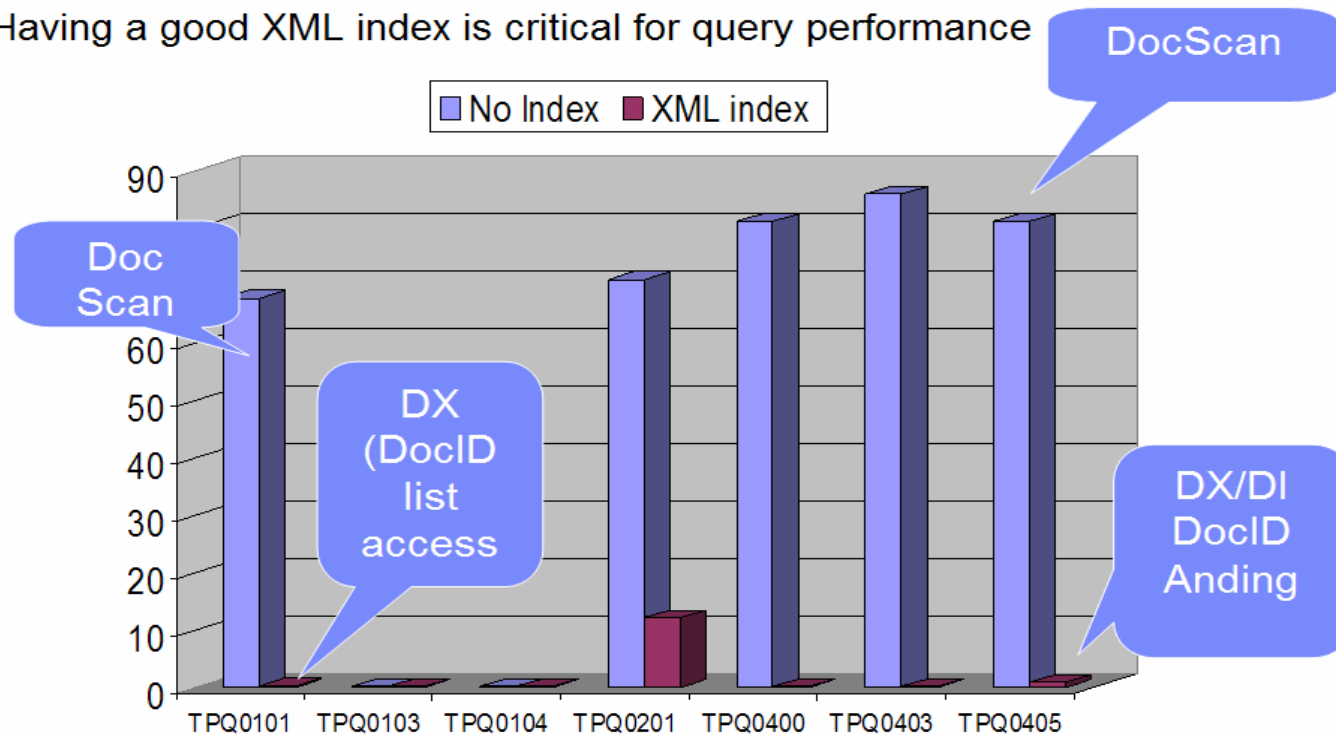


**4K doc Single thread 3000 repeat**



## XML Index Exploitation

- Having a good XML index is critical for query performance





## Future Directions in DB2 for z/OS\*

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

\* All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.



## References on DB2 z/OS pureXML

- Detailed introduction presentation:  
<ftp://ftp.software.ibm.com/software/data/db2zos/db29zospureXMLgzhang.pdf>
- DB2 z/OS XML Guide:  
<http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.db29.doc.xml/bknxspsh.htm>
- IBM developerWorks DB2 XML (LUW):  
<http://www.ibm.com/developerworks/db2/zones/xml/index.html>



## 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.



## XQuery v.s. SQL/XML (1)

- Simplest FLWOR expressions => XPath path expressions

for \$x in /purchaseorder/items/item  
where contains(\$x/desc, "Shoe")  
return \$x

=>

/purchaseorder/items/item[contains(desc,  
"Shoe")]



## XQuery v.s. SQL/XML (2)

- Use SQL/XML Constructors
  - ▶ Assume table DOCUMENTS(DOCURI, DOC)

```
<results>
{
  for $t in doc("books.xml")/(chapter | section)/title
  where contains($t/text(), "XML")
  return $t
}
</results>

=>
SELECT XMLDOCUMENT(
  XMLELEMENT(NAME "results",
    XMLQUERY('/*[name(.) = "chapter" or name(.) = "section"]
              /title[contains(text(), "XML")]')
    PASSING DOC)
  )
FROM DOCUMENTS
WHERE DOCURI = 'books.xml';
```



## XQuery v.s. SQL/XML (3)

- Use XMLTABLE, XMLAGG, and XML constructors

```
<bib>
{
  for $b in doc("http://bstore1.example.com/bib.xml")//book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  order by $b/title
  return
    <book>
      { $b/@year }
      { $b/title }
    </book>
}
</bib>
```

```
SELECT XMLDOCUMENT(
  XMLELEMENT(NAME "bib",
    XMLAGG(
      XMLELEMENT(NAME "book", "@year", "title")
      ORDER BY XMLCAST("title" as VARCHAR(100))
    ) ) )
FROM DOCUMENTS,
  XMLTABLE('/bib/book[publisher = "Addison-Wesley"
    and @year > 1991]' PASSING DOC

  COLUMNS
    "@year" XML,
    "title" XML) AS X
WHERE DOCURI = 'http://bstore1.example.com/bib.xml';
```



## XPath support in DB2 9 z/OS

- XPath used in XMLEXISTS, XMLQUERY, XMLTABLE, indexing
- XPath 1.0 + -
  - ▶ XPath 1.0 constructs in XPath 2.0 semantics (no value comp)
  - ▶ + more numeric data types and some generic types.
  - ▶ + namespace declaration from XQuery prolog
  - ▶ - Axes: only forward axes (child, descendant, descendant-or-self, self, attribute, //, /, @) & parent axis (..) are supported.
- Types supported in XPath expressions:
  - ▶ xs:boolean, xs:integer, xs:decimal, xs:double, xs:string
- Functions supported: fn:abs, fn:boolean, fn:compare, fn:concat, fn:contains, fn:count, fn:data, fn:string-length, fn:normalize-space, fn:not, fn:round, fn:string, fn:substring, fn:sum, **fn:distinct-values**, **fn:max**, **fn:min**, **fn:upper-case**, **fn:lower-case**, **fn:translate**, **fn:tokenize**, **fn:matches**, **fn:replace**, **fn:position**, **fn:last**, **fn:name**, **fn:local-name**

\* Functions in blue are being delivered post V9 GA.



## Application Interfaces

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



## Java JDBC Example

Use standard interface:

```
PreparedStatement pstmt = connection.prepareStatement("INSERT INTO PurchaseOrders  
VALUES(?, ?); // second column: XML type
```

```
...  
InputStream fin = new FileInputStream(file);  
pstmt.setBinaryStream( 2, fin, flen );  
pstmt.execute();
```

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



## 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)**



## 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.	XMLEExists('/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.	XMLEExists('/Catalog/Categories/Product[RegPrice > 100 and Discount > 0.1]' ... ) With indexes on: '//RegPrice' as SQL DECFLOAT and '//Discount' as SQL DECFLOAT



## System Configurations

- Basic XML parsing requires z/OS XMLSS: z/OS 1.8 or z/OS 1.7 with APAR OA16303
- XML schemas requires IBM 31-bit SDK for z/OS, Java 2 Technology Edition, V5 (5655-N98), SDK5. And Java stored procedure setup.
- Zparms for virtual storage: XMLVALA and XMLVALS
  - ▶ Default: 200MB and 10GB.
  - ▶ LOBVALA and LOBVALS also impact XML bind-in and bind-out size
- Buffer pool for XML tables (default BP16K0), authorization for users who create or alter tables with XML columns.
  - ▶ DEFAULT BUFFER POOL FOR USER XML DATA ==> BP16K0 BP16K0 - BP16K9



## Using SPUFI or JCL

```
SELECT Cid, Info  
FROM DSN8910.CUSTOMER  
WHERE XMLEXISTS ('declare default element namespace  
"http://posample.org";  
//addr[city="Toronto"]' passing INFO)
```

- XML and XPath are **case-sensitive**: CAPS off
  - ▶ **SQLCODE = -16002**, ERROR: AN XQUERY EXPRESSION HAS AN UNEXPECTED TOKEN **DEFAULT** FOLLOWING **DECLARE**.
- Terminal session CCSID setting has to be consistent with application encoding scheme as “[” and “]” have different code points in different code pages.
  - ▶ **SQLCODE = -16002**, ERROR: AN XQUERY EXPRESSION HAS AN UNEXPECTED TOKEN FOLLOWING **"Toronto"**.



## Command Line Processor (CLP) - Setup

- Official support on USS, but works on Windows
- Set up JCC driver on Windows
- Set up CLP:
  - ▶ Put `clp.jar` in a directory, say `...\path`
  - ▶ Set up file `clp.properties` (optional):  
`TerminationChar=#`  
`mydb2=hostname.svl.ibm.com:446/LOC,USER,PASSWD`
  - ▶ Set environment variables:
    - `CLASSPATH = ...; ...path\clp.jar`
    - `CLPPROPERTIESFILE=...path\clp.properties`
  - ▶ (Bind metadata routines – from the console: `S JOB,J=V91TIJMS`)
- For convenience, on Windows:  
`set db2=java com.ibm.db2.clp.db2`  
Then use `%db2%` to invoke CLP



## XSR (XML Schema Repository) Setup

- A set of DB2 user tables, stored procedures, and a user-defined function:
  - ▶ XSR\_REGISTER, XSR\_ADDSCHEMADOC, XSR\_COMPLETE, XSR\_REMOVE
  - ▶ XDBDECOMPXML
  - ▶ DSN\_XMLVALIDATE (UDF)
- Setup required:
  - ▶ Bind package SYSXSR (install/migration jobs DSNTIJSJG and DSNTIJNX, or manual run for PTF)
  - ▶ Java 2 SDK V1.5, JCC for DB2 9 for z/OS, and bind the JCC packages to SYSXSR collection ID
  - ▶ WLM for stored procedures and functions (default WLMENV3)
  - ▶ WLM for Java stored procedure (needed for XSR\_COMPLETE)



## XML Type and DDL

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

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

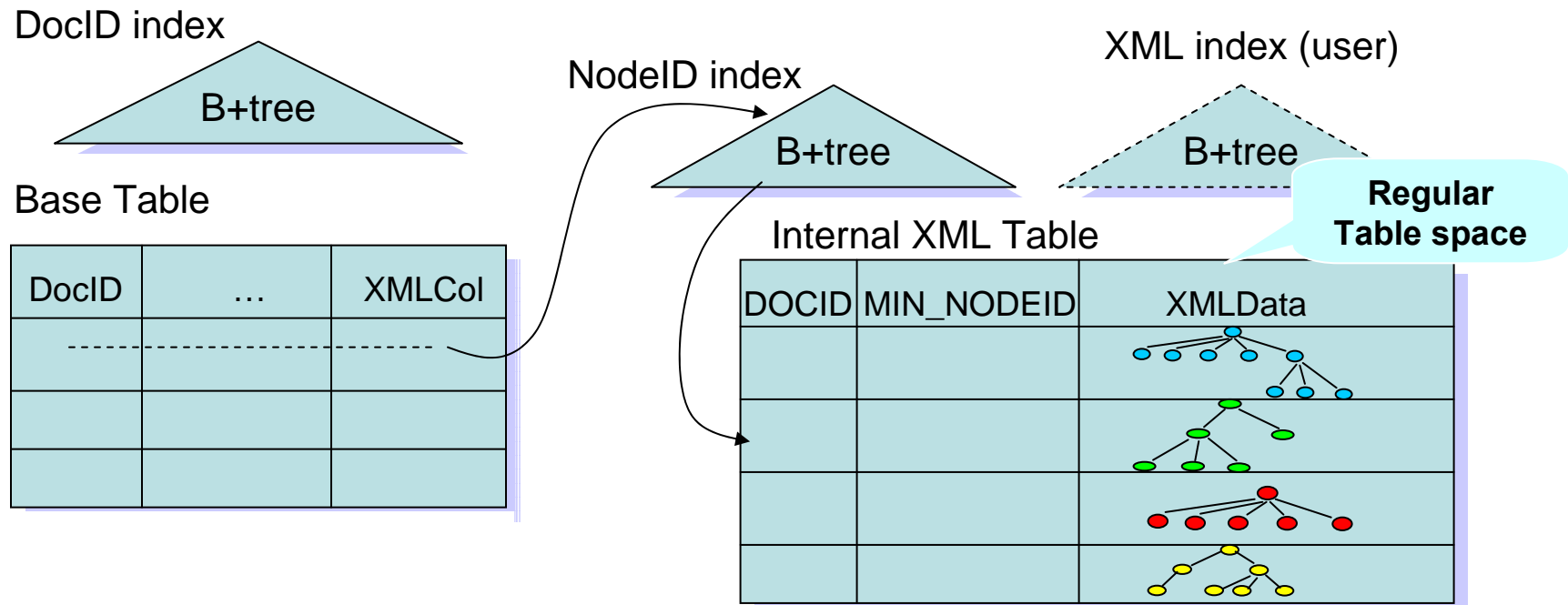
```
CREATE TABLE PO LIKE PurchaseOrders;
```

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

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



## 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 XML 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.



## Storing XML Trees - Tree Packing

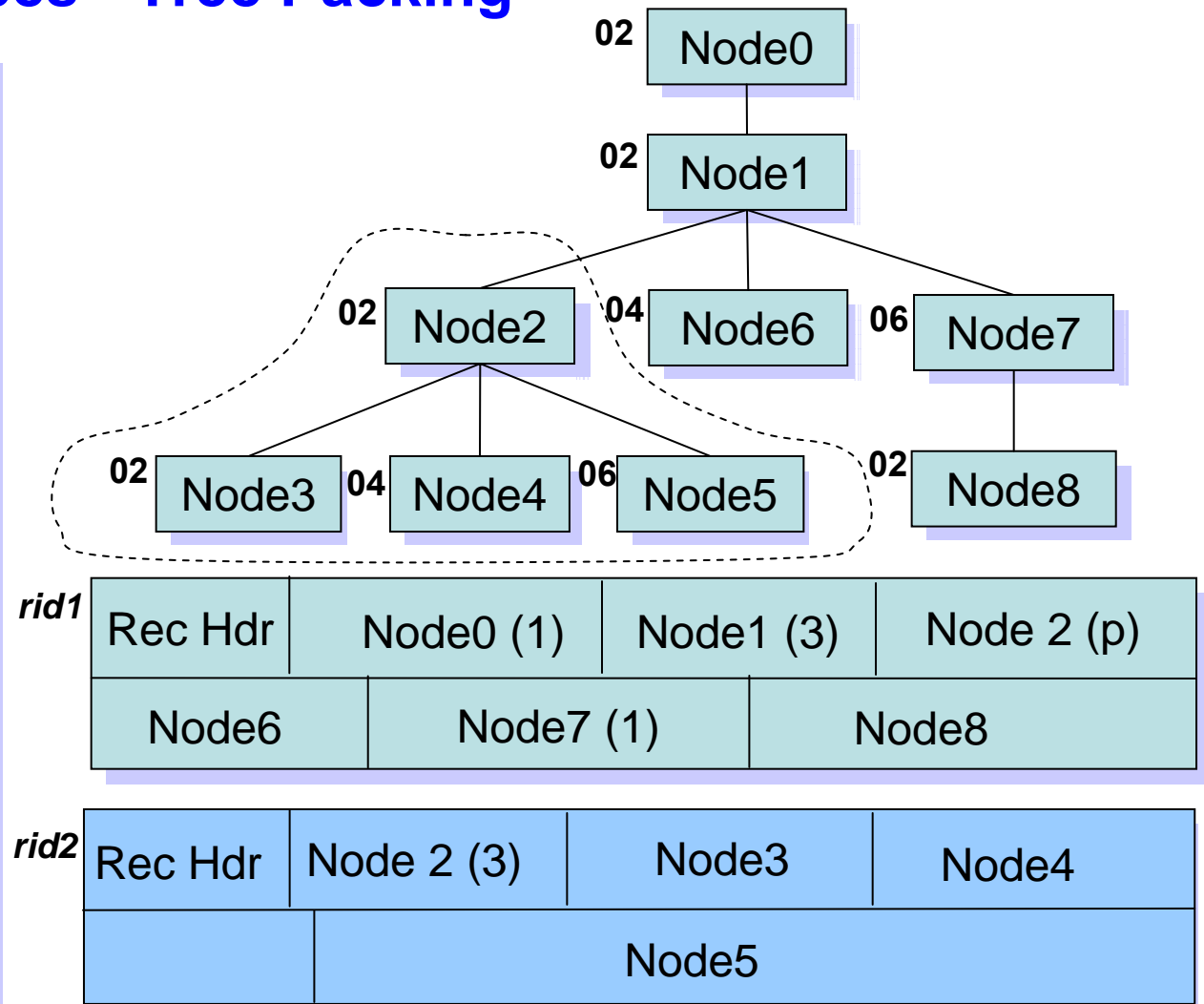
Each node contains local node id, length and optional number of children.

Proxy nodes are used as placeholder for subtrees in a separate record.

It supports traversal using *firstChild*, *nextSibling*, or *nextNode*.

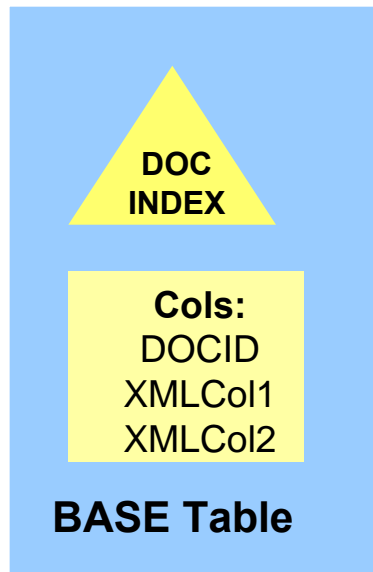
RecHdr contains context path information for the record – absolute ID, path, in-scope namespaces

All names use stringIDs.

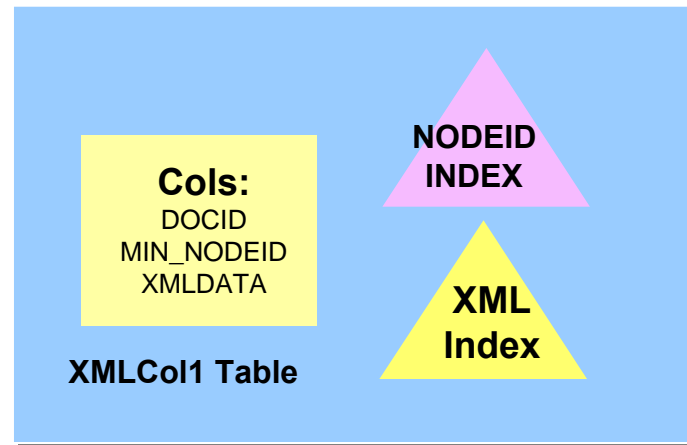




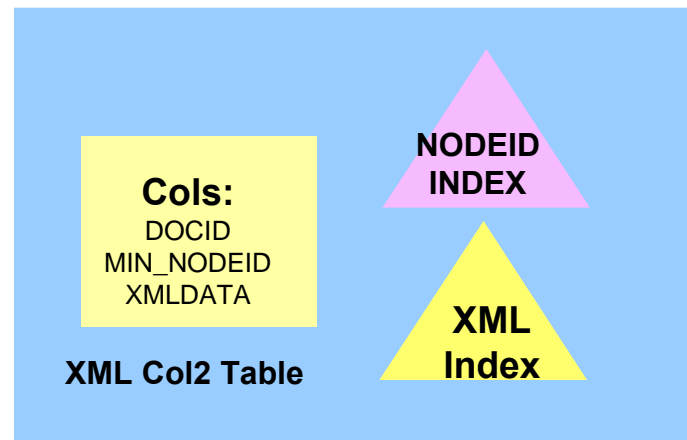
# XML Objects for Non-partitioned Base Table



**Non-Partitioned Base TS**  
(simple, segmented, PBG)



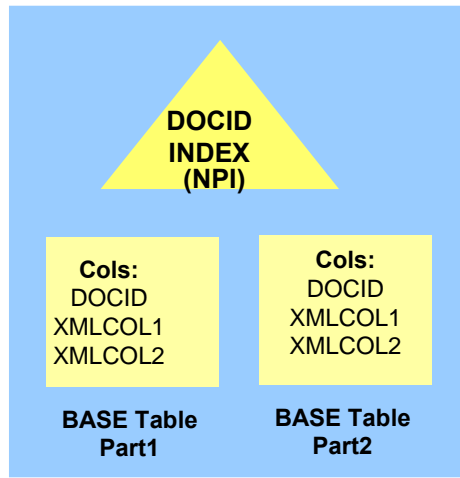
**PBG TS for XMLCol1**



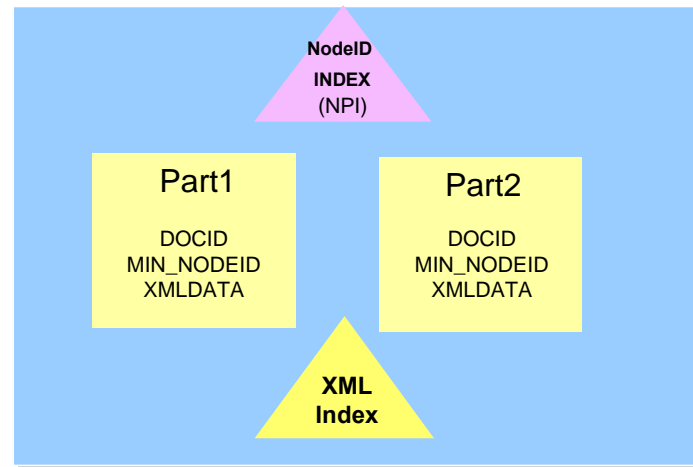
**PBG TS for XMLCol2**



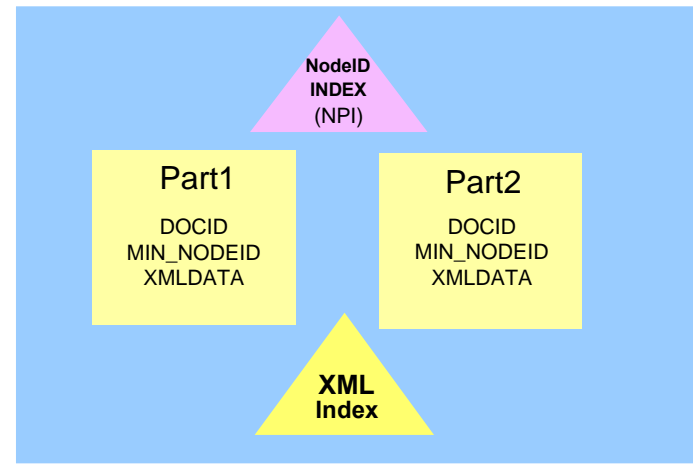
# XML Objects for Partitioned Base Table



**Partitioned Base TS**  
2 Parts, Table has 2  
XML Columns



**Partitioned TS for XMLCol1**



**Partitioned TS for XMLCol2**



## REPORT TABLESPACESET - Output

Contains both a LOB column and an XML column. (New XML text is shown in red)

### **TABLESPACE SET REPORT:**

**TABLESPACE : DB1.BASETS1**

**TABLE : SYSADM.BTAB1**

**INDEXSPACE : DB1.XXXXXXXX**

**INDEX : SYSADM.I\_DOCIDBTAB1**

### **LOB TABLESPACE SET REPORT:**

**TABLESPACE : DB1.BASETS1**

**BASE TABLE : SYSADM.BTAB1**

**LOB TABLESPACE : DB1.LOBTs1**

**AUX TABLE : SYSADM.AUXTAB1**

**AUX INDEXSPACE : DB1.AUXIX1**

**AUX INDEX : SYSADM.AUXIX1**

### **XML TABLESPACE SET REPORT:**

**TABLESPACE : DB1.BASETS1**

**BASE TABLE : SYSADM.BTAB1**

**XML TABLESPACE : DB1.XBAS0001**

**XML TABLE : SYSADM.XBTAB1001**

**NODEID INDEXSPACE : DB1.YYYYYYYY**

**NODEID INDEX : SYSADM.I\_NODEID\_XBTAB1001**

**XMLVALUESINDEXSPACE:DB1XPTH1REPORT**

**XML VALUES INDEX : SYSADM.XPTH1**



## Encoding for XML

- Textual XML Data Encoding
  - ▶ Internally-encoded (within XML data itself)
    - Encoding Declaration option or **Byte Order Mark** within XML data
    - Default: UTF-8
    - Applies to **binary** variables (DB2 detects encoding)
  - ▶ Externally-encoded
    - Character host variables CCSID (override internal encoding)
- DB2 uses UTF-8 to handle XML data. Character data in XML always stored as UTF-8 in XML column.
- During insert, DB2 converts XML data to UTF-8 from the corresponding CCSID.
- During select, DB2 converts XML in UTF-8 to host var encoding locally, or sends serialized XML data in UTF-8 format to remote requester.



# 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 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.



## Table Space Size Consideration

- Basic XML storage is about 0.3 (strip ws w/ compression) to 1.5 (preserve ws w/o compression) of original doc size
- An XML table space always use 16KB pages.
  - ▶ For non range-partitioned base table spaces, PBG table space is used for XML.
- Range-partitioned base table spaces: XML partitioning follows base table partitioning.
- The number of rows to fit into a relational partition is limited by the number of documents to fit into an XML partition.
  - ▶ For example, 4K doc size, 32GB partition can roughly store 8M documents (or 6M to be safe).



## Compression

- XML Tablespace compression
  - ▶ Inherit from base table Compress YES
  - ▶ You can alter XML table space compress attribute
- Significant disk storage savings, especially for preserve whitespace option (70%)
- CPU cost similar to relational
  - ▶ Significant CPU impact if you select/scan many documents (DocScan)
- Initial LOAD will trigger base table compression but not for XML table space
- XML table space compression happens at next REORG



## Utilities Overview

- No new special utilities for XML.
- DB2 utilities support for the XML data type and the related database objects
  - ▶ The XML data type in LOAD and UNLOAD, with file reference support
  - ▶ Support for the XML table spaces and tables used to store XML column values
  - ▶ Support for auxiliary relationships used to connect base tables to XML tables
  - ▶ Support for the base table space DocID index
  - ▶ Support for the XML table space NodeID index and XML indexes
- No partition level checking for XML PBG UTS(APAR pk49033)



## New Keywords for XML

- LISTDEF
  - ▶ implements a new XML keyword for the building of lists with and without XML objects
- LOAD
  - ▶ implements a new field type, XML, in the field-spec.
  - ▶ Additional keywords for XML fields to specify how white space is to be handled (PRESERVE WHITESPACE)
- UNLOAD
  - ▶ implements a new field type, XML, in the field-spec



## 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.



## Diagnosing Problem Related to XML Objects

- Identify XML tables and their related objects
  - ▶ Run REPORT TABLESPACESET
- Validate that the auxiliary index is consistent with the underlying table spaces
  - ▶ Run CHECK INDEX on all indexes, DocID, NodeID and XML value indexes
- Validate the logical connection between the base table and XML table.
  - ▶ Run CHECK DATA against the base table space.
- Use Repair to diagnose problem related to base table spaces with XML columns and their DocID index
  - ▶ Use REPAIR LOCATE KEY to locate a row using DocID key in the DocID index
- Use Repair to diagnose problem related to XML table spaces and their NodeID index or XML Value Index
  - ▶ Use REPAIR LOCATE RID to locate a row using a RID.



## LOAD XML Data

- LOAD uses internal INSERT for XML data.
- To load XML directly from input record, specify XML as the field type.
  - ▶ `LOAD DATA INDDN(INFILE) LOG NO RESUME(NO) FORMAT DELIMITED INTO TABLE PURCHASEORDERS`
  - ▶ `LOAD DATA INDDN(INFILE) LOG NO RESUME(NO) ... XMLPO POSITION(20) XML PRESERVE WHITESPACE INTO TABLE PURCHASEORDERS`
- To load XML from a file, specify CHAR or VARCHAR along with either BLOBF, CLOBF or DBCLOBF.
- Schema validation not supported for LOAD.
- XML compression takes effect after first REORG, not on initial LOAD. Same for FREEPAGE, PCTFREE.



## UNLOAD XML Data

- To unload XML data directly to output record, specify XML as the field type.
  - ▶ non-delimited format: a 2-byte length will precede the value of the XML.
  - ▶ For delimited output, no length field is present.
  - ▶ Limit to 32K length
- To unload XML data to a separate file:
  - ▶ Specify CHAR(n)/VARCHAR(n) BLOBF, CLOBF or DBCLOBF for file names
  - ▶ Use the template control statement to create the XML output file and filename



## File References – UNLOAD to PDS

```

TEMPLATE TCLOBF UNIT(SYSDA) DISP(MOD,CATLG,DELETE)
                      DSN(SAMPLE.&TS..T&TI..&SN..UFILEREf)
                      DSNTYPE(PDS) DIR(15) VOLUMES(SCR03)
UNLOAD TABLESPACE SAMPLEDB.SAMPLETS
PUNCHDDN SYSPUNCH UNLDDN SYSREC
FROM TABLE ADMF001.SAMPLETB
  ( MYCOL1          POSITION(*) DECIMAL(5,2)
    ,MYXML1          POSITION(*) VARCHAR CLOBF TCLOBF
  )

```

&TS. Table space name

&TI. Time

&SN. Space name

DIR number of 256-byte  
records for directory

SAMPLE.UFILEREf.SYSREC

```

ß@  SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREf(BI0MSW37)
Ñ@  SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREf(BI0MSW5K)
`@  SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREf(BI0MSW55)
i@  SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREf(BI0MSW6S)
r@  SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREf(BI0MSW7E)

```

```

SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREf:
BI0MSW37
BI0MSW5K
BI0MSW55
BI0MSW6S
BI0MSW7E

```



## File References – LOAD from PDS

```

TEMPLATE TCLOBF UNIT
          DSN(
          DSNT
UNLOAD TABLESPACE SAM
PUNCHDDN SYSPUNCH
FROM TABLE ADMF00
  ( MYCOL1
    ,MYXML1
  )

```

```

LOAD DATA INDDN SYSREC LOG NO RESUME YES
EBCDIC CCSID(00037,00000,00000)
SORTKEYS 10
INTO TABLE "ADMF001"."SAMPLETB"
WHEN(00001:00002) = X'0003'
( "DSN_NULL_IND_00001" POSITION( 00003) CHAR(1)
  , "MYCOL1"
    POSITION( 00004:00006) DECIMAL PACKED
    NULLIF(DSN_NULL_IND_00001)=X'FF'
  , "DSN_NULL_IND_00002" POSITION( 00007) CHAR(1)
  , "MYXML1"
    POSITION( 00008) VARCHAR CLOBF
    NULLIF(DSN_NULL_IND_00002)=X'FF'

```

```

SAMPLE.UFILEREFSYS
  ß@ SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREFSYS
  Ñ@ SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREFSYS
  `@ SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREFSYS
  i@ SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREFSYS
  r@ SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREFSYS

```

```

SAMPLE.XSAM0000.T054854.XSAM0000.UFILEREFSYS
BI0MSW37
BI0MSW5K
BI0MSW55
BI0MSW6S
BI0MSW7E

```



## File References – UNLOAD to HFS

```
//CRTMNT      EXEC PGM=BPXBATCH,
// PARM='sh mkdir /u/sample; chmod 777 /u/sample;
//           chown sysadm /u/sample'

//STEP1       EXEC PGM=BPXBATCH,
// PARM='sh mkdir /u/sample/clobf`
...
TEMPLATE TCLOBF DSN /u/sample/clobf
              DIR(5) DSNTYPE(HFS)
UNLOAD TABLESPACE SAMPLEDB.SAMPLETS
PUNCHDDN SYSPUNCH UNLDDN SYSREC
FROM TABLE ADMF001.SAMPLETB
  ( MYCOL1      POSITION(*) DECIMAL(5,2)
    ,MYXML1      POSITION(*) VARCHAR CLOBF TCLOBF
  )
```

SAMPLE.UFILEREFSYSREC:

```
ß@    /u/sample/clobf/BI10FEHQ
Ñ@    /u/sample/clobf/BI10FEH0
`@    /u/sample/clobf/BI10FEH1
i@    /u/sample/clobf/BI10FEIN
r@    /u/sample/clobf/BI10FEIO
```

```
$ cd /u/sample/clobf
```

```
$ ls
```

```
BI10FEH0 BI10FEH1 BI10FEHQ BI10FEIN BI10FEIO
```



## Performance Monitoring and Tuning

- Since XML native storage is built on top of regular tablespace structure, there are no special changes in DB2 Performance Expert to support XML other than minor points - such as new XML locks.
- XML performance problem can be analyzed through [accounting traces and performance traces](#).
- There is a new LOAD MODULE for XML: DSNXML
- XML indexes have the same consideration as other indexes.
- The REORG utility should be used to maintain order and free space.
- Run RUNSTATS for statistics to help pick XML indexes.



## Use RUNSTATS

- Use RUNSTATS to collect statistics for XML data and indexes so the optimizer can pick the right access methods

```
LISTDEF DBACORDTSLIST INCLUDE  
TABLESPACES DATABASE DBACORD  
RUNSTATS TABLESPACE LIST  
DBACORDTSLIST TABLE(ALL) INDEX(ALL)
```



# 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 POIDX 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)**

133





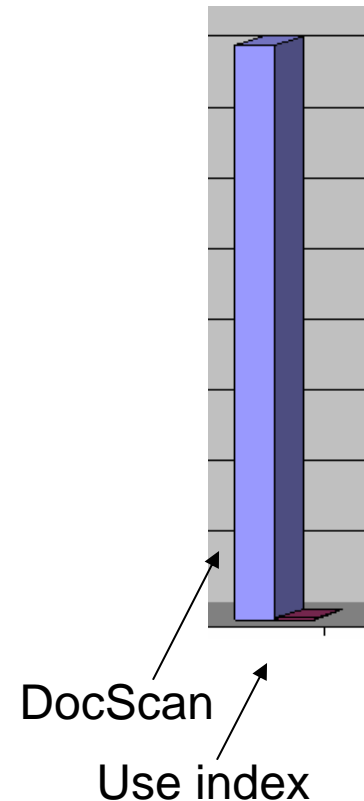
## Something Special for XML Index

- The number of keys for each document (each base row) depends on the document and XMLPattern.
- For a numeric index, if a string from a document cannot be converted into a number, it is ignored.
  - ▶ `<a><b>X</b><b>5</b></a>`, 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.



# Index Exploitation

- Indexes are critical for query performance.
- Indexes are not used for the XMLQuery function.
- Indexes are used for XMLEXists and XMLTable.
- `/po/items/item[price > 123.5]` can match XMLPattern `/po/items/item/price`
- `/po[billTo/city="London"]/items/item[price > 123.5]` matches two indexes:  
`/po/billTo/city` and  
`/po/items/item/price`  
(also `//city` and `//price` - not recommended)





## 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.	XMLEExists('/Catalog/Categories/Product[Reg Price > 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.	XMLEExists('/Catalog/Categories/Product[Reg Price > 100 and Discount > 0.1]' ... ) With indexes on: '//RegPrice' as SQL DECFLOAT and '//Discount' as SQL DECFLOAT



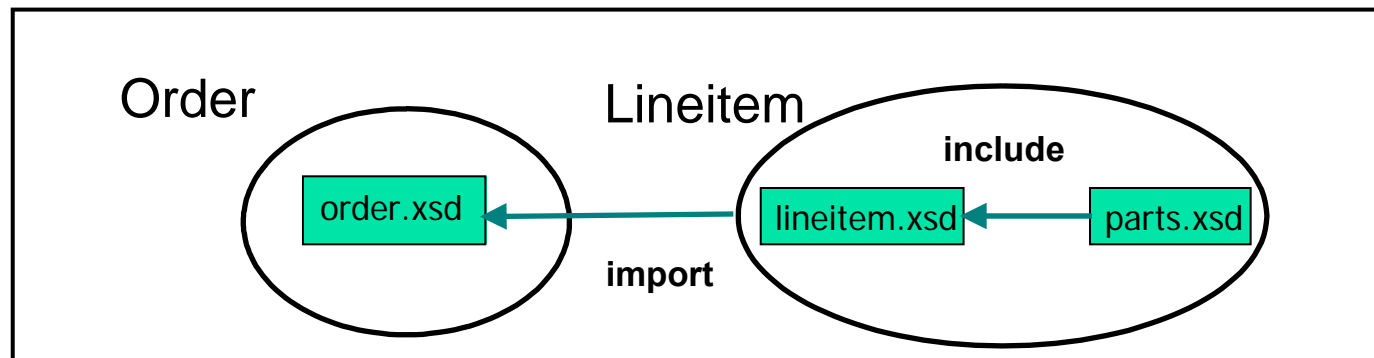
## XML Schema Support

- XML Schema adds constraints on XML data.
- Register a schema in XML Schema Repository (XSR)
  - ▶ XSR is a set of DB2 provided user tables (not catalog).
- External names
  - ▶ target namespace: e.g.,  
"http://www.ibm.com/software/catalog"
  - ▶ schema location: e.g.,  
"http://www.ibm.com/schemas/software/catalog.xsd"
- SQL identifier - used to reference schemas in SQL
  - ▶ unique identifier in DB, e.g., SYSXSR.ORDERSCHEMA
- Where are schemas used?
  - ▶ SYSFUN.DSN\_XMLValidate in SQL (UDF for XMLValidate)
  - ▶ Decomposition



## Example: Registering an XML Schema

### Orderschema



- XSR\_REGISTER('SYSXSR', 'ORDERSHEMA',  
'http://www.test.com/**order.xsd**', :xsd, :docproperty)
- XSR\_ADDSCHEMADOC('SYSXSR', 'ORDERSHEMA',  
'http://www.test.com/**lineitem.xsd**', :xsd, :docproperty)
- XSR\_ADDSCHEMADOC('SYSXSR', 'ORDERSHEMA',  
'http://www.test.com/**parts.xsd**', :xsd, :docproperty)
- XSR\_COMPLETE ('SYSXSR', 'ORDERSHEMA', :schemaproperty, 0)



## Example: Use CLP to Register XML Schema

- Register schema

```
REGISTER XMLSCHEMA
```

```
http://www.test.com/order.xsd
```

SchemaLocation

```
FROM file://C:/xmlschema/order.xsd
```

```
AS ORDERSHEMA
```

SQL Identifier

SchemaLocation

```
ADD http://www.test.com/lineitem.xsd
```

```
FROM file://C:/xmlschema/lineitem.xsd
```

```
ADD http://www.test.com/parts.xsd
```

```
FROM file://C:/xmlschema/parts.xsd
```

```
COMPLETE [ENABLE DECOMPOSITION];
```

- Remove schema

```
REMOVE XMLSCHEMA ORDERSHEMA;
```

Recommended  
for DBAs



# Using XML Schemas

- Schema validation (type annotation not kept)

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

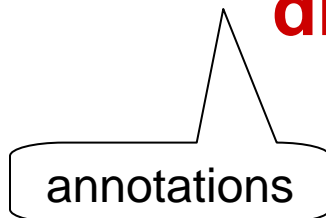
- Annotated schema-based decomposition – store using tables. (XDBDECOMPXML stored proc)

E.g. orderID -> PORDER.ORDERID

<attribute name="orderId" type="xs:string"

**db2-xdb:rowSet** = "PORDER"

**db2-xdb:column**= "ORDERID" />



PORDER	
ORDERID	ORDE
19991201-ZFG	1999-

140





## Enforcing Schema Validation for INSERT

- Create SQL PL stored procedure

```
CREATE PROCEDURE INSERTMYTABLEVALID
(IN col1 INT, IN xmlcol BLOB(1M))
LANGUAGE SQL
BEGIN
    INSERT INTO MYTABLE VALUES( col1,
XMLPARSE(DOCUMENT
SYSFUN.DSN_XMLVALIDATE(xmlcol,
`SYSXSR.ORDERSHEMA' )) );
END
```

- Applications call

```
CALL INSERTMYTABLEVALID(1, :blob);
```

- Or use instead-of trigger on a dummy view for XML docs up to 32KB (using a VARCHAR column).



## Well-formedness v.s. Validation

- Validation CPU time is 2-3 times as expensive as well-formedness checking only.
  - ▶ We are working on removing 50MB size limit on validation
- There is no difference in stored XML data whether you validate or not, unless there are default values defined in the XML schema.
- Avoid validation if you can, or
- Offload to DataPower (from application).



## XML Decomposition

- Calling XDBDECOMPXML stored procedure
- In CLP, use the following command:  

```
DECOMPOSE XML DOCUMENT  
C:/$User/IOD2006/xmldecomp/bookdetail.xml  
XMLSCHEMA SYSXSR.BOOKDETAIL;
```
- CLP calls XDBDECOMPXML stored proc implicitly.



## DB2 z/OS XML Unique Features

- Mature optimized storage infrastructure, compact storage with optional high-ratio compression
  - ▶ As little as 0.3 of original document size
- Synergy with z/OS XML system services for highly efficient parsing, achieving unparalleled XML insertion and load performance
- Schema validation exploiting new next-generation XML validating parser (XLXP-C);
- Efficient XML indexes for scalability; Patent pending highly efficient XPath algorithm for query performance;
- Partitioned table space and data sharing support;
- Redirection of XML processing to the zIIP specialty engine from DRDA workload;
- 100% redirection of XML parsing to the zIIP or zAAP specialty engine in the near future.



## Some Restrictions and Limits

- Update: whole document replacement
- Largest document size: ~2GB
- Type annotation not kept in storage after validation.
- XML indexes:
  - ▶ Numeric and string types
  - ▶ String index key length: 1000 bytes
  - ▶ Keys do not span records
- XMLEXISTS is indexable, but stage 2, always re-evaluated by DocScan after index access.
- Range-partitioned table spaces: XML partitioning follows base table partitioning
- Triggers: XML columns cannot be transition vars
- Stored procedures: no XML type arguments
- Initial sweet spot: a large number of small documents.

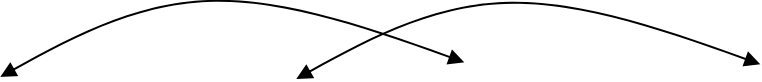


## Choose XML Document Granularity

- Choose XML document based on predominant access unit
  - ▶ Frequently searched and retrieved
  - ▶ Frequently updated
- For example, each document contains:
  - ▶ One project
  - ▶ One employee
  - ▶ One department
  - ▶ One purchase order
- However, small documents incur per document cost.



# Implicitly Created Objects



Create Table option	Base table	XML table	DocID index	NodeID index	XML Index
IN database. tablespace	NM: given, DB: given, TS: given, ST: from DB	NM: G, DB: Base, TS: G, ST: from Base	NM: G, TB: Base, IS: G, ST: from DB	NM: G, TB: XML, IS: G, ST: from Base	NM: given, TB: Base, IS: G, ST: given or DB
IN DATABASE database	NM: given, DB: given, TS: G, ST: from DB	NM: G, DB: Base, TS: G, ST: from Base	NM: G, TB: Base, IS: G, ST: from DB	NM: G, TB: XML, IS: G, ST: from Base	NM: given, TB: Base, IS: G, ST: given or DB
IN tablespace	NM: given, DB: DSNDB04, TS: given, ST: from TS	NM: G, DB: DSNDB04, TS: G, ST: from Base	NM: G, TB: Base, IS: G, ST: SYSDEFLT	NM: G, TB: XML, IS: G, ST: from Base	NM: given, TB: Base, IS: G, ST: given or SYSDEFLT
none	NM: given, DB: G, TS: G, ST:SYSDEFLT	NM: G, DB: Base, TS: G, ST:SYSDEFLT	NM: G, TB: Base, IS: G, ST: SYSDEFLT	NM: G, TB: XML, IS: G, ST: SYSDEFLT	NM: given, TB: Base, IS: G, ST: given or SYSDEFLT

NM: Name, TS: Table Space, ST: Stogroup, TB: Table, IS: Index Space, G: Generated.





## Other DDLs Support

- **ALTER TABLESPACES**
  - ▶ Only some of the attributes on the implicit XML table space can be altered
  - ▶ LOG attribute updated on the base table spaces will be propagated to the XML tablespace.
- **ALTER INDEX**
  - ▶ Allowed on the implicit NodeID index
  - ▶ Not all attributes can be altered
- **DROP TABLE**
  - ▶ Not allowed on the implicit table
  - ▶ DROP TABLE of a table with XML columns will drop all the associated implicit objects for all the XML columns
- **DROP INDEX**
  - ▶ Not allowed on the implicit index
- **DROP TABLESPACES**
  - ▶ Not allowed on the implicit table space created



## Be Careful Creating Indexes on Non-leaf Nodes

- Indexing everything is not supported. SQLCODE -20305 if a key value spans multiple rows in the internal XML table.
- Always safe to index leaf nodes ( $\leq 1000$  bytes).
- Indexing non-leaf nodes will result in concatenation of values for the key: `/.../name`, the key value will be “JohnJoe” with strip whitespace, or “ John Joe ” (w/ LFs) with preserve whitespace
  - ▶ Query has to use the same string to match:  
`.../customer[name = “JohnJoe”]`

```
<name>  
  <first>John</first>  
  <last>Joe</last>  
</name>
```



## Use Fully Specified Paths in XPath

- If **customerinfo** were a record in a programming language, what would you do to address “city”?
  - ▶ **customerinfo.addr.city**
- This is what XPath should be:  
**/customerinfo/addr/city**, not **//city**
- Wildcard **\*** and **//** should be used with a real reason.
  - ▶ **\*** for generic search need
  - ▶ **//** for recursive documents or nodes appearing at different paths or levels.

```
<customerinfo Cid="1004">
  <name>Matt Foreman</name>
  <addr country="Canada">
    <street>1596 Baseline</street>
    <city>Toronto</city>
    <state>Ontario</state>
    <pcode>M3Z-5H9</pcode>
  </addr>
  <phone type="work">905-555-4789</phone>
  <phone type="home">416-555-3376</phone>
</customerinfo>
```





## XML Indexing

- Each index adds 15-20% CPU time to the basic INSERT cost. Create indexes that are only needed.
- Specify full path for index XML patterns, avoid wild card, or descendant axis
- Rebuild index is recommended over create index
- Code XPath conditions that will match index patterns.



## Another Example for Indexes and Query

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

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

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



## Another Example for Indexes and Query (cont'ed)

```

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

**Find participant information  
about a policy.**

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

153





# DSSIZE for XML Table Space (future PTF)

- PBG XML DSSIZE = 4GB (base is simple, SEG, or PBG)
- PBR XML DSSIZE based on base DSSIZE and page size

Base DSSIZE	Base Page Size 4KB	Base Page Size 8KB	Base Page Size 16KB	Base Page Size 32KB
1-GB-4GB	4GB	4GB	4GB	4GB
8GB	32GB	16GB	16GB	16GB
16GB	64GB	32GB	16GB	16GB
32GB	64GB	64GB	32GB	16GB
64GB	64GB	64GB	64GB	32GB

- Rationale: not to limit base table max number of partitions.

SQL Ref Table 74. Maximum number of partitions allowed

DSSIZE	Page Size 4KB	Page Size 8KB	Page Size 16KB	Page Size 32KB
1-GB-4GB	4096	4096	4096	4096
8GB	2048	4096	4096	4096
16GB	1024	2048	4096	4096
32GB	512	1024	2048	4096
64GB	256	512	1024	2048

154





## Utilities

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



## XML Data Exchange in DRDA

- Support for DRDA XML data access
  - ▶ DB2 for z/OS V9 (ODBC driver support is provided)
  - ▶ DB2 Universal Database for Linux, Unix, and Windows Version 9.1 (including CLI support).
  - ▶ DB2 Universal Driver (JDBC, SQLJ) V3.1
- Transport is also in serialized XML format
- Encoding either within XML value itself (DRDA internally-encoded) or in DRDA descriptor (DRDA externally-encoded)
- Sending input data from DRDA Requester to Server
  - ▶ BLOB as host variable
    - Data is in DRDA internally encoded string value
  - ▶ DBCLOB, CLOB as host variables
    - Data is in the DRDA externally encoded string value , i.e. CCSID in DRDA descriptor
  - ▶ Server side converts XML string value to UTF-8
- Sending output data from DRDA Server to Requester
  - ▶ Always with UTF-8 DRDA externally encoded
  - ▶ Requester side converts XML string value into application host variable CCSID
- Separate send/receive for XML data if remote result set contains XML data



## Registering an XML Schema (Stored Procedures)

- XSR\_REGISTER (rschema, name, schemalocation, xsd, docproperty)
- XSR\_ADDSCHEMADOC (rschema, name, schemalocation, xsd, docproperty)
- XSR\_COMPLETE (rschema, name, schemaproperties, isUsedForDecomp)
- XSR\_REMOVE(rschema, name)
  
- Parameters:

rschema	– null or 'SYSXSR';
identifier	– SQL name (VARCHAR(128));
schemalocation	– VARCHAR(1000);
xsd	– XML schema document
(BLOB(30M));	
docproperty	– BLOB(5M), may be used by tools;
schemaproperties	– same as docproperties
isUsedForDecomp	– INTEGER, 1 yes, 0 no.
  
- Java Driver (JCC) provides a set of APIs for schema registration



# Relational View of XML Data

```
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;

XMLTABLE function available after GA

```
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;
```





# XML Related Locks

- New XML lock type, value '34'x in instrumentation

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

159





## Decomposition for XML

- Consider decomposing XML into relational data, only if
  - ▶ The schema is regular, fixed and very stable.
  - ▶ The follow-on processing is all relational and there is no need to get data back to XML.
  - ▶ Existing software requires efficient relational storage
- Decomposition may not be a good choice, if
  - ▶ Schema changes frequently
  - ▶ Schema is irregular and complex, decomposition requires dozens or even hundred tables.
  - ▶ Significant new application development and generates XML
  - ▶ Need to preserve digital signatures
- Decomposition CPU time is more expensive than native XML.



## 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 collect statistics for XML data and indexes
- Tip 9: How to use SQL/XML publishing views to expose relational data as XML
- Tip 10: How to use XMLTABLE views to expose XML data in relational format
- Tip 11: For short queries or OLTP applications, use SQL/XML statements with parameter markers

Adapted from LUW's Top 15 best practices – 11 applies to z/OS  
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0610nicola/>



# Agenda

- *Introduction*
  - ▶ *What is XML?*
  - ▶ *XML storage options:*  
*XML-enabled databases vs. hybrid database (DB2 9)*
- *Processing XML with DB2 pureXML*
  - ▶ *Publishing*
  - ▶ *Storing*
  - ▶ *Querying*
  - ▶ *Indexing*
  - ▶ *Updating*
- *Specifics for DB2 for z/OS*
- **Specifics for DB2 for LUW**



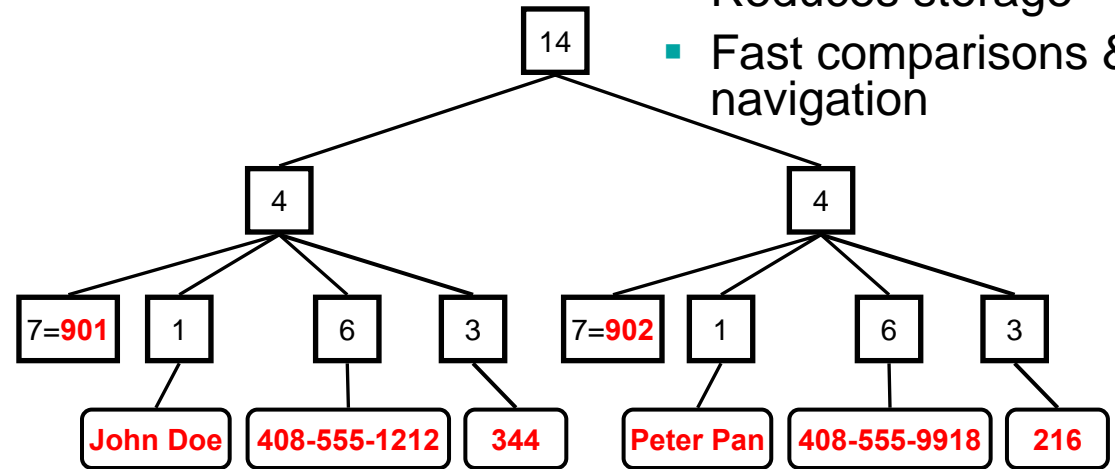
# Efficient Document Tree Storage

```

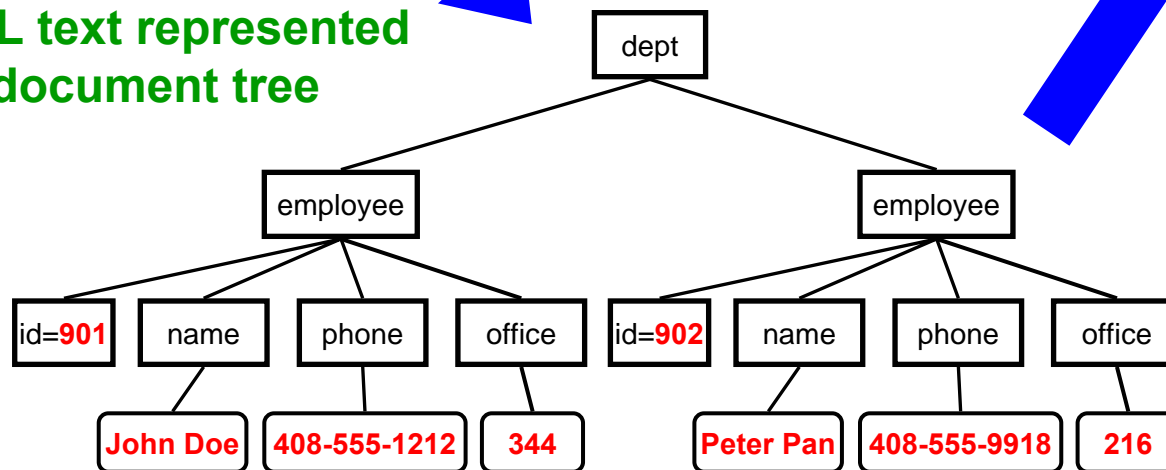
<dept>
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>

```

- Reduces storage
- Fast comparisons & navigation



XML text represented  
as document tree

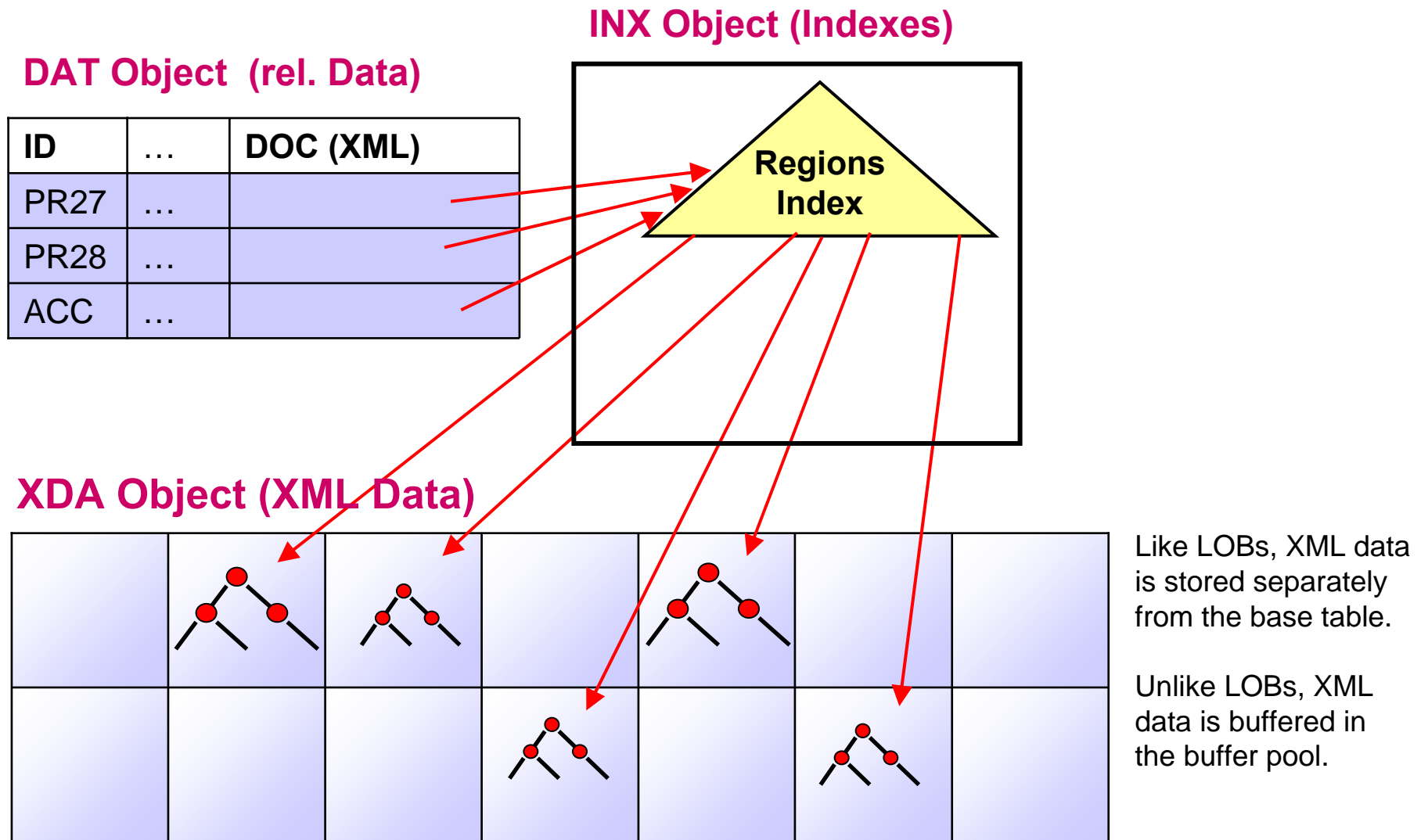


Tag names encoded  
as unique integers

SYSIBM.SYSXMLSTRINGS	
String table	
14	dept
4	employee
1	name
7	id
6	phone
3	office



## Recap: XML Storage in DB2 9





## Base Table Inlining and Compression

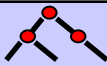
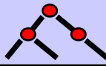
```
create table dept (deptID char(8),...,deptdoc XML inline length 10000);  
alter table dept compress yes;
```

Now let's see how that works....

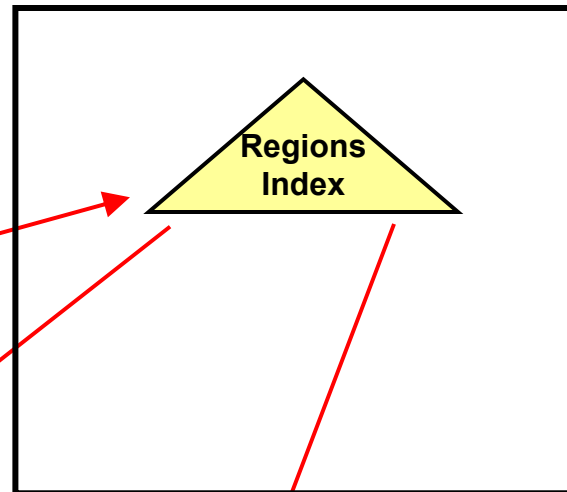


## DB2 9.5: Base Table Inlining for small docs

### DAT Object (rel. Data)

ID	...	DOC (XML)
PR27	...	
PR28	...	
ACC	...	

### INX Object (Indexes)

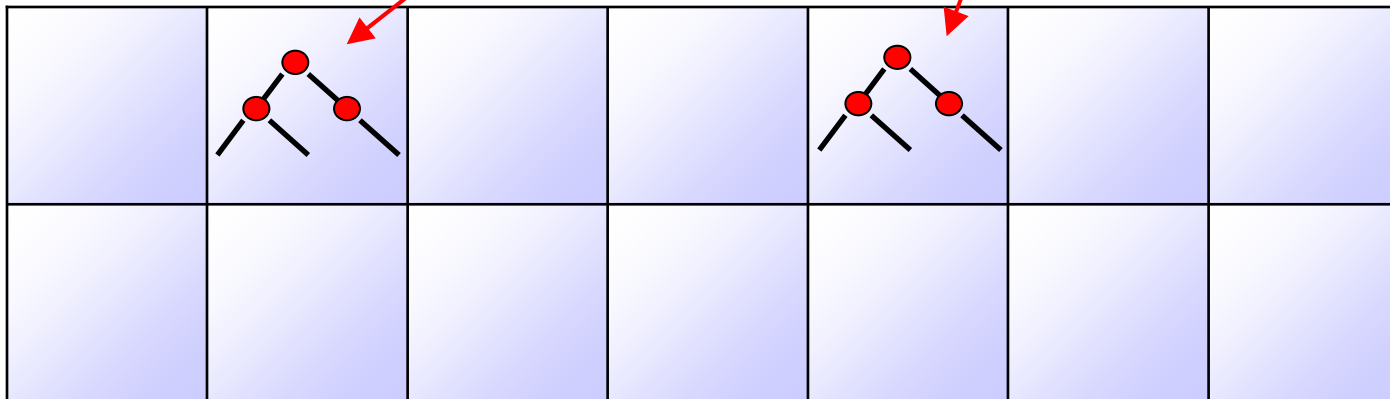


Documents which are small enough can be stored in the base table....

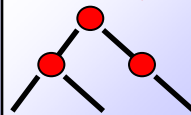
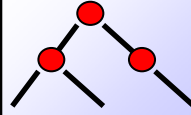
...and can be compressed !

80% less storage +  
2x performance  
increase over DB2 9

### XDA Object (XML Data)






The diagram shows a grid of cells representing the XDA Object. The grid has 2 rows and 7 columns. The first row has XML tree diagrams in the second and fifth cells. The second row is empty. Red arrows point from the XML tree diagrams in the DAT Object table to the XML tree diagrams in the XDA Object grid, indicating that the XML data is stored in the XDA Object.

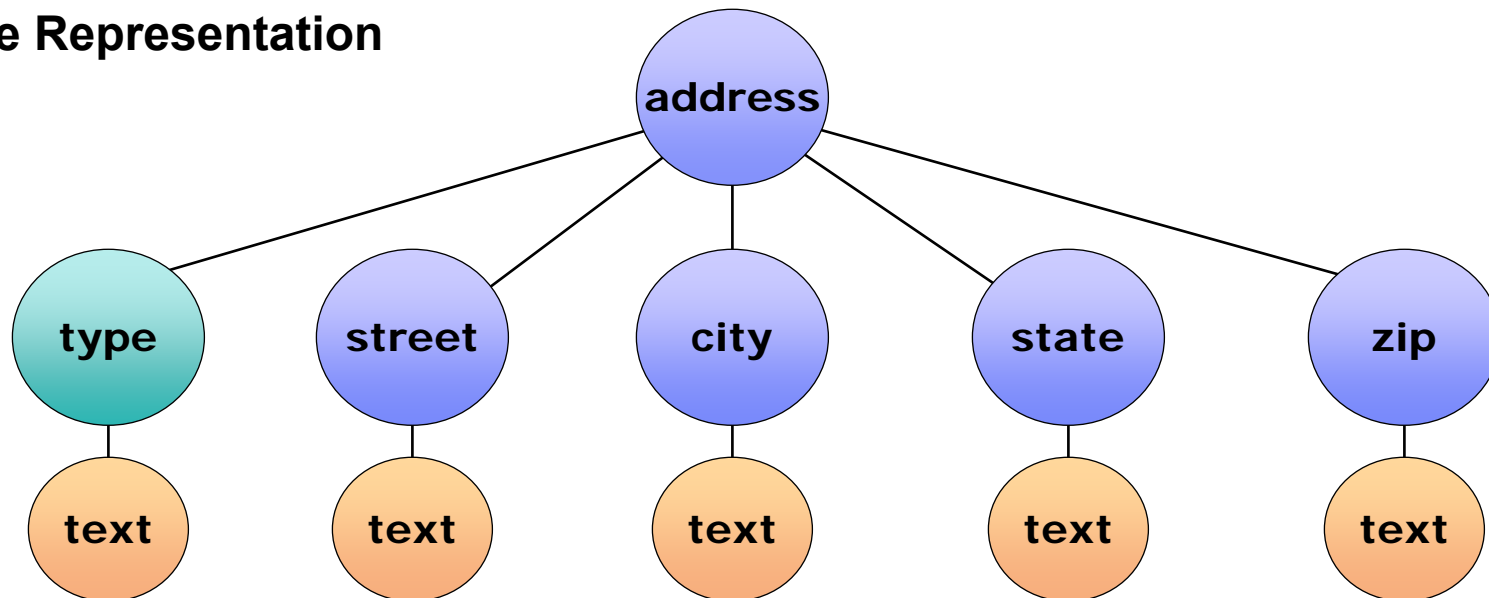


# Structural Inlining

```
<address type="US">  
  <street>555 Bailey Ave</street>  
  <city>San Jose</city>  
  <state>CA</state>  
  <zip>95141</zip>  
</address>
```

-  - Element Node
-  - Attribute Node
-  - Text Node

## Tree Representation

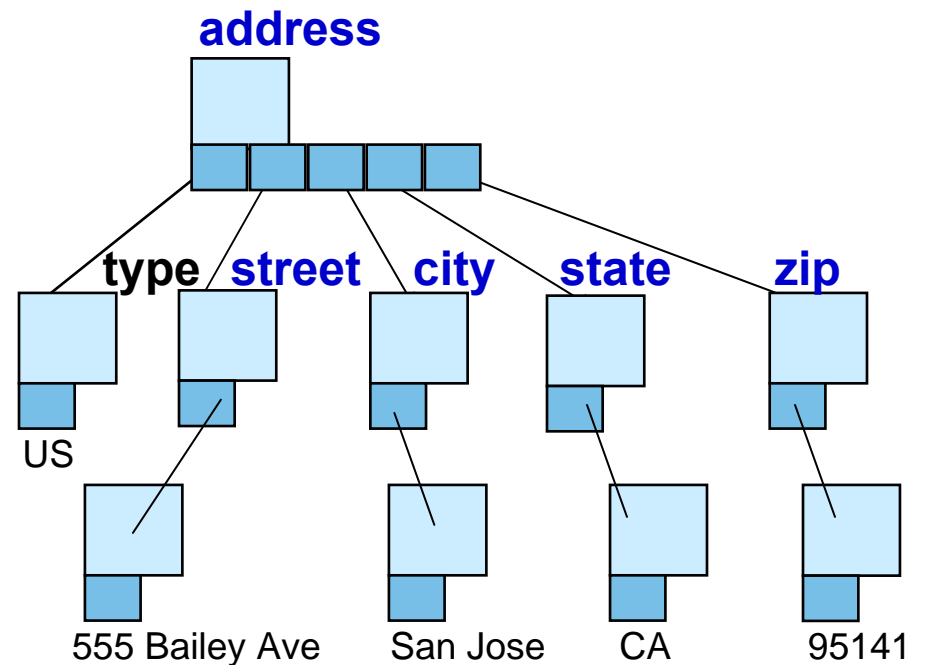




## Hierarchical Storage: the simple way

```
<address type="US">  
  <street>555 Bailey Ave</street>  
  <city>San Jose</city>  
  <state>CA</state>  
  <zip>95141</zip>  
</address>
```

Size of Document: 113 bytes



**Space requirement:** 508 bytes

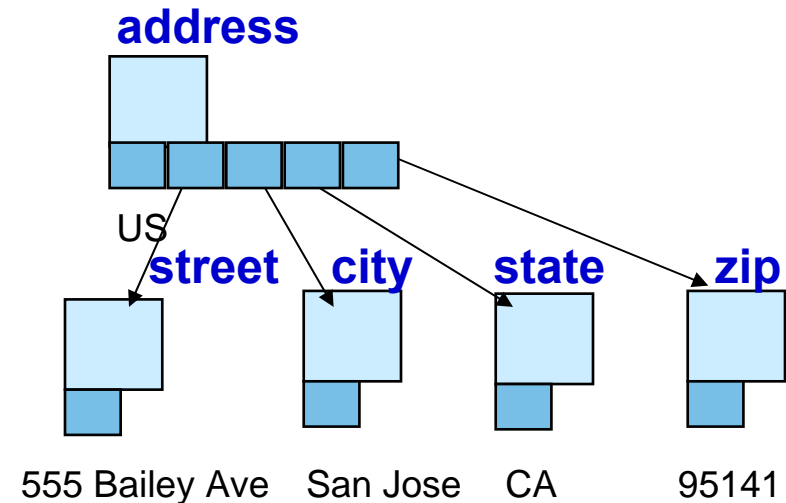
**Storage Ratio:** ~5



## DB2 9 on-disk Storage: Attribute & Text node Inlining

```
<address type="US">  
  <street>555 Bailey Ave</street>  
  <city>San Jose</city>  
  <state>CA</state>  
  <zip>95141</zip>  
</address>
```

Size of Document: 113 bytes



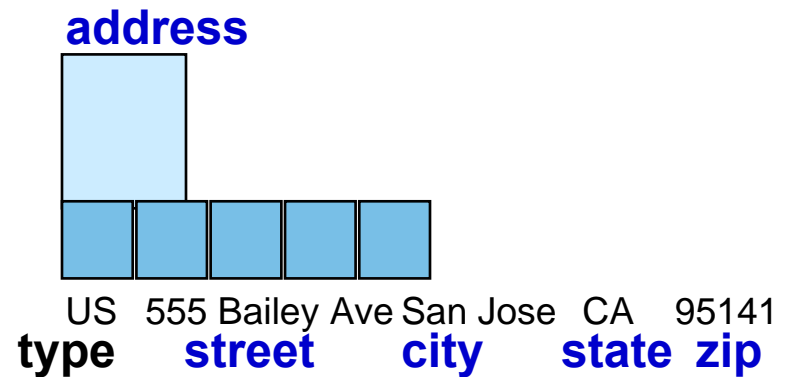
**Space requirement:** 308 bytes  
**Storage Ratio:** ~3



# DB2 9.5 on-disk Storage Structure Inlining Feature

```
<address type="US">  
  <street>555 Bailey Ave</street>  
  <city>San Jose</city>  
  <state>CA</state>  
  <zip>95141</zip>  
</address>
```

Size of Document: 113 bytes



**Space requirement:** 124 bytes

**Storage Ratio:** ~1

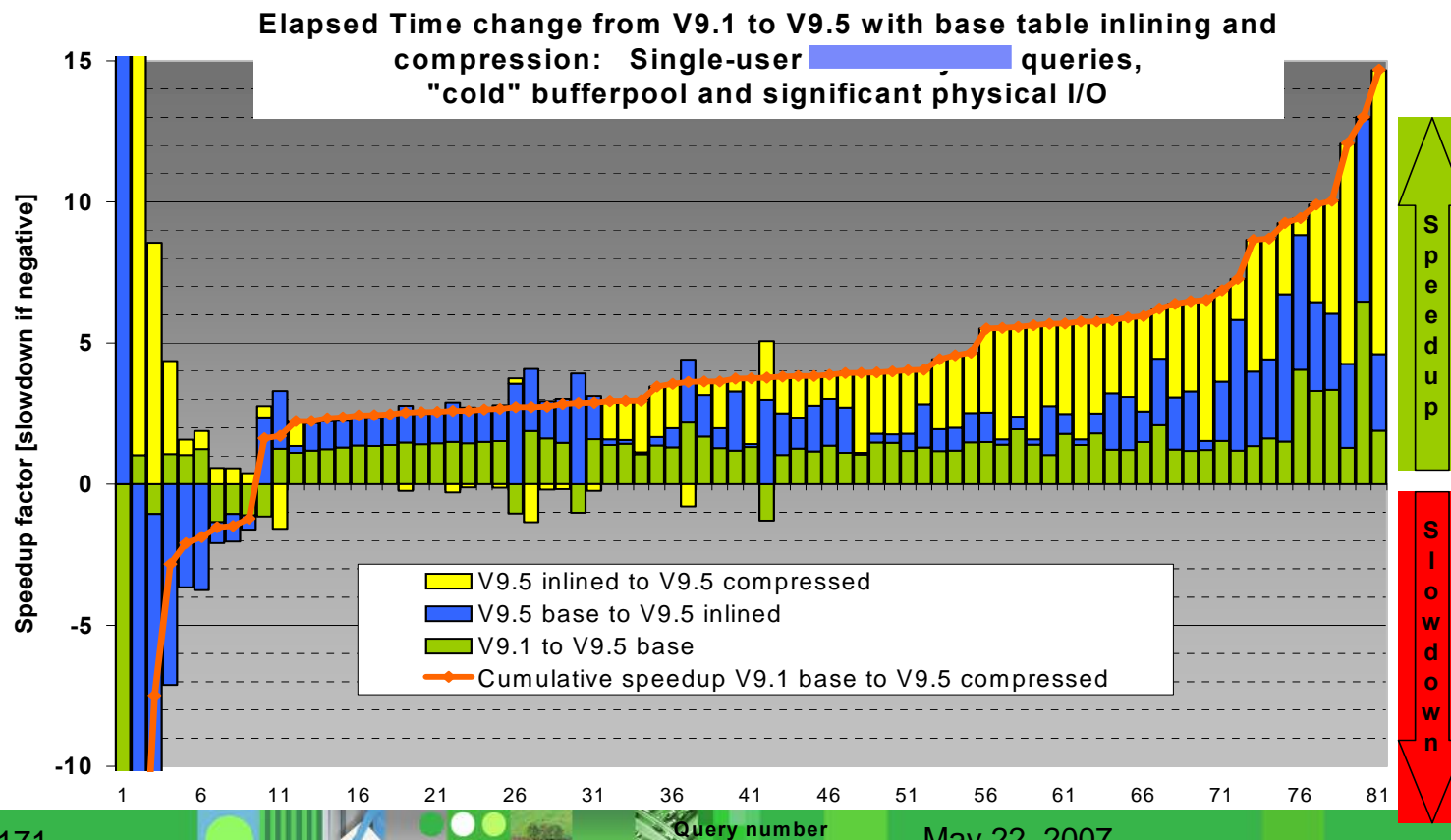
## Conditions for Inlining Elements:

- ▶ Should either have NO children OR only 1 text child
- ▶ Should NOT have any attributes
- ▶ Should NOT have a URI or Prefix or Typing information



## Financial Transaction Application – Internal Test

- Storage: Database shrinks: 9.5 database with inline and compression is 20% size of 9.1
- Cold bufferpool run: Queries require physical I/O
  - Speed up of 1.5X – 15X (average 3.5X) from V9.1 to V9.5 with inlining and compression
- Warm bufferpool run: Queries with little/no physical I/O
  - 2.5X improvement for these queries from V9.1 to V9.5. Inlining/compression does not contribute more





## Controlling XML indexing behavior

- Example

```
create index x1 on T(doc)  
generate key using xmlpattern  
'/a/b/@id' as sql double
```

- In DB2 9: one can insert docs where @id is not numeric

- In DB2 9.5, new option to control this

- Example

```
create index x2 on T(doc)  
generate key using xmlpattern  
'/a/b/@id' as sql double reject invalid values
```

- Applies to double, date, and timestamp index



## Triggers used to XMLValidate

- Enable DBAs to force validation with triggers

```
CREATE TRIGGER TR1 NO CASCADE BEFORE INSERT ON T
REFERENCING NEW AS n
FOR EACH ROW MODE DB2SQL WHEN (n.C2 is not validated)
BEGIN ATOMIC
  set (n.c1) = (500);
  set (n.c2) = xmlvalidate(n.c2 ACCORDING TO XMLSCHEMA
    URI 'http://my-namespace');
  set (n.c3) = NULL;
END%
```

- Only the simplest "validation triggers" will not affect insert performance
- Triggers with conditions will certainly affect insert performance
  - ▶ Order of magnitude: 10%



## How Triggers and XML co-exists

- DB2 9:
  - ▶ Cannot reference XML transition variables in triggers
  - ▶ Cannot reference an XML value from NEW or OLD
  - ▶ Cannot reference an XML value in the WHEN condition.
- Viper 2
  - ▶ Before-triggers supported
  - ▶ Only XMLValidate can reference XML transition variables
  - ▶ Trigger condition can only be "is validated" when referencing an XML transition variable
    - This is useful to avoid overwriting user specified XMLValidate



## Two ways to query XML data in DB2 9

- SQL/XML

- ▶ SQL as the primary language
- ▶ Optional: XQuery embedded in SQL



SQL

SQL/XML  
XQuery

- XQuery

- ▶ XQuery as the primary language
- ▶ Optional: SQL embedded in XQuery



XQuery

XQuery  
SQL

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0606nicola/>



## XMLQuery, XMLEExists, XMLTable simplification: default passing mechanism

- Implicit XQuery variable binding
  - ▶ Named variables are the names of the columns in the tables
  - ▶ Used in XMLQuery, XMLEExists, XMLTable
- XQuery has automatically access to column values via the column name.
- Example
  - ▶ create table friendly (**J** int, **doc** xml)
  - ▶ select XMLQuery('\$**DOC**/a/b, <j>{**J**}</j>') from friendly
- Remember:
  - ▶ XQuery is case sensitive
  - ▶ SQL is not case sensitive, unless you double quote it...
- Observations:
  - ▶ If there is ambiguity about which column an XQuery variable refers to, then use the explicit passing mechanism.



## Querying XML using SQL or SQL/XML

create table dept(deptID char(8), deptdoc xml)

```
select deptID,  
       xmlquery('for $i in $deptdoc/dept  
                let $j := $i//name  
                return $j' passing deptdoc as "deptdoc")  
from dept  
where deptID LIKE "PR%"  
and xmlexists('$deptdoc/dept[@bldg = 101]' passing deptdoc  
as "deptdoc")
```

```
select deptID,  
       xmlquery('for $i in $DEPTDOC/dept  
                let $j := $i//name  
                return $j')  
from dept  
where deptID LIKE "PR%"  
and xmlexists('$DEPTDOC/dept[@bldg = 101]')
```



## XMLTable() - default column specification

- When "flattening" XML on a path, no column spec is needed anymore.
  - ▶ Rows of one column of XML type are created by XMLTable
  - ▶ The 'default column' has no name.

- Example:

- ▶ list all the assembly steps of a Mustang
- ▶ example uses default passing mechanism

```
select t.*  
from cars,  
      xmltable('$ASSEMBLY/steps/step') as t  
where cars.type = 'Mustang'
```

- ▶ Is equivalent to:

```
select t.*  
from cars,  
      xmltable('$ASSEMBLY/steps/step'  
               columns x xml path '.') as t  
where cars.type = 'Mustang'
```



## User-friendly XML Publishing functions

- Goal:
  - ▶ Easy to use "convenience" publishing functions
  - ▶ For more difficult usages, use existing functions (e.g. XMLElement).
- Functions:
  - ▶ select XMLRow(c1, c2) from T
    - Returns an XML doc per row of the table T where each column is mapped to an element
  - ▶ select XMLRow(c1, c2 option as attributes) from T
    - Returns an XML doc per row of the table T where each column is mapped to an attribute
  - ▶ select XMLGroup(c1, c2) from T
    - Returns an XML doc for the entire table T where each row is mapped to an element
  - ▶ select XMLGroup(c1, c2 option as attributes) from T
    - Returns an XML doc for the entire table T where each row is mapped to an element, each column is mapped to an attribute
  - ▶ select XMLElement(name "names", XMLNamespaces('http://www.ibm.com/xml/'), XMLRow(c1 as "ibm:first", c2 as "ibm:last")) from T
    - Controls the element names and namespaces
- Observations:
  - ▶ If column value is null, then no element/attribute created
  - ▶ Existing functions have an option to control that behavior

```
<row><C1>3</C1><C2>text</C2></row>
```

```
<row C1="3" C2="text"/>
```

```
<rowset>
<row C1="3" C2="text"/>
...
</rowset>
```



## XSLT built-in function

### ■ New function: XSLTransform

- ▶ Parameters: XML doc to be transformed, XSLT stylesheet, optional XSLT parameters in XML doc
- ▶ Supports XSLT Version 1.0
- ▶ All input parameters can be XML, CLOB, BLOB, or VARCHAR
- ▶ Return type is by default CLOB but can be overwritten to CHAR, VARCHAR, BLOB
  - No XML return type because the return might not even be XML

### ■ Example:

```
select XSLTransform (xmldoc USING  
xsltdoc)  
from T
```

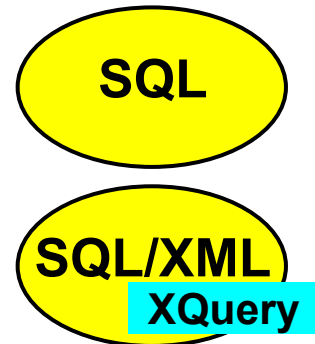
```
select XSLTransform (xmldoc USING  
xsltdoc WITH paramdoc AS CLOB)  
from T
```



## Two ways to query XML data in DB2 9

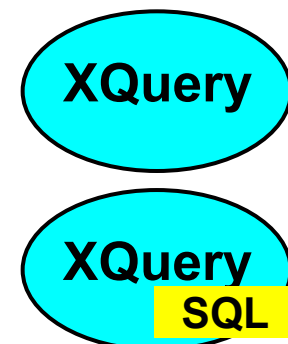
- SQL/XML

- ▶ SQL as the primary language
- ▶ Optional: XQuery embedded in SQL



- XQuery

- ▶ XQuery as the primary language
- ▶ Optional: SQL embedded in XQuery



<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0606nicola/>



## XQuery Support in DB2 9

```
create table dept(deptID char(8), deptdoc xml);
```

```
for $d in db2-fn:xmlcolumn('dept.deptdoc')/dept
let $emp := $d//employee/name
where $d/@bldg > 95
order by $d/@bldg
return <EmpList>
      {$d/@bldg, $emp}
    </EmpList>
```

XQuery



## XQuery Support in DB2 9

```
create table dept(deptID char(8), deptdoc xml);
```

```
for $d in db2-fn:xmlcolumn('dept.deptdoc')/dept
let $emp := $d//employee/name
where $d/@bldg > 95
order by $d/@bldg
return <EmpList>
      {$d/@bldg, $emp}
    </EmpList>
```

XQuery

```
for $d in db2-fn:sqlquery('select deptdoc from dept
                           where deptID LIKE "PR%" ')/dept...
```

```
for $d in db2-fn:sqlquery('select dept.deptdoc from dept, unit
                           where dept.deptID=unit.ID
                           and unit.headcount > 200')/dept...
```

XQuery  
SQL

Optional:  
SQL embedded



## Passing parameters from XQuery to sqlquery()

- XQuery expressions can be passed as parameters to the SQL statement in sqlquery().

- Example:

```
let $id := 9
for $doc in db2-fn:sqlquery(
    "select doc from T where id = parameter(1)", $id)
return $doc
```

- New function "parameter()" can only be used in this specific context
  - ▶ Return type is calculated by the compiler
    - the parameter marker, i.e. "?", rules are used
  - ▶ Return type can be specified explicitly via casts
    - just like parameter markers can
    - db2-fn:sqlquery("select doc from T where id = cast(parameter(1) as integer)", \$id)
  - ▶ The XQuery expression must be castable to the parameter() type
    - The XMLCast() rules are used



## Check-constraints for XML validation

- Viper 2 enables DBAs to enforce validation with check constraint.

```
ALTER TABLE T2
  ADD CONSTRAINT CK_VALIDATED
  CHECK (XMLCOL IS VALIDATED
        ACCORDING TO XMLSCHEMA ID RICK.MYSCHEMA
```
- DB2 9 already supported a limited IS VALIDATED predicate that can also be used in check constraints
  - ▶ 

```
SELECT XMLCOL
  FROM T1
 WHERE XMLCOL IS VALIDATED
```
  - ▶ 

```
ALTER TABLE T1
  ADD CONSTRAINT CK_VALIDATED
  CHECK (XMLCOL IS VALIDATED)
```
- Note that constraints are executed **AFTER** the statement is executed and **AFTER** triggers are fired



## Compatible schema evolution

register xmlschema **sample\_v1**

register  
schema

insert into T values (xmlvalidate(?))

Insert  
validated docs

update xmlschema **sample\_v1** with **sample\_v2**

Evolve  
schema

insert into T values (xmlvalidate(?))

Insert more  
validated docs

- XML data is not touched
- Still only one XML schema
- Docs continue pointing to the same schema
- SQL does not need changes






ID	...	DOC (XML)
PR27	...	
PR28	...	
ACC	...	
Z04	...	
Z05	...	

Table with  
XML column

### Schema repository

[http://www.sample\\_v2.com](http://www.sample_v2.com)

[http://www.PO\\_v1.com](http://www.PO_v1.com)

[http://www.emp\\_v1.com](http://www.emp_v1.com)



## XML Schema Validation Improvement in 9.5

- Compatible schema evolution
  - ▶ Helps in evolving the XML schema
  - ▶ `update xmlschema mySchema with myNewSchema`
- Dramatically improved validation performance for complex XML Schemas
  - ▶ Standard XML schemas are particularly improved because they have a lot of "comments" as XML Schema annotation
  - ▶ XML Validation cpu overhead is now between 10-25%
- Use CONSTRAINTS to enforce XML validation
  - ▶ ...  
`check (doc is validated according to xmlschema id mySchema)`  
...
- Use TRIGGERS to force XML validation.
  - ▶ ...  
`set (n.doc) = xmlvalidate(n.doc according to xmlschema id mySchema);`  
...



## Other XML support in DB2

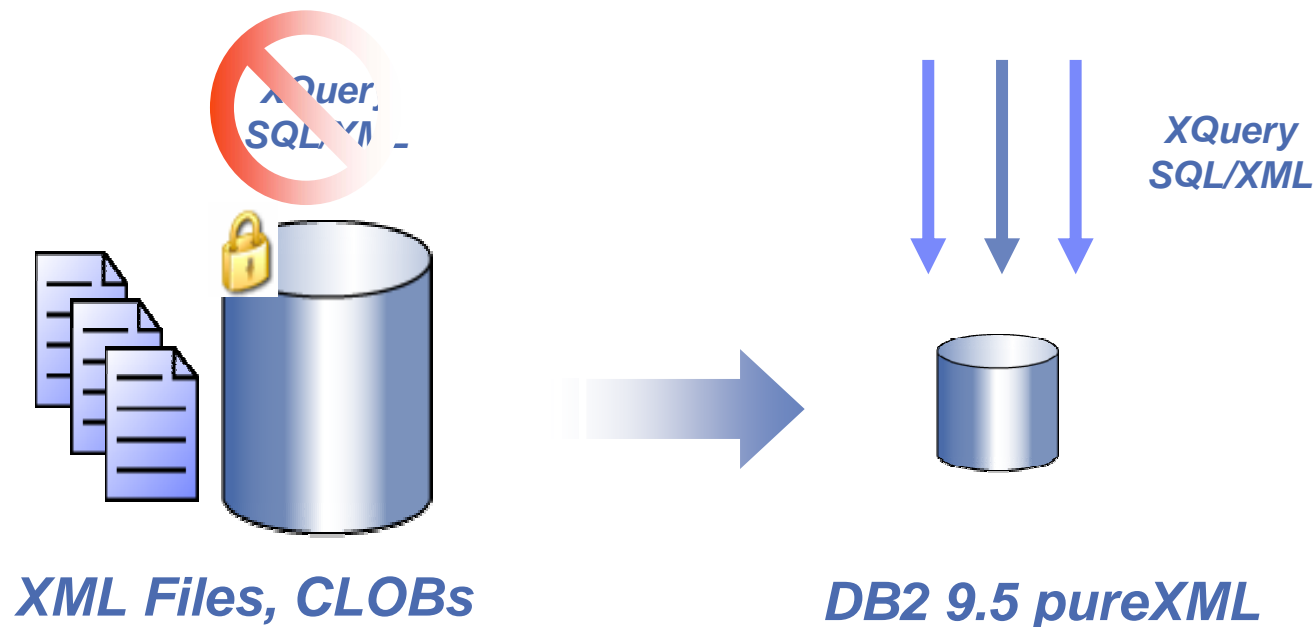
- Import, Export, Load, Backup/Restore
- HADR
- Federation, Replication
- Runstats, Explain
- XML Type in Stored Procedures
- API Extensions for SQL/XML and XQuery (JDBC, ODBC/CLI, .NET, etc.)
- New shredding capabilities
- *and more...*
- Future: DPF, Range Partitioning, MDC, etc.



## **DB2 9.5 “Transactional pureXML”**

**Half the storage** of XML Flat Files/CLOBs

**Double the throughput** of DB2 9





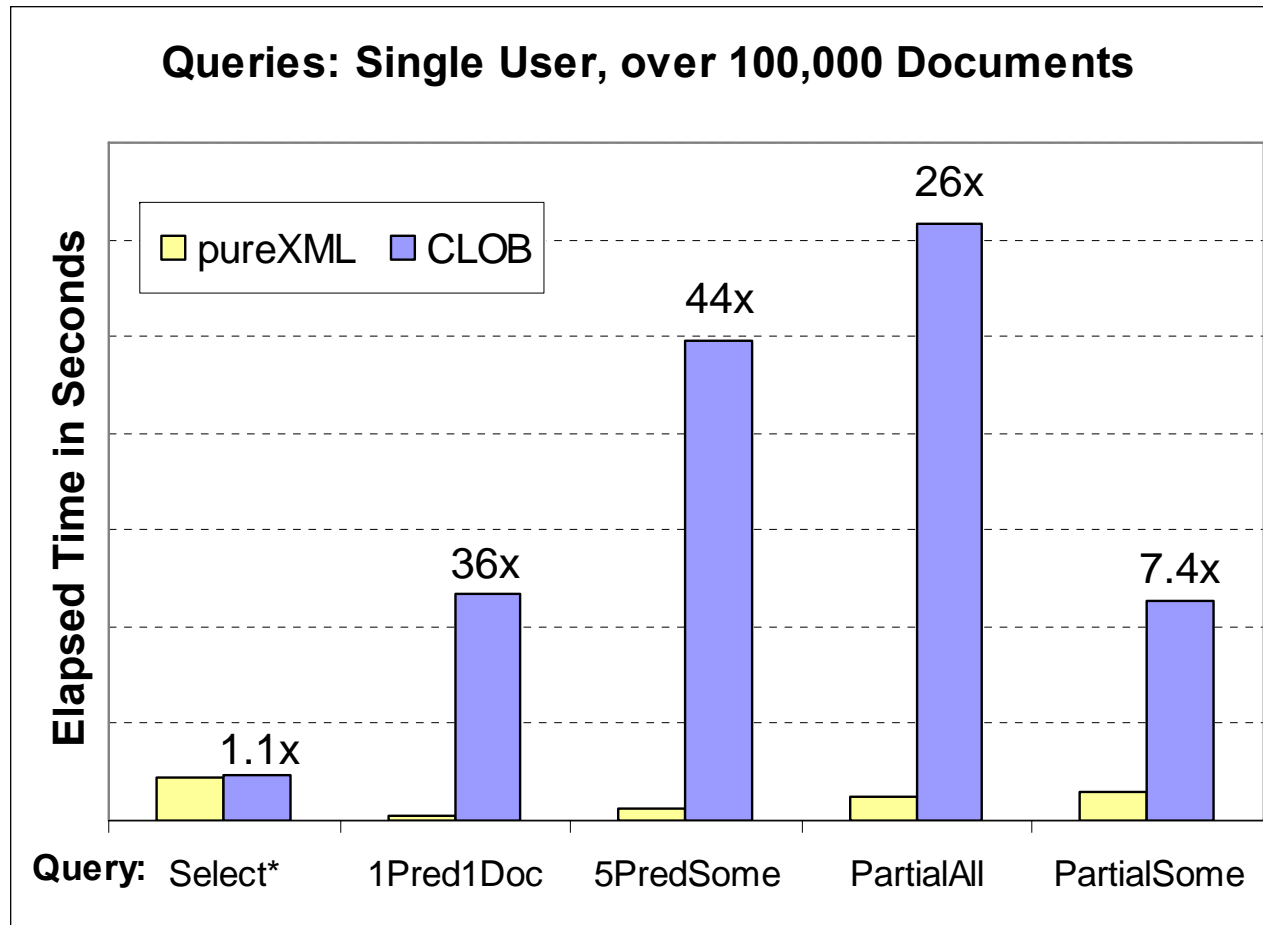
## DB2 9.5 Transactional XML

- **Inlining and compression**
  - ▶ Major NA bank gets **6x** storage savings and **2x** faster inserts than DB2 9
  - ▶ Transaction Processing over XML ( TPOX ) benchmark: **1/3** storage, **2x** throughput than DB2 9
- **Industry first XQuery Update**
  - ▶ **2-3** times faster than the stored procedure approach in DB2 9
- **Faster insert with schema validation**
  - ▶ Up to **5** times faster than DB2 9
- **Instant compatible schema evolution**
- **Enable existing customer base**
  - ▶ Non-Unicode, Offline Load, Replication, Federation
- **Richer tool support:**
  - ▶ IBM Data Studio, RDA, DB2 Warehouse, and many more
  - ▶ Info 2.0
  - ▶ Altova, Skytide, and many more



# pureXML™ vs CLOB Query Performance

**Lower is better**



## Five Test Queries

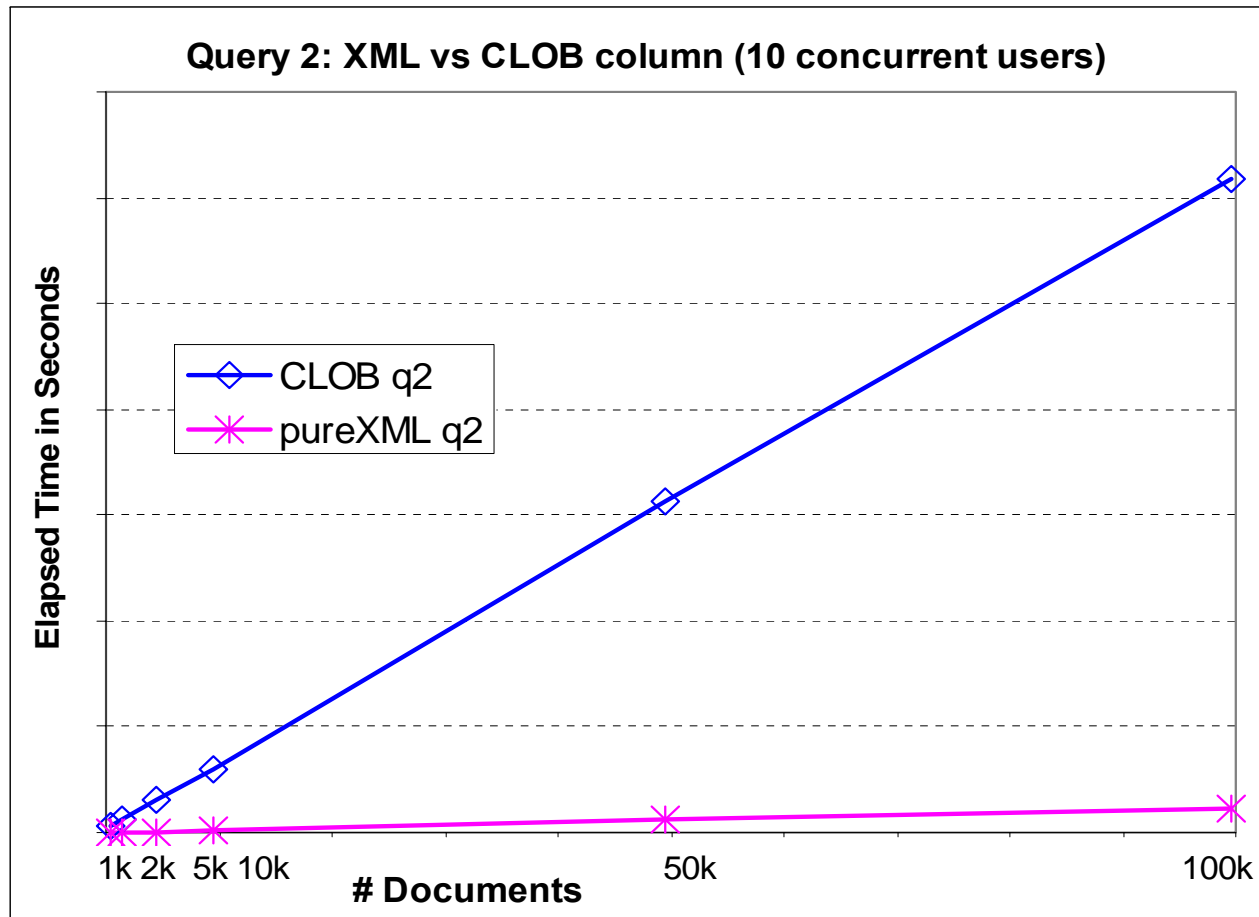
### Query details:

- Full document retrieval, all documents
- Full document retrieval of one doc, one predicate
- Full document retrieval of multiple documents, 5 predicates
- Partial retrieval of all documents
- Partial retrieval of some documents, 1 predicate



# pureXML™ vs CLOB Query Performance

**Lower is better**



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

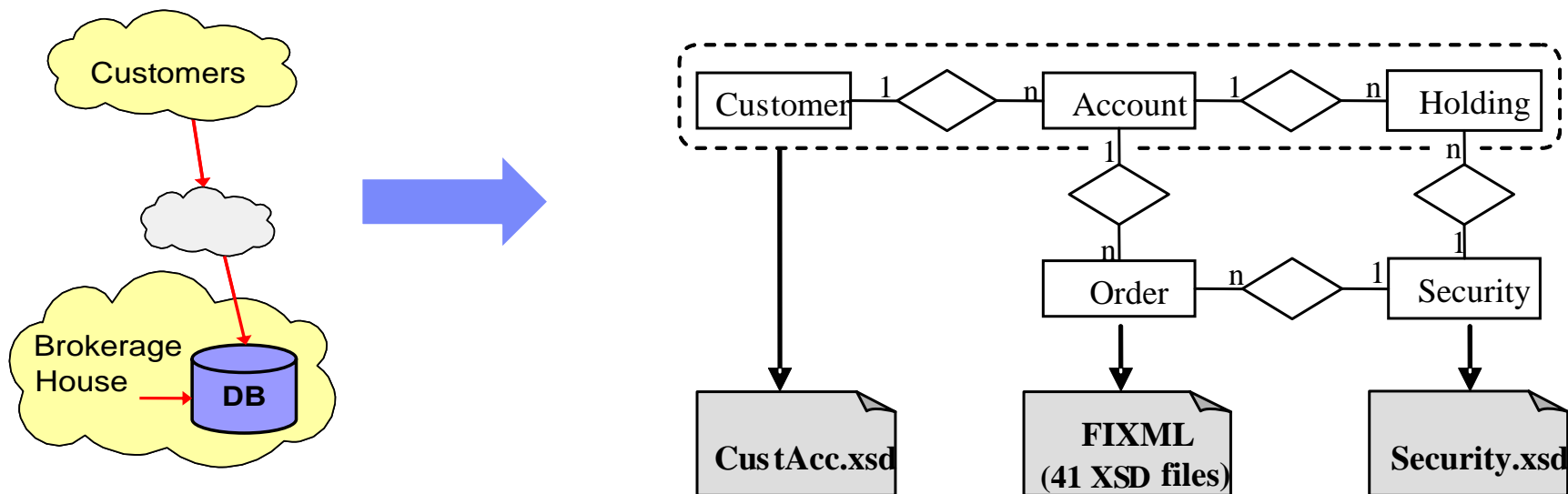
No index is used.

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



## DB2 9 Performance and Scalability

### Financial transaction processing scenario (FIXML)

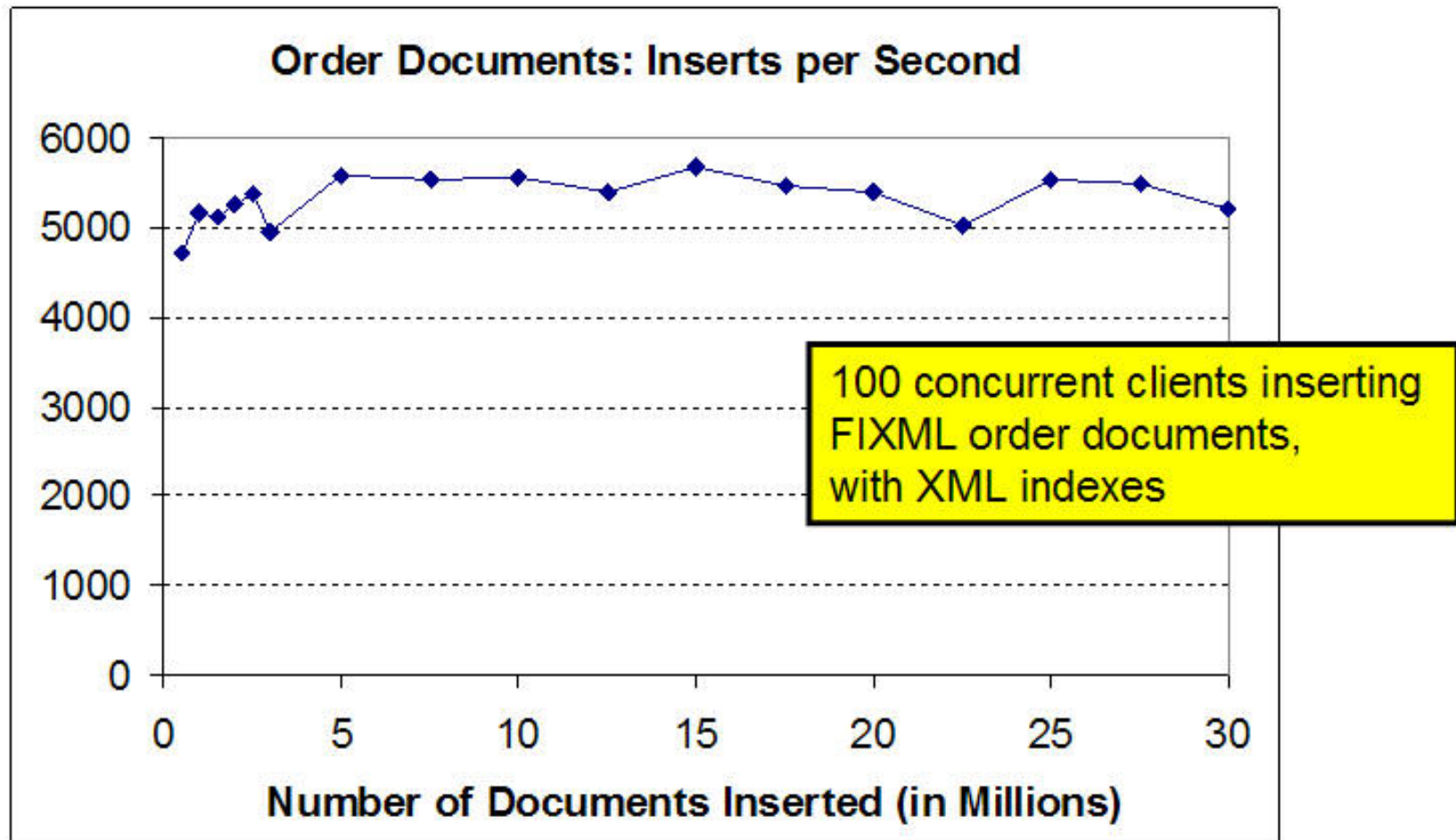


- Database: 36 million XML documents, 2kb to 20kb
- 23 XML Indexes
- High concurrency, queries/insert/update/delete

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



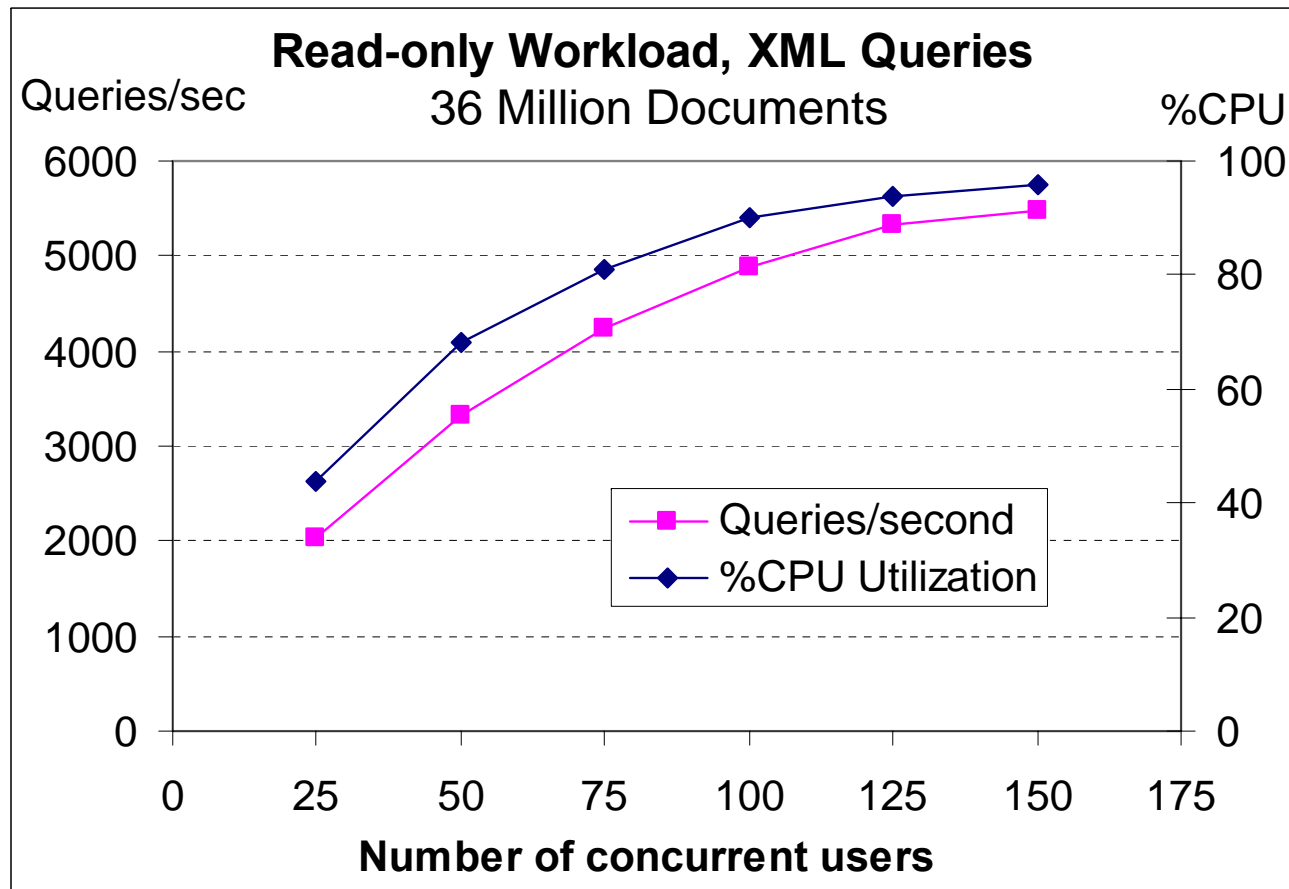
## Constantly High XML Insert Throughput



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



## XML Query Scalability



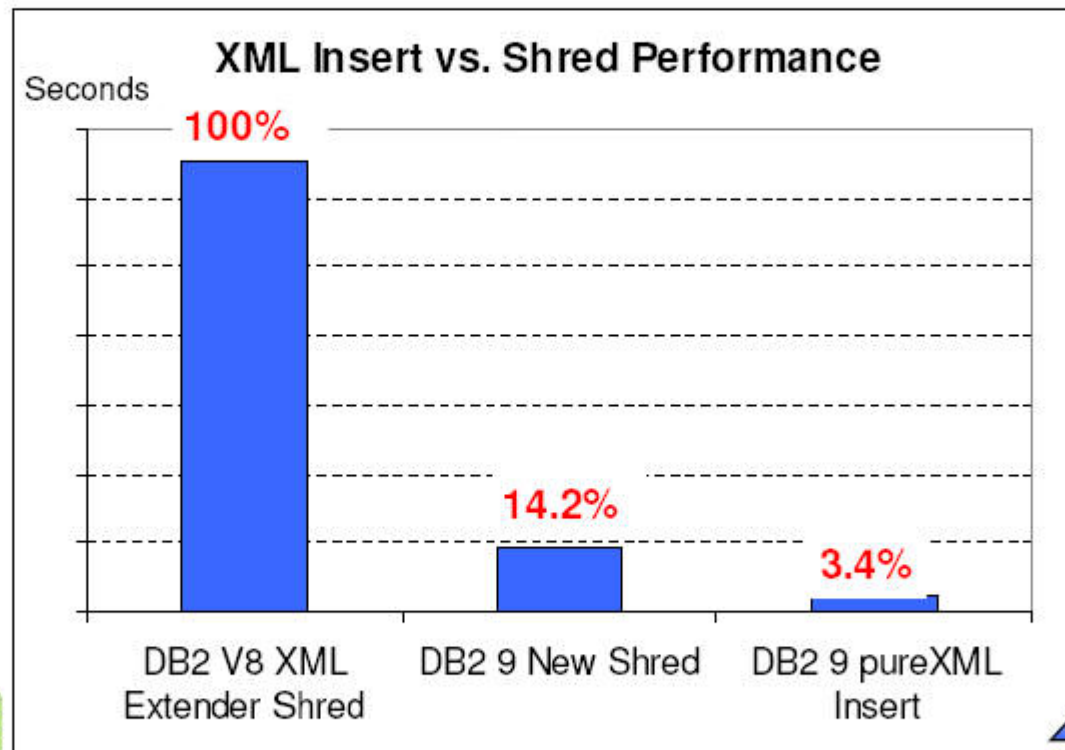
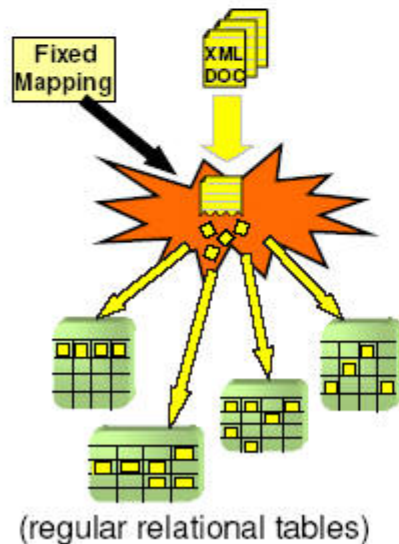
Throughput increases with number of users until machine capacity is max'ed out.  
This is text-book scalability.



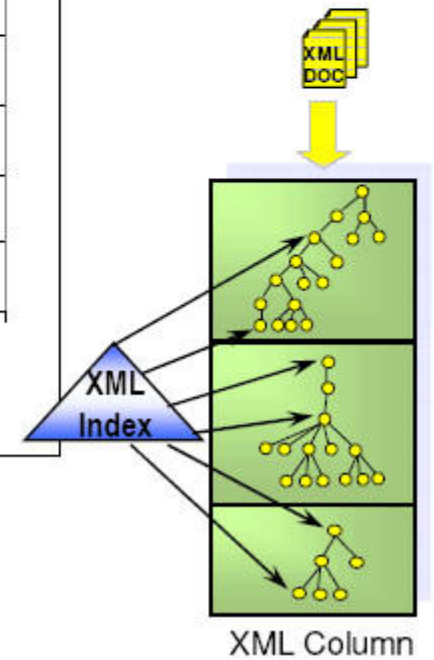
# pureXML Insert vs. Shred Performance

10,000 Customer Documents, 4Kb to 20Kb...

...shredded to  
87 columns in  
12 tables.



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





## Replication support for XML

- Queue replication is supported in Viper 2
  - ▶ Capture (log reading) and Apply (logical application) technology
  - ▶ Uses MQSeries to move the operations to the destination system
  - ▶ Replication cannot be filtered based on XML predicates
  
- HADR is already supported in DB2 9
  - ▶ Disaster recovery
  - ▶ Typically full database replicated
  - ▶ Log shipping technology
  - ▶ Different levels of synchronization between primary and backup
  
- SQL replication is not supported
  - ▶ Staging table technology
  - ▶ SQL connections to transfer data



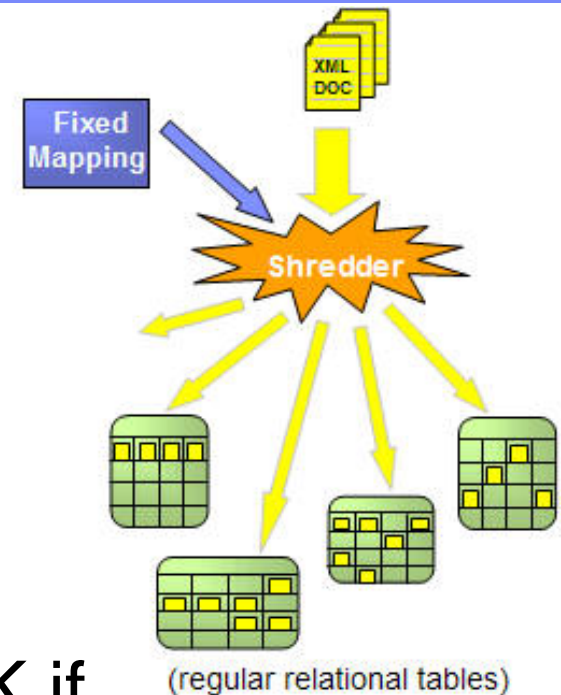
## Federation support for XML

- Federation will be supported between DB2 servers
- One can create a "nickname" over a table that has XML typed columns
- That nickname can be used in SQL/XML and XQuery statements



## Why Shredding?

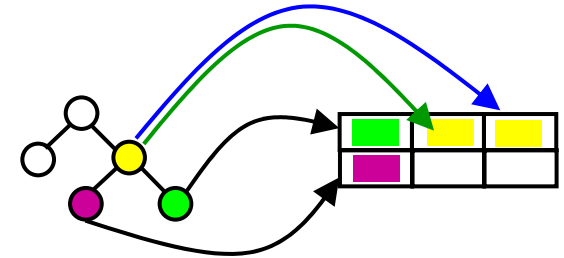
- Shredding is bad if
  - ▶ XML Schema & Mapping is complex
  - ▶ XML Schema changes
- Shredding *may* still be needed & OK if
  - ▶ Existing SQL applications need the data
  - ▶ Your BI & Reporting Tools only have relational APIs
  - ▶ XML structure no longer relevant after shredding
  - ▶ XML structure is simple & very regular
  - ▶ XML schema changes are rare and benign





## The “New Shred” in DB2 9 (XML Decomposition)

- Mapping from XML to relational:  
annotations to the XML schema
  - ▶ aka “Annotated Schema Shred”
  - ▶ uses XML schemas from the schema repository
- Replaces XML Extender shred - a lot faster !



*"Our tests of the new XML Decomposition in DB2 Viper showed an 8 times performance improvement over the DB2 V8 XML Extender."*

**Dr. Andreas Birkendorf, DOUGLAS Informatik & Service GmbH**



## UNICODE & XML

- XML is always stored internally as UTF-8
- In DB2 9: database encoding had to be UTF-8
- In DB2 9.5: database can be any encoding
  
- However, UNICODE databases are the way of the future
  - ▶ for DB2 and its competitors
- In DB2 9.5: `CREATE DATABASE` defaults to UTF-8 encoding.





## XML Full Text Search

- DB2 Net Search Extender enhanced for XML
- XML-aware text index of full or partial documents

```
create index idx5 for text on customer(info) ;
```

- Usage:

```
select info from customer where  
contains(info, 'sections("/customerinfo/comment") "happy" ')= 1;
```

- Complex search criteria supported
- 37 languages, fuzzy search, stemming, etc.
- Queries can use a mix of full-text & regular XML indexes



## DB2 pureXML integrates well with other IBM offerings, including . . . .

- **DB2** Information Management Software
  - ▶ DB2 Developer Workbench
  - ▶ DB2 Control Center
  - ▶ DB2 Alphablox
- **Rational** software
  - ▶ Rational Data Architect
  - ▶ Rational Application Developer
- **WebSphere** software
  - ▶ WebSphere Enterprise Service Bus
  - ▶ WebSphere Process Server
  - ▶ WebSphere DataPower
  - ▶ WebSphere Federation Server
  - ▶ WebSphere Transformation Extender
- **Lotus** software
  - ▶ Workplace Forms



## DB2 pureXML Partners (partial list)



Database Auditing and Performance Solutions



## Further Reading: Administering XML in DB2 9

- "Native XML Support in DB2 Universal Database"  
<http://www.vldb2005.org/program/paper/thu/p1164-nicola.pdf>
- "Indexing XML Documents in DB2 9 pureXML™ "  
<http://www-03.ibm.com/developerworks/wikis/download/attachments/1824/indexingXMLdocuments.pdf>
- "XML Schema Registration and Validation"  
<http://www-03.ibm.com/developerworks/wikis/download/attachments/1824/XMLSchema+Registration+and+Validation.pdf>
- "EXPORT and IMPORT for XML Data Type Support"  
[http://www-03.ibm.com/developerworks/wikis/download/attachments/1824/tutor\\_expimp.pdf](http://www-03.ibm.com/developerworks/wikis/download/attachments/1824/tutor_expimp.pdf)
- "DB2 goes hybrid: Integrating XML and XQuery with relational data and SQL"  
<http://www.research.ibm.com/journal/sj/452/beyer.html>
- "DB2 9 XML Performance Characteristics"  
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0606schiefer>
- DB2 9 Documentation & Resources:  
<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp>  
<http://www.ibm.com/developerworks/wikis/display/db2xml/Technical+Papers+and+Articles>



## Further Reading: Querying & Updating XML

- "pureXML™ in DB2 9: Which way to query your XML Data?"  
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0606nicola/>
- "Query DB2 XML Data with SQL"  
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0603saracco2/>
- "Query DB2 XML Data with XQuery"  
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0604saracco/>
- "Query XML data that contains namespaces"  
<http://www-128.ibm.com/developerworks/db2/library/techarticle/dm-0611saracco/>
- "Integration of SQL and XQuery in IBM DB2"  
<http://www.research.ibm.com/journal/sj/452/ozcan.html>
- "XMLTABLE by Example", Part 1 & 2  
<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0708nicola/>  
<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0709nicola/>
- "Update XML in DB2 9.5":  
<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0710nicola/>



## Further Reading on DB2 XML Performance

### **15 best practices for pureXML performance in DB2 9**

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0610nicola/>

### **Performance of DB2 9 pureXML vs. CLOB and shredded XML storage**

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0612nicola/>

### **XML Database Benchmark: “Transaction Processing over XML (TPoX)”**

<http://tpox.sourceforge.net/> , [http://tpox.sourceforge.net/Sigmod2007\\_TPoX.pdf](http://tpox.sourceforge.net/Sigmod2007_TPoX.pdf)

### **DB2 9 XML performance characteristics**

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

### **DB2 9 pureXML Scalability on Intel® Xeon® MP Platforms Using IBM N Series Storage**

By Intel: <http://www.intel.com/cd/ids/developer/asmo-na/eng/dc/mobile/technologies/326328.htm>

### **Exploit XML indexes for XML query performance in DB2**

<http://www.ibm.com/developerworks/db2/library/techarticle/dm-0611nicola/>

### **On the Path to Efficient XML Queries**

<http://www.vldb.org/conf/2006/p1117-balmin.pdf>



# Questions?





# Thank You!

