Native XML Support in DB2 Universal Database

Matthias Nicola, Bert van der Linden
IBM Silicon Valley Lab
mnicola@us.ibm.com

Sept 1, 2005

Agenda

- Why “native XML” and what does it mean?
- Native XML in the forthcoming version of DB2
  - Native XML Storage
  - XML Schema Support
  - XML Indexes
  - XQuery, and the Integration with SQL
- Summary

XML Databases

- **XML-enabled Databases**
  - The core data model is not XML (but e.g. relational)
  - Mapping between XML data model and DB's data model is required, or XML is stored as text
  - E.g.: DB2 XML Extender (V7, V8)

- **Native XML Databases**
  - Use the hierarchical XML data model to store and process XML internally
  - No mapping, no storage as text
  - Storage format = processing format
  - E.g.: Forthcoming version of DB2

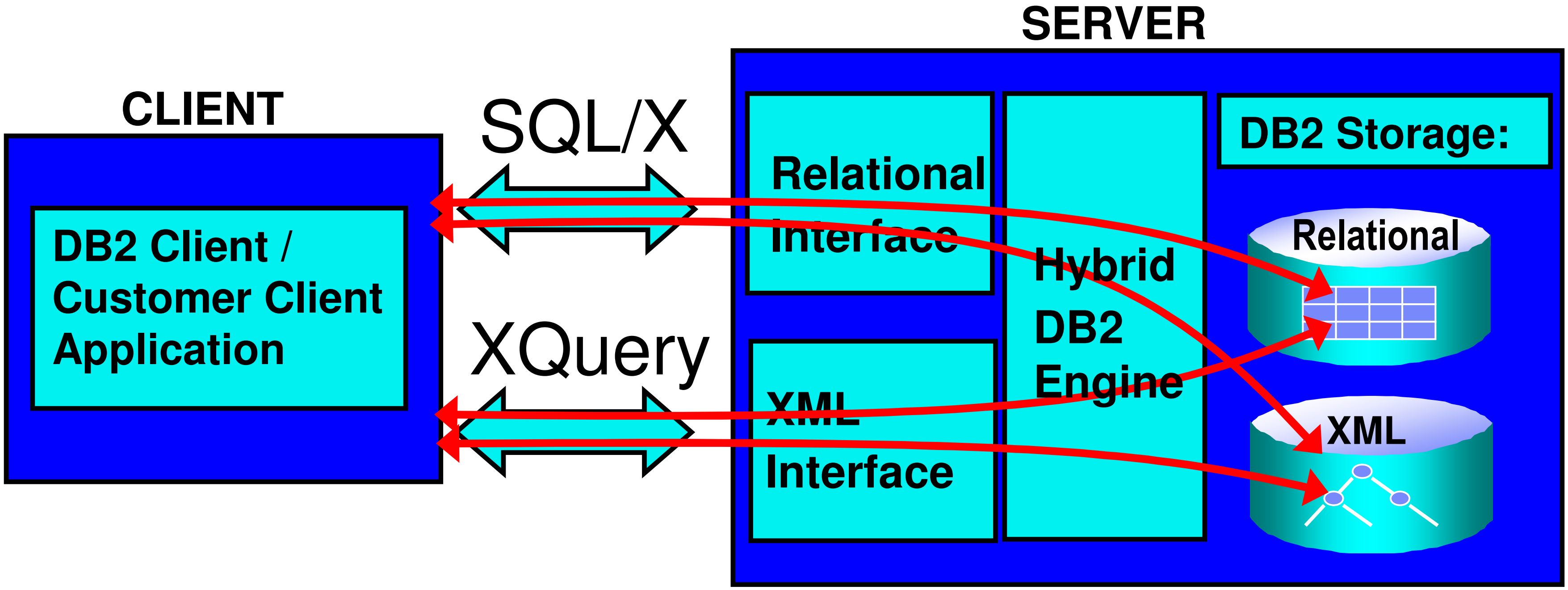
Problems of XML-enabled Databases

- CLOB storage:
  - Query evaluation & sub-document level access requires costly XML Parsing – too slow !

- Shredding:
  - Mapping from XML to relational often too complex
  - Often requires dozens or hundreds of tables
  - Complex multi-way joins to reconstruct documents
  - XML schema changes break the mapping
    - no schema flexibility !
    - For example: Change element from single- to multi-occurrence requires normalization of relational schema & data

Integration of XML & Relational Capabilities in DB2

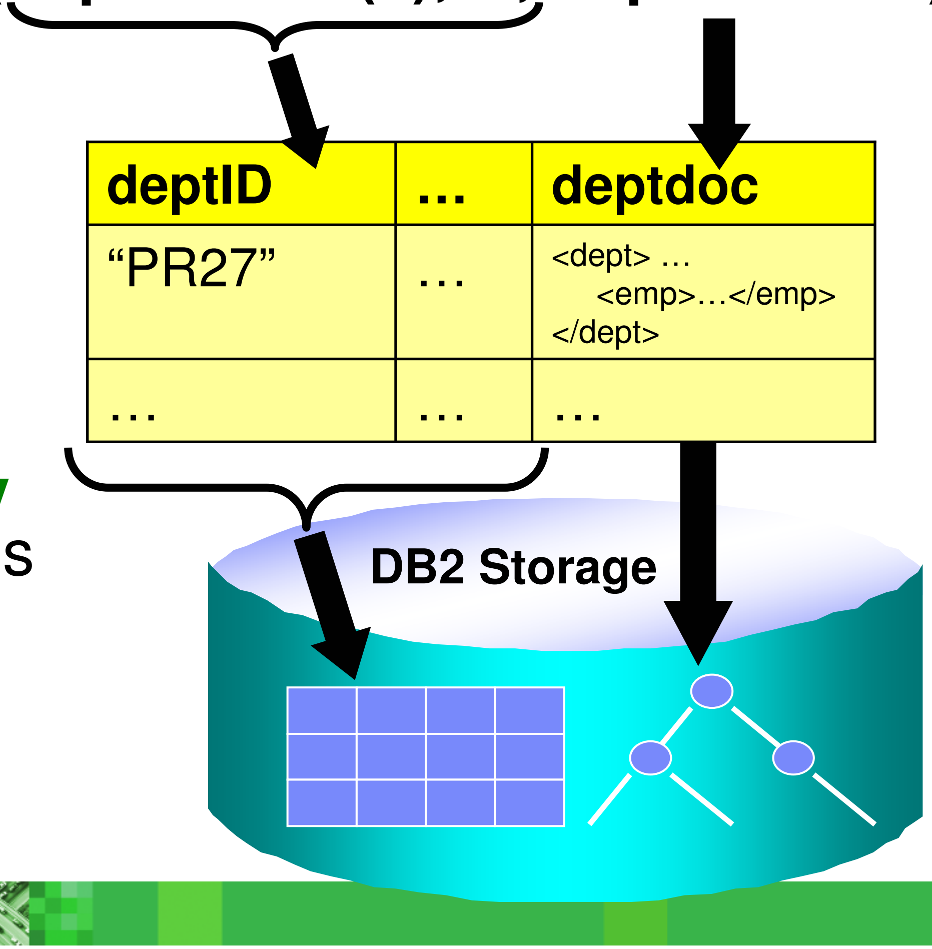
- **Native XML data type**
  - (not Varchar, not CLOB, not object-relational)
- XML Capabilities in all DB2 components
- Applications combine XML & relational data



Native XML Storage

- DB2 stores XML in **parsed hierarchical** format (similar to the DOM representation of the XML infoset)

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



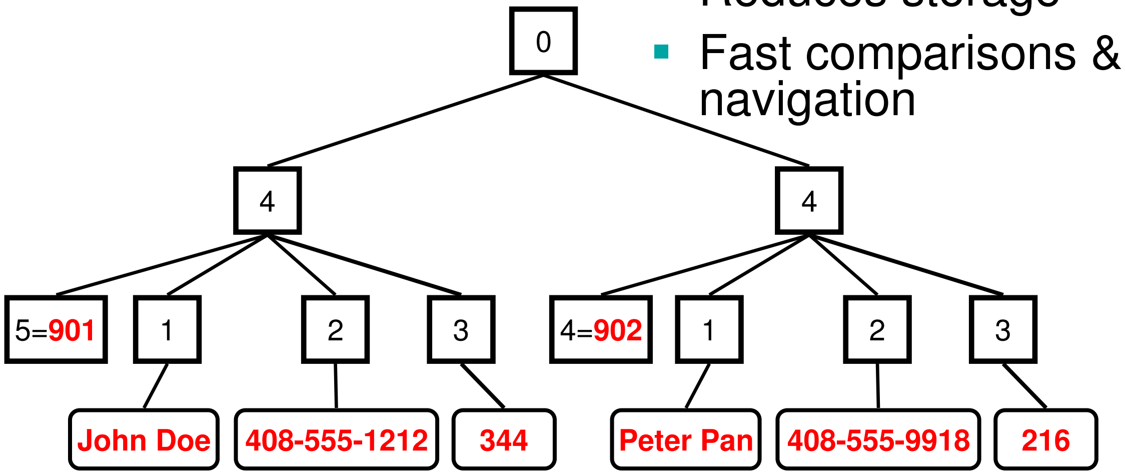
deptID	…	deptdoc
"PR27"	…	<dept> … <emp>…</emp> </dept>
…	…	…

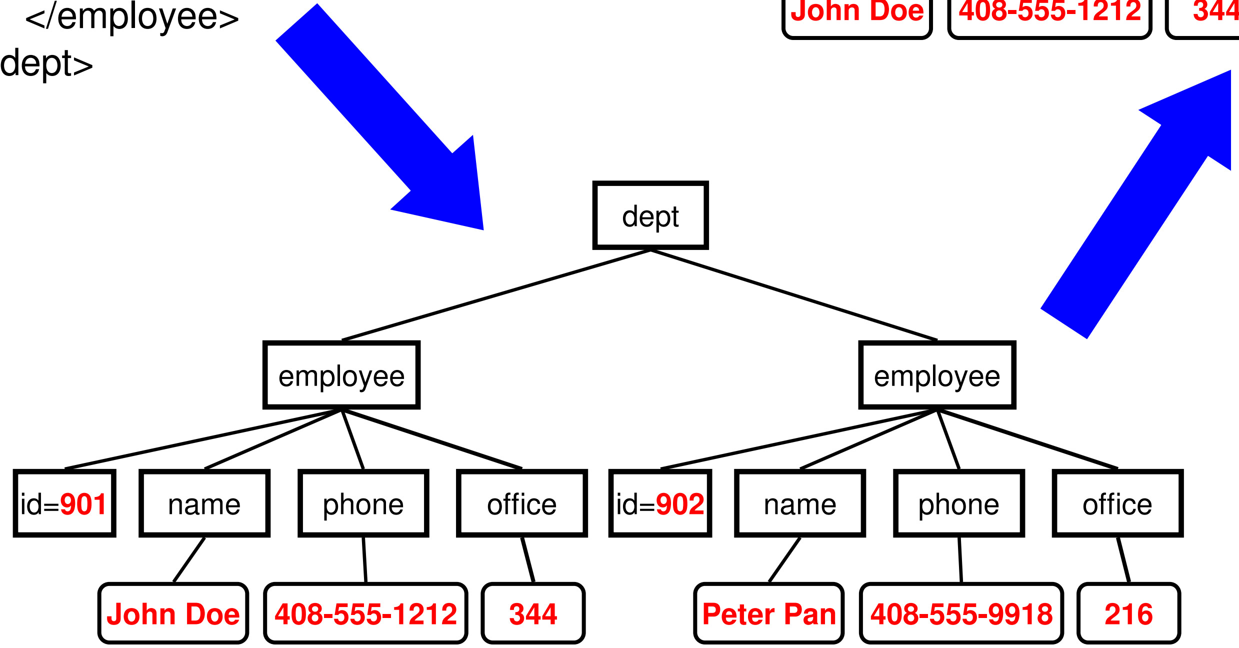
- Relational columns are stored in relational format (tables)
- XML is stored **natively** as type-annotated trees (representing the XQuery Data Model).

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





SYSIBM.SYSXMLSTRINGS

String table	
0	dept
4	employee
1	name
5	id
2	phone
3	office

- 1 String table per database
- Database wide dictionary for all tags in all XML columns

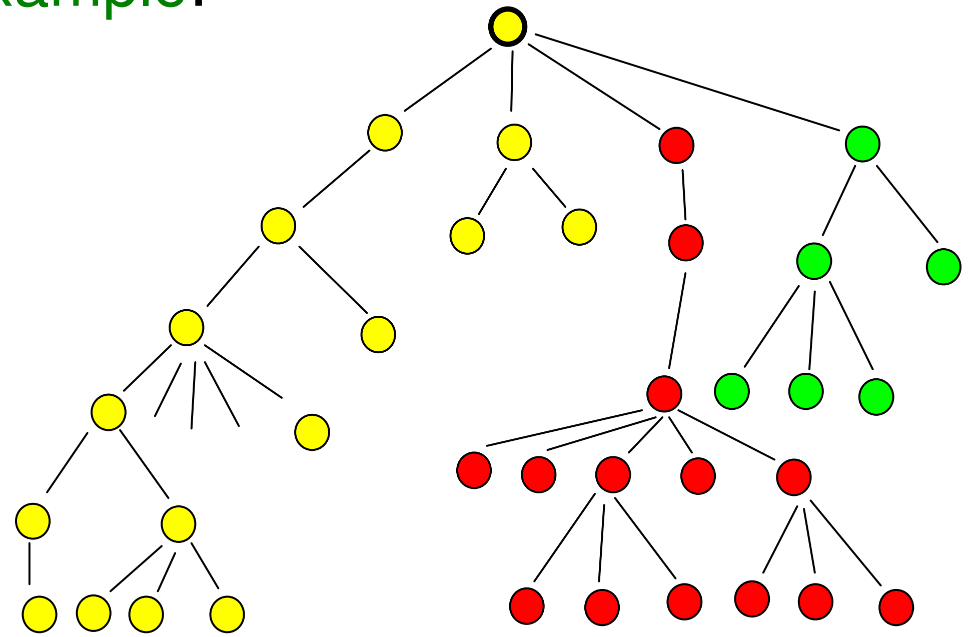
Information for Every Node

- Tag name, encoded as unique StringID
- A nodeID
- Node kind (e.g. element, attribute, etc.)
- Namespace / Namespace prefix
- Type annotation
- Pointer to parent
- Array of child pointers
- Hints to the kind & name of child nodes (for early-out navigation)
- For text/attribute nodes: the data itself

XML Node Storage Layout

- Node hierarchy of an XML document stored on DB2 pages
- Documents that don’t fit on 1 page: split into regions/pages
- Docs < 1 page: 1 region, multiple docs/regions per page

Example:

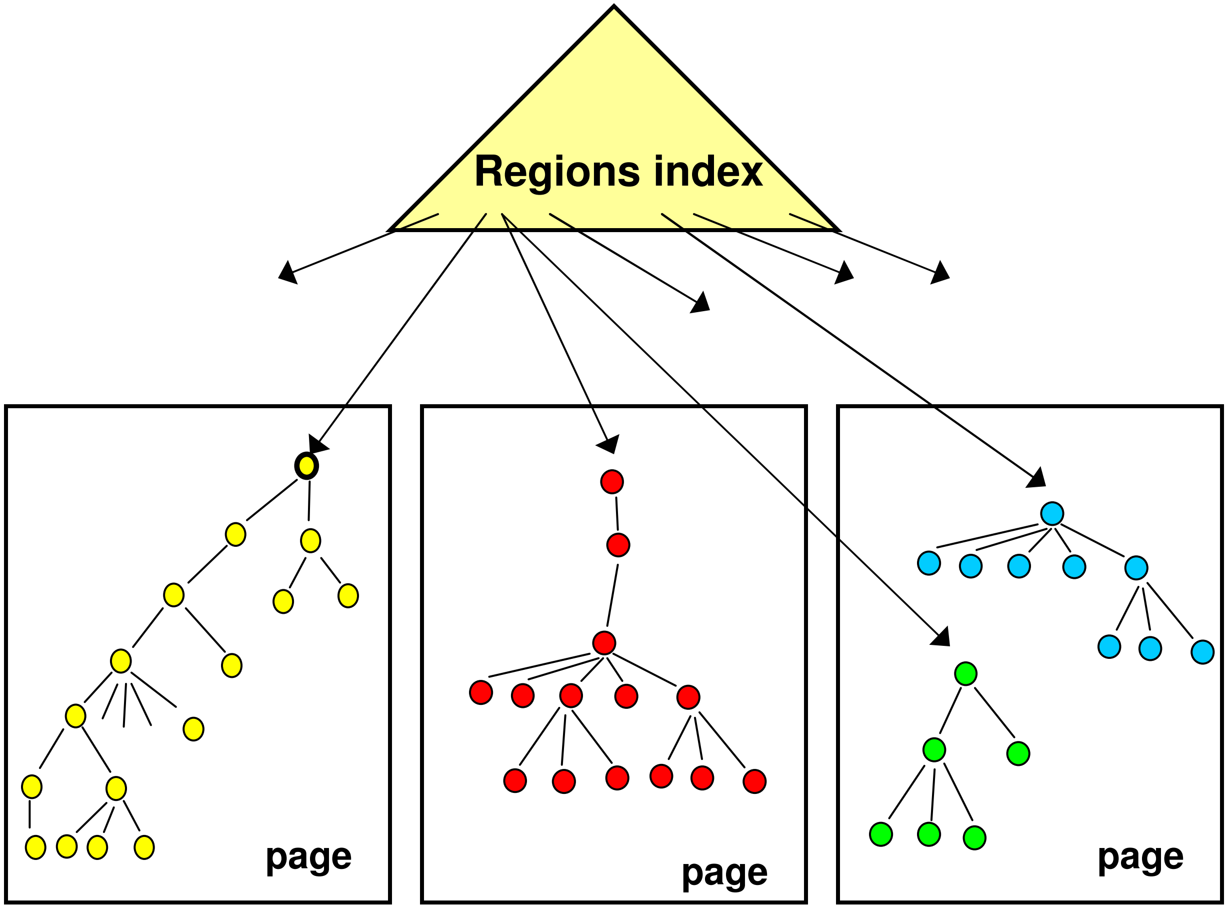


Document split into 3 regions, stored on 3 pages

Split can be at any level of the document

XML Storage: “Regions Index”

- not user defined, default component of XML storage layer



- maps nodeIDs to regions & pages
- allows to fetch required regions instead of full documents
- allows intelligent prefetching

XML Schema Support & Validation in DB2

- 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
- Schemas are registered & stored in the DB2 Schema Repository (XSR) for fast and stable access.

Validation using Schemas

Validate XML from a parameter marker using xsi:schemaLocation:

```
insert into dept(deptdoc) values (xmlvalidate(?))
```

Override schema location by referencing a schema ID or URI:

```
insert into dept(deptdoc) values (
        xmlvalidate(? according to xmlschema id departments.deptschema)

insert into dept(deptdoc) values (
        xmlvalidate(? according to xmlschema uri 'http://my.dept.com’)
```

Identify schema for a given document:

```
  select deptid, xmlxsrobjectid(deptdoc)
   from dept      where deptid = “PR27”
```

XML Indexes for High Query Performance

- Define 0, 1 or multiple XML Value Indexes per XML column
- XML index maps: (pathID, value) → (nodeID, rowID)
- Index any elements or attributes, incl. mixed content
- Index definition uses an XML pattern 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

xmlpattern = XPath without predicates, only child axis (/) and descendent-or-self axis (//)

XML Indexing: Examples

```
<dept bldg=101>
   <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>
```

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

```
create index idx1 on dept(deptdoc) generate key
using xmlpattern '/dept/@bldg' as sql double;

create unique index idx2 on dept(deptdoc) generate key
using xmlpattern '/dept/employee/@id' as sql double;

create index idx3 on dept(deptdoc) generate key
using xmlpattern '/dept/employee/name' as sql varchar(35);
```

```
…xmlpattern '//name' as sql varchar(35);      (Index on ALL “name” elements)
…xmlpattern '//@*' as sql double;              (Index on ALL numeric attributes)
…xmlpattern '//text()‘ as sql varchar(hashed);   (Index on ALL text nodes, hash code)
…xmlpattern ‘/dept//name' as sql varchar(35);

…xmlpattern '/dept/employee//text()' as sql varchar(128);  (All text nodes under employee)

…xmlpattern 'declare namespace m="http://www.myself.com/";  /m:dept/m:employee/m:name‘
                                                              as sql varchar(45);
```

Querying XML Data in DB2

The following options are supported:

- XQuery/XPath as a stand-alone language
- SQL embedded in XQuery
- XQuery/XPath embedded in SQL/XML
- Plain SQL for full-document retrieval

Compiler/Optimizer Details: *Beyer et al. "System RX", SIGMOD 2005* [2]

Example: XQuery as a stand-alone Language in DB2

```
create table dept(deptID char(8) primary key,  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>
```

db2-fn:xmlcolumn returns the sequence of all documents in the specified XML column

```
<dept bldg=101>
   <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>
```

Examples: SQL embedded in XQuery

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

- **Identify XML data by a SELECT statement**
- **Leverage predicates/indexes on relational columns**

```
for $d in db2-fn:sqlquery('select deptdoc from dept                (single document)
                                 where deptID = “PR27” ')…

for $d in db2-fn:sqlquery('select deptdoc from dept                (some documents)
                                 where deptID LIKE “PR%” ')…

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

  for $d in db2-fn:xmlcolumn(‘dept.deptdoc’)/dept ,
      $e in db2-fn:sqlquery('select  xmlforest(name, desc)     (constructed documents)
                             from unit u’)…                    (join & combine XML
                                                                and relational data)
```

Example: XQuery embedded in SQL/XML

SQL/XML Standard Functions: **xmlexists**, **xmlquery**, **xmltable**

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

select deptID,
        xmlquery(‘for $i in $d/dept
                  let $j := $i//name
                  return $j’   passing deptdoc as “d”)
from dept
where  deptID LIKE “PR%”
  and  xmlexists(‘$d/dept[@bdlg = 101]’ passing deptdoc as “d“)
```

Other Features in DB2 native XML

- XML Text Search Support
- XML Import/Export
- XML Type in Stored Procedures
- API Extensions (JDBC, CLI, .NET, etc.)
- XML Schema Repository
- Full SQL/XML support
- Visual XQuery Builder
- Annotated schema shredding
- …and more

Summary

CLOB and shredded XML storage restrict performance and flexibility

New **native** XML support in DB2:

- Better Performance through
  - Hierarchical & parsed XML representation at all layers
  - Path-specific XML Indexing
  - New XML join and query methods
- Higher Flexibility through:
  - Integration of SQL and XQuery
  - Schemas are optional, per document, not per column
  - Zero, one, or many XML schemas per XML column

Questions?

mnicola@us.ibm.com