SWG BetaWorks

DB2 9

Technical Education Series

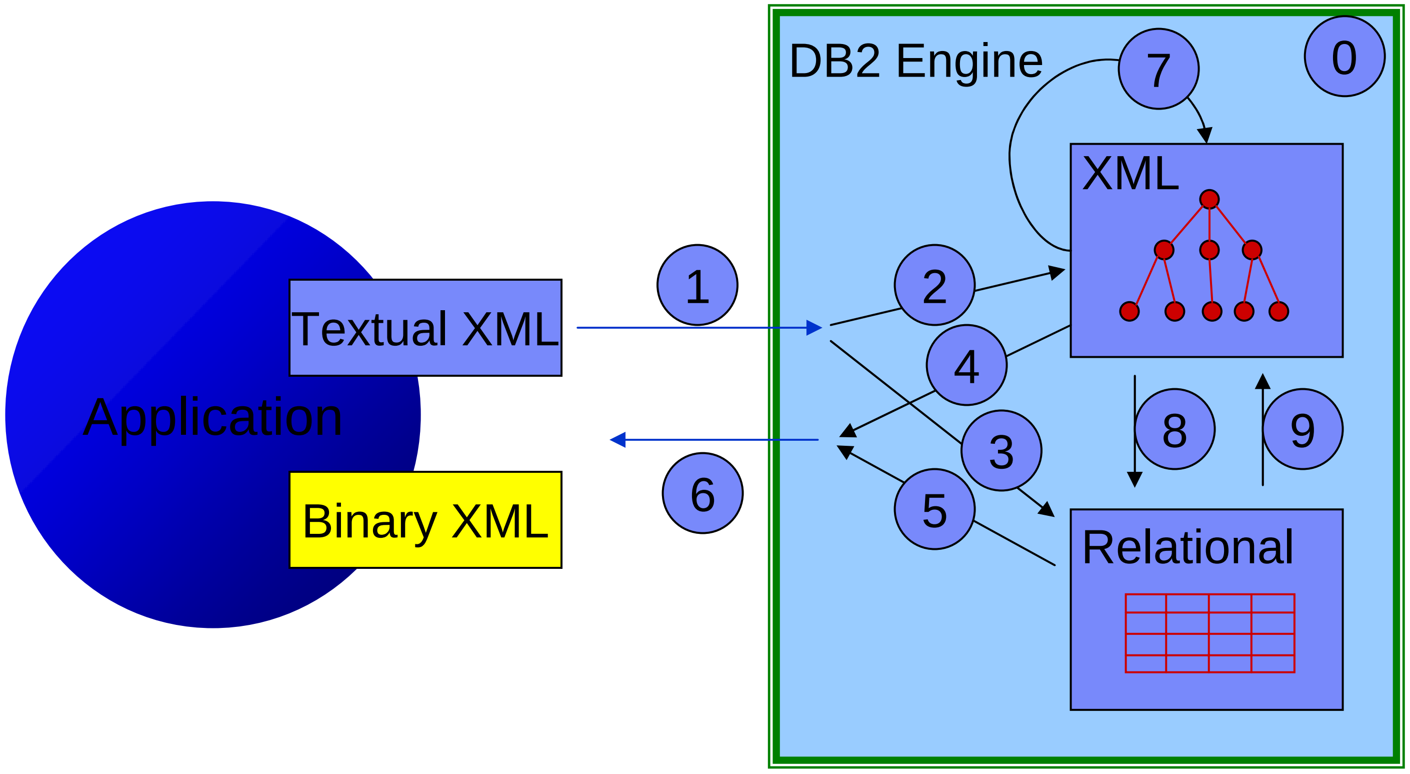
“XML Part 2 (Application Development)”

DB2 9 for z/OS Technical Education Series

XML Support in DB2 9 for z/OS

- Part 1 (Foundation and DBA)
  - What is XML?
  - XML support in DB2 9
  - Storage Infrastructure
  - XML Schema Support
  - Utilities
- **Part 2 (Application Development)**
  - **XML in DB2 Application**
  - **Inserting, Updating, Deleting XML documents**
  - **Querying XML documents**
    - **Using the SQL/XML interface and XPath**
  - **Publishing XML documents**
  - **Programming Access**

Overview of XML Possibilities in DB2 9



0	XQuery Data model
1	Bind in XML
2	Store as XML
3	Shred into SQL
4	Retrieve XML
5	Publish XML
6	Bind out XML
7	XML to XML
8	XML to SQL
9	SQL to XML

INSERT/UPDATE/DELETE XML Data

- All data inserted into an XML column has to be a well-formed XML document
- Document is parsed and stored in an internal DB2 XML format
- Document format may be changed during parsing due to white space stripping and default values filling in
  - You can choose to preserve or strip whitespace (default is to strip) with the STRIP WHITESPACE or PRESERVE WHITESPACE option in XMLPARSE
- You can also use the XMLVALIDATE function to validate documents as they are inserted
- Partial Update is not supported in current release

INSERT/UPDATE/DELETE XML Data (cont.)

INSERT: Validation is optional and can be at per document (per row) level;

```
INSERT into emp (id, info) values(1001,XMLPARSE(DOCUMENT :hv PRESERVE
WHITESPACE))
INSERT into emp (id, info)  values (1001, XMLVALIDATE(‘SYSXSR’,’
MYXMLSCHEMA’, :hv))
```

UPDATE: Language only support full document replace

```
UPDATE emp
      SET info = XMLPARSE (DOCUMENT ’<doc>...</doc>’ PRESERVE
WHITESPACE)
```

DELETE: Delete entire object

```
DELETE FROM emp WHERE …;
```

DB2 9 SQL/XML functions

- Based on SQL/XML 2006
- XMLPARSE and XMLSERIALIZE
- Other SQL/XML functions with XPath :
  - XMLQUERY
  - XMLEXISTS
  - XMLPATTERN
- New SQL/XML constructors
  - New options and binary type support for XMLELEMENT, XMLFOREST
  - XMLPI, XMLCOMMENT, XMLTEXT, XMLDOCUMENT
  - Transient data type becomes real XML type
- Enhanced SQL/XML Publishing Functions

XMLPARSE

- Parse a string expression and return an XML value

```
INSERT INTO EMP(id, xvalue)
  VALUES(1001,XMLPARSE(DOCUMENT :hv PRESERVE
  WHITESPACE))
```

  – Options to 'STRIP WHITESPACE | PRESERVE WHITESPACE'

```
INSERT INTO EMP(id, xvalue) VALUES(1002, :hv);
```

- Implicitly invoked by DB2 if the source data to be inserted is not an XML data type
- Can be invoked explicitly to override the default parsing options used by DB2 (e.g. strip/preserve whitespace)

XMLSERIALIZE

- Convert an XML value from its internal tree format into the corresponding textual XML
- Inverse operation of XMLPARSE
- With or without XML declaration

Serialize XML into a string of CLOB type

```
SELECT e.id, XMLSERIALIZE(XMLELEMENT ( NAME "Emp", e.fname || ' ' || e.lname)
                    AS CLOB(100) EXCLUDING XMLDECLARATION) AS "result"
          FROM employees e;

ID        result
--------  ----------------------------------
1001      <Emp>John Smith</Emp>
1206      <Emp>Mary Martin</Emp>
```

Querying XML Documents with XPath

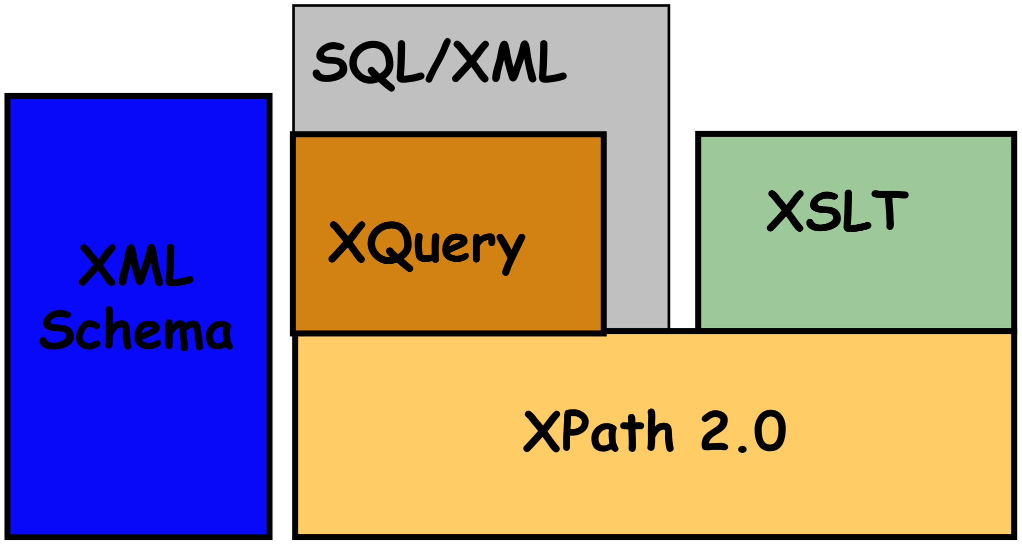
- Why a new query language?
  - XML data is sufficiently different than relational data
  - Hierarchical nature of XML data requires a navigation language
  - SQL cannot handle the heterogeneous nature of XML data
  - XPath is used to navigate through the tree structure format of an XML documents and address nodes in a tree

What is XPath?

- The primary purpose of XPath is to navigate through the tree structure of XML documents and address the nodes in the trees.
- XPath 2.0 is based on XQuery Data Model.
- The basic building block of XPath is the expression.
  - Path expressions, Arithmetic Expressions, Comparisons, Function calls, Atomic type constructor, etc.
- XPath 2.0 is a functional language
- XPath 2.0 is a strongly-typed language

XPath 2.0

- XPath 2.0 is designed to be embedded in a host language. e.g XQuery, XSLT or SQL/XML

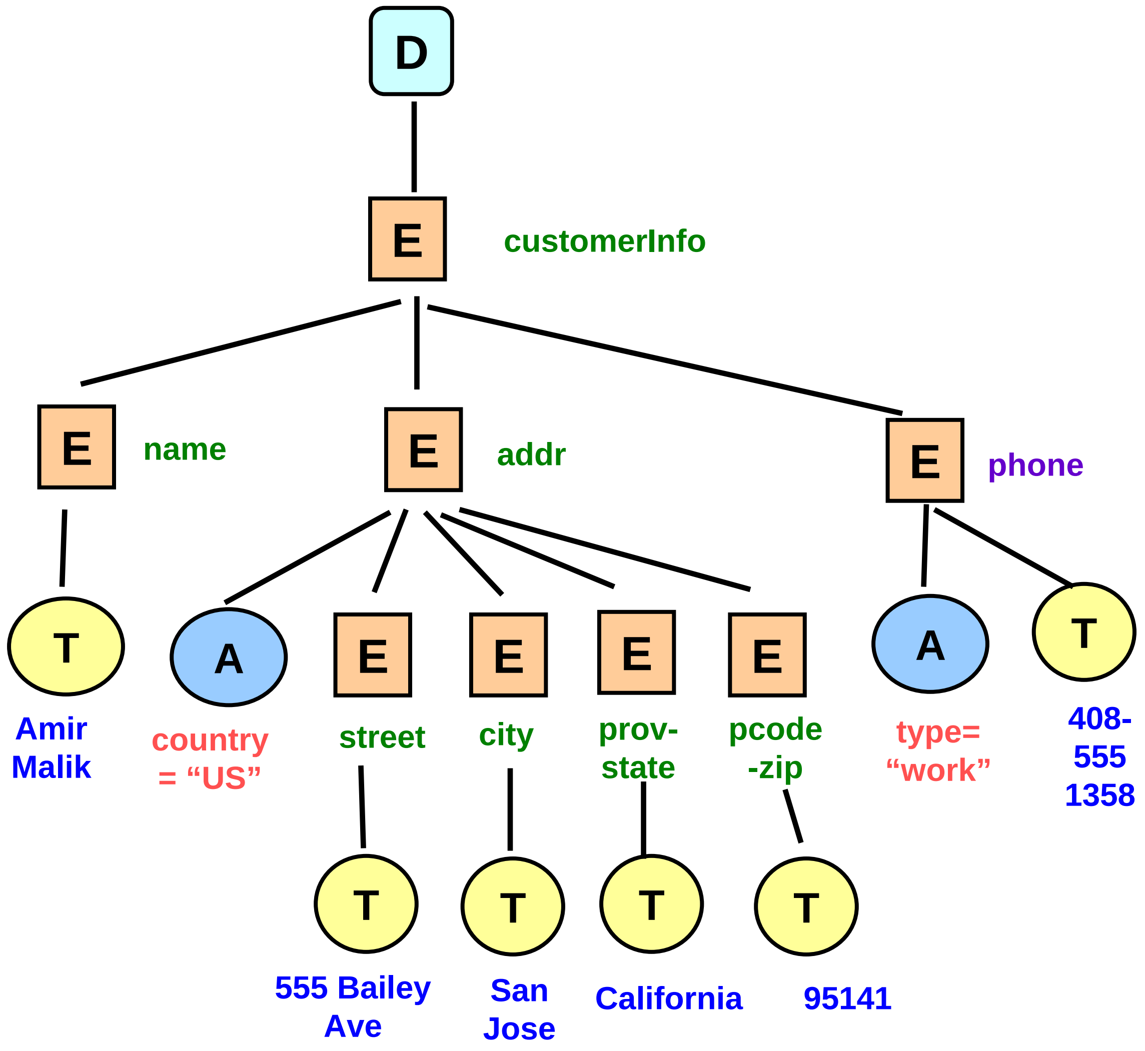


XML Data Model (XDM) Terminology

- XML data in DB2 XML column is represented as the pureXML data model (follows XQuery 1.0 and XPath 2.0 Data Model)
  - Abstract representation of XML documents or fragments
  - Described in terms of sequences and items, atomic values and nodes
- **XPath Language** is used to search and manipulate the XML data model
- **Sequence** is an instance of XML data model, can have zero of more items in it, e.g. ("Nathan", 1.32e0, true())
- **Item** is either an atomic value or a node
- **Node** is an instance of one of the six kinds of nodes defined in the XQuery/XPath Data Model (XDM): Document, Element, Attribute, Text, Comment, Processing Instruction
- **Atomic Value** is a value in the value space of an Atomic Type
- **Atomic Type** is defined in the XML Schema data types, e.g. xs:date, xs:time, xs:decimal

Navigate XML Documents using XPath

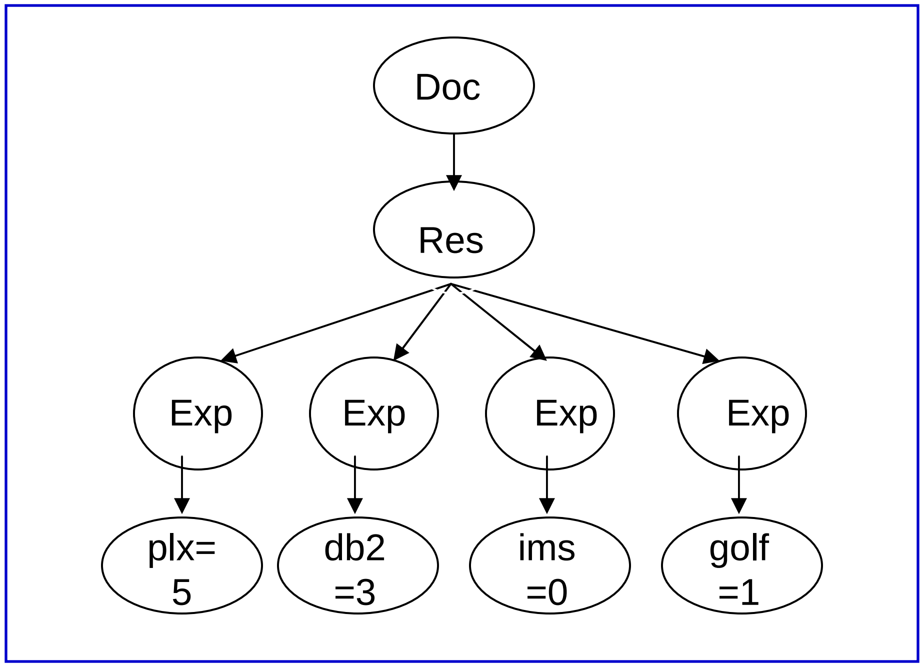
- **Use XPath to navigate nodes of the XML tree [XDM]**
- **6 kinds of nodes in an XML document**
  - Document Node
  - Element Node
  - Attribute Node
  - Text Node
  - Comment Node
  - Processing Instruction Node
- **Each node has:**
  - Node identity
  - A type (e.g. “xs:decimal")
  - A string value (e.g. "47")
  - A typed value (e.g. 47)
  - …..



Xpath is a full expression language

- XPaths works with the XDM model to find parts of an XML structure

```
/resume[xs:decimal(experience/@plx) >
      (fn:sum(experience/@*) div fn:count(experience/@*))]
```



Node-tree representation

find a resume whose plx experience is better than the average of all of its experiences

XPath Support in DB2 9

- **DB2 9 provides subset of XPath 2.0 support**
  - e.g. - Axes: only 5 forward axes (child, attribute, descendant, self, self-or-descendant), & parent axis are supported.
- **XPath is used in the following SQL Context in DB2 9**
  - XMLQUERY
    - Scalar function that executes an XPath query against an XML value
  - XMLEXISTS
    - Test if the XPath expression has non-empty sequence return
  - XMLPATTERN (to create XML user index)
    - Specify which nodes in an XML document are present in the index

XMLQUERY (Querying and Extracting)

- Query and extract parts of document from XML column
- First parameter has to be an XPath expression constant
  - parameters follow the passing keyword is used to pass values to the XPath expression
- Construct new documents from existing documents

```
SELECT XMLQUERY('//item[productName = $n]'
        PASSING PO.POrder, P.name AS "n")
    FROM PurchaseOrders PO, Product P;
```

XMLEXISTS –New Predicate

- Use XMLEXISTS with XPATH to find documents
- Checks for existence of a node that matches certain criteria
  - Return true if result not empty
  - Return false if empty sequence
  - Return error if XPath expression is incorrect
- Support subset of XPath 2.0 => same as XMLQUERY

```
SELECT S.prodno, count(*) as result
   FROM PurchaseOrders PO, Products S
WHERE XMLEXISTS ('//item[@partNum = $n]'  PASSING PO.POrder, S.prodno AS "n")
   AND  S.prod_name = 'Baby Monitor';

Prodno                 result
------------------------------------------------------------
926-AA                 1
```

XML (Value) Index

- User defined index for XML document – improve query performance
- Indexing values are from element/attribute/text nodes inside an XML document that match the specific XPath
- Composite indexes not supported
- Multiple indexes allowed per XML column
- CREATE INDEX statement with
  - XMLPATTERN keyword
  - Data Type (mismatch)
- Index key
  - Concatenate the values extracted from the node with the document id and node id
  - DocID => identify the XML document
  - NodeID => identify the node position
  - Value extracted from the node

Use of XMLPATTERN in creating XML User Index

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

Index values are stored as varchar(20)

Document stored in the XML column DEPTDOCS

```
<department name="Department1" id="OF2">
         <emp id="12345" gender="Female">
            <name>
                  <first>Kathy</first>
                  <last>Chen</last>
            </name>
            <DOB>1972-12-31</DOB>
         </emp>
         <emp id="67890" gender="Male">
            <name>
                  <first>John</first>
                  <last>Joe</last>
            </name>
           <DOB>1975-01-01</DOB>
         </emp>
</department>
```

Then queries with predicates of the form '/department/emp/name[last]="**Joe**" ' could utilize the XML values index. => search by last name.

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.

XML Index Usage

- Criteria:
  - Index pattern is equal to or less restrictive than the query predicate:

    index: //product/regpricev.s.

    query:/catalog//product[regprice> 10]
  - Data types have to match.
- Use internal “between” for better performance.
  - //item[@size> 5 and @size < 10]
  - //product[wt> 10 and wt < 20] =>

    /product[wt[. > 10 and . <20]]

Publishing - Producing XML Data from Relational Data

- Publishing functions in V8 have been enhanced to construct nodes in XQuery Data Model
- The SQL/XML publishing functions essentially take relational data and create XML Node types and possibly an entire XML document
  - XMLELEMENT()

    - creates an XML element
  - XMLATTRIBUTES()

    - used within XMLELEMENT to create attributes

Example:

```
SELECT
      XMLELEMENT(NAME “empname", e.firstnme)
      AS "Result"
FROM employee e
```

Result
<empname>John Smith</empname>
<empname>Sue Ellen</empname>
<empname>Joe Blow</empname>
<empname>Abe Lincoln</empname>
<empname>Matt Foreman</empname>

SQL/XML Publishing Functions in Version 8 - Revisit

- Based on SQL/XML 2003
- Scalar functions
  - XMLELEMENT - generates an XML element
  - XMLATTRIBUTES - used within XMLELEMENT to specify attributes for the XML element
  - XMLFOREST - produces a forest of XML elements from SQL values
  - XMLCONCAT - concatenates a variable number of XML values
  - XMLNAMESPACES – produces a namespace declaration
  - XMLAGG – aggregate function to group or aggregate XML data
- XML Serialization function
  - XML2CLOB - converts the transient XML value into a string value (should use XMLSERIALIZE in DB2 9)
- Based on Transient XML data type, not the native XML data type

SQL/XML Publishing Functions in Version 9

- Based on SQL/XML 2006
- Publishing functions in V8 have been enhanced to construct nodes in XQuery Data Model (XDM)
  - XMLPI
  - XMLCOMMENT
  - XMLDOCUMENT
  - XMLTEXT
- New options (Null Handling) and binary type support (HEX or BASE64) for XMLELEMENT, XMLFOREST

Construct Invoice from Purchase Order

```
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>
```

XMLVALIDATE() Function

- User defined function – SYSFUN.DSN_XMLVALIDATE()
  - test XML values for validity against XML schema
  - obtain default values from XML schema

```
INSERT INTO T1(C1) VALUES
          (XMLPARSE (DOCUMENT    SYSFUN.DSN_XMLVALIDATE(:host_var,
                                 'SYSXSR.ORDERSCHEMA')));
```

Select XML Data

- Simple select:

```
SELECT XMLPO INTO :xmlPo
        FROM PurchaseOrders
WHERE ponumber = ‘200300001’;
```

- Select with condition:

```
SELECT XMLPO INTO :xmlPo
        FROM PurchaseOrders
WHERE XMLEXISTS(‘//items/item[desc = “Shoe”]’   PASSING XMLpo);
```

- Extract from a document:

```
SELECT XMLQUERY(‘//items/item/quantity’   PASSING XMLpo)
        FROM PurchaseOrders WHERE …;
```

Application Access via SQL Statement

- **XML data input to and receive from SQL statements via**
  - XML Host Variables Types, or
    - XML AS BLOB(n), XML AS CLOB(n), XML as DBCLOB(n)
  - non-XML Host Variable Types
    - Character types and binary string types, e.g. CLOB, DBCLOB, BLOB, BINARY, CHAR, GRAPHIC. etc.
- **XMLPARSE and XMLSERIALIZE APPLY (implicitly or explicitly)**
  - Implicitly by DB2 on behalf of application - (XML and non-XML host variables)
    - Similar to XMLPARSE/XMLSERIALIZE with their default options
  - Explicitly by application via XMLPARSE/XMLSERIALIZE - (if for non-XML host variables)
    - CLOB, DBCLOB, GRAPHIC or character types => external-encoded
    - BLOB, binary, or for bit data => internal-encoded

Application Access via SQL Statement

- **XML type is supported in**
  - Java (JDBC, SQLJ), ODBC,
  - C/C++, COBOL, PL/I, Fortran, Assembly
  - .NET
- **Application access via**
  - XML Host Variables Types, or
    - XML AS BLOB(n), XML AS CLOB(n), XML as DBCLOB(n)
  - non-XML Host Variable Types
    - CLOB, DBCLOB, BLOB, BINARY, CHAR, GRAPHIC. etc.
- **XMLPARSE and XMLSERIALIZE APPLY (implicitly or explicitly)**
  - Implicitly by DB2 on behalf of application
  - Explicitly by application via XMLPARSE/XMLSERIALIZE

COBOL Application with Embedded SQL Example

```
EXEC SQL BEGIN DECLARE SECTION END-EXEC.
    01 xmlBuff USAGE IS SQL TYPE IS XML as CLOB(5K).
EXEC SQL END DECLARE SECTION END-EXEC.
```

```
EXEC SQL SELECT xmlCol INTO :xmlBuf from myTable where id = '001'.
```

After Translation

```
01 xmlBuff.
    49 xmlBuff-LENGTH              PIC 9(9) COMP.
    49 xmlBuff-DATA                PIC X(length).
```

If the length of the host variables is greater than 32K, translation becomes

```
01 xmlBuff.
    02 xmlBuff-LENGTH              PIC 9(9) COMP.
    02 xmlBuff-DATA.
        49 filler                  PIC X(32767).
        49 filler                  PIC X(32767).
      :
        49 filler                  PIC X(length - n * 32767).
```

Java JDBC Example

```
PreparedStatement pstmt= connection.prepareStatement("INSERT INTO
    PurchaseOdersVALUES(?, ?)");        // second column: XML type
…
BufferedReader br= new BufferedReader( new InputStreamReader( fin ) );
pstmt.setCharacterStream( 2, br, fileLen);
pstmt.execute();

Statement s = connection.createStatement();
ResultSet rs= s.executeQuery("select ponumber, xmlpo from purchaseOrders");
while (rs.next()) {
    intpo_no= rs.getInt("ponumber");
    com.ibm.db2.jcc.DB2Xml xml = (com.ibm.db2.jcc.DB2Xml) rs.getObject("xmlpo");
    System.out.println(xml.getDB2String()); //  or
    System.out.println(xml.getDB2XmlString());
}
```

More XML support in DB2 9 in APARs
XMLTABLE: Return XML in tabular format

```
SELECT X.* FROM dept,
    XMLTABLE ('$d/dept/employee' passing deptdoc as "d")
      COLUMNS
      "empID"          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>
```

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

When to use XML

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

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

"Thank You for listening"

If you have any questions on this DB2 9 for z/OS session, then please send them to the BetaWorks team at:

Ian_Cook@uk.ibm.com
FLETCHPL@uk.ibm.com

ON DEMAND BUSINESS™