



XML in DB2 9 for z/OS Overview

Hunter Cobb

**Friday, August 28, 2009
Session # 6152**

**The Trainer's Friend, Inc
6790 East Cedar Avenue, Suite 201
Denver, Colorado 80224 USA
303.393.8716**

www.trainersfriend.com

Presentation Agenda



- XML overview
- XML and DB2
 - DB2 objects that support XML data
- XML and SQL
 - Version 8 XML functions
 - Version 9 XML functions
- XML in application programs
- XML indexes
- XML schema support
- XML catalog tables
- XML impact on utilities
- Recapitulation

XML Overview



- XML – eXtensible Markup Language
- Technique for storing information with embedded “markup” – self describing data
- Intended for communication among different architectures
- By convention, we refer to XML “documents”
 - A document corresponds to what we would loosely call a logical record, or row in a table
 - Although you could create a (much larger) document that corresponds to all the records in a file (all the rows in a table)
- Historically, we've separated the description of the data from the data
 - Economy of storage
 - Data independence
 - Structure embedded in applications
 - Structure embedded in table definitions
- XML documents include the names of data fields, as well as the contents

XML Example



```
<?xml version="1.0" encoding="ibm-037" ?>  
<!-- example employee info -->  
<emprec>  
  <empno emptype="exempt">000500</empno>  
  <name>Phineas T. Albatross</name>  
  <dept>C01</dept><phoneno>5538</phoneno>  
  <salary mode="biweekly">31415.92</salary>  
</emprec>
```




XML Structure, 2

- XML is case-sensitive, white space is significant
- Five characters have a special representation called “Named character entities”

<	<
>	>
&	&
'	'
"	"



XML Structure, 3

- An XML document is well-formed if:
 - It has exactly one root element
 - Each opening tag is matched with a closing tag (matching case)
 - Elements are properly nested
 - Attribute values are quoted, and uniquely named within elements
 - All < > and & characters are present only via their named character entities
 - All " characters are present only via its named character entity inside attribute values
- An XML document is valid if:
 - It is well-formed
 - It complies with a document type definition or XML schema (specifications for element and attribute names, either external or contained in the prolog)

XML and z/OS High Level Languages



- The two basic operations:
 - Parsing – extracting the desired elements and attributes (and structure, if necessary) from an XML document
 - Generation – creation of an XML document from component elements
- COBOL
 - The XML PARSE statement for parsing
 - The XML GENERATE statement for creating an XML document from the contents of a structure
- PL/I
 - PLIXSAXA subroutine for parsing an XML document in a variable
 - PLIXSAXB subroutine for parsing a file that contains an XML document
 - XMLCHAR built-in function for creating an XML document from the contents of a structure



XML and DB2

- **In Version 8 and earlier, DB2 supported XML via the XML extender**
 - **A collection of user-defined data types and user-defined functions**
 - **Two choices for storing XML data**
 - **Intact placement in a CLOB or VARCHAR column**
 - **Deconstruction of elements and attributes into column values in one or more tables (“shredding“)**
- **In Version 9, DB2 supports XML as a native data type**
- **An XML column is like a LOB column**
 - **Stored in a separate table space – an XML table space**
 - **Processed in the DBM1 address space**
 - **Functions that were previously UDFs are now built-in**
 - **Several new functions for increased, well, functionality**



Defining an XML column

- You define an XML column in a table with the data type XML

```
CREATE TABLE ALBUM_XML  
  (ORDER_NO          CHAR(12) NOT NULL,  
   CONSTRAINT ALBPRI PRIMARY KEY(ORDER_NO),  
   SONGS_XML         XML          NOT NULL)  
IN TRAINDBA.TRAINTSA;
```

- You can add an XML column to a table with ALTER TABLE

```
ALTER TABLE ALBUM_XML  
  ADD COLUMN DANCES_WITH_DATA XML;
```

Notes:

- 1) There is no length specified for an XML column
- 2) There is no architectural limit on the size of an XML value in the table, but there is an effective limit of 2G (sounds like a distinction without a difference)
- 3) XML column data is stored in Unicode UTF-8 format



Behind the Scenes with XML columns

- **With an XML column, several supporting objects are defined / created for you automatically**
 - **A hidden BIGINT column in the base table named DB2_GENERATED_DOCID_FOR_XML**
 - **An unique index on DB2_GENERATED_DOCID_FOR_XML**
 - **Named I_DOCIDxxx, where xxx is the table name**
 - **An XML table space – partitioned, if the base table is partitioned; universal if the base table is universal**
 - **Named Xyyyynnnn, where yyy are the first three characters of the base table space name, and nnnn are a sequence number: 0000 for the first XML column, 0001 for the second XML column, etc.**

Behind the Scenes with XML columns, 2



- **With an XML column, several supporting objects are defined / created for you automatically**
 - **An XML table in the XML table space**
 - **Three columns: DOCID, MIN_NODEID, and XMLDATA**
 - **Table name is base table, prefixed by an X, and possibly suffixed with a number; no suffix for the first XML column, 000 for the second XML column, etc.**
 - **An index on the XML table**
 - **Index key: DOCID, MIN_NODEID**
 - **Index name begins with I_NODEIDXTxxx, where xxx is the XML table name, possibly suffixed by a three digit number**

XML and SQL in Version 8



- **DB2 Version 8 had several UDFs for working with XML values (actually, XML UDTs)**
 - These still exist in Version 9 – converted to be built-in functions, working with the native XML data type
 - **XMLELEMENT** – constructs an XML element
 - **XMLATTRIBUTES** – constructs an XML attribute
 - **XML2CLOB** – returns a CLOB representation of an XML value (superseded by XMLSERIALIZE)
 - **XMLCONCAT** – concatenates two or more XML elements
 - **XMLFOREST** – constructs a series of XML elements
 - **XMLNAMESPACES** – declares one or more XML namespaces
 - **XMLAGG (column function)** – returns a concatenation of XML elements from several table rows

XML and SQL in Version 9



- **DB2 Version 9 adds several addition XML-related functions (built-in, of course)**
 - **XMLSERIALIZE** – returns a CLOB/BLOB/DBCLOB version of an XML value
 - **XMLCOMMENT** – generates a comment
 - **XMLDOCUMENT** – generates a complete document
 - **XMLPI** – generates a processing instruction
 - **XMLTEXT** – generates a text node (content, with named character entities, if necessary)
 - **XMLPARSE** – parses an argument as an XML document
 - **XMLQUERY** – applies an XPATH expression to an XML value
 - **XMLEXISTS** – tests whether an XPATH expression returns a sequence of one or more items (used in a WHERE clause)
 - **DSN_XMLVALIDATE** – validates an XML document against an XML schema
 - **XMLTABLE** – returns a DB2 result set with rows derived from one or more XML documents, based on an XPATH expression
 - There is also a new variation of the CAST operator – **XMLCAST** – that converts to/from an XML expression

XMLELEMENT – Build an XML element



XMLELEMENT by Itself

```
SELECT XMLELEMENT(NAME "Dptname", DEPTNAME)  
FROM DEPT WHERE DEPTNO = 'C01';
```

```
<?xml version="1.0" encoding="IBM037"?><Dptname>INFORMATION  
CENTER</Dptname>
```

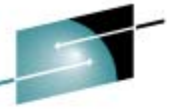
XMLELEMENT with XML2CLOB / XMLSERIALIZE

```
SELECT XML2CLOB (XMLELEMENT(NAME "Dptname", DEPTNAME) )  
FROM DEPT WHERE DEPTNO = 'C01';
```

```
SELECT XMLSERIALIZE(XMLELEMENT(NAME "Dptname", DEPTNAME)  
AS CLOB(1K) )  
FROM DEPT WHERE DEPTNO = 'C01';
```

```
<Dptname>INFORMATION CENTER</Dptname>
```

XMLATTRIBUTES – Construct an XML attribute



S H A R E
Technology • Connections • Results

```
SELECT XMLSERIALIZE(XMLELEMENT(NAME "DPT",  
    XMLATTRIBUTES(DEPTNO AS "DeptID" ,ADMRDEPT AS "Owner" ),  
    DEPTNAME ) AS CLOB(1K) )  
FROM DEPT WHERE DEPTNO = 'C01';
```

```
<DPT DeptID="C01" Owner="A00">INFORMATION CENTER</DPT>
```

```
SELECT XMLSERIALIZE(XMLELEMENT(NAME "DPT",  
    XMLATTRIBUTES(DEPTNO AS "DeptID" ) ) AS CLOB(1K) )  
FROM DEPT WHERE DEPTNO = 'C01';
```

```
<DPT DeptID="C01"/> or <DPT DeptID="C01"></DPT>
```

Note: Only appropriate context is as an argument to XMLELEMENT

XMLSERIALIZE / XMLCLOB – Build a CLOB / BLOB / DBCLOB version of an XML value



```
SELECT XML2CLOB(SONGS_XML) FROM ALBUM
WHERE ORDER_NO = 'Ar-1059';
```

<or>

```
SELECT XMLSERIALIZE(SONGS_XML AS CLOB(2K)) FROM ALBUM
WHERE ORDER_NO = 'Ar-1059';
```

```
<SONGS><SONGCT>11</SONGCT><SONG SEQ="01"> <TITLE>Matchsticks
In The Ashtray</TITLE> <TIME>0319</TIME></SONG><SONG
SEQ="02"><TITLE>There's No Light In The Men's
Room</TITLE><TIME>0352</TIME> </SONG> ... <SONG
SEQ="11"><TITLE>Song For Papua-New
Guinea</TITLE><TIME>0445</TIME></SONG></SONGS>
```

Note: XMLSERIALIZE supersedes XMLCLOB, and lets you specify result data type and size; it also is what changed the ' to ' in “Men’s”

XMLCONCAT – Concatenate two or more XML elements



```
SELECT XMLSERIALIZE (  
    XMLCONCAT(  
        XMLELEMENT(NAME "Frst", FIRSTNME),  
        XMLELEMENT(NAME "Mid", MIDINIT),  
        XMLELEMENT(NAME "Last", LASTNAME))  
    AS CLOB(1K) )  
FROM EMP WHERE SALARY > 40000;
```

```
<Frst>CHRISTINE</Frst><Mid>I</Mid><Last>HAAS</Last>  
<Frst>MICHAEL</Frst><Mid>L</Mid><Last>THOMPSON</Last>  
<Frst>JOHN</Frst><Mid>B</Mid><Last>GEYER</Last>  
<Frst>VINCENZO</Frst><Mid>G</Mid><Last>LUCCHESI</Last>  
<First>DIANE</Frst><Mid>J</Mid><Last>HEMMINGER</Last>
```


XMLNAMESPACES – Constructs XML namespace declarations



```
SELECT XMLSERIALIZE(  
    XMLELEMENT(NAME "beep:employee",  
    XMLNAMESPACES('http://inhouse.docs' as "beep"),  
    'Hi mom') AS CLOB(1K) )  
FROM SYSIBM.SYSDUMMY1;
```

```
<beep:employee xmlns:beep="http://inhouse.docs">Hi mom  
</beep:employee>
```

Note: Namespace URIs are only used by DB2 when using XPATH expressions in XMLQUERY and XMLEXISTS functions



XMLAGG – Returns a concatenation of XML elements from several table rows

```
SELECT XMLSERIALIZE(  
    XMLAGG(XMLELEMENT(NAME "DPT",  
        XMLATTRIBUTES(DEPTNO AS "DptID"), DEPTNAME)  
    ORDER BY DEPTNO)  
    AS CLOB(1K) )  
FROM DEPT  
WHERE ADMRDEPT = 'D01';
```

```
<DPT DptID="D11">MANUFACTURING SYSTEMS</DPT><DPT DptID="D21">ADMINISTRATION SYSTEMS</DPT>
```

or, more visibly

```
<DPT DptID="D11">MANUFACTURING SYSTEMS</DPT><DPT  
DptID="D21">ADMINISTRATION SYSTEMS</DPT>
```

Note: This is a column function, aggregating multiple row's column values into a result set row

XMLCOMMENT – Generates a comment



```
SELECT XMLSERIALIZE(  
        XMLCOMMENT('HI MOM')  
        AS CLOB(1K) )  
FROM SYSIBM.SYSDUMMY1;
```

```
<!--HI MOM-->
```

XMLDOCUMENT – Generate a complete document



```

CREATE TABLE T1 (C1 CHAR(5) NOT NULL, C2 XML);
INSERT INTO T1 VALUES('AAAAA','<a>Hello</a>');
INSERT INTO T1 VALUES('BBBBB',
    XMLDOCUMENT(XMLELEMENT(NAME "b", 'Goodbye')) );
INSERT INTO T1 VALUES('CCCCC',
    XMLELEMENT(NAME "c", 'Not going to happen') );
SELECT C1, XMLSERIALIZE(C2 AS CLOB(1K)) AS C2 FROM T1;

```

C1	C2
AAAAA	<a>Hello
BBBBB	Goodbye

Note: The third INSERT fails, since XMLELEMENT does not produce a document node

XMLPI – Generate a processing instruction



```
SELECT XMLSERIALIZE(  
    XMLPI(NAME "DISCIPLINE", 'Tough Love')  
    AS CLOB(1K) )  
FROM SYSIBM.SYSDUMMY1;
```

```
<?DISCIPLINE Tough Love?>
```

XMLTEXT – Generate a text node



```
SELECT XMLSERIALIZE(XMLTEXT('Hel>lo&the<re')
  AS CLOB(1K) ) FROM SYSIBM.SYSDUMMY1;
```

```
Hel&gt;lo&amp;the&lt;re
```

More complicated example:

```
SELECT XMLSERIALIZE(XMLELEMENT(NAME "Request",
  XMLCONCAT(XMLTEXT('Help '), XMLTEXT(LASTNAME),
    XMLTEXT(' please') ) ) AS CLOB(1K) )
FROM EMP WHERE SALARY > 40000;
```

```
<Request>Help HAAS please</Request>
<Request>Help THOMPSON please</Request>
<Request>Help GEYER please</Request>
<Request>Help LUCCHESI please</Request>
<Request>Help HEMMINGER please</Request>
```

XMLPARSE – Parse a string argument, returning an XML document



```
SELECT XMLSERIALIZE(XMLPARSE(DOCUMENT  
    '<A> xyz<B> pdq</B> <C>stuff</C> </A>' PRESERVE WHITESPACE)  
    AS CLOB(1K) )  
FROM SYSIBM.SYSDUMMY1;
```

```
<A> xyz<B> pdq</B> <C>stuff</C> </A>
```

```
SELECT XMLSERIALIZE(XMLPARSE(DOCUMENT  
    '<A> xyz<B> pdq</B> <C>stuff</C> </A>' STRIP WHITESPACE)  
    AS CLOB(1K) )  
FROM SYSIBM.SYSDUMMY1;
```

```
<A> xyz<B> pdq</B><C>stuff</C></A>
```

XMLQUERY – Apply an XPATH expression to an XML value



```
SELECT XMLSERIALIZE(  
    XMLQUERY('//TIME' PASSING SONGS_XML)  
    AS CLOB(1K) )  
FROM ALBUM  
WHERE ORDER_NO LIKE 'Ar%';
```

```
<TIME>0319</TIME>      ...      <TIME>0445</TIME>  
<TIME>0250</TIME>      ...      <TIME>0215</TIME>  
<TIME>0312</TIME>      ...      <TIME>0445</TIME>
```

Note: Example shows using an XPATH expression to specify which elements to extract from an XML document

XMLQUERY – Apply an XPATH expression to an XML value, 2



```
SELECT XMLSERIALIZE(  
    XMLQUERY('//TIME/text()') PASSING SONGS_XML)  
    AS CLOB(1K) )  
FROM ALBUM  
WHERE ORDER_NO LIKE 'Ar%';
```

```
03190352025803150405021703050300024503100445  
02500259025203150315022003100235044503150215  
031202520515031902500315033302540220034002190445
```

Note: Example shows retrieving text nodes under an element

XMLQUERY – Apply an XPATH expression to an XML value, 3



```
SELECT XMLSERIALIZE(  
    XMLQUERY('//SONG[@SEQ<="02"]/TIME' PASSING SONGS_XML)  
    AS CLOB(1K) )  
FROM ALBUM  
WHERE ORDER_NO LIKE 'Ar%';
```

```
<TIME>0319</TIME><TIME>0352</TIME>  
<TIME>0250</TIME><TIME>0259</TIME>  
<TIME>0312</TIME><TIME>0252</TIME>
```

Note: Example shows applying an XPATH predicate to an attribute; this predicate can be specified via a host variable (not shown)

XMLEXISTS – Test whether an XPATH expression returns one or more items



```
SELECT ORDER_NO, NO_SONGS  
FROM ALBUM  
WHERE XMLEXISTS('//SONG[@SEQ>="13"]' PASSING SONGS_XML);
```

ORDER_NO	NO_SONGS
In-2029	15
Ept-0300	18
Clo-125	13
Mm-10351	20
Mm-11503	14
Car-781	22

Note: Only appropriate context for XMLEXISTS is a WHERE clause

XMLEXISTS – Test whether an XPATH expression returns one or more items, 2



```
EXEC SQL DECLARE CURS1 CURSOR FOR
SELECT ORDER_NO, NO_SONGS
FROM ALBUM
WHERE XMLEXISTS(
  '//SONG[@SEQ>=$sq]' PASSING SONGS_XML,
  CAST(:seqthresh as VARCHAR(2)) AS "sq");
```

Notes:

- 1) You can use host variables within an XPATH expression to get search flexibility while still using static SQL
- 2) Only choices for casting are VARCHAR(n) and DECFLOAT

DSN_XMLVALIDATE – Validate an XML document against an XML schema



```
INSERT INTO T1(C1) VALUES  
(XMLPARSE (DOCUMENT SYSFUN.DSN_XMLVALIDATE  
(:hv1,'SYSXSR.ORDERSCHEMA')));
```

Notes:

- 1) The schema must have been previously registered (discussed later)
- 2) The only context for using DSN_XMLVALIDATE is as input to the XMLPARSE function

XMLTABLE – Return a result set, based on an XPATH expression



```

SELECT A.ORDER_NO, S.NO_SONGS, S.SEQNO, S.TITLE, S.TIME
FROM ALBUM AS A,
     XMLTABLE('$SO/SONGS/SONG' PASSING A.SONGS_XML AS "SO"
     COLUMNS "NO_SONGS "  SMALLINT      PATH './SONGCT',
              "SEQNO"      SMALLINT      PATH './@SEQ',
              "TITLE"       CHAR(40)      PATH './TITLE',
              "TIME"        CHAR(4)       PATH './TIME'
     ) AS S
WHERE A.ORDER_NO LIKE 'Ar-%';

```

ORDER_NO	NO_SONGS	SEQNO	TITLE	TIME
Ar-0319	11	1	Two Suns And Three ...	0250
Ar-0319	11	2	Star Crossed	0259
...				
Ar-0319	11	11	Crossed Words	0215
...				
Ar-1720	12	12	Song For Austria	0445

XMLCAST – Convert to/from an XML expression



```
SELECT ORDER_NO, XMLCAST(  
    XMLQUERY('//SONGCT' PASSING SONGS_XML)  
    AS SMALLINT)          AS THE_COUNT  
FROM ALBUM  
WHERE ORDER_NO LIKE 'Ar%';
```

ORDER_NO	THE_COUNT
Ar-1059	11
Ar-0319	11
Ar-1720	12

XML in Application Programs



- When you DCLGEN a table with an XML column, you get a DECLARE TABLE that specifies the column as XML

```
EXEC SQL DECLARE ALBUM TABLE
      (ORDER_NO          CHAR(12) NOT NULL,
      SONGS_XML          XML)
END-EXEC
```

- You also get a host variable for the XML column (part of a structure) that is very similar to the variable generated for a LOB column

COBOL Example:

```
10 SONGS_XML          USAGE SQL TYPE IS XML AS CLOB(1M).
```

- You might want to change the specification, if a one megabyte character host variable is not appropriate for your needs
- There are no XML locator variables

XML in Application Programs, 2



COBOL Example:

```
xclob USAGE IS SQL TYPE IS XML AS CLOB(400K).
```

This results in:

```
01  xclob.  
    02  xclob-LENGTH  PIC S9(9) COMP.      <--- fullword length prefix  
    02  xclob-DATA.  
        49  FILLER PIC X(32767).  
        <repeated 11 more times>  
        49  FILLER PIC X(409600-12*32767).
```

Note: Other languages result in similar data structures

XML in Application Programs, 3



- You can **SELECT** or **FETCH INTO** an XML variable
 - Best to use the **XMLSERIALIZE** function, if destination is a **CLOB**, **BLOB**, or **DBCLOB** (i.e., not XML) variable
- You can **UPDATE** an XML column from an XML variable
 - All XML column updates are done via complete replacement of the column value
- You can **INSERT** a row in a table that has an XML column, specifying an XML variable as the data source for the column

XML in Application Programs, 4



- When you use an AS CLOB or AS DBCLOB variable, you are requesting that DB2 do character conversion
 - From the code page of the variable to UTF-8 Unicode for an INSERT or UPDATE
 - To the code page of the variable from UTF-8 Unicode for a SELECT or FETCH
- When you use a AS BLOB variable, you are requesting that DB2 do no character conversion
 - For an INSERT or UPDATE, you are providing an XML document in UTF-8 Unicode
 - For a SELECT or FETCH, DB2 returns the XML document in UTF-8 Unicode

XML in Application Programs, 5



- You can use file reference variables, if your program only needs to move XML documents between the data base and an external file
- XML file reference variables work exactly like LOB file reference variables (see session #6142 for examples)
- **Definition:**
 - `SQL TYPE IS XML AS BLOB_FILE`
 - `SQL TYPE IS XML AS CLOB_FILE`
 - `SQL TYPE IS XML AS AS DBCLOB_FILE`
- Code page conversion is done for CLOB_FILE and DBCLOB_FILE variables, and not done for BLOB_FILE variables
 - The conversion (or lack thereof) applies to the file's contents

XML SQL – UPDATE (COBOL)



```
EXEC SQL DECLARE ALBUM TABLE
  (ORDER_NO      CHAR(12) NOT NULL,
   SONGS_XML     XML)  END-EXEC.
```

```
01 txorder      PIC X12
01 xclob        USAGE IS SQL TYPE IS XML AS CLOB 10K
01 xblob        USAGE IS SQL TYPE IS XML AS BLOB 10K
01 clobber      USAGE IS SQL TYPE IS CLOB 10K
```

```
EXEC SQL UPDATE ALBUM  SET SONGS_XML = :xclob
  WHERE ORDER_NO = :txorder  END-EXEC
```

```
EXEC SQL UPDATE ALBUM  SET SONGS_XML = :xblob
  WHERE ORDER_NO = :txorder  END-EXEC
```

```
EXEC SQL UPDATE ALBUM
  SET SONGS_XML = XMLPARSE(DOCUMENT :clobber)
  WHERE ORDER_NO = :txorder  END-EXEC
```

XML SQL – SELECT (COBOL)



```
EXEC SQL DECLARE ALBUM TABLE
  (ORDER_NO      CHAR(12) NOT NULL,
   SONGS_XML     XML)  END-EXEC.
```

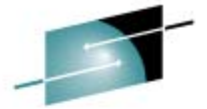
```
01 txorder      PIC X12
01 xclob        USAGE IS SQL TYPE IS XML AS CLOB 10K
01 xblob        USAGE IS SQL TYPE IS XML AS BLOB 10K
01 clobber      USAGE IS SQL TYPE IS CLOB 10K
```

```
EXEC SQL SELECT SONGS_XML INTO :xclob
  FROM ALBUM WHERE ORDER_NO = :txorder END-EXEC
```

```
EXEC SQL SELECT SONGS_XML INTO :xblob
  FROM ALBUM WHERE ORDER_NO = :txorder END-EXEC
```

```
EXEC SQL SELECT XMLSERIALIZE(SONGS_XML AS CLOB(10K))
  INTO :clobber FROM ALBUM WHERE ORDER_NO = :txorder
END-EXEC
```

XML SQL – Cursor FETCH (COBOL)



SHARE
Technology - Connections - Results

```
EXEC SQL DECLARE ALBUM TABLE  
  (ORDER_NO      CHAR(12) NOT NULL,  
   SONGS_XML     XML)  END-EXEC.
```

```
01 txorder      PIC X12  
01 xclob        USAGE IS SQL TYPE IS XML AS CLOB 10K  
01 xblob        USAGE IS SQL TYPE IS XML AS BLOB 10K
```

```
EXEC SQL DECLARE XCURS1 CURSOR FOR  
  SELECT SONGS_XML FROM ALBUM  
  WHERE ORDER_NO = :txorder END-EXEC.
```

```
EXEC SQL OPEN XCURS1 END-EXEC
```

```
EXEC SQL FETCH XCURS1 INTO :xclob END-EXEC  
EXEC SQL FETCH XCURS1 INTO :xblob END-EXEC
```

```
EXEC SQL CLOSE XCURS1 END-EXEC
```

XML SQL – Cursor FETCH (COBOL), 2



```
EXEC SQL DECLARE ALBUM TABLE  
    (ORDER_NO      CHAR(12) NOT NULL,  
     SONGS_XML     XML)  END-EXEC.
```

```
01 txorder      PIC X12  
01 clobber      USAGE IS SQL TYPE IS CLOB 10K
```

```
EXEC SQL DECLARE XCURS2 CURSOR FOR  
    SELECT XMLSERIALIZE(SONGS_XML  
    AS CLOB(10K)) FROM ALBUM  
WHERE ORDER_NO = :txorder END-EXEC.
```

```
EXEC SQL OPEN XCURS2 END-EXEC
```

```
EXEC SQL FETCH XCURS2 INTO :clobber END-EXEC
```

```
EXEC SQL CLOSE XCURS2 END-EXEC
```



XML Indexes

- You can create indexes to support faster access to data stored in XML columns
 - Indexes are based on XPATH statements that specify particular element nodes and / or attributes
 - Indexes are used in conjunction with XMLEXISTS predicates, presumably with the same element / attribute nodes used in predicates

```
CREATE INDEX XINDEX1 ON ALBUM(SONGS_XML)  
GENERATE KEY USING XMLPATTERN  
'//SONG/@SEQ' AS SQL VARCHAR(2);
```

```
CREATE INDEX XINDEX2 ON ALBUM(SONGS_XML)  
GENERATE KEY USING XMLPATTERN  
'//SONG/TITLE/text()' AS SQL VARCHAR(40);
```

XML Indexes, 2



- The index based on the XPATH expression '//SONG/@SEQ' would be useable in the following queries:

```
SELECT ORDER_NO, SONGS_XML FROM ALBUM  
WHERE XMLEXISTS('//SONG[@SEQ>="13"]' PASSING SONGS_XML);
```

```
SELECT ORDER_NO, SONGS_XML FROM ALBUM  
WHERE XMLEXISTS('//SONG[@SEQ>="$sq"]' PASSING SONGS_XML,  
CAST(:seqthresh as VARCHAR(2)) AS "sq");
```

- But not in the following query, since the XPATH expression occurs in the SELECT list, and not in the WHERE clause

```
SELECT ORDER_NO, XMLSERIALIZE(  
XMLQUERY('//SONG[@SEQ>="15"]/TITLE/text()'  
PASSING SONGS_XML) AS CLOB(1K) )  
FROM ALBUM;
```

XML Indexes – Notes



- 1) The data type must be specified as either VARCHAR(n) or DECFLOAT
- 2) You can specify UNIQUE for an XML index
Uniqueness is over the data type, the path to the node, the value of the node
- 3) No multi-predicate indexes are allowed
- 4) No predicates are allowed in the index definition
- 5) Indexes can only access element nodes, attribute nodes, or text nodes (i.e., not comment or processing instruction nodes)
- 6) You can't have a partitioned index on an XML column, although you can have an XML index on a partitioned table
- 7) If an XML index is used for an access path, the ACESSTYPE column of PLAN_TABLE contains the value DX

XML Schemas and DB2



- **DB2 provides an XML Schema repository that supports storing schemas, as well as the ability to validate an XML document against a schema**
- **Three DB2 catalog tables for storing schema information**
 - **SYSIBM.XSROBJECTS – Registered XML schemas**
 - **SYSIBM.XSROBJECTCOMPONENTS – Schema documents**
 - **SYSIBM.XSROBJECTHIERARCHIES – Schema document hierarchy relationships**
- **Four stored procedures for schema registration**
 - **XSR_REGISTER – Start registering a schema**
 - **XSR_ADDSCHEMADOC – Add additional schema documents**
 - **XSR_COMPLETE – Complete schema registration process**
 - **XSR_REMOVE – Removes all components of an XML schema**



XML Catalog Tables

- **There are several new DB2 catalog tables to support XML data**

SYSIBM.SYSXMLRELS – one row for each XML table that is created for an XML column

SYSIBM.SYSXMLSTRINGS – one row contains a single string and its unique ID that are used to condense XML data

SYSIBM.XSROBJECTS – one row for each registered XML schema

SYSIBM.XSROBJECTCOMPONENTS – one row for each component (document) in an XML schema

SYSIBM.XSROBJECTHIERARCHIES – one row for each component (document) in an XML schema to record the XML schema document hierarchy relationship

XML Impact on Utilities



Many DB2 utilities have been enhanced to support XML data

- **CHECK DATA** – In addition to checking LOB relationships, the CHECK DATA utility also checks XML relationships
- **CHECK INDEX** – You can use the CHECK INDEX utility to check XML indexes, DocID indexes, and NodeID indexes
- **CHECK INDEX** does not check any user defined XML indexes
- **COPY** – You can use the COPY utility to copy XML table spaces and indexes
 - XML table spaces and indexes are not automatically copied when you COPY the base table space

XML Impact on Utilities, 3



Still more ...

- **QUIESCE – QUIESCE TABLESPACESET includes related XML objects**
- **REBUILD INDEX – REBUILD INDEX can rebuild XML indexes, DocID indexes, and NodeID indexes**
- **RECOVER INDEX and RECOVER TABLESPACE – RECOVER works for XML indexes and XML tablespaces**
- **REORG INDEX and REORG TABLESPACE – REORG works for XML indexes and XML tablespaces**
 - **REORGing the base table does not automatically REORG any XML table spaces or indexes**

XML Impact on Utilities, 4



Last, but not least ...

- **REPAIR** – REPAIR works for XML indexes and XML tablespaces
- **REPORT TABLESPACESET** – The output from REPORT TABLESPACESET includes related XML objects
- **RUNSTATS** – RUNSTATS will gather statistics on XML table spaces and indexes
- **UNLOAD** – You can unload XML data with the UNLOAD utility
 - Output may be a file reference variable, with file name specified via a template

Summarizing DB2 and XML



- **With Version 9, XML is ready for prime time in DB2**
 - Processing is inboard of the DBM1 address space
 - Objects are native
 - Functions are built-in
 - Embedded SQL fully supports the XML data type
 - XPATH enables intra-document searching and retrieval (with supporting indexes)
 - Schema support
 - Utility support

For More Information ...



- **IBM Publications:**
 - **DB2 Version 9 XML Guide – SC18-9858**
 - **DB2 9: pureXML Overview and Fast Start – SG24-7298**
 - **DB2 Version 9 XML Extender Administration and Programming – SC18-9857**
- **For details about table definitions and contents shown in this presentation, contact:**

hunter@trainersfriend.com
303.393.8716

- **For more information about our DB2 curriculum, including Version 8 and Version 9 transition, DB2's Greatest Hits, and Native SQL Stored Procedures (plus some free technical papers), see our web site:**

www.trainersfriend.com