IBM Information On Demand 2006

IBM

Leveraging Native XML Support in DB2 9 for z/OS

Guogen (Gene) Zhang, IBM SVL

Session 1730A Data Servers – DB2 for z/OS

October 17, 2006

TAKE BACK CONTROL

IBM INFORMATION ON DEMAND 2006
October 15 – 20, 2006
Anaheim Convention Center
Anaheim, California

Agenda

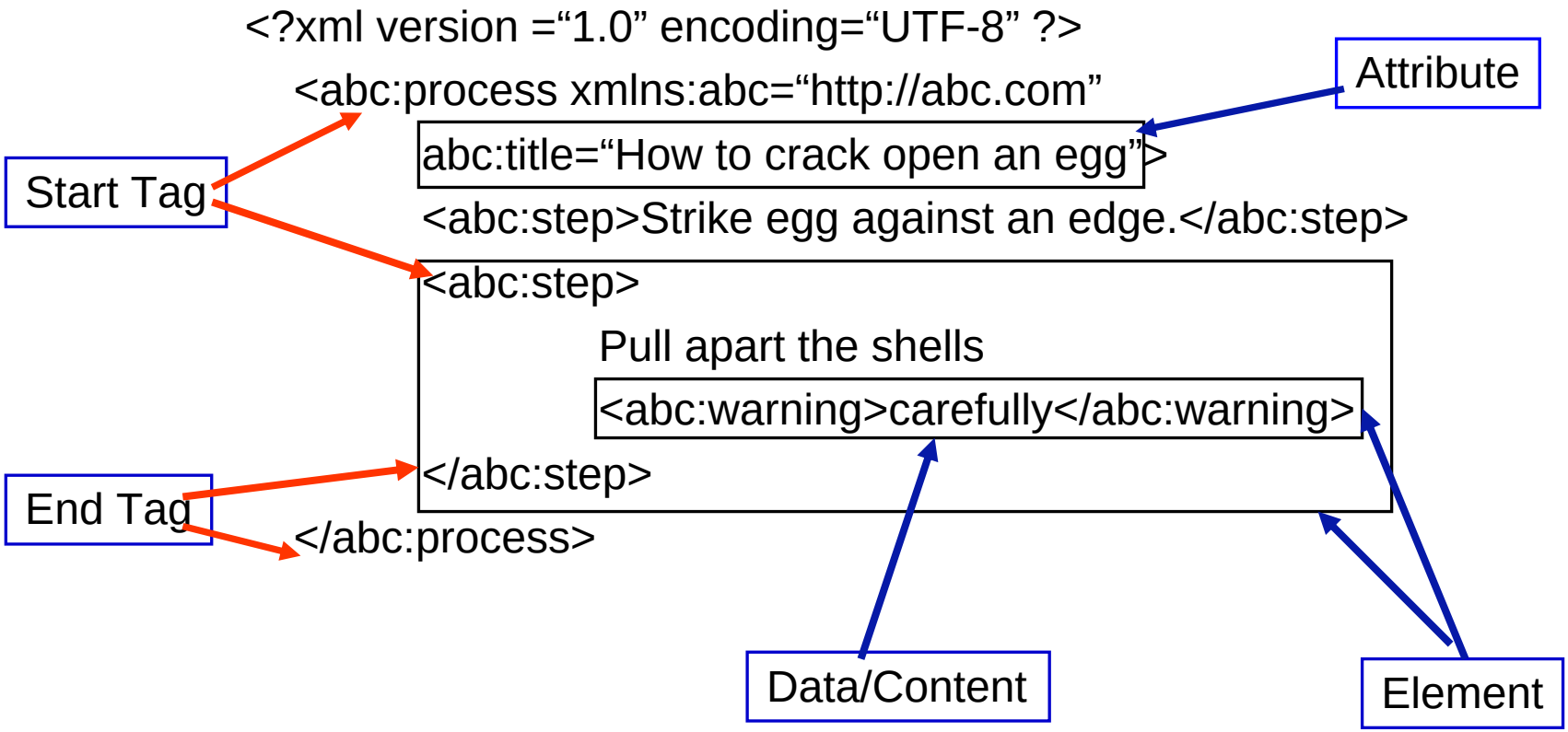
- Basics of Native XML Support in DB2 9 for z/OS
  - XML Data Type, DDL, and Storage
  - Query Language and API
  - XML Schema Validation and Decomposition
  - Utilities
- Advanced Topics
  - Indexing and Access Methods
  - XPath Typing and Cardinality
- Scenarios to Use Native XML
- Connecting to the Web
- Summary

Basics of Native XML Support

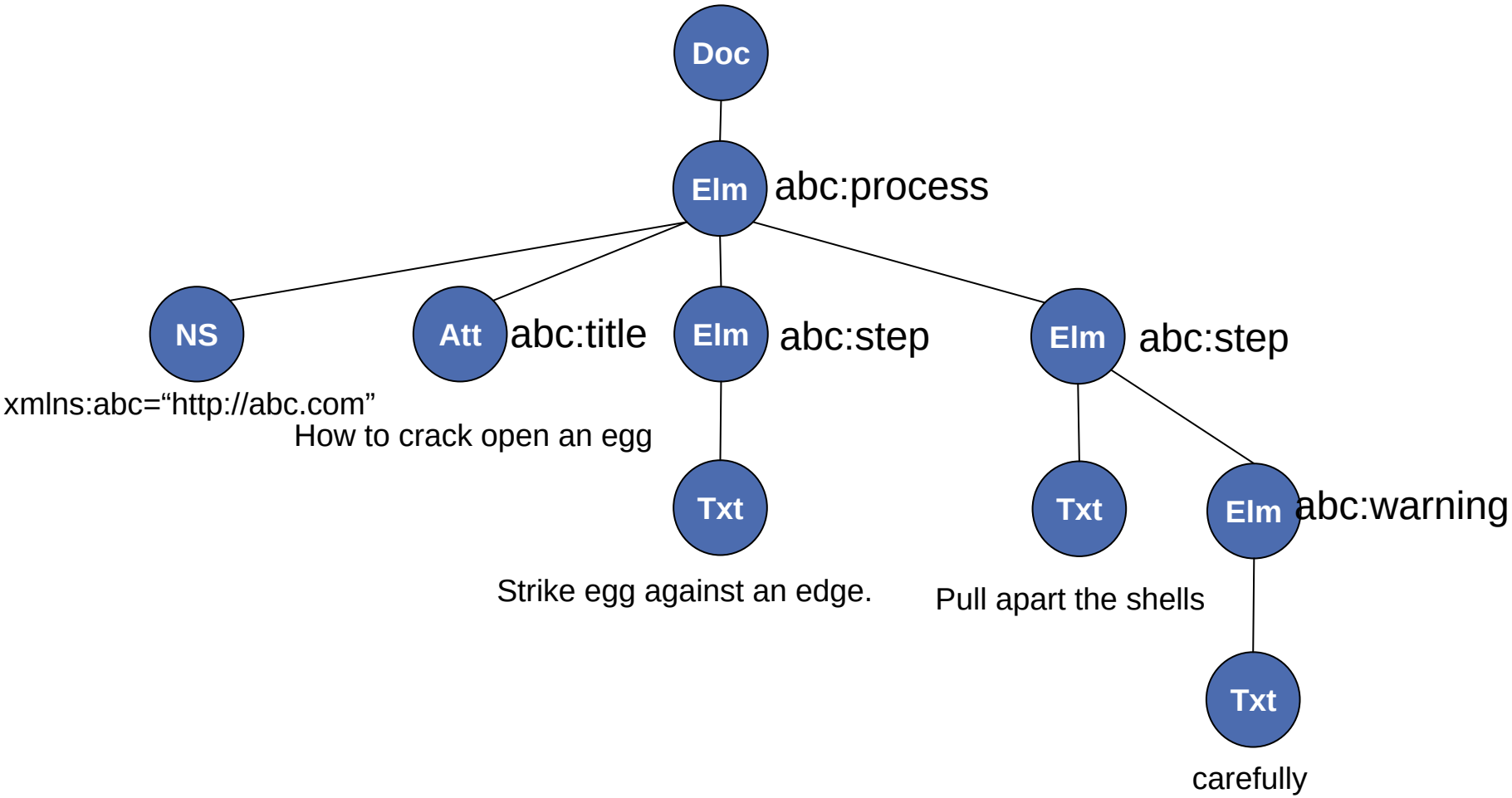
TAKE BACK CONTROL

What is XML

XML = Extensible Markup Language



XML (XQuery) Data Model

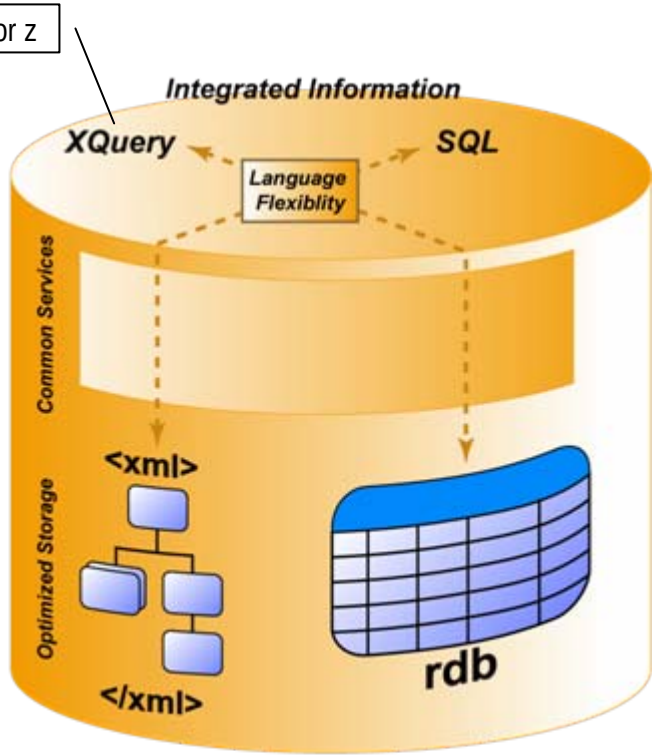


Native XML and pureXML® in DB2

- Native XML
  - Hierarchical data model: XDM (XQuery Data Model)
  - XML query languages: XQuery, XPath, (XSLT)
- pureXML® in DB2
  - Designed specifically for XML from the ground up
    - Supports XML hierarchical structure storage
    - Native operations and languages: XPath, SQL/XML, XQuery
  - Not transformation into relational
  - Not using objects or nested tables
  - Not using LOBs

XML in DB2 - A Long-term View

SQL Developer… "I see a sophisticated RDBMS that also supports XML"



XML Developer… "I see a sophisticated XML repository that also supports SQL"

XML integrated in all facets of DB2!

Storage, indexing, queries, utilities, tools

Native XML Features

- First-class XML data type, native storage of XQuery Data Model
- SQL/XML constructor functions
  - Construct XML from relational data in V8:
    XMLElement, XMLAttributes, XMLNamespaces, XMLForest, XMLConcat, XMLAGG
  - New constructor functions in V9:
    XMLText, XMLPI, XMLComment, XMLDocument, and binary string support and more null handling options
- XMLPARSE and XMLSERIALIZE
- XML indexes
- Important SQL/XML functions with XPath
  - XMLEXISTS, XMLQUERY
- XML Schema repository, Validation UDF, (and decomposition)
- DRDA (distributed support) and application interfaces
- Utilities

XML Type and DDL

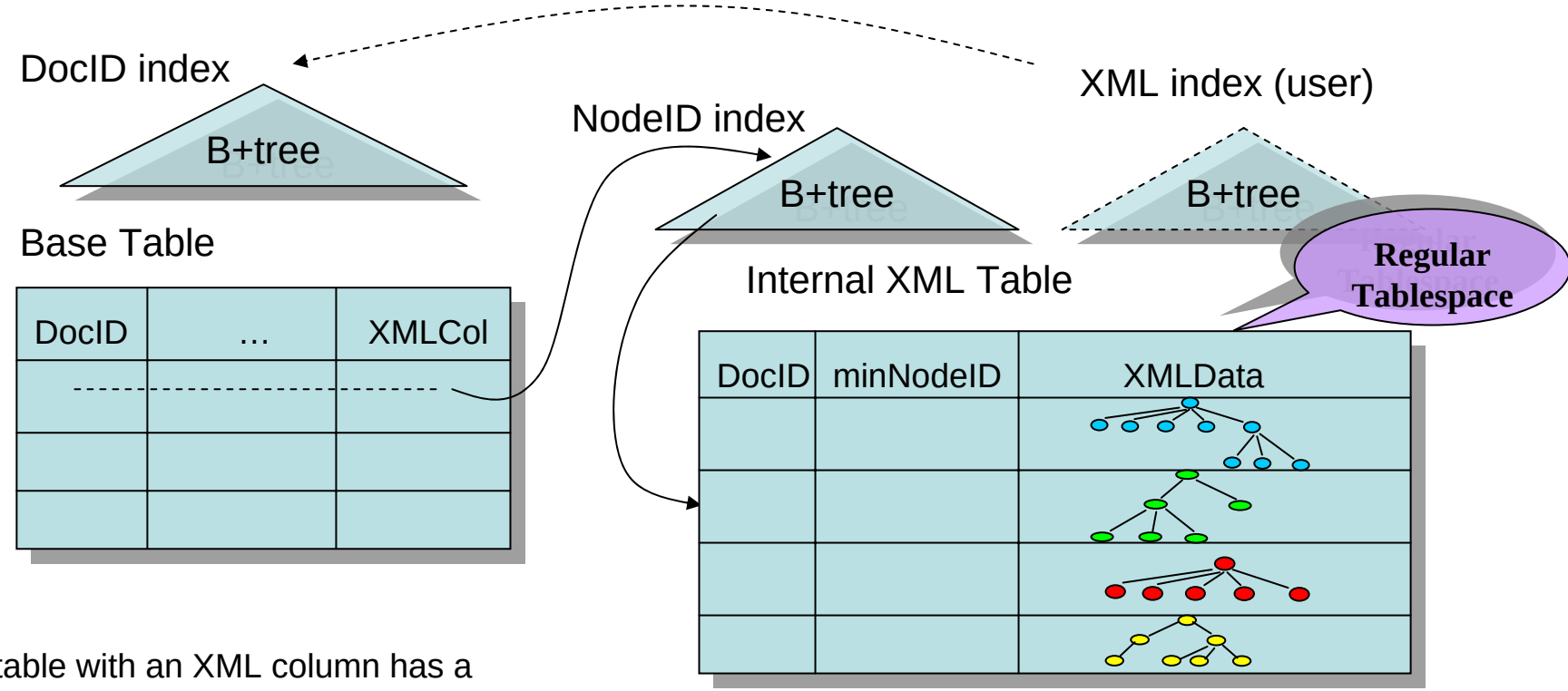
```
CREATE TABLE PurchaseOrders (
  ponumber    varchar(10) not null,
  podate      date not null,
  status      char(1),
  XMLpo       xml)
    IN MYDB.MYTS;
```

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

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

ALTER TABLE PurchaseOrders
    ADD revisedXMLpo xml;
```

XML Storage



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

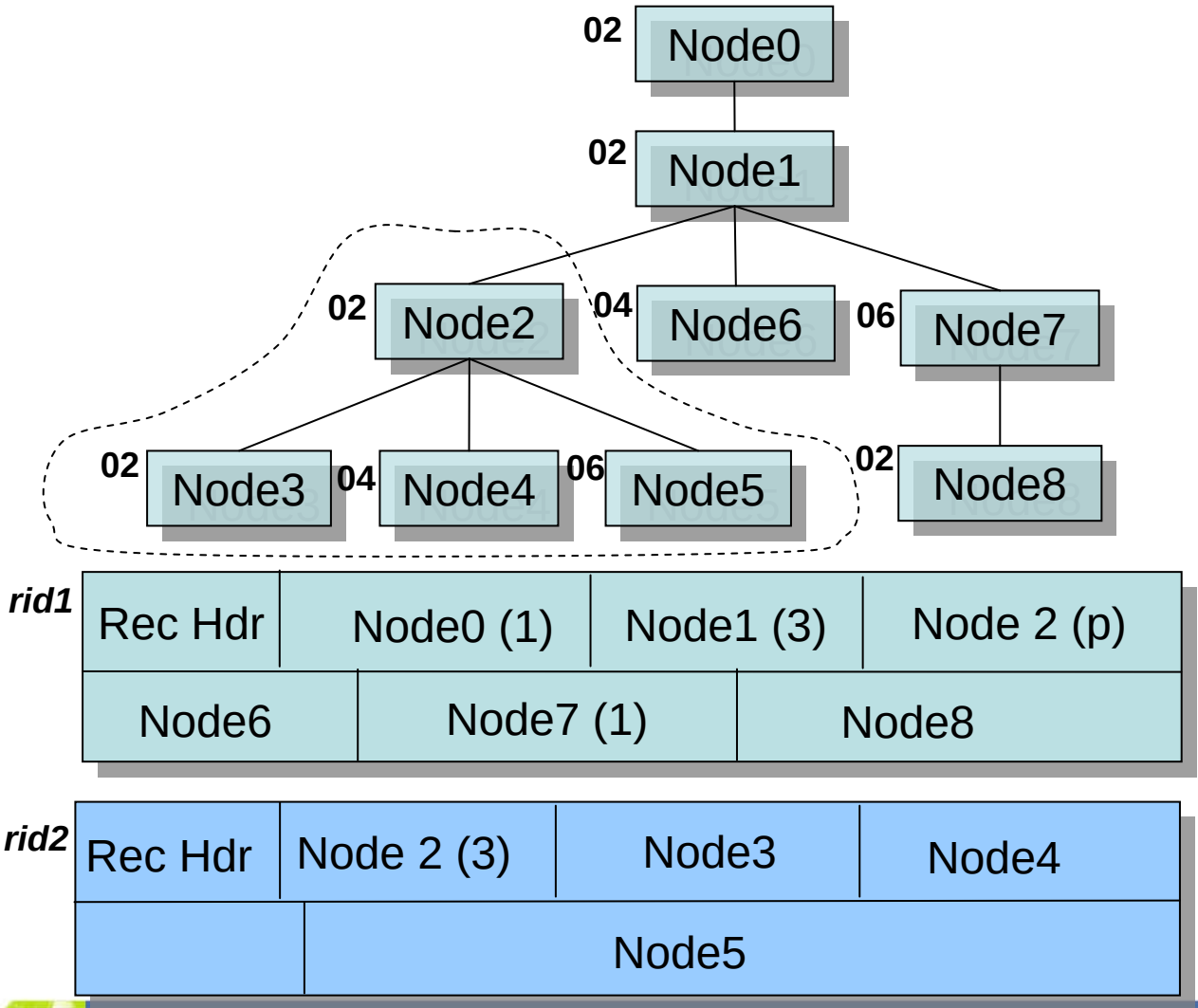
Each node contains local node id, length and optional number of children.

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.



Manipulating XML Data

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

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

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

DELETE FROM PurchaseOrders WHERE ponumber = '12345';

LOAD into PurchaseOrders …
```

Host var of XML type

String literal is OK

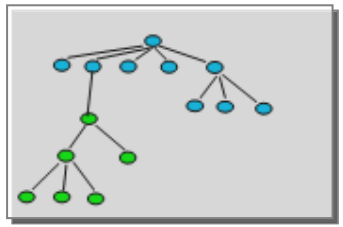
Whole document replacement

XMLParse and XMLSerialize

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

XMLParse →

← XMLSerialize



- XMLParse
  - Allows strip whitespace or preserve whitespace
  - Implicit XMLParse applies for bind-in XML hostvar or inserting hostvar or string literal.
- XMLSerialize
  - With XML declaration or without
  - Implicit XMLSerialize applies for bind-out XML type (w/ XML declaration)

Retrieving XML Data

- Simple select:

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

- Select with condition:

```
SELECT 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 …;
```

XPath Support

- Used in XMLEXISTS, XMLQUERY, and XML indexing

- XPath 1.0 + - (subset of XPath 2.0)
  - XPath 1.0 constructs in XPath 2.0 semantics
  - + more data types: xs:boolean, xs:integer, xs:decimal, xs:double, xs:string
  - + namespace declaration from XQuery prolog
  - - Axes: only 5 forward axes & parent axis are supported.

- All stored XML data are untyped initially (in V9).
  - Explicit type casting may be needed in some cases

Constructor Example

```
SELECT XMLDOCUMENT(
            XMLELEMENT(NAME “hr:Department",
             XMLNAMESPACES(‘http://example.com/hr’ as “hr”),
             XMLATTRIBUTES (e.dept AS "name" ),
             XMLCOMMENT(‘names in alphabetical order’),
             XMLAGG(XMLELEMENT(NAME “hr:emp", e.lname)
                          ORDER BY e.lname  )
                        ) ) AS "dept_list”
FROM employees e
GROUP BY dept;
```

```
<?xml version=“1.0” encoding=“UTF-8”>
<hr:Department xmlns:hr=“http://example.com/hr”
    name="Shipping">
   <!-- names in alphabetical order -->
   <hr:emp>Lee</hr:emp>
   <hr:emp>Martin</hr:emp>
   <hr:emp>Oppenheimer</hr:emp>
</hr:Department>
```

API Support

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

Java JDBC Example

```
PreparedStatement pstmt = connection.prepareStatement("INSERT INTO
     PurchaseOders VALUES(?, ?)");     // 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()) {
  int po_no = rs.getInt ("ponumber");
  com.ibm.db2.jcc.DB2Xml xml = (com.ibm.db2.jcc.DB2Xml) rs.getObject ("xmlpo");
  System.out.println (xml.getString()); // uninterpreted flat xml text
}
```

XML Schema Support

- Register a schema in XML Schema Repository (XSR)
- External names
  - target namespace: e.g., "http://www.ibm.com/software/catalog"
  - schema location: e.g., "http://www.ibm.com/schemas/software/catalog.xsd"
- SQL identifier - used to reference schemas in SQL
  - unique identifier in DB, e.g., SYSXSR.ORDERSCHEMA
- Where are schemas used?
  - DSN_XMLValidate in SQL (UDF for XMLValidate)
  - Decomposition

Registering an XML Schema (Procedure)

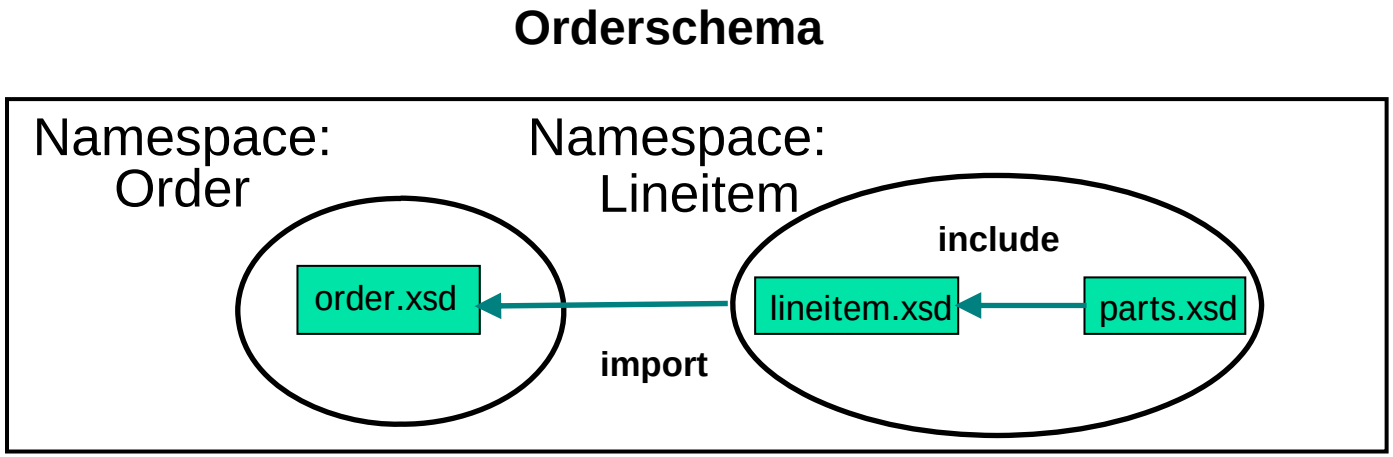
- XSR_REGISTER (rschema, name, schemalocation, xsd, docproperty)
- XSR_ADDSCHEMADOC (rschema, name, schemalocation, xsd, docproperty)
- XSR_COMPLETE (rschema, name, schemaproperties, isUsedForDecomp)
- XSR_REMOVE(rschema, name)

- Parameters:

rschema	– null or ‘SYSXSR’;
identifier	– SQL name (VARCHAR(128));
schemalocation	– VARCHAR(1000);
xsd	– XML schema document (BLOB(30M));
docproperty	– BLOB(5M), may be used by tools;
schemaproperties	– same as docproperties
isUsedForDecomp	– INTEGER, 1 yes, 0 no.

- Java Driver (JCC) provides a set of APIs for schema registration

Example: Registering an XML Schema



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

Using XML Schema

- Schema validation – type annotation not kept

  INSERT into PurchaseOrders
  VALUES('200300001', CURRENT DATE, 'A',
  DSN_XMLValidate(:xmlPo,SYSXSR.ORDERSchema));

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

  E.g. orderID ->PORDER.ORDERID

  <attribute name="orderID" type="xs:string"
  sql:relation = "PORDER"
  sql:field = "ORDERID" />

annotations

PORDER

ORDERID	ORDE
19991201-ZFG	1999-

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

Advanced Topics

TAKE BACK CONTROL

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>
```

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>X5, 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.
- Restriction: Index key value cannot span multiple rows. Always safe to index leaf nodes with short values.

Examples of XPath - Typing

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

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

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

Examples of XPath - Cardinality

- No cardinality problem: “Find all the products in the Catalog with RegPrice > $price”
  XMLQUERY(‘/Catalog/Categories/Product[RegPrice > $price]’ PASSING XCatalog, 200 as “price”)

- To avoid cardinality violation: “Find all the products on sale in the Catalog”
  XMLQUERY(‘/Catalog/Categories/Product[RegPrice > SalePrice/xs:double(.))]’ PASSING XCatalog)

- To avoid cardinality violation: “Find all the products with more than 10% discount in the Catalog”
  XMLQUERY(‘/Catalog/Categories/Product[RegPrice/(. * 0.9) > SalePrice]’ PASSING XCatalog)

Performance and Scalability

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

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.	'/Catalog/Categories/Product[RegPrice > 100]' with index on '/Catalog/Categories/Product/RegPrice' as SQL DECFLOAT
DocID ANDing/ORing "DX/DI/DU" intersect or union (unique) DocID lists from XML indexes, then access the base table and XML table.	'/Catalog/Categories/Product[RegPrice > 100 and Discount > 0.1]' With indexes on: '//RegPrice' as SQL DECFLOAT and '//Discount' as SQL DECFLOAT

XML Index Usage

- Criteria:
  - Index pattern is equal to or less restrictive than the query predicate:
    index: //product/regprice v.s.
    query: /catalog//product[regprice > 10]
  - Data types have to match.
- Use internal “between” for better performance.
  - //item[@size > 5 and @size < 10]
  - //product[wt > 10 and wt < 20] =>
    //product[wt[. > 10 and . <20]]

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 to construct new documents from existing documents.

Example: 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>
```

FETCH CONTINUE for XML and LOB

- No size associated with XML values
- Hard to allocate large memory
- Shortcomings with LOB Locator
- New FETCH CONTINUE statements: (one of two ways)
  - DECLARE CURSOR1 CURSOR FOR SELECT C2 FROM T1;
  - OPEN CURSOR1;
  - FETCH WITH CONTINUE CURSOR1 into :clobhv;
  - if (sqlcode >= 0) & sqlcode <> 100
  - Loop if truncation occurs until lob/xml complete (total length)
  - FETCH CURRENT CONTINUE CURSOR1 into :clobhv;
  - Consume :clobhv content
  - end loop
- Another way is to use FETCH … INTO DESCRIPTOR :SQLDA

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 Timer 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.

Scenarios to Use Native XML

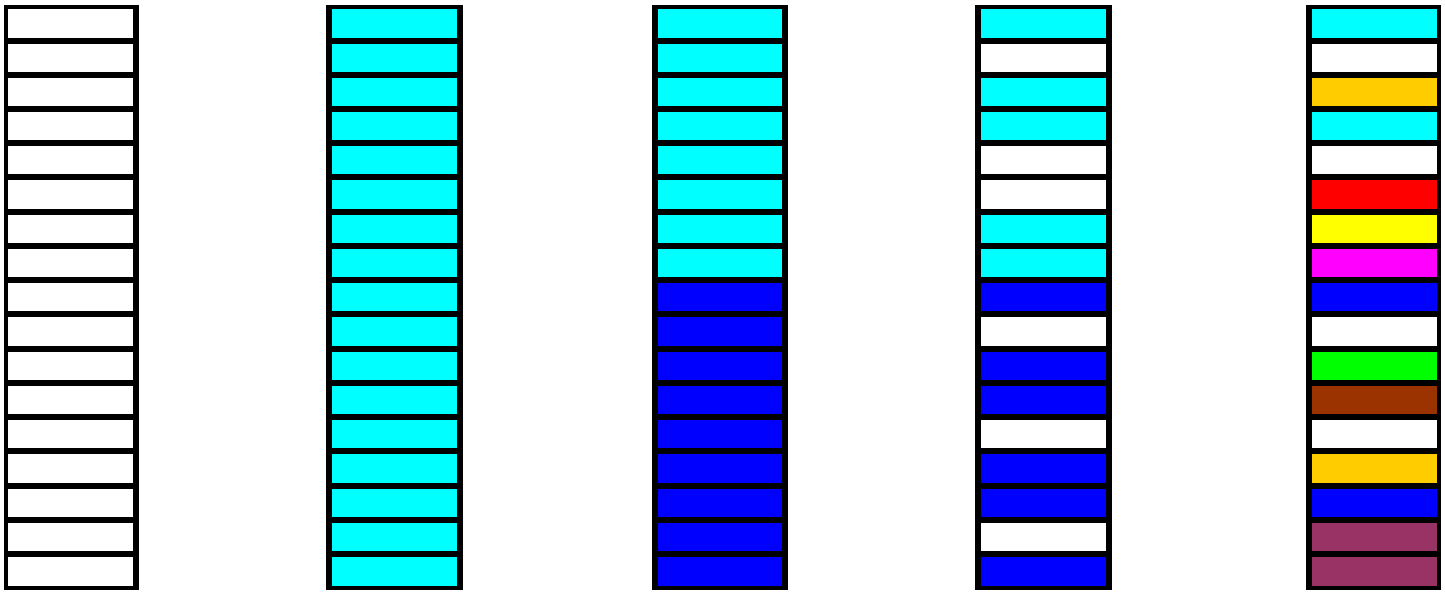
TAKE BACK CONTROL

XML Characteristics

- XML Characteristics
  - Flexible hierarchical data structures
  - Self-describing, no fixed schema for a column
  - Ordering is important
- Flexibility
  - Any XML documents can be put into a column
  - Indexing and query with different types on the same data
- Search capability
  - Indexing and efficient search into XML documents (you cannot achieve the same with VARCHAR or LOB)

XML Schema Flexibility

Mix of documents in an XML column → Many Options:



No Schema

One Schema

Schema V1 & Schema V2

Documents w/ and w/o schema

Any mix you want!

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.

Processing XML Data

- Processing XML data directly:
  - ACORD, FIXML, FpML, MIMSO, XBRL,
  - DJXDM, HR-XML, HL7, ARTS, HIPAA, NewsML, XForms
  - Insurance policy, contract, purchase order, emails etc
- Insert/Update/Delete/Select/Extract/Construct
- Indexing/Search
- All XML solutions
  - From one angle: trade-off - speed of development v.s. storage space

Scenario 1 Trading Exceptions

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

Scenario 2 Auto Insurance Policy Variations

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

Scenario 3 Email Marketing

- Emails are tagged with keywords
- Keywords are searched to identify the potential sales leads
- Instead of side table and CLOB, use XML and indexing on the tagged keywords
- Benefit: flexible, high performance

Scenario 4 Senate Bills & Report

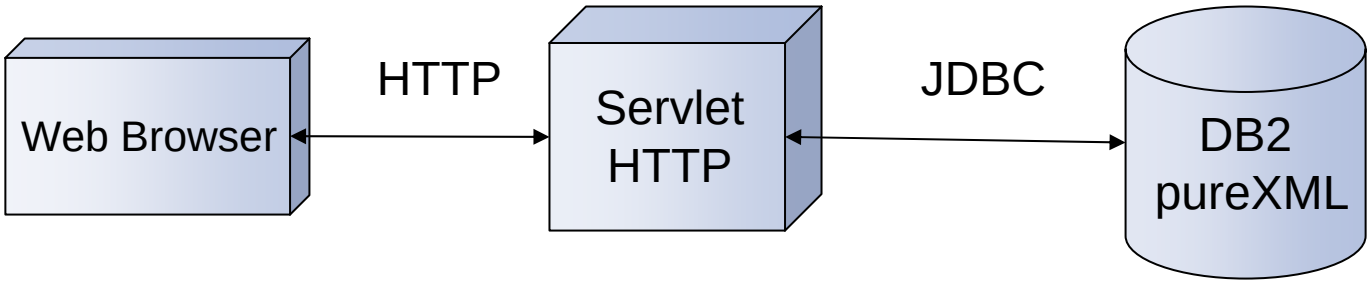
- Committee, sub-committees, and assignment
- Legislation bills, titles, documents, sponsors, and actions
- Bills and actions are in XML
- Generate report on the bills, committees and actions

Connecting to the Web

TAKE BACK CONTROL

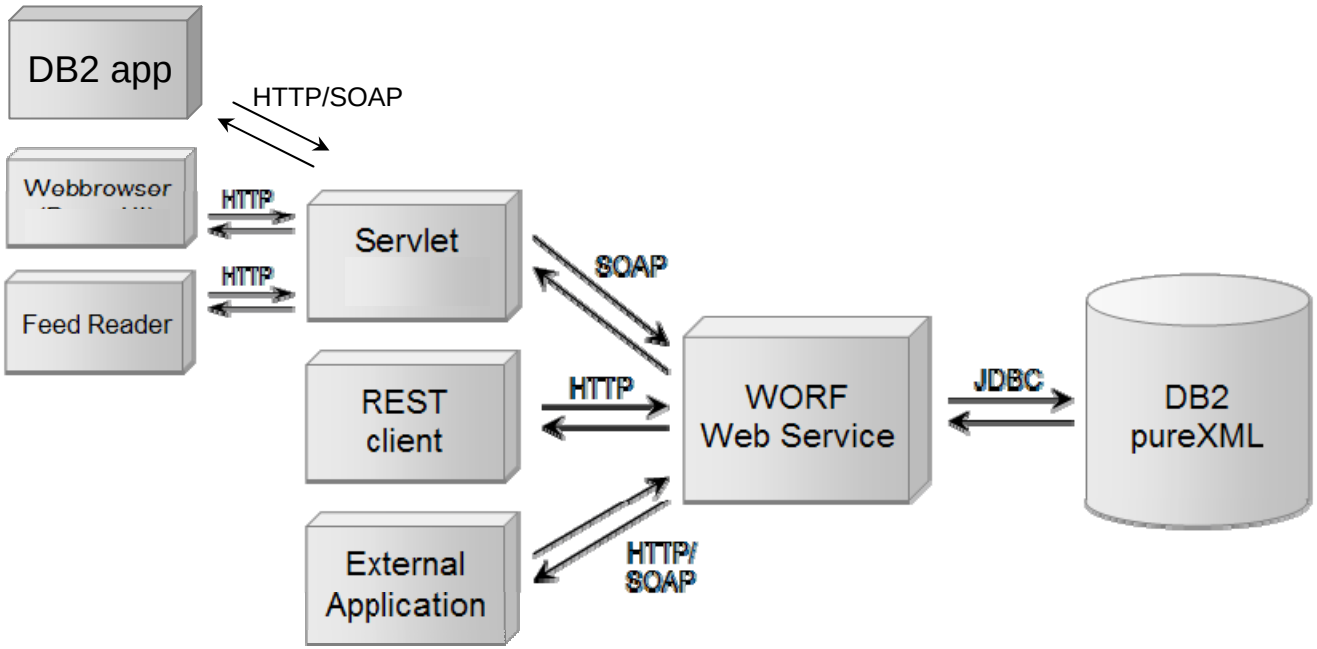
Config 1 Generate XHTML

- Instead of using Java or other languages to generate XHTML, use SQL directly to generate dynamic web pages
- (Query examples will be shown in Session 2206A)



Config 2 Sending and Receiving SOAP

- In web services, SQL statements (as consumer) can directly send and receive SOAP XML messages through web services UDF
- (Consumer query examples will be shown in Session 2206A)



Summary

TAKE BACK CONTROL

Summary

- Native XML type and storage
- SQL/XML with XPath
- API and host language support
- Utilities
- Indexing, performance and scalability
- Usage scenarios