Advanced Aspects and New Trends in XML (and Related) Technologies

RNDr. Irena Holubová, Ph.D.
holubova@ksi.mff.cuni.cz

Lecture 4. Support of XML in Oracle DB

http://www.ksi.mff.cuni.cz/~holubova/NPRG039/
Oracle XML Database (XML DB)

- Collection of XML technologies built into Oracle DB
  - XML-enabled database
  - Storage, retrieval, processing of XML data
- Topics:
  - XML data storage
    - XML Schema annotations
    - XML storage comparison
  - Generating XML from SQL data
  - XML data retrieval
  - XML data updates
  - XML data indexing
  - Evolution of XML applications
XML Data Storage
XMLType

- Native data type for storing XML data
  - Similar to, e.g., data type DATE
    - Table column type, parameter, return value or variable in PL/SQL procedures, ...
  - Content must be well-formed

- Types of storage:
  1. Storing intact into a CLOB XMLType column
  2. Native storing of XMLType column
  3. Shredding into relational tables (object-relational storage)

- Built-in member functions:
  - Creating an XMLType instance from various resources
  - Extracting XML content
  - Operating XML content
  - Validating against XSDs
  - XSL transformations
XMLType
CLOB storage

- Advantages:
  - Best preserves original format
  - Maximum flexibility for XML schema evolution

- Disadvantages:
  - Expensive querying and updates
  - Require building a DOM tree

CREATE TABLE product (
  id VARCHAR(10),
  name VARCHAR2(100),
  description XMLType
) XMLTYPE COLUMN description
STORE AS CLOB;

INSERT INTO product (id, name, description) VALUES ('XDK', 'XML Developer''s Kit',
  XMLTYPE('<DESCRIPTION><KEYWORD>xdk</KEYWORD> is a set of standards-based utilities that help to build XML applications.</DESCRIPTION>'));
XMLType
XML Schema-based Storage

- Object-relational structure specified by an XSD
  - Defines how to shred an XML document into SQL objects
- Advantages:
  - Speeding up queries and updates (= exploitation of efficient implementation of SQL queries)
- Each XSD must be first registered under a unique URL
  - To be referenced from XML Schema-based XMLTypes
- Registration:
  - Creating SQL objects
  - (Optional) generation of a default table
    - XML DB Repository
Registration of an XSD

BEGIN

  DBMS_XMLSCHEMA.registerSchema(
    SCHEMAURL=>'http://xmlns.oracle.com/xml/content.xsd',
    SCHEMADOC=>'<?xml version="1.0" encoding="UTF-8"?>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="DESCRIPTION">
        <xs:complexType mixed="true">
          <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element name="KEYWORD" type= "xs:string"
              maxOccurs="unbounded"/>
          </xs:choice>
        </xs:complexType>
      </xs:element>
    </xs:schema>',
    LOCAL=>true,
    GENTYPES=>true,
    GENTABLES=>false);

END;
During Registration

- SQL objects to store XMLType are generated
- Default tables are created

```sql
SELECT object_name, object_type FROM user_objects;
```

<table>
<thead>
<tr>
<th>OBJECT_NAME</th>
<th>OBJECT_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYWORD209_COLL</td>
<td>TYPE</td>
</tr>
<tr>
<td>DESCRIPTION208_T</td>
<td>TYPE</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

```sql
describe KEYWORD209_COLL;
```

"KEYWORD209_COLL" AS VARRAY(2147483647) OF VARCHAR2(4000 CHAR)
CREATE TABLE product (  
id VARCHAR(10),
name VARCHAR2(100),
description XMLType )
XMLType COLUMN description
XMLSCHEMA "http://xmlns.oracle.com/xml/content.xsd"
ELEMENT "DESCRIPTION"

INSERT INTO product (id, name, description)
    VALUES ('XDK', 'XML Developer''s Kit',
        XMLTYPE('<DESCRIPTION><KEYWORD>xdk</KEYWORD> is a set of
        standards-based utilities that help to build XML
        applications.</DESCRIPTION>').createSchemaBasedXML
        ('http://xmlns.oracle.com/xml/content.xsd'));
CREATE TABLE product2 OF XMLType;

CREATE TABLE product2 OF XMLType
XMLSCHEMA "http://xmlns.oracle.com/xml/content.xsd"
ELEMENT "DESCRIPTION";

INSERT INTO product2 VALUES (
  XMLTYPE(
    bfilename('XMLDIR', 'productdata1.xml'),
    nls_charset_id('AL32UTF8')));
Default XML Schema-based DB Structure

- complexType → SQL type
  - Oracle is an object-relational RDBMS
  - Tables are created from objects
- An element/attribute defined by the complexType → SQL attribute of the SQL type
  - By default inlined
- 47 XML Schema data types → 19 scalar SQL data types
- Element with maxOccurs > 1 → collection attribute
  - By default VARRAY stored in a LOB
- XML Schema type/element/attribute names → SQL type/attribute names
Introducing Binary XML Storage

- Oracle 10g:
  - **Structured** (object-relational) – a set of objects (shredding)
  - **Unstructured** (text-based) – CLOB

- Oracle 11g adds:
  - **Binary** (native) – post-parse, binary format specifically designed for XML data

- **Hybrid** – mix of storage models for different parts of XML data
Binary Storage Models

- **Structured storage** = default
- **Unstructured storage**: STORE AS CLOB
- **Binary storage** + usage of XML schema:
  1. **Non-schema-based** – possibly existing schema is ignored
     - But validation can be made explicitly
  2. **Single XML schema** – all rows/columns must conform to the schema
  3. **Multiple XML schemas** – each row can reference own XML schema
     - Non-schema based documents can be allowed
- **CREATE TABLE options**:
  - STORE AS BINARY XML
  - STORE AS BINARY XML XMLSCHEMA ...
  - STORE AS BINARY XML XMLSCHEMA ... ALLOW NONSCHEMA
  - STORE AS BINARY XML ALLOW ANYSCHEMA
  - STORE AS BINARY XML ALLOW ANYSCHEMA ALLOW NONSCHEMA
Structured Storage

- By default:
  - Collections are mapped into SQL VARRAY values
  - Entire contents of such a VARRAY is serialized using a single LOB column

- Idea:
  - Optimal insertion and retrieval of entire document

- Limitations: indexing, updating, retrieval of individual members of the collection

- Overriding of default storage:
  - STORE AS clause of CREATE TABLE
  - Annotations of XML schema
<xs:schema
targetNamespace="http://xmlns.oracle.com/xdb/documentation/purchaseOrder"
xmlns:po="http://xmlns.oracle.com/xdb/documentation/purchaseOrder"
xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xs:element name="PurchaseOrder" type="po:PurchaseOrderType"/>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Reference" type="po:ReferenceType"/>
      <xs:element name="Actions" type="po:ActionsType"/>
      <xs:element name="Reject" type="po:RejectionType" minOccurs="0"/>
      <xs:element name="Requestor" type="po:RequestorType"/>
      <xs:element name="User" type="po:UserType"/>
      <xs:element name="CostCenter" type="po:CostCenterType"/>
      <xs:element name="ShippingInstructions" type="po:ShippingInstructionsType"/>
      <xs:element name="SpecialInstructions" type="po:SpecialInstructionsType"/>
      <xs:element name="LineItems" type="po:LineItemsType" maxOccurs="unbounded"/>
      <xs:element name="Notes" type="po:NotesType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LineItemsType">
    <xs:sequence>
      <xs:element name="LineItem" type="po:LineItemType" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
...</xs:schema>
CREATE TABLE purchaseorder_as_table
   OF XMLType
      (UNIQUE ("XMLDATA"."Reference"),
       FOREIGN KEY ("XMLDATA"."User")
                    REFERENCES hr.employees (email))
ELEMENT "http://xmlns.oracle.com/purchaseOrder.xsd#PurchaseOrder"
   VARRAY "XMLDATA"."Actions"."Action"
       STORE AS TABLE action_table1
          ((PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))
   VARRAY "XMLDATA"."LineItems"."LineItem"
       STORE AS TABLE lineitem_table1
          ((PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))
   LOB ("XMLDATA"."Notes")
       STORE AS (TABLESPACE USERS ENABLE STORAGE IN ROW
                  STORAGE(INITIAL 4K NEXT 32K));
CREATE TABLE purchaseorder_as_column (  
id NUMBER,  
xml_document XMLType,  
UNIQUE (xml_document."XMLDATA"."Reference"),  
FOREIGN KEY (xml_document."XMLDATA"."User")  
    REFERENCES hr.employees (email))  

XMLTYPE COLUMN xml_document  
XMLSCHEMA "http://xmlns.oracle.com/xdb/documentation/purchaseOrder.xsd"  
ELEMENT "PurchaseOrder"  

VARRAY xml_document."XMLDATA"."Actions"."Action"  
    STORE AS TABLE action_table2  
        ((PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))  
VARRAY xml_document."XMLDATA"."LineItems"."LineItem"  
    STORE AS TABLE lineitem_table2  
        ((PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))  
LOB (xml_document."XMLDATA"."Notes")  
    STORE AS (TABLESPACE USERS ENABLE STORAGE IN ROW  
        STORAGE(INITIAL 4K NEXT 32K));
XML Schema Annotations
XML Schema Annotations

- To control the mapping between XSD and storage methods
- Namespace: http://xmlns.oracle.com/xdb
  - Prefix: xdb
- Attribute types:
  - Default table – name and storage options of the default XMLType table
  - SQL names – SQL names for element in the schemas
  - SQL types – SQL data types for simple and complex XML Schema types
- Maintain DOM – whether to preserve DOM fidelity of an element
  - XML DB adds position descriptor for comments, PIs, sibling element order, … → increases storage overhead
- Storage options – for optimizing storage
Annotating Attributes (1)

- **Element** `schema`:
  - `xdb:storeVARRAYAsTable`
    - `true` = store collection elements (maxOccurs > 1) in nested object tables
    - `false` = collection is serialized as a VARRAY in a LOB column
  - `xdb:mapUnboundedStringToLob`
    - `true` = unbounded strings/binary data are stored to BLOB/CLOB
    - Default: false → VARCHAR2(4000) / RAW(2000)

- **Global complex types**:
  - `xdb:SQLType`
    - Name of the SQL type
      - To avoid XML DB-generated names for complex types
      - To change storage from object-relational to VARCHAR2, RAW, CLOB, BLOB
  - `xdb:maintainDOM`
    - `true` (default) = the complex type should maintain DOM fidelity
Annotating Attributes (2)

- **XML elements:**
  - **xdb:SQLName**
    - Specifies the name of the attribute within the SQL object that maps to this XML element
  - **xdb:SQLType**
    - Name of SQL type corresponding to the element
  - **xdb:SQLCollType**
    - The name of the SQL collection type
      - For elements with `maxOccurs > 1`
  - **xdb:maintainDOM**
Annotating Attributes (3)

- **xdb:SQLInline**
  - **true** (default) = embedded attribute
  - **false** = a REF value is stored (or a collection of REF values, if `maxOccurs > 1`)

- **xdb:maintainOrder**
  - **true** (default) = collection is mapped to VARRAY
  - **false** = collection is mapped to a nested table

- **xdb:defaultTable**
  - Name of the table into which XML instances are stored
    - A link from XML DB repository is created to this table

- **xdb:tableProps, xdb:columnProps**
  - Default table properties in SQL appended to the CREATE TABLE
<xs:complexType name="LineItemsType" xdb:SQLType="LINEITEMS_T">
    <xs:sequence>
        <xs:element name="LineItem" type="LineItemType" maxOccurs="unbounded"
            xdb:SQLName="LINEITEM" xdb:SQLCollType="LINEITEM_V"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="LineItemType" xdb:SQLType="LINEITEM_T">
    <xs:sequence>
        <xs:element name="Description" type="DescriptionType"
            xdb:SQLName="DESCRIPTION"/>
        <xs:element name="Part" type="PartType" xdb:SQLName="PART"/>
    </xs:sequence>
    <xs:attribute name="ItemNumber" type="xs:integer"
        xdb:SQLName="ITEMNUMBER" xdb:SQLType="NUMBER"/>
</xs:complexType>

<xs:complexType name="PartType" xdb:SQLType="PART_T">
    <xs:attribute name="Id" xdb:SQLName="PART_NUMBER" xdb:SQLType="VARCHAR2">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="10"/>
                <xs:maxLength value="14"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="Quantity" type="moneyType" xdb:SQLName="QUANTITY"/>
    <xs:attribute name="UnitPrice" type="quantityType" xdb:SQLName="UNITPRICE"/>
</xs:complexType>
Too many options → the most common strategy:

- Specify the name of the default table
  - defaultTable
- Specify the name and data type for each element and data type
  - SQLName, SQLType, SQLCollType

Note that:

- SQLName – SQL names for XML elements
- SQLCollName – SQL names for elements with maxOccurs > 1
- SQLType – SQL names for all simple types or complex types that do not use default mapping
XML Schema Annotations

- **Recommendations:**
  - Avoid preserving DOM fidelity
  - Store subtrees/complex types as CLOB when no XPath queries are required
    - `xdb:SQLType="CLOB"`
  - Store large unbounded XML elements using nested tables
    - `xdb:storeVARRAYAsTable="true"

- **Important:**
  - SQL is case-insensitive, but names in SQL code are implicitly uppercase, unless you enclose them in double-quotes
  - XML is case-sensitive – you must refer to SQL names in XML code using the correct case: uppercase SQL names must be written as uppercase
XML Storage Comparison
How to Store XML Data?

- XMLType or Relational tables?
  - Data-centric documents: structured
    - Efficient retrieval
    - No need to preserve XML format
    - If necessary: XMLType Views
  - Document-centric documents: XMLType
  - Hybrid documents: ?
- XMLType schema based storage?
  - No (CLOB): DTD based documents, changing XSDs
  - Yes: intensive data retrieval/updates
- XMLType table or column?
  - Column: We need to store relational data along with XML documents
    - Example: create time, create owner, ...
<table>
<thead>
<tr>
<th>Quality</th>
<th>Structured</th>
<th>Unstructured</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>– XML decomposition can result in reduced throughput when ingesting retrieving the entire content of an XML document.</td>
<td>+ High throughput when ingesting and retrieving the entire content of an XML document.</td>
<td>++ High throughput. Fast DOM loading.</td>
</tr>
<tr>
<td>Space efficiency (disk)</td>
<td>++ Extremely space-efficient.</td>
<td>– Consumes the most disk space, due to insignificant whitespace and repeated tags.</td>
<td>+ Space-efficient.</td>
</tr>
<tr>
<td>Data flexibility</td>
<td>– Limited flexibility. Only documents that conform to the XML schema can be stored in the XMLType table or column.</td>
<td>+ Flexibility in the structure of the XML documents that can be stored in an XMLType column or table.</td>
<td>+ Flexibility in the structure of the XML documents that can be stored in an XMLType column or table.</td>
</tr>
<tr>
<td>XML schema flexibility</td>
<td>– Relatively inflexible. Data and metadata are stored separately. Cannot use multiple XML schemas for the same XMLType table.</td>
<td>+ Flexible. Data and metadata are stored together. Cannot use multiple XML schemas for the same XMLType table.</td>
<td>++ Flexible. Can store data and metadata together or separately. Can use multiple XML schemas for the same XMLType table.</td>
</tr>
<tr>
<td>Update operations (DML)</td>
<td>++ In-place, piecewise update.</td>
<td>– When any part of the document is updated, the entire document must be written back to disk.</td>
<td>+ In-place, piecewise update for SecureFile LOB storage.</td>
</tr>
<tr>
<td>Quality</td>
<td>Structured</td>
<td>Unstructured</td>
<td>Binary</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>XML fidelity</td>
<td>– DOM fidelity: A DOM created from an XML document that has been stored in the database will be identical to a DOM created from the original document. However, insignificant whitespace may be discarded.</td>
<td>+ Document fidelity: Maintains the original XML data, byte for byte. In particular, all original whitespace is preserved.</td>
<td>– DOM fidelity (see structured storage description).</td>
</tr>
<tr>
<td>XPath-based queries</td>
<td>++ XPath operations can often be evaluated using XPath rewrite, leading to significantly improved performance, particularly with large collections of documents.</td>
<td>– XPath operations are evaluated by constructing a DOM from the CLOB data and using functional evaluation. Expensive when performing operations on large documents or large collections of documents. XMLIndex indexing can improve performance of XPath-based queries.</td>
<td>+ Streaming XPath evaluation avoids DOM construction and allows evaluation of multiple XPath expressions in a single pass. Navigational XPath evaluation is significantly faster than with unstructured storage. XMLIndex indexing can improve performance of XPath-based queries.</td>
</tr>
<tr>
<td>SQL constraint support</td>
<td>+ SQL constraints are supported.</td>
<td>– SQL constraints are not available.</td>
<td>+ SQL constraints are supported.</td>
</tr>
<tr>
<td>Quality</td>
<td>Structured</td>
<td>Unstructured</td>
<td>Binary</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------</td>
<td>---------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Support for SQL scalar data types</td>
<td>+ Yes</td>
<td>– No</td>
<td>+ Yes</td>
</tr>
<tr>
<td>Indexing support</td>
<td>B-tree, Oracle Text, and function-based indexes.</td>
<td>XMLIndex, function-based, and Oracle Text indexes.</td>
<td>XMLIndex, function-based, and Oracle Text indexes.</td>
</tr>
<tr>
<td>Optimized memory management</td>
<td>+ XML operations can be optimized to reduce memory requirements.</td>
<td>– XML operations on the document require creating a DOM from the document.</td>
<td>+ XML operations can be optimized to reduce memory requirements.</td>
</tr>
<tr>
<td>Validation upon insert</td>
<td>XML data is partially validated when it is inserted.</td>
<td>XML schema-based data is partially validated when it is inserted.</td>
<td>+ XML schema-based data is fully validated when it is inserted.</td>
</tr>
</tbody>
</table>
Generating XML from SQL Data
SQL/XML in Oracle (1)

- **XMLELEMENT**
  - Returns an XML element in an XMLType when given:
    - XML element name
    - Optional list of XML attributes (**XMLATTRIBUTES**)
    - Optional list of values as the content of the new element
      - Other XML elements or
      - XML fragments (**XMLFOREST**)

- **XMLATTRIBUTES**
  - Used within **XMLELEMENT()** to specify attributes for the element.
CREATE TABLE employees (    employee_id VARCHAR(10),    first_name VARCHAR2(100),    last_name VARCHAR2(100),    job_id VARCHAR2(100),    hire_date DATE,    salary NUMBER,    email VARCHAR2(100),    phone_number VARCHAR2(100),    department_id VARCHAR(10));

SELECT
    XMLELEMENT("Employee",
        XMLATTRIBUTES
            (employee_id AS "empno",
             job_id AS "job"),
        XMLELEMENT("Name",
                first_name || ' ' || last_name),
        'is hired on ',
        hire_date) AS result
FROM employees WHERE rownum=1;

<Employee empno="100" job="AD_PRES">
    <Name>Steven King</Name>is hired on 17-JUN-87
</Employee>
SQL/XML in Oracle (2)

- **XMLFOREST**
  - Returns an XML fragment in an XMLType when given a list of named expressions for the XML elements
  - Each expression specifies the name of an XML element and its content

- **XMLCONCAT**
  - Returns an XML fragment in an XMLType by concatenating a list of XML elements/values

- **XMLAGG**
  - Returns an XML fragment in an XMLType by aggregating XML fragments, with the option of XML element sorting
XML Data Retrieval
XMLType
Member Functions (1)

- **XMLType, createXML, createSchemaBasedXML, createNonSchemaBasedXML**
  - Create XMLTypes from XML data stored in VARCHAR2, CLOB, or other XMLTypes

- **getClobVal, getBlobVal, getNumberVal, getStringVal**
  - Gets the CLOB, BLOB, NUMBER, or String value in VARCHAR2 from the XMLType
  - `getNumberVal` can be used only when the content of XMLType is numeric

- **transform**
  - Transforms the XML content in XMLType with the XSL stylesheet specified
CREATE TABLE xml_table OF XMLType;
CREATE TABLE table_with_xml_column
  (filename VARCHAR2(64), xml_document XMLType);

INSERT INTO xml_table
  VALUES (XMLType(bfilename('XMLDIR', 'purchaseOrder.xml'),
                   nls_charset_id('AL32UTF8')));
INSERT INTO table_with_xml_column (filename, xml_document)
  VALUES ('purchaseOrder.xml',
           XMLType(bfilename('XMLDIR', 'purchaseOrder.xml'),
                   nls_charset_id('AL32UTF8')));

SELECT x.OBJECT_VALUE.getCLOBVal()
  FROM xml_table x;
SELECT x.xml_document.getCLOBVal()
  FROM table_with_xml_column x;
XMLType
Member Functions (2)

- **isFragment**
  - Checks if the XMLType is an XML document fragment or a well-formed document

- **isSchemaBased, getSchemaURL, getRootElement, getNamespace**
  - Checks the XML schema–related information of XMLType and returns respective information if exists

- **isSchemaValidated, isSchemaValid, schemaValidation, setSchemaValidated**
  - Checks and updates the XML Schema validation status of XMLType
Oracle SQL Extensions

- **existsNode**
  - Checks if the XML nodes or node sets specified by an XPath expression exist

- **extract**
  - Extracts nodes or node sets based on the XPath expression and returns an XMLType instance containing the resulting node(s)

- **extractValue**
  - Returns scalar content, such as numbers or strings, when passed an XPath expression pointing to an XML element with only a single text child
SQL/XML
XMLTable, XMLQuery

- **XMLQuery**
  - XQuery-expression evaluation

- **XMLTable**
  - Shreds the result of an XQuery-expression evaluation into the relational rows and columns of a new, virtual table
  - Further inserting, SQL querying, ...
SELECT warehouse_name,
    XMLQuery(
        'for $i in /Warehouse
            where $i/Area > 80000
            return <Details>
                <Docks num="{$i/Docks}"/>
                <Rail>{if ($i/RailAccess = "Y")
                    then "true" else "false"}
            </Details>'
    )
FROM warehouses;
SELECT xtab.poref, xtab.priority, xtab.contact
FROM purchaseorder,
XMLTable('for $i in /PurchaseOrder
    let $spl := $i/SpecialInstructions
    where $i/CostCenter eq "A10"
    return <PO>
        <Ref>{$i/Reference}</Ref>
        {if ($spl eq "Next Day Air" or
            $spl eq "Expedite") then
            <Type>Fastest</Type>
        else if ($spl eq "Air Mail") then
            <Type>Fast</Type>
        else ()
        }<Name>{$i/Requestor}</Name>
    </PO>'
PASSING OBJECT_VALUE
COLUMNS poref    VARCHAR2(20) PATH '/PO/Ref',
    priority VARCHAR2(8)  PATH '/PO/Type'
    DEFAULT 'Regular',
    contact  VARCHAR2(20) PATH '/PO/Name')
xtab;
## XMLTable

<table>
<thead>
<tr>
<th>POREF</th>
<th>PRIORITY</th>
<th>CONTACT</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKING-20021009123336</td>
<td>Fastest</td>
<td>Steven A. King</td>
</tr>
<tr>
<td>SMCCAIN-200210091233</td>
<td>Regular</td>
<td>Samuel B. McCain</td>
</tr>
<tr>
<td>SMCCAIN-200210091233</td>
<td>Fastest</td>
<td>Samuel B. McCain</td>
</tr>
</tbody>
</table>

- **The result without **COLUMNS**: Table with single column containing the resulting XML fragments
Oracle XQuery Extension
ora:contains

ora:contains (input_text, text_query
 [, policy_name] [, policy_owner])

- Returns:
  - Positive integer when the input_text matches text_query
    - The higher the number, the more relevant the match
  - Zero otherwise

- Can be used in:
  - XPath expression inside XQuery expression
  - SQL functions existsNode, extract, extractValue

- input_text – single text node/attribute
SELECT ID
    FROM PURCHASE_ORDERS_xmltype
WHERE existsNode(DOC,
    '/purchaseOrder/comment[
        ora:contains(text(),
            "($lawns AND wild) OR flamingo") > 0",
    'xmlns:ora="http://xmlns.oracle.com/xdb"
    ) = 1 ;

- Full-text operators
  - e.g., $ = all words with the same linguistic stem, i.e., lawn or lawns
Oracle XQuery Extension
ora:matches

ora:matches (target_string, match_pattern [, match_parameter])

- Returns:
  - **true** – if target_string argument matches regular-expression match_pattern argument
  - **false** – otherwise
- Optional match_parameter is a code that qualifies matching
  - Case-sensitivity etc.
Oracle XQuery Extension
ora:replace

ora:replace (target_string, match_pattern, replace_string [, , match_parameter])

- Each occurrence in target_string that matches match_pattern is replaced by replace_string
- Returns: new string that results from the replacement
- Optional match_parameter is a code that qualifies matching
  - Case-sensitivity etc.
Oracle XQuery Extension
ora:view

- Creates XML views over the relational data, on the fly
- Parameters:
  - db-schema – optional database schema
  - db-table – database table or view
- Returns: unordered sequence of document nodes, one for each row of db-table
- SQL/XML standard is used:
  - Relational column names become XML element names
  - Column elements are wrapped together in a ROW element
**ora:view Example**

```sql
SELECT XMLQuery(
    'for $i in ora:view("REGIONS"), $j in ora:view("COUNTRIES")
        where $i/ROW/REGION_ID = $j/ROW/REGION_ID and
        $i/ROW/REGION_NAME = "Asia"
        return $j'
    RETURNING CONTENT) AS asian_countries
FROM DUAL;
```

**ASIAN_COUNTRIES**

```
<ROW>
  <COUNTRY_ID>AU</COUNTRY_ID>
  <COUNTRY_NAME>Australia</COUNTRY_NAME>
  <REGION_ID>3</REGION_ID>
</ROW>
<ROW>
  <COUNTRY_ID>CN</COUNTRY_ID>
  <COUNTRY_NAME>China</COUNTRY_NAME>
  <REGION_ID>3</REGION_ID>
</ROW>
```
XML Data Updates
Oracle SQL Extensions

- No support for standard **XQuery Update Facility**
- `updateXML`
  - Replaces XML nodes of any kind
- `insertChildXML`
  - Inserts XML element or attribute nodes as children of a given element node
- `insertXMLbefore`
  - Insert XML nodes of any kind immediately before a given node
- `appendChildXML`
  - Inserts XML nodes of any kind as the last child nodes of a given element node
- `deleteXML`
  - Deletes XML nodes of any kind
updateXML

- Accepts:
  - XMLType instance
  - Set of XPath expression – string value pairs
- Updates XPath-referred elements or attributes with the provided values

```
SELECT updateXML (column_name, 'XPath1', 'text1',
  ..., 'XPathN', 'textN',
  'Namespace1 NamespaceN')
FROM table_name;
```
SELECT `updateXML`(`XMLType(
  '<Employee xmlns:app1="www.example.com/ns1"
    xmlns:app2="www.example.com/ns2">
    <Name app1:type="Customer">Janet Jones</Name>
    <Job app2:type="IT">Manager</Job>
    <Salary app2:type="Hidden">12000</Salary>
    <Commission app2:type="Hidden">3400</Commission>
  </Employee>
)`,
'/Employee/Name/text()', 'Janet Lee',
'/Employee/Name/@app1:type', 'Important Customer',
'/Employee/Job/@app2:type', 'Hidden',
'/Employee//*[@app2:type="Hidden"]', null,
'xmlns:app1="www.example.com/ns1"
  xmlns:app2="www.example.com/ns2"') AS result
FROM dual;

<Employee xmlns:app1="www.example.com/ns1"
  xmlns:app2="www.example.com/ns2">
  <Name app1:type="Important Customer">Janet Lee</Name>
  <Job/>
  <Salary/>
  <Commission/>
</Employee>
UPDATE purchaseorder
SET OBJECT_VALUE =
insertChildXML(OBJECT_VALUE,
'/PurchaseOrder/LineItems',
'LineItem',
XMLType(
'&lt;LineItem ItemNumber="222">'\n &lt;Description>The Harder They Come&lt;/Description&gt;
 &lt;Part Id="953562951413"
  UnitPrice="22.95"
  Quantity="1"/&gt;
 &lt;/LineItem&gt;'))
WHERE existsNode(OBJECT_VALUE,
'/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]') = 1;
UPDATE purchaseorder
SET OBJECT_VALUE =
    insertXMLbefore(OBJECT_VALUE,
        '/PurchaseOrder/LineItems/LineItem[1]',
        XMLType('<LineItem ItemNumber="314">
            <Description>Brazil</Description>
            <Part Id="314159265359"
                UnitPrice="69.95"
                Quantity="2"/>
        </LineItem>'))
WHERE existsNode(OBJECT_VALUE,
        '/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]') = 1;
UPDATE purchaseorder
SET OBJECT_VALUE =
    deleteXML(OBJECT_VALUE,
        '/PurchaseOrder/LineItems/LineItem[@ItemNumber="222"]')
WHERE existsNode(OBJECT_VALUE,
    '/PurchaseOrder[Reference="AMCEWEN-20021009123336171PDT"]')
   = 1;
Indexing XMLType Data
Indexing XMLType Data

● Idea: improvement of performance of often and expensive queries

● Storage types:
  ● Structured: B-tree indexes on columns and tables
  ● Unstructured and hybrid: XMLIndex
    ● For CLOB parts
  ● Function-based indexes:
    ● XMLQuery, XMLExists, XMLCast, extract, extractValue, existsNode
  ● Oracle Text Indexes: ora:contains
    ● full-text
Function-based Index

- On an XMLType table or column
  - XML schema-based or not
  - Storage structured, unstructured, or binary XML
- Function-based index: Created by evaluating the specified function for each row in the target table or column and storing the value in the index
  - B-tree index or bitmap index

CREATE TABLE po_clob OF XMLType
XMLTYPE STORE AS CLOB
ELEMENT
  "http://localhost:8080/purchaseOrder.xsd#PurchaseOrder";

CREATE UNIQUE INDEX po_fn_based_ix ON po_clob
(extractValue(OBJECT_VALUE, '/PurchaseOrder/Reference'));
Function-based Index

- Structured storage:
  - Element/attribute being targeted by the function must be a singleton
  - Not a collection
  - If the function being indexed is extractValue, Oracle tries to rewrite it to not function-based index

```sql
CREATE INDEX po_fn_based_ix ON purchaseorder
(extractValue(OBJECT_VALUE, '/PurchaseOrder/Reference'));
```

```sql
CREATE INDEX po_fn_based_ix ON purchaseorder p
(p."XMLDATA"."REFERENCE");
```
Components of an XMLIndex

- **Path index** – indexes the XML tags of a document and identifies its various document fragments
- **Order index** – indexes the hierarchical positions of the nodes in an XML document
  - parent–child, ancestor–descendant, sibling relations
- **Value index** – indexes values of an XML document
- **Not indexed parts:**
  - Applications of XPath functions
    - Except ora:contains
  - Other axes
  - Expressions using the union operator
Path Table

- Implements XMLIndex
- Columns:
  - **PATHID** – unique identifier for the XPath path to the node
  - **RID** – rowid of the table used to store the XML data
  - **ORDER_KEY** – decimal order key that identifies the hierarchical position of the node
  - **LOCATOR** – Fragment-location information. Used for fragment extraction
  - **VALUE** – text of attribute/simple element node
# Path Table

<table>
<thead>
<tr>
<th>PATHID</th>
<th>Indexed XPath</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/PurchaseOrder</td>
</tr>
<tr>
<td>2</td>
<td>/PurchaseOrder/Reference</td>
</tr>
<tr>
<td>3</td>
<td>/PurchaseOrder/Actions</td>
</tr>
<tr>
<td>4</td>
<td>/PurchaseOrder/Actions/Action</td>
</tr>
<tr>
<td>5</td>
<td>/PurchaseOrder/Actions/Action/User</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PATHID</th>
<th>RID</th>
<th>ORDER_KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R1</td>
<td>1.1</td>
<td>SBELL-2002100912333601PDT</td>
</tr>
<tr>
<td>3</td>
<td>R1</td>
<td>1.2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>R1</td>
<td>1.2.1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>R1</td>
<td>1.2.1.1</td>
<td>SVOLLMAN</td>
</tr>
<tr>
<td>1</td>
<td>R2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>R2</td>
<td>1.1</td>
<td>ABEL-20021127121040897PST</td>
</tr>
</tbody>
</table>
CREATE INDEX po_xmlindex_ix ON po_clob (OBJECT_VALUE)
INDEXTYPE IS XDB.XMLIndex;

CREATE INDEX po_xmlindex_hybrid_ix ON li_clob
(extract(OBJECT_VALUE, '/PurchaseOrder/LineItems'))
INDEXTYPE IS XDB.XMLIndex;

DROP INDEX po_xmlindex_hybrid_ix;
Evolution of XML Applications
Evolution of XML Applications

- **XML application** – an application processing XML data
  - Well-formed, valid against a schema

**Problem:**
- Applications are usually dynamic, user requirements change, new information come, ...
- Structure of data needs to be changed
  - Old schema $S_{old}$, new schema $S_{new}$
- We still want to work with both old and new data
  - Without any loss if possible
Evolution of XML Applications

- Approaches:
  - XML schema/data evolution
    - The old data valid against $S_{\text{old}}$ are transformed to be valid against $S_{\text{new}}$
    - User poses queries over $S_{\text{new}}$
  - XML data versioning
    - We must preserve all versions $V_1, V_2, \ldots, V_n$ of the data
    - User poses queries over any $V_i; 1 \leq i \leq n$
      - Retrieves data from all versions $V_1, V_2, \ldots, V_n$
      - Retrieves data from $V_1, V_2, \ldots, V_i$
Oracle Types of Schema Evolution

- **Copy-based:**
  - All documents that conform to $S_{old}$ are copied to a temporary location
  - $S_{old}$ is deleted
  - $S_{new}$ is registered
  - Instance documents are inserted into their new locations from the temporary area
    - Before insertion the data are transformed to conform new schema

- **In-place:**
  - Does not require copying, deleting, inserting existing data
  - Much faster, but restricted:
    - Not changing the storage model
    - Changes do not invalidate existing documents
      - Existing documents are/can be made conformant with the new schema
Copy-Based Schema Evolution

- Existing XML instance documents need to be revalidated
- PL/SQL function `DBMS_XMLSCHEMA.copyEvolve`
  - Copies existing instance documents to temporary XMLType tables
  - Drops $S_{old}$ and deletes the associated instance documents
  - Registers $S_{new}$
  - Transforms and copies the backed-up instance documents to new XMLType tables
procedure copyEvolve(
schemaURLs IN XDB$STRING_LIST_T,
newSchemas IN XMLSequenceType,
transforms IN XMLSequenceType := NULL,
preserveOldDocs IN BOOLEAN := false,
mapTabName IN VARCHAR2 := NULL,
generateTables IN BOOLEAN := true,
force IN BOOLEAN := false,
schemaOwners IN XDB$STRING_LIST_T := NULL
parallelDegree IN PLS_INTEGER := 0,
options IN PLS_INTEGER := 0);
DBMS_XMLSCHEMA.copyEvolve

- **schemaURLs** – VARRAY of URLs of XML schemas to be evolved
  - VARCHAR2(4000)
- **newSchemas** – VARRAY of new XML schema documents
  - XMLType instances
  - The same order as corresponding URLs
- **transforms** – VARRAY of XSL documents
  - XMLType instances
  - Applied to XML schema based documents to make them conform to the new schemas
  - The same order as the corresponding URLs
- **preserveOldDocs**
  - true = the temporary tables holding old data are not dropped
- **mapTabName** – name of table that maps old XMLType table or column names to names of corresponding temporary tables
DBMS_XMLSCHEMA.copyEvolve

- **generateTables**
  - false = XMLType tables or columns will not be generated after registering new schemas
    - preserveOldDocs must be true and mapTabName must not be NULL
- **force**
  - true = errors during the registration of new schemas are ignored
    - e.g., circular dependencies
- **schemaOwners** – VARRAY of names of schema owners
- **parallelDegree** – degree of parallelism to be used during the data-copy stage
- **options** – miscellaneous options
  - COPYEVOLVE_BINARY_XML – register the new XML schemas for binary XML data and create the new tables or columns with binary XML as the storage model
BEGIN
    DBMS_XDB.createResource(
        '/schemas/revisedPurchaseOrder.xsd',
        bfilename('XMLDIR', 'revisedPurchaseOrder.xsd'),
        nls_charset_id('AL32UTF8'));
    DBMS_XDB.createResource(
        '/schemas/evolvePurchaseOrder.xsl',
        bfilename('XMLDIR', 'evolvePurchaseOrder.xsl'),
        nls_charset_id('AL32UTF8'));
    DBMS_XMLSCHEMA.copyEvolve(
        xdb$string_list_t('http://localhost:8080/schemas/purchaseOrder.xsd'),
        XMLSequenceType(XDBURIType('/schemas/revisedPurchaseOrder.xsd').getXML()),
        XMLSequenceType(XDBURIType('/schemas/evolvePurchaseOrder.xsl').getXML()));
END;

- Indexes, triggers, constraints related to the XMLType tables that are dependent on the schemas are not preserved
In-Place XML Schema Evolution

- Changes to an XML schema without requiring that existing data be copied, deleted, and reinserted
- \texttt{DBMS_XMLSHEMA.inPlaceEvolve}
  - Constructs $S_{\text{new}}$ by applying changes specified in a diffXML document
  - Validates $S_{\text{new}}$
  - Constructs DDL statements to evolve the disk structures used to store the XML instance documents associated with $S_{\text{new}}$
  - Executes these DDL statements
  - Replaces $S_{\text{old}}$ with $S_{\text{new}}$
- Backward compatibility
  - Any possible instance document that would validate against $S_{\text{old}}$ must also validate against $S_{\text{new}}$
Supported Operations

- Add an optional element to a complex type or group
- Add an optional attribute to a complex type or attribute group
- Convert an element from simple type to complex type with simple content
  - Supported only if the storage model is binary XML
- Modify the value attribute of an existing `maxLength` element
  - The value can only be increased, not decreased
- Add an enumeration value
- Add a global element
- Add a global attribute
Supported Operations

- Add or delete a global complex type
- Add or delete a global simple type
- Change the `minOccurs` attribute value
  - The value of `minOccurs` can only be decreased
- Change the `maxOccurs` attribute value
  - The value of `maxOccurs` can only be increased, and only for data stored as binary XML
- Add or delete a global `group` or `attributeGroup`
- Change the `xdb:defaultTable` attribute value
- Add, modify, or delete a comment or processing instruction
procedure inPlaceEvolve(scalaURL IN VARCHAR2,
                        diffXML IN XMLType,
                        flags IN NUMBER);

- `schemaURL` – URL of the XML schema to be evolved
  - VARCHAR2
- `diffXML` – XML document that conforms to the xdiff XML schema
  - XMLType instance
- `flags` – bit mask that controls the behavior of the procedure
  - INPLACE_EVOLVE – perform in-place XML schema evolution
  - INPLACE_TRACE – perform all steps necessary for in-place evolution, except executing the DDL statements and overwriting the old XML schema with the new, then write both the DDL statements and the new XML schema to a trace file
diffXML Parameter

- XML document that specifies the changes to be applied to an XML schema for in-place evolution
- Sequence of operations that describe the changes between $S_{\text{old}}$ with $S_{\text{new}}$.
- The changes are applied in order
- Can be created using:
  - The XMLDiff JavaBean
  - The xmldiff command-line utility
  - SQL function XMLDiff
<?xml version="1.0" encoding="UTF-8"?>
<xd:xdiff xsi:schemaLocation="http://xmlns.oracle.com/xdb/xdiff.xsd
xmlns:xd="http://xmlns.oracle.com/xdb/xdiff.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:oraxdfns_0="http://booklist.oracle.com">
  <?oracle-xmldiff operations-in-docorder="true" output-model="snapshot"
diff-algorithm="global"?>
  <xd:delete-node xd:node-type="element" xd:xpath="/oraxdfns_0:booklist[1]/oraxdfns_0:book[2]"/>
  <xd:update-node xd:node-type="attribute"
    xd:parent-xpath="/oraxdfns_0:booklist[1]/oraxdfns_0:book[3]/oraxdfns_0:title[1]"
    xd:attr-local="country">
    <xd:content>US</xd:content>
  </xd:update-node>
  <xd:append-node xd:node-type="element"
    xd:parent-xpath="/oraxdfns_0:booklist[1]/oraxdfns_0:book[4]"/>
    <xd:content>
      <oraxdfns_0:description>This is a classic</oraxdfns_0:description>
    </xd:content>
  </xd:append-node>
  <xd:insert-node-before xd:node-type="element"
    xd:xpath="/oraxdfns_0:booklist[1]/oraxdfns_0:book[5]/oraxdfns_0:author[1]"/>
    <xd:content>
      <oraxdfns_0:edition>Hardcover</oraxdfns_0:edition>
    </xd:content>
  </xd:insert-node-before>
  <xd:update-node xd:node-type="text"
    xd:xpath="/oraxdfns_0:booklist[1]/oraxdfns_0:book[5]/oraxdfns_0:price[1]/text()[1]"/>
    <xd:content>12.99</xd:content>
  </xd:update-node>
</xd:xdiff>
Creating `diffXML`

```sql
SELECT XMLDIFF(
XMLTYPE('<?xml version="1.0"?>
  <bk:tr>
    <bk:td>
      <bk:chapter>Chapter 1.</bk:chapter>
    </bk:td>
    <bk:td>
      <bk:chapter>Chapter 2.</bk:chapter>
    </bk:td>
  </bk:tr>
</bk:book>'),
XMLTYPE('<?xml version="1.0"?>
  <bk:tr>
    <bk:td>
      <bk:chapter>Chapter 1.</bk:chapter>
    </bk:td>
    <bk:td/>
  </bk:tr>
</bk:book>'))
FROM DUAL;
```
Creating diffXML

```xml
xmns:xd="http://xmlns.oracle.com/xdb/xdiff.xsd"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmns:bk="http://nosuchsite.com">
  <?oracle-xmldiff operations-in-docorder="true" output-model="snapshot"
diff-algorithm="global"?>
  <xd:delete-node xd:node-type="element"
</xd:xdiff>
```
References

  [http://books.google.cz/books?id=cHt2YN5_JI8C](http://books.google.cz/books?id=cHt2YN5_JI8C)
- Oracle XML DB Developer's Guide 10g Release 2:
  [http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14259/toc.htm](http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14259/toc.htm)
- Oracle XML DB Developer's Guide 11g Release 1:
  [http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28369/toc.htm](http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28369/toc.htm)
- Oracle XML DB Developer's Guide 11g Release 2:
  [http://download.oracle.com/docs/cd/E11882_01/appdev.112/e10492/toc.htm](http://download.oracle.com/docs/cd/E11882_01/appdev.112/e10492/toc.htm)