XQuery,
XML and databases

Patryk Czarnik

XML and Applications 2014/2015
XML support in databases – categorisation

- Classic (usually relational) database with XML support
  - logical structure – relations and references
  - additional XML-related features
  - used for application integration or storing XML data as part of larger data structures

- ‘Native' XML database
  - logical structure – collection of XML document trees
  - XQuery (or XPath) as native query language
  - natural XML-related features
  - used for storing XML data (or structural data easily mapped to XML tree)
XML support in relational databases

- Possible functionalities
  - import and export of data in XML format
  - special treatment of XML data stored in fields
  - XML validation as part of integrity constraints checking
  - XPath or XQuery for querying field contents
  - XSLT applied to query result

- How to store XML data?
  - whole document (fragment) stored in single field
  - split into prima factors
    - each XML node in separate field
    - tables structure reflects tree structure of XML
Example – XML support in Oracle database

http://www.oracle.com/xml

- Since Oracle 8i
  - details differ from version to version
- XML parsers
  - for database programming (PL/SQL)
  - or middleware programming (Java, C++)
- XML-SQL Utility
  - XML data import and export
- XMLType data type and XML-specific operations
XML-SQL Utility

- `getXML()` function – XML data export

```sql
SELECT xmlgen.getXML('select * from emp') FROM dual;
```

```xml
<rowset>
  <row id="1">
    <empno>10</empno>
    <name>Scott Tiger</name>
    <title>specialist</title>
  </row>
  ...
</rowset>
```
XML in Oracle DB – XMLType

- **XMLType** - special datatype:
  - to be stored as LOB or used for columns, variables, etc.
  - indexing XML content
  - XPath expressions
  - validation against XML Schema
  - XSLT

- Available functions:
  - extract, extractValue, existsNode, transform, updateXML, XMLSequence
XMLType applications – some examples

CREATE TABLE warehouses(
    warehouse_id NUMBER(4),
    warehouse_spec XMLTYPE,
    warehouse_name VARCHAR2(35),
    location_id NUMBER(4));

CREATE TABLE po_xtab of XMLType;

UPDATE po_xml_tab
SET poDoc = UPDATEXML(poDoc,
    '/PO/CUSTNAME/text()', 'John');

INSERT INTO warehouses VALUES
(100, XMLType(
    '<Warehouse whNo="100">
        <Building>Owned</Building>
    </Warehouse>'), 'Tower Records', 1003);

SELECT e.poDoc.getClobval() AS poXML
    FROM po_xml_tab e
    WHERE e.poDoc.existsNode('/PO[PNAME = "po_2"]') = 1;

CREATE INDEX city_index ON po_xml_tab
    (poDoc.extract('//PONO/text()').getNumberVal());
XML support in database engines

  new data type **XML:**
  - only well-formed XML documents allowed
  - parsing and serialisation
  - implementation may add XML-specific operations

- Substantial support
  - IBM DB2 (since v.9 – *pureXML*)
  - Oracle (since 8i)
  - Microsoft SQL Server (since v.2000)
  - Sybase ASE (since v.12)

- Minimal support
  - MySQL – XPath queries over text fields containing XML
  - PostgreSQL – as above plus XML datatype but with no special operations
Native XML database

- Logical layer
  - XML document as basic data entity
  - collections of documents build a database
  - XML schema (or equivalent) as structure definition
  - XQuery (or XPath) as “native” query language

- Physical layer – not necessarily “files with XML text“

- More than just a collection XML files:
  - transactions and concurrent access
  - security (access privileges etc.), versioning, replication, ...
  - API for data access and update
  - additional means of data access
    - e.g. REST-compliant HTTP server
  - indexing for efficient access to selected nodes
Standards for XML databases

- High level query languages:
  - XQuery – primary language for queries
    - versions 1.0 and 3.0 in use
  - XQL – former approach to make XML query language
  - XPath – poor stub for XQuery

- High level update languages:
  - XQuery Update Extension
  - XUpdate

- Programmer APIs
  depend additionally on programming language
  - XML Database API (XAPI)
  - XQJ (for Java, expected to become XML equivalent of JDBC)
  - vendor-specific APIs...
XQuery – the query language for XML

- **Status**
  - XQuery 1.0 – W3C Recommendation, I 2007
  - XQuery 3.0 – W3C Recommendation, IV 2014
  - Data model, functions and ops – shared with XPath 2.0 / 3.0
  - Syntax defined in a separate document
    - In practice: extension of XPath

- **Main features:**
  - Picking up data from XML documents
  - Constructing new result nodes
  - Sorting, grouping
  - Defining custom functions
  - Various output methods (XML, HTML, XHTML, text)
    - shared with XSLT
(Unexpectedly) XQuery is not an XML application
- There exists a verbose XML syntax for XQuery, not intended to be written by hand

Typical file extensions: .xquery, .xq, .xqm (for modules)

Text format, header and body
- header optional in normal queries
- units declared as *modules* do not have body

Minimalistic example
2 + 2
XQuery headers

- Header part consists of declarations:
  - version declaration
  - import
  - flags and options
  - namespace declaration
  - global variable or query parameter
  - function

Example

xquery version "1.0" encoding "utf-8";
declare namespace foo = "http://example.org";
declare variable $id as xs:string external;
declare variable $doc := doc("example.xml");
$doc//foo:object[@id = $id]
FLWOR expression

- **For**, **Let**, **Where**, **Order by**, **Return**
  - More clauses in XQuery 3.0, but the acronym remains.
- Replaces **for** from XPath
- Obvious influence of SQL SELECT

**Example**

```xml
for $obj in doc("example.xml")/list/object
let $prev := $obj/preceding-sibling::element()
let $prev-name := $prev[1]/@name
where $obj/@name
order by $obj/@name
return
  <div class="result">
    Object named {xs:string($obj/@name)}
    has count({$prev}) predecessors.
    The nearest predecessor name is
    {xs:string($prev-name)}.
  </div>
```
Node constructors – direct

- **XML document fragment within query**

```xml
for $el in doc("example.xml")//* return
  <p style="color: blue">I have found an element.
  <?pi bla Bla ?>
  <!-- Comments and PIs also taken to result -->!
</p>
```

- **Expressions nested within constructors – braces**

```xml
<result> {
for $el in doc("example.xml")//* return
  <elem depth="{count($el/ancestor::node())}">
    {name($el)}
  </elem>
} </result>
```
Node constructors – computed

The same example again to show the syntax

```
for $el in doc("example.xml")//* return
element p {
    attribute style {"color: blue"},
    text { "I have found an element."},
    processing-instruction pi { "bla Bla" }
    comment { "Comments and PIs also taken to result" }
}
```

Application example – dynamically computed name

```
<result> {
    for $el in doc("example.xml")//* return
        element {concat("elem-", name($el))} {
            attribute depth {count($el/ancestor::node())},
            text {name($el)}
        }
} </result>
```
Custom function definitions

- Simple example:

```xml
declare function local:factorial($n) {
    $n * local:factorial($n – 1)
};
```

- Example using type declarations:

```xml
declare function local:factorial($n as xs:integer) as xs:integer {
    $n * local:factorial($n – 1)
};
```
Type constraints

- Type declarations possible (but not obligatory) for:
  - variables
  - function arguments and result
  - also in XSLT 2.0 (variables and parameters)

- Dynamic typing used in practical applications
  - $13 + if ( aCondition ) then 'not a valid number' else 1$
    may fail or not depending on input data
  - some $x$ in (1+1, xs:date('long long time ago'))
    satisfies $x=2$  fails or not depending on the processor

- Static typing discussed, but rarely deployed
  ("academic" solutions, for XQuery rather than XSLT)
Type declarations

- Capabilities:
  - type name
    - built-in – always available
    - user-defined – schema aware processors only
  - kind of node | node() | item()
  - occurrence modifier (? , *, +, exactly one occurrence by default).

- Examples:
  - `xs:double`
  - `element()`
  - `node()*`
  - `xs:integer?`
  - `item()+`
Minor features

- string concatenation operator `||`
  
  `'Her name is ' || name`

- mapping operator `!`
  
  `(1, 2, 3) ! (. + 5) → (6, 7, 8)`

- switch

- count clause in FLWOR

More serious features

- group-by clause and grouping “windows” in FLWOR

- higher-level functions,

- more predefined functions
  
  - in particular, more math available in separate namespace
Initiative for XML database interfaces specification

- XML Database API (XAPI)
- accessing XML databases from programs
- resource collections (resource = XML document)
- reading and writing documents via DOM or SAX
- pluggable “services”; specified: XPath, transactions, operations on collections
- last version: 2001

XML Update Language (XUpdate)

- XML application (format) for updating XML databases
- inserting, updating and removing nodes
- XPath used for node addressing
- last version: 2000
Example (from XUpdate documentation)

```xml
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
    xmlns:xupdate="http://www.xmldb.org/xupdate">
    <xupdate:insert-after select="/addresses/address[1]" >
        <xupdate:element name="address">
            <xupdate:attribute name="id">2</xupdate:attribute>
            <fullname>Lars Martin</fullname>
            <born day='2' month='12' year='1974' />
            <town>Leizig</town>
            <country>Germany</country>
        </xupdate:element>
    </xupdate:insert-after>
</xupdate:modifications>
```
## XML database products – overview

<table>
<thead>
<tr>
<th>Product</th>
<th>Licence</th>
<th>Queries</th>
<th>XML:DB API</th>
</tr>
</thead>
<tbody>
<tr>
<td>eXist</td>
<td>open source</td>
<td>XPath, XQuery</td>
<td>yes</td>
</tr>
<tr>
<td>BaseX</td>
<td>open source</td>
<td>XPath, XQuery</td>
<td>yes</td>
</tr>
<tr>
<td>MarkLogic</td>
<td>commercial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache Xindice</td>
<td>open source</td>
<td>XPath</td>
<td>yes</td>
</tr>
<tr>
<td>Sedna</td>
<td>open source</td>
<td>XPath, XQuery</td>
<td>yes</td>
</tr>
<tr>
<td>Gemfire Enterprise</td>
<td>commercial</td>
<td>XQuery, OQL</td>
<td>yes</td>
</tr>
<tr>
<td>Tamino</td>
<td>commercial</td>
<td>XQuery, XPath</td>
<td>part</td>
</tr>
</tbody>
</table>

Source: Wikipedia and providers' websites

**In addition:**

- Saxon – just a query processing engine, works on files (or other XML sources accessible in Java).
eXist DB

- One the most popular and elaborated XML database engines
- Open-source, but developed and supported by a (German) company; commercial support available
- Features include:
  - storage of XML and binary entities
  - various means of access, including: human-readable Web interface, direct HTTP access (REST-compliant), SOAP and XML-RPC, Java API (XQJ, elements of XAPI)
  - full XML model available in XPath, XQuery, and XSLT code
  - full XQuery support with majority of new 3.0 features, Update extension and some other non-standard extensions
  - XForms support using betterFORM or XSLTForms plugins
  - extensible with custom Java code
eXist – eXide

- XQuery programmer SDK running within a browser
  - supports also (to some extent...) XSLT, XML Schema, XHTML, XForms
eXist – template mechanism

- Easy integration of XQuery logic and HTML interface

```xquery
declare function app:hello( $node as node(), $model as map(*), ... ) {
  'Hello world!'
};
```

```html
<div class="app:hello"> </div>
```