XML Support in Relational Databases

Mitchell W. Smith
Array BioPharma Inc.
msmith@arraybiopharma.com
Agenda

- Introduction to SQL/XML
- SQL/XML Publishing Functions
- SQL/XML Predicates
- SQL/XML Shredding

- Vendor specific implementations
  - Oracle
  - DB2
  - SQL Server 2005
What Is SQL/XML?

- Bridges XML and SQL.
  - Allows XML instances to be stored into a database and retrieved from a database.
  - Creation of XML instances from relational tables.
  - Creation of relational tables from XML instances.
  - Provides the ability to query XML instances.
SQL/XML Is New

- Only been around officially since 2003.
- The 2005 standard has yet to be ratified and contains numerous additions and adjustments.
- Each database vendor implements a portion of the standard and has many non-standard features.
- Many of the non-standard features will be deprecated over time, but many will remain.
- New SQL/XML features are being implemented with each vendor’s release and many are new with the current releases.
Example DB Schema

- Used for rest of talk.
- Not a real schema – just one to use during this talk.
Example DB Schema w/Values

INSERT INTO location VALUES (1, 'Boulder');
INSERT INTO location VALUES (2, 'Longmont');

INSERT INTO department VALUES (1, 'HR');
INSERT INTO department VALUES (2, 'SciComp');
INSERT INTO department VALUES (3, 'IT');
Example DB Schema w/VALUES

(Continued)

<table>
<thead>
<tr>
<th>Location</th>
<th>Employee</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK2</td>
<td>employee</td>
<td>department</td>
</tr>
<tr>
<td></td>
<td>location</td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>id</td>
<td>not null</td>
</tr>
</tbody>
</table>

Example DB Schema w/VALUES

| INSERT INTO employee VALUES (1, 'Jennifer', to_date('05/09/2000', 'mm/dd/yyyy'), 2, 1); |
| INSERT INTO employee VALUES (2, 'Dan', to_date('02/05/2001', 'mm/dd/yyyy'), 1, 2); |
| INSERT INTO employee VALUES (3, 'Ron', to_date('01/01/2004', 'mm/dd/yyyy'), 1, 3); |
| INSERT INTO employee VALUES (4, 'Mike', to_date('01/01/2001', 'mm/dd/yyyy'), 1, 3); |
| INSERT INTO employee VALUES (5, 'Wayne', to_date('11/15/2001', 'mm/dd/yyyy'), 1, 3); |
| INSERT INTO employee VALUES (6, 'Wayne', to_date('11/15/2001', 'mm/dd/yyyy'), 2, 3); |
| INSERT INTO employee VALUES (7, 'Joey', to_date('02/15/2003', 'mm/dd/yyyy'), 1, 3); |
| INSERT INTO employee VALUES (8, 'Laurie', to_date('03/28/2002', 'mm/dd/yyyy'), 1, 3); |
| INSERT INTO employee VALUES (9, 'Michelle', to_date('05/21/2002', 'mm/dd/yyyy'), 1, 2); |
| INSERT INTO employee VALUES (10, 'Matt', to_date('11/15/2001', 'mm/dd/yyyy'), 1, 2); |
XML Type

- New database type
  - Can be used as a column in a table.
  - Can be used as the type of a table.
- Constructed via XML publishing functions, casting, or serialization.
- Methods for node extraction, node existence, and validation.
XML Type – Examples

CREATE TABLE employeeXML OF XMLType;

CREATE TABLE employee_info (Employee_id NUMBER NOT NULL, Employee_data XMLType);

select XMLType(''<employee>John'</employee>'') from ... 

NOTE: Use of "XMLType" is specific to Oracle. SQL standard specifies this as "XML".
XML Type – Insert of Doc

```
INSERT INTO employeeXML VALUES ('<?xml version="1.0" standalone="yes" ?>
<Employees>
  <Employee>
    <Name>Jennifer</Name>
    <DateHired>2000-05-09</DateHired>
    <Location>Longmont</Location>
    <Department>HR</Department>
  </Employee>
  <Employee>
    <Name>Dan</Name>
    <DateHired>2001-02-05</DateHired>
    <Location>Boulder</Location>
    <Department>SciComp</Department>
  </Employee>
  <Employee>
    <Name>Ron</Name>
    <DateHired>2004-01-01</DateHired>
    <Location>Boulder</Location>
    <Department>IT</Department>
  </Employee>
  <Employee>
    <Name>Laurie</Name>
    <DateHired>2002-03-28</DateHired>
    <Location>Longmont</Location>
    <Department>IT</Department>
  </Employee>
  <Employee>
    <Name>Michelle</Name>
    <DateHired>2002-05-21</DateHired>
    <Location>Boulder</Location>
    <Department>SciComp</Department>
  </Employee>
  <Employee>
    <Name>Matt</Name>
    <DateHired>2001-11-15</DateHired>
    <Location>Boulder</Location>
    <Department>SciComp</Department>
  </Employee>
</Employees>');```

XML Publishing Functions

- Generate XML fragments from relational tables.
  -XMLElement
  -XMLAttributes
  -XMLForest
  -XMLConcat
  -XMLAgg
  -XMLPI
  -XMLComment
  -XMLRoot
XMLElement

- Generates XML element nodes.

```sql
SELECT XMLElement("Department", name) AS DepartmentNames
FROM   department;
```

DepartmentNames
-------------------------------
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
<br/>
XMLAttributes

- Generates XML attributes inside of an XMLElement.

```sql
SELECT XMLElement("Employee",
    XMLAttributes(Employee.name AS "employeeName"),
    XMLElement("Location", Location.name)) AS "Employee"
FROM (employee inner join location on employee.location_id = location.location_id);
```

Employee

-----------------------------
<Employee employeeName="Matt"><Location>Boulder</Location></Employee>
<Employee employeeName="Michelle"><Location>Boulder</Location></Employee>
<Employee employeeName="Joey"><Location>Boulder</Location></Employee>
<Employee employeeName="Wayne"><Location>Boulder</Location></Employee>
<Employee employeeName="Wayne"><Location>Longmont</Location></Employee>
<Employee employeeName="Jennifer"><Location>Longmont</Location></Employee>
```
XMLForest

- Generates a forest of XML Elements.

```sql
SELECT XMLElement("Employee",
    XMLAttributes(Employee.name AS "employeeName"),
    XMLForest(Location.name AS "Location",
        Employee.date_hired AS "HireDate")) AS "Employee"
FROM (employee inner join location on employee.location_id = location.location_id);
```

```
<Employee employeeName="Matt"><Location>Boulder</Location><HireDate>2001-11-01</HireDate></Employee>
<Employee employeeName="Michelle"><Location>Boulder</Location><HireDate>2001-05-01</HireDate></Employee>
<Employee employeeName="Joey"><Location>Boulder</Location><HireDate>2003-02-03</HireDate></Employee>
<Employee employeeName="Wayne"><Location>Boulder</Location><HireDate>2001-11-01</HireDate></Employee>
```
XMLConcat

- Concatenates all of its Element arguments to generate an XML fragment.

```sql
SELECT XMLConcat(XMLElement("EmployeeName", Employee.name),
    XMLElement("Location", Location.name)) AS "Employee"
FROM (employee inner join location on employee.location_id = location.location_id);
```

Employee

-----------------------------------------------
.EmployeeName>Matt</EmployeeName><Location>Boulder</Location>
.EmployeeName>Michelle</EmployeeName><Location>Boulder</Location>
.EmployeeName>Joey</EmployeeName><Location>Boulder</Location>
.EmployeeName>Wayne</EmployeeName><Location>Boulder</Location>
.EmployeeName>Mike</EmployeeName><Location>Boulder</Location>
.EmployeeName>Ron</EmployeeName><Location>Boulder</Location>
.EmployeeName>Dan</EmployeeName><Location>Boulder</Location>
.EmployeeName>Laurie</EmployeeName><Location>Longmont</Location>
.EmployeeName>Wayne</EmployeeName><Location>Longmont</Location>
.EmployeeName>Jennifer</EmployeeName><Location>Longmont</Location>
XMLAgg

- Generates a forest of XMLElements from a collection of XMLElements.

```sql
SELECT XMLElement("EmployeeByLocation",
    XMLAttributes(Location.name AS "locationName"),
    XMLAgg(XMLForest(Employee.name AS "Employee",
        Department.name AS "DepartmentName"))) AS "EmployeeByLocation"
FROM (employee inner join location on employee.location_id = location.location_id)
    join department on employee.department_id = department.department_id
GROUP BY location.name;

EmployeeByLocation

...<EmployeeByLocation locationName="Longmont">
    <Employee>Jennifer</Employee><DepartmentName>HR</DepartmentName>
    <Employee>Laurie</Employee><DepartmentName>IT</DepartmentName>
    <Employee>Wayne</Employee><DepartmentName>IT</DepartmentName>
</EmployeeByLocation>
XMLPI

- Generates XML processing instructions.

```sql
SELECT XMLConcat(XMLPI(NAME "LocationComp", 'imported'),
                   XMLMLElement("Location", Location.name)) AS "Location"
FROM   location;
```

Location

-------------------------------
<?LocationComp imported?><Location>Boulder</Location>
<?LocationComp imported?><Location>Longmont</Location>
**XMLComment**

- Generates XML comments.

```sql
SELECT XMLConcat(XMLComment('This is a comment'), XMLElement("Location", Location.name)) AS "Location"
FROM location;
```

<table>
<thead>
<tr>
<th>Location</th>
</tr>
</thead>
</table>
| -----------
| <!--This is a comment--><Location>Boulder</Location> |
| <!--This is a comment--><Location>Longmont</Location> |
XMLRoot

- Generates an XML root node.

```sql
SELECT XMLRoot(XMLElement("EmployeeByLocation",
    XMLAttributes(Location.name AS "locationName"),
    XMLAgg(XMLForest(Employee.name AS "Employee",
        Employee.date_hired AS "HireDate"))),
    VERSION '1.0', STANDALONE YES) AS "EmployeeDoc"
FROM (employee inner join location on employee.location_id = location.location_id)
GROUP BY location.name;
```

EmployeeDoc

```xml
<?xml version="1.0" standalone="yes"?>
<EmployeeByLocation locationName="Longmont">
    <Employee>Jennifer</Employee>  <HireDate>2000-05-09</HireDate>
    <Employee>Dwayne</Employee>  <HireDate>2001-11-15</HireDate>
    <Employee>Laurie</Employee>  <HireDate>2002-03-28</HireDate>
</EmployeeByLocation>
```
What can these be used for?
XML Documents from DB

- With publishing functions covered so far, creation of XML instances in the database is now possible.
  - No need for middle-tier to generate XML.

```sql
CREATE VIEW EmployeeByLocation AS
SELECT XMLRoot(
    XMLElement("EmployeeByLocation",
        XMLAttributes(Location.name AS "locationName"),
        XMLAgg(XMLForest(Employee.name AS "Employee",
                      Employee.date_hired AS "HireDate")))
    VERSION '1.0', STANDALONE YES) AS "EmployeeDoc"
FROM (employee inner join location on employee.location_id = location.location_id)
GROUP BY location.name;
```
XML Doc Fragments in DB

- In addition, can support web services more intelligently by XML fragments that can be used in a SOAP message return:

```sql
CREATE VIEW EmployeeByLocationDocFragment AS
SELECT location.name,
    XMLElement("EmployeeByLocation",
        XMLAttributes(Location.name AS "locationName"),
        XMLAgg(XMLForest(Employee.name AS "Employee",
            Department.name AS "DepartmentName")))
AS "EmployeeByLocation"
FROM (employee inner join location on employee.location_id = location.location_id) 
join department on employee.department_id = department.department_id
GROUP BY location.name;
```

- Request to get employees in Longmont becomes:

```sql
SELECT * FROM EmployeeByLocationDocFragment where name = 'Longmont';
```
More XML Publishing Functions

- XMLParse
- XMLSerialize
- XMLCast
- XMLQuery
XMLParse

- Takes a serialized XML instance and converts it to an XML type.

```
SELECT XMLParse(CONTENT '<Department>HR</Department>' WELLFORMED) AS "HRNode"
FROM location;

HRNode
------------------------
<Department>HR</Department>
<Department>HR</Department>
<Department>HR</Department>
```
XMLSerialize

- Takes an XML instance and converts it to a serialized string.

```sql
SELECT XMLSerialize(DOCUMENT XMLElement("Department", name) AS CLOB) AS "Dept"
FROM department;
```

Dept
-------------------
<Department>HR</Department>
<Department>SciComp</Department>
<Department>IT</Department>
XMLCast

- Allows casting from one XML type to another XML type.

  SQL/XML has five different XML document types as a result of using the XQuery data model rather than the Infoset.
  - XML(Sequence), XML(Any Content), XML(Untyped Content), XML(Any Document), and XML(Untyped Document)

XMLCast(value-expression AS type)
XMLQuery

- Evaluates an XQuery expression and returns the result.

```sql
SELECT XMLQUERY('for $Employee in /Employees/Employee
    let $Name := $Employee/Name,
    $Location := $Employee/Location
    where $Location eq $LOC
    return $Name'
    --object_value is specific to Oracle
    PASSING BY VALUE object_value, XMLELEMENT("LOC", 'Longmont') AS LOC
    RETURNING CONTENT) AS EmployeesAtLocation
FROM   employeeXML;

EmployeesAtLocation
----------------------
<Name>Jennifer</Name><Name>Dwayne</Name><Name>Laurie</Name>
```
XML Predicates

- **XML IS DOCUMENT**
  - Determines whether an XML instance is well-formed.

- **XML IS VALID**
  - Determines whether an XML instance is valid against an XML schema.
    - Can also specify whether identity constraints are checked.

- **XMLExists**
  - Evaluates an XQuery expression on an XML instance and returns whether there was a match or not.
XMLTable

Transforms XML data into relational data.

```
SELECT employeename, datehired, locationname, departmentname
FROM   employeexml, XMLTable('for $i in /Employees/Employee

   return <employee><name>{$i/Name}</name>

        <datehired>{$i/DateHired}</datehired>

        <locationname>{$i/Location}</locationname>

        <departmentname>{$i/Department}</departmentname>

    </employee>'PASSING employeexml.object_value COLUMNS
employeename   VARCHAR2(100) PATH '/employee/name',
datehired      DATE          PATH '/employee/datehired',
locationname   VARCHAR2(100) PATH '/employee/locationname',
departmentname VARCHAR2(100) PATH '/employee/departimentname');
```

<table>
<thead>
<tr>
<th>EMPLOYEENAME</th>
<th>DATEHIRED</th>
<th>LOCATIONNAME</th>
<th>DEPARTMENTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jennifer</td>
<td>05/09/2000</td>
<td>Longmont</td>
<td>HR</td>
</tr>
<tr>
<td>Dan</td>
<td>02/05/2001</td>
<td>Boulder</td>
<td>SciComp</td>
</tr>
<tr>
<td>Ron</td>
<td>01/01/2004</td>
<td>Boulder</td>
<td>IT</td>
</tr>
<tr>
<td>Mike</td>
<td>01/01/2001</td>
<td>Boulder</td>
<td>IT</td>
</tr>
<tr>
<td>Ron</td>
<td>01/01/2004</td>
<td>Boulder</td>
<td>IT</td>
</tr>
<tr>
<td>Wayne</td>
<td>11/15/2004</td>
<td>Boulder</td>
<td>IT</td>
</tr>
<tr>
<td>Dwayne</td>
<td>11/15/2004</td>
<td>Longmont</td>
<td>IT</td>
</tr>
<tr>
<td>Joey</td>
<td>02/03/2003</td>
<td>Boulder</td>
<td>HR</td>
</tr>
<tr>
<td>Laurie</td>
<td>03/28/2002</td>
<td>Longmont</td>
<td>IT</td>
</tr>
<tr>
<td>Michelle</td>
<td>05/21/2002</td>
<td>Boulder</td>
<td>SciComp</td>
</tr>
<tr>
<td>Matt</td>
<td>11/15/2001</td>
<td>Boulder</td>
<td>SciComp</td>
</tr>
</tbody>
</table>
Relational Views from XML Data

- Transform XML data into a relational view for systems that expect relational data.

```sql
CREATE or replace VIEW EmployeeInfoRelational AS
SELECT employeename, datehired, locationname, departmentname
FROM employeexml,
     XMLTable('for $i in /Employees/Employee
return <employee><name>{$i/Name}</name>
   <datehired>{$i/DateHired}</datehired>
   <locationname>{$i/Location}</locationname>
   <departmentname>{$i/Department}</departmentname>
</employee>'
PASSING employeexml.object_value COLUMNS
employeename VARCHAR2(100) PATH '/employee/name',
datehired DATE PATH '/employee/datehired',
locationname VARCHAR2(100) PATH '/employee/locationname',
departmentname VARCHAR2(100) PATH '/employee/departmentname');
```

- Request to get employees in Longmont becomes:

```sql
SELECT * FROM EmployeeInfoRelational WHERE locationname = 'Longmont';
```
XML Schemas Are Registered

- To validate an XML instance to an XML Schema, the Schema must be registered with the database.
- Avoids security issues (reverse engineer schema).
- Registration involves a XML Schema descriptor which describes the target namespace URI, the schema location URI, the registered name of the XML Schema, etc.
- The xs, xsi, and sqlxml schemas are permanently registered.
Oracle Specific XML
Oracle Supported XML Predicates

- **XMLIsValid**
  - Determines whether an XML instance is valid for an XML Schema.

- **schemaValidate**
  - Validates the XML instance, if needed.

- **isSchemaValidated**
  - Returns whether the XML instance has been validated, or not.

- **setSchemaValidated**
  - Manually sets the validation state of the XML instance.

- **existsNode**
Oracle Predicate `existsNode`

- SQL function to determine whether an XPath expression returns at least one element or text node given an XML document.
  - Superceded by `XMLExists` in standard.

```sql
SELECT COUNT(*)
FROM employeeXML
WHERE existsNode(object_value, '/Employees/Employee/Name="Joey"') = 1;

COUNT(*)
----------
        1
```
Oracle Specific XML Publishing Functions

- **XMLSequence**
  - Returns a VARRAY of XML instances. Useful to generate a table in the FROM clause.

- **XMLCollAttVal**
  - Generates a forest of XML column elements.

- **XMLCDATA**
  - Generates an XML CDATA section.

- **SYS_XMLGen**
  - Generates an XML fragment for each row returned.
    - Lots of options.
Oracle Specific XML Publishing Functions (Continued)

- SYS_XMLAgg
  - Generates an XML instance for all of the XML fragments.
- XMLTransform
- extract
- extractValue
Oracle – XMLTransform

- Applies an XSL style sheet to an XML instance.
  - Example to generate an HTML document.

```sql
SELECT XMLtransform(OBJECT_VALUE, xmltype(''?
  <xsl:stylesheet version="1.0" encoding="ISO-8859-1"?
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
      <xsl:output method = "html"
        omit-xml-declaration = "no"
        doctype-public = "-//W3C//DTD HTML 3.2 Final//EN"
        doctype-system = "http://www.w3.org/TR/html40/strict.dtd"/>
      <xsl:template match="/">
        <xsl:element name="html">
          <xsl:element name="head">
            <xsl:element name="title">
              Employee Info
            </xsl:element>
          </xsl:element>
        </xsl:element>
        <xsl:element name="body">
          <xsl:element name="center">
            <xsl:element name="h1">Employee Info</xsl:element>
          </xsl:element>
        </xsl:element>
      </xsl:template>
    </xsl:stylesheet>
'))
```
Oracle – XMLTransform (Continued)

```
...</xsl:template>
<xsl:template match="Employee">
  <xsl:element name="tr">
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
<xsl:template match="Name|Location|DateHired|Department">
  <xsl:element name="td">
    <xsl:element name="small">
      <xsl:value-of select="."/>
    </xsl:element>
  </xsl:element>
</xsl:template>
</xsl:stylesheet>')FROM EmployeeXML;
```
### Oracle – XMLTransform Result

#### Employee Info

<table>
<thead>
<tr>
<th>Name</th>
<th>Date Hired</th>
<th>Location</th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jennifer</td>
<td>2000-05-09</td>
<td>Longmont</td>
<td>HR</td>
</tr>
<tr>
<td>Dan</td>
<td>2001-02-05</td>
<td>Boulder</td>
<td>SciComp</td>
</tr>
<tr>
<td>Ron</td>
<td>2004-01-01</td>
<td>Boulder</td>
<td>IT</td>
</tr>
<tr>
<td>Mike</td>
<td>2001-01-01</td>
<td>Boulder</td>
<td>IT</td>
</tr>
<tr>
<td>Ron</td>
<td>2004-01-01</td>
<td>Boulder</td>
<td>IT</td>
</tr>
<tr>
<td>Wayne</td>
<td>2004-11-15</td>
<td>Boulder</td>
<td>IT</td>
</tr>
<tr>
<td>Dwayne</td>
<td>2004-11-15</td>
<td>Longmont</td>
<td>IT</td>
</tr>
<tr>
<td>Joey</td>
<td>2003-02-03</td>
<td>Boulder</td>
<td>HR</td>
</tr>
<tr>
<td>Lauren</td>
<td>2002-03-28</td>
<td>Longmont</td>
<td>IT</td>
</tr>
<tr>
<td>Michelle</td>
<td>2002-05-21</td>
<td>Boulder</td>
<td>SciComp</td>
</tr>
<tr>
<td>Matt</td>
<td>2001-11-15</td>
<td>Boulder</td>
<td>SciComp</td>
</tr>
</tbody>
</table>
Oracle – extract

- SQL function that returns a node set given an XPath expression applied to an XML instance.

```
SELECT extract(object_value, '/Employees/Employee/Name') AS "EmpName"
FROM   employeeXML;
```

```
EmpName
-----------------------
<Name>Jennifer</Name><Name>Dan</Name><Name>Ron</Name><Name>Mike</Name>...
```
Oracle – extractValue

- SQL function that returns a scalar value given an XPath expression applied to an XML instance.
  - Must return one value.

```
SELECT extractValue(XMLType('<val>3</val>'), '/val')
FROM   dual;
```

```
EXTRACTVALUE(XMLTYPE('<VAL>3</VAL>'),'/VAL')
--------------------------
3
```
Oracle – XML Modification

- `updateXML`
- `insertChildXML`
- `insertXMLBefore`
- `appendChildXML`
- `deleteXML`

--Joey now in Longmont instead of Boulder
```
UPDATE employeeXML
SET OBJECT_VALUE = updateXML(OBJECT_VALUE,
   '/Employees/Employee[Name/text()="Joey"]/Location/text()','
   'Longmont')
WHERE existsNode(OBJECT_VALUE, '/Employees/Employee/Name="Joey"') = 1;
```
Oracle XML DB

- Generates relational tables and objects from an XML Schema.
  - Lots of options for table and object generation.
- XML instances are stored in a relational form instead of a LOB field. Benefits include:
  - Reduced disk usage.
  - XML instance validation against XML Schema.
  - Reduced memory requirements during queries.
  - Increased query performance.
Oracle XML DB – Schema Registration

declare
doc varchar2(32767);
begin
doc:= '<?xml version="1.0"?>
<x:schema id="Employees"
     xmlns:x="http://www.w3.org/2001/XMLSchema" version="1.0"
     attributeFormDefault="qualified"
     elementFormDefault="qualified">
     <xs:simpleType name="Department">
     <xs:restriction base="xs:string">
     <xs:enumeration value="HR"/>
     <xs:enumeration value="SciComp"/>
     <xs:enumeration value="IT"/>
     </xs:restriction>
     </xs:simpleType>
     <xs:simpleType name="Location">
     <xs:restriction base="xs:string">
     <xs:enumeration value="Boulder"/>
     <xs:enumeration value="Longmont"/>
     </xs:restriction>
     </xs:simpleType>
</x:schema>
'
Oracle XML DB – Schema Registration

(Continued)

```xml
<xs:element name="Employees">
  <xs:complexType>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="Employee">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Name" type="xs:string" minOccurs="1" />  
            <xs:element name="DateHired" type="xs:dateTime" minOccurs="1" />  
            <xs:element name="Location" type="Location" minOccurs="1" />  
            <xs:element name="Department" type="Department" minOccurs="1" />  
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

dbms_xmlschema.registerSchema('http://tempuri.org/employees.xsd', doc, 
true, true, false, true);
end;
```
Oracle XML DB – Table Creation and Insert

CREATE TABLE EmployeeXMLSchema OF XMLType
XMLSCHEMA "http://tempuri.org/employees.xsd"
ELEMENT "Employees";

-- Review table and types that have been created.

INSERT INTO EmployeeXMLSchema VALUES (XMLType(''<?xml version="1.0" standalone="yes"?>
<Employees xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://tempuri.org/employees.xsd">
   <Employee>
   <Name>Jennifer</Name>
   <DateHired>2000-05-09</DateHired>
   <Location>Longmont</Location>
   <Department>HR</Department>
   </Employee>
   ...
</Employees>''));
Oracle XML DB – Queries and Execution Plans

```sql
SELECT extract(object_value, '/Employees/Employee/Name') AS "EmpName"
FROM EmployeeXMLSchema;

SELECT XMLQUERY('for $Employee in /Employees/Employee
let $Name := $Employee/Name,
    $DateHired := $Employee/DateHired
where $Employee/Location eq $LOC
return <employee><name>{$Name}</name>
    <datehired>{$DateHired}</datehired>
</employee>'
    PASSING BY VALUE object_value,
    XMLELEMENT("LOC", 'Longmont') AS LOC
RETURNING CONTENT).getstringval() AS EmployeesAtLocation
FROM EmployeeXMLSchema;
```
Oracle XML Support

- PL/SQL
  - DBMS_XMLDOM
  - DBMS_XMLPARSER
  - DBMS_XSLPROCESSOR

- Java

- C and C++ libraries to mirror Java capabilities.
DB2 Specific XML
DB2 – XML Types

- DB2 has three XML types:
  - XMLVARCHAR – small XML instances.
  - XMLCLOB – large XML instances.
  - XMLFile – XML instances stored outside of DB2.
DB2 – Side Tables

- Maps XML Type to Relational Tables.
  - Use DAD file to map the XML elements and attributes to side tables.
  - XPath used to map XML instances to relational columns.

```xml
<Xcolumn>
<table name="employee">
  <column name="name" type="varchar" path="/Employees/Employee/Name" multi_occurrence="NO"/>
  <column name="date_hired" type="DATE" path="/Employees/Employee/DateHired" multi_occurrence="NO"/>
  ...
</table>
</Xcolumn>
```
DB2 – Side Tables (Continued)

- XML Data is maintained as both an XML type and relational format.
  - Updates to the XML instance result in changes to the relational data as well.
- Can validate XML instance against a DTD or an XML Schema.
- Performance advantages for querying.
- Allows mapping of new XML instances to existing relational database schema.
DB2 – Generating XML

- Generate XML using a DAD file that specifies the SQL mapping.
  - Provide a SQL query and an XML instance that maps the results of the SQL call to an XML instance.
  - Can validate resulting XML instance against a DTD or an XML Schema.
DB2 – Extracting Contents

- XML publishing functions not supported.
- Extract the value of an element or attribute node.
  - Special functions for each datatype:
    - `extractInteger`, `extractIntegers`
    - `extractDate`, `extractDates`
    - ...

```sql
SELECT DB2XML.extractVarChar(employee_data, '/Employees/Employee/Name')
FROM employee_info;
```
DB2 – Searching

- Perform a direct query on a side table.
- Use the extractXXX function to get a value and test against it in a condition.
DB2 – Miscellaneous

- **XSLTransformToFile**
  - Transforms an XML document.

- **DTD Repository**
  - Stores DTDs in the database.

- **Update XML Instance Elements**
  - Use Update function.

- **Validation functions**
  - `DB2XML.SValidate` – XML Schema validation
  - `DB2XML.DValidate` – DTD validation
DB2 – VIPER

- **Native XML Store**
  - Store XML in manner similar to a DOM.
  - Native XML type in database.

- **XQuery, XPath, and SQL/XML support.**
  - Mix SQL with XQuery in same query.

- **Not necessarily XML Schema based.**
  - Indices on XML columns based on XPath.
SQL Server 2005 Specific XML
SQL Server – XML Types

- SQL Server has two XML types:
  - CONTENT – untyped XML.
  - DOCUMENT – one top-level elements.
    - Can create check constraint to enforce compliance to XML Schema.
SQL Server – XML Type Methods

- **Publishing Functions**
  - Query() – runs an XQuery on the XML instance.
  - Value() – evaluates an XQuery expression to return a scalar value.
  - Nodes() – generates node contexts within the XML instance. Used with Query(), Value(), and Exists().

- **Predicates**
  - Exists() – used as a condition. Runs an XQuery on the XML instance and returns whether or not a matching node list was found.

- **Modification Function**
  - Modify() – modifies the XML instance.
SQL Server – XML Modification

- Modify method on xml type object.
  - Perform inserts, updates, and deletes of elements attributes, comments, cdata, etc.

```
UPDATE T
SET x.modify('insert <Material>Aluminium</Material> as first
  into   (/Root/ProductDescription/Features)[1]
');

UPDATE T
SET x.modify('replace value of (/Root/ProductDescription/@ProductName)[1]
  with "New Road Bike" ');

UPDATE T
SET x.modify('delete /Root/ProductDescription/Features/*[2]')
```
SQL Server Specific XML Publishing Functions

- FOR XML
  - Returns a series of XML document fragments from a relational query.
  - Lots of options.

SELECT *
FROM employee
FOR XML RAW, ELEMENTS;

<row>
  <ProductModelID>122</ProductModelID>
  <Name>All-Purpose Bike Stand</Name>
</row>
...

...
SQL Server – Conversions to and from XML and Relational Database

- Can generate an XML view of a relational database.
  - Create XML schema document and annotate it with the corresponding tables, columns, and relationships between the tables.
  - Call OLEDB Provider to generate XML from query.

- OpenXML
  - Converts XML document to a relational table.
  - Very similar to XMLTable.
Using an updategram (an XML instance that specifies a series of insert, updates, and deletes), changes to the relational data or XML instance can be performed.

```xml
--Example insert.
<ROOT xmlns:updg="urn:schemas-microsoft-com:xml-updategram">
  <updg:sync [mapping-schema="SampleSchema.xml"]>
    [<updg:before>]
    <updg:after [updg:returnid="x y ..."]>
      <ElementName [updg:id="value"]
        [updg:at-identity="x"]
        [updg:guid="y"]
        attribute="value"
        attribute="value"
        ...
      />
      [<ElementName .../>...]
    </updg:after>
  </updg:sync>
</ROOT>
```
SQL Server Miscellaneous

- Native SOAP handling.
- XML Schema repository.
  - XML table columns can be strongly typed with an XML Schema.
- SQLCLR
  - Allows procedure creation within SQL Server written in any CLR supported language.
    - Run XSL transformations using the TransformXml.ApplyXslTransform function.
    - Generate new XML documents in a standard programming language.
    - Full XML support.
Resources

- Advancements in SQL/XML
- ISO/IEC FCD 9075-14 – SQL/XML Standard
  - http://www.ansi.org
- W3C Page on XQuery
  - http://www.w3.org/xml/Query.html
- DB2
- SQL Server
  - Microsoft SQL Server 2005 Books Online
- Oracle