Database modeling
SQL/XML

Reference:

Querying XML, Jim Melton, Oracle Corp,Morgan Kaufmann Publishers

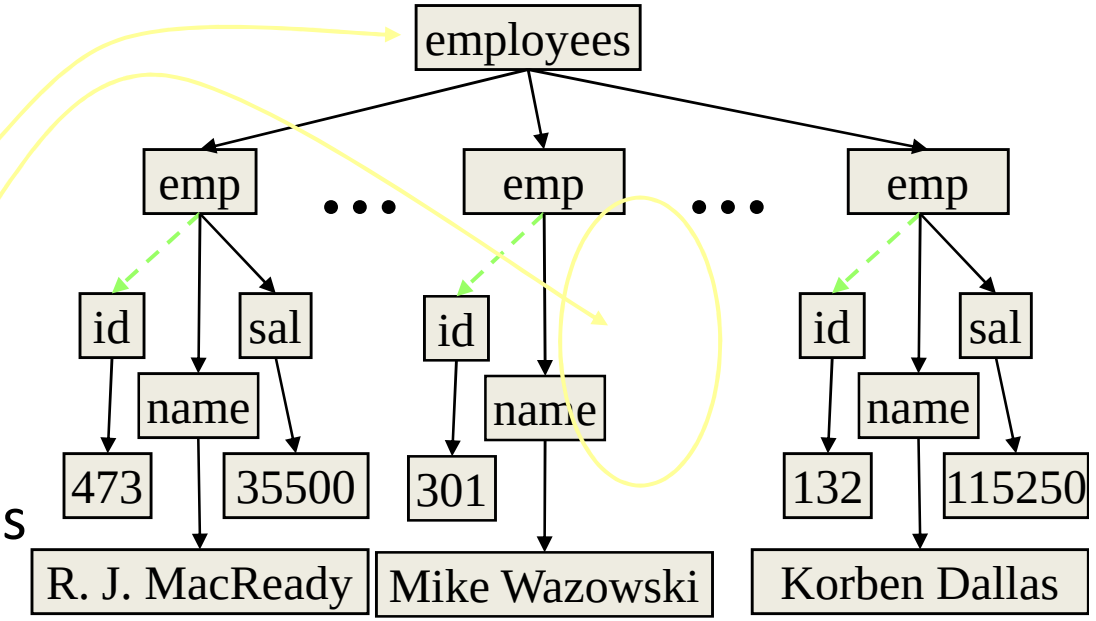
SQL—Structured Data

- Based on relational model
- Tables of rows and columns
- Every "cell" has a value (possibly null)
- SQL Query Language extremely powerful (probably too big)
- Products extremely well-developed: scalable, robust, manageable, *etc.*

EMP_ID	NAME	SALARY
473	R. J. MacReady	35500.00
921	Sam Loomis	26350.00
301	Mike Wazowski	*(null)*
17	David Kessler	22600.00
1284	Laurie Strode	14000.00
132	Korben Dallas	115250.00

XML—Semi-Structured Data

- Foundation for “User” Markup Languages
- Tree-structured, semi-structured
- DTD, XML Schema *etc.*
- Rapid Acceptance, Tools Widely Available



SQL/XML

- Goal of SQLX Group was to define extensions to SQL standard to integrate SQL and XML

- Allow SQL programmers to manage and query XML data.

I. Query SQL data and publish the results as XML.
II. Store and manage XML natively in a SQL database.
III. Query XML and publish the results as either XML or as SQL data.
IV. Map SQL data and identifiers to XML, and vice versa.

I. Query SQL data and publish the results as XML. Relational ➔ XML

Publishing Functions

- XMLParse
- XMLSerialize
- XMLQuery
- XMLTable

- XMLDocument
- XMLElement
  - XMLAttribute
- XMLForest
- XMLAgg
- XMLConcat

- XMLComment
- XMLPI
- XMLText

Relational → XML
Example: XMLElement

```
XMLELEMENT ( NAME = "emp"
  XMLCOMMENT ( "Example 1" ),
  XMLATTRIBUTES
    ( EMP_ID AS "id" ),
  XMLELEMENT ( NAME = "name",
    NAME ),
  XMLELEMENT ( NAME = "sal",
    SALARY ) )
```

In serialized form:

```
<emp id="473">
  <!-- Example 1 -->
  <name>R. J.
    MacReady</name>
  <sal>35500.00</sal>
</emp>
```

XML Functions by Example – 1. simple element

```
<Emp>Jack Lee</Emp>
```

```
SELECT
    XML2CLOB(
        XMLELEMENT(NAME "Emp",
                   e.fname ||' '|| e.lname )
        ) AS "result"
 FROM employees e WHERE ...;
```

XML Functions by Example – 1’. simple element

```
<Emp name="Jack Lee"></Emp>
```

```
SELECT
    XML2CLOB(
          XMLELEMENT(NAME "Emp",
            XMLATTRIBUTES(e.fname ||' '|| e.lname AS "name") )
            ) AS "result"
 FROM employees e WHERE ...;
```

XML Functions by Example – 2. namespace

```
<doc:Emp xmlns:doc="http://www.ibm.com/emp.xsd">
  Jack Lee
</doc:Emp>
```

```
SELECT
     XML2CLOB(
          XMLELEMENT(NAME "doc:Emp",
               XMLNAMESPACES ('http://www.ibm.com/emp.xsd'
                                   AS "doc" ),
                         e.fname ||' '|| e.lname)
           ) AS "result"
 FROM employees e WHERE ...;
```

XML Functions by Example – 3. nested elements

```
<Emp name="Jack Lee">
  <BIRTHDAY>1960-10-28</BIRTHDAY>
  <department>Shipping</department>
</Emp>
```

```
SELECT
     XML2CLOB(
           XMLELEMENT(NAME "Emp",
             XMLATTRIBUTES(e.fname ||' '|| e.lname AS "name")
               XMLELEMENT(NAME BIRTHDAY, e.birthday),
                 XMLELEMENT(NAME “department”, e.dept) )
             ) AS "result"
 FROM employees e WHERE ...;
```

XML Functions by Example – 3’. alternative

```
<Emp name="Jack Lee">
  <BIRTHDAY>1960-10-28</BIRTHDAY>
  <department>Shipping</department>
</Emp>
```

```
SELECT
     XML2CLOB(
           XMLELEMENT(NAME "Emp",
             XMLATTRIBUTES(e.fname ||' '|| e.lname AS "name")
               XMLFOREST(e.birthday, e.dept as “department”) )
             ) AS "result"
 FROM employees e WHERE ...;
```

XML Functions by Example – 4. mixed content

```
<Emp>Employee <name>Jack Lee</name>
was hired on <hiredate>2000-05-24</hiredate>
</Emp>
```

```
SELECT XML2CLOB(
          XMLELEMENT(NAME "Emp",
                              'Employee ',
          XMLELEMENT(NAME "name", e.fname ||' '|| e.lname),
           ' was hired on ',
          XMLELEMENT(NAME "hiredate", e.hire) )
        ) AS "result"
FROM employees e WHERE ...;
```

XML Functions by Example – 5. concat

```
<Emp name="Jack Lee"></Emp>
<department>Shipping</department>
```

```
SELECT
     XML2CLOB(
       XMLCONCAT(
           XMLELEMENT(NAME "Emp",
             XMLATTRIBUTES(e.fname ||' '|| e.lname AS "name") ),
           XMLELEMENT(NAME “department”, e.dept) )
         ) AS "result"
FROM employees e WHERE ...;
```

XML Functions by Example – 6. grouping

```
<Department name="Shipping">
    <emp>Oppenheimer</emp>
    <emp>Martin</emp>
    <emp>Lee</emp>
</Department>
```

```
SELECT XML2CLOB(
            XMLELEMENT(NAME "Department",
              XMLATTRIBUTES (e.dept AS "name" ),
                XMLAGG(XMLELEMENT(NAME "emp", e.lname)  )
                        ) ) AS "dept_list"
FROM employees e
GROUP BY dept;
```

XML Functions by Example – 6’. grouping with order

```
<Department name="Shipping">
    <emp>Lee</emp>
    <emp>Martin</emp>
    <emp>Oppenheimer</emp>
</Department>
```

```
SELECT XML2CLOB(
            XMLELEMENT(NAME "Department",
              XMLATTRIBUTES (e.dept AS "name" ),
                XMLAGG(XMLELEMENT(NAME "emp", e.lname)
                              ORDER BY e.lname  )
                          ) ) AS "dept_list"
FROM employees e
GROUP BY dept;
```

Example

```
<?xml version="1.0" ?>
<myDatabase>
    <customers>
        <custRec>
          <custName type="String">Robert Roberts</custName>
          <custAge type="Integer">25</custAge>
        </custRec>
        <custRec>
          <custName type="String">John Doe</custName>
          <custAge type="Integer">32</custAge>
        </custRec>
    </customers>
</myDatabase>
```

Generating XML from relational data using Java and JDBC

Step 1 : Set up the database connection

```
// Create an instance of the JDBC driver so that it has
// a chance to register itself
   Class.forName(sun.jdbc.odbc.JdbcOdbcDriver).newInstance();

// Create a new database connection.
  Connection con =
       DriverManager.getConnection(jdbc:odbc:myData, “”, “”);

// Create a statement object that we can execute queries with
  Statement stmt = con.createStatement();
```

Generating XML (cont.)

Step 2 : Execute the JDBC query

```
String query = “Select Name, Age from Customers”;
ResultSet rs = stmt.executeQuery(query);
```

Generating XML (cont.)

Step 3 : Create the XML!

```
StringBuffer xml = “<?xml
   version=‘1.0’?><myDatabase><customers>”;

while (rs.next()) {
   xml.append(“<custRec><custName>”);
      xml.append(rs.getString(“Name”));
      xml.append(“</custName><custAge>”);
      xml.append(rs.getInt(“Age”));
   xml.append(“</custAge></custRec>”);
}
xml.append(“</customers></myDatabase>”);
```

II. Store and manage XML natively in a SQL database XML - Relational

Step 1 : Set up the parser

```
StringReader stringReader = new
  StringReader(xmlString);
InputSource inputSource = new
  InputSource(stringReader);
DOMParser domParser = new DOMParser();
domParser.parse(inputSource);
Document document = domParser.getDocument();
```

XML - Relational

Step 2 : Read values from parsed XML document

```
NodeList nameList =
   doc.getElementsByTagName(“custName”);
NodeList ageList =
   doc.getElementsByTagName(“custAge”);
```

XML - Relational

Step 3 : Set up database connection

```
Class.forName(sun.jdbc.odbc.JdbcOdbcDriver).newInsta
   nce();
Connection con =

   DriverManager.getConnection(jdbc:odbc:myDataBase,
   “”, “”);
Statement stmt = con.createStatement();
```

XML - Relational

Step 4 : Insert data using appropriate JDBC update query

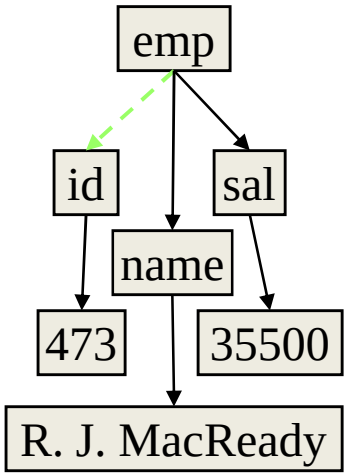
```
String sql = “INSERT INTO Customers (Name, Age) VALUES
   (?,?)”;
PreparedStatement pstmt = conn.prepareStatement(sql);
int size = nameList.getLength();
for (int i = 0; i < size; i++) {
        pstmt. setString(1,
   nameList.item(i).getFirstChild().getNodeValue());
        pstmt.setInt(2,
   ageList.item(i).getFirstChild().getNodeValue());

        pstmt.executeUpdate(sql);
}
```

XML ➔ Relational Storage Considerations

- VARCHAR
- CLOB
- BLOB

- “Native” XML

```
<emp id="473">
  <!-- Example 1 -->
  <name>R. J.
        MacReady</name>
  <sal>35500.00</sal>
</emp>
```



Parsing & Serialization

- XMLParse: Parses a string value using an XML parser; produces value whose specific type is XML(DOCUMENT(ANY)), or …CONTENT…, or …UNTYPED…
- XMLSerialize: transforms an XML value into a string value (CHAR, VARCHAR, CLOB, or BLOB)

XQuery Within SQL

- Problem: How can SQL applications locate and retrieve information in XML documents stored in an SQL database cell?
- Answer: Invoke XQuery from SQL statements (retrieve—in SELECT list; locate—in WHERE clause)!

XQuery Within SQL

- XMLQuery: A new SQL expression, invoked as a pseudo-function, whose data type can be an XML type—such as XML(CONTENT(ANY))—or an ordinary SQL type
- XMLExists: A new SQL predicate, invoked as a pseudo-function, returning *true* when the contained XQuery expression returns anything other than the empty sequence (*false*) or SQL null value (*unknown*)

XQuery Within SQL

- XMLValidate: Validates an XML value against an XML Schema (or target namespace), returning new XML value with type annotations
- IS VALID: Tests an XML value to determine whether or not it is valid according to an XML Schema (or target namespace); return *true*/*false* without altering the XML value itself

XQuery Within SQL

- Other new predicates:
  - IS DOCUMENT: determines whether an XML value satisfies the (SQL/XML) criteria for an XML document
  - IS CONTENT: determines whether an XML value satisfies the (SQL/XML) criteria for XML content

XMLTable

- Provides an SQL *view* of XML data
- Evaluates an XQuery “row pattern” with optional arguments (as with XMLQuery)
- Element/attribute values mapped to columns using XQuery “column patterns”
- Names & types of columns required; default values optional