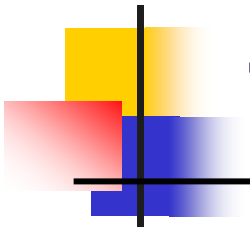
Java, Databases, and XML Query Languages

Jonathan Robie

DataDirect Technologies

jonathan.robie@datadirect.com



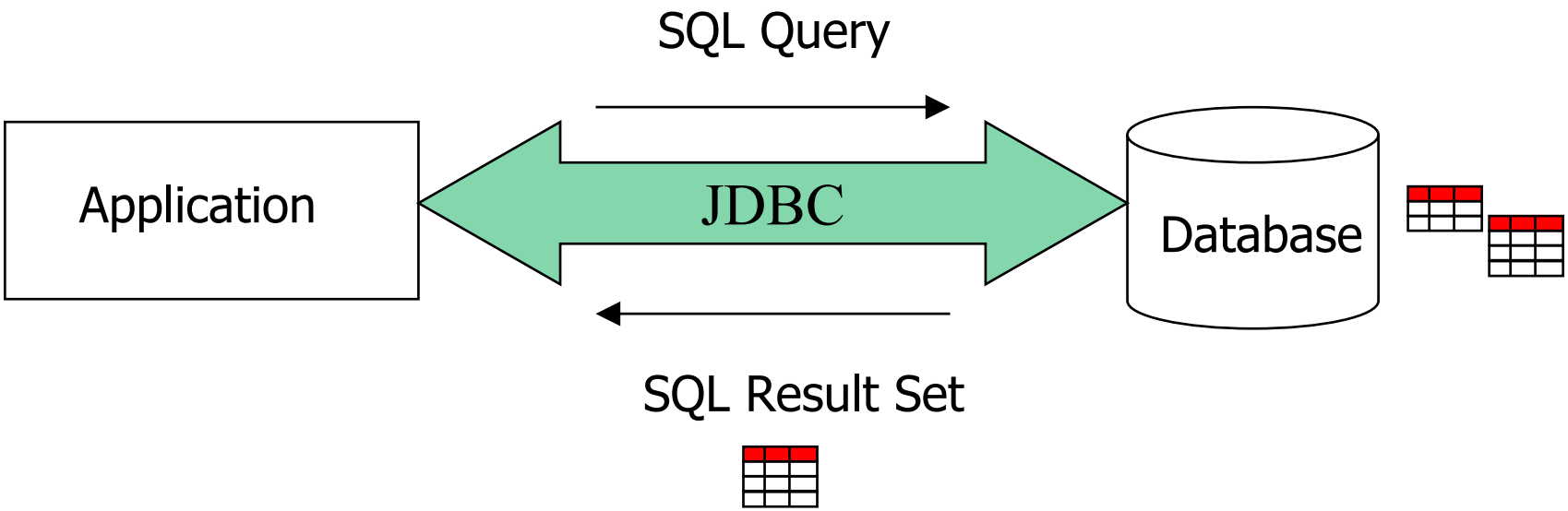
The Need for XML Queries

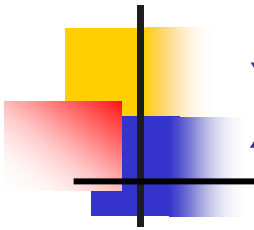
- Much of the content on Web pages is the result of queries
  - This data is structured hierarchically on Web pages, not as a set of tables that the user should join together mentally.
  - The result of a relational query is always a two dimensional table.
- Most Web message payloads are hierarchically structured
  - These formats are almost never relational.
  - To exchange data, you have to use the agreed format.
- In an XML query, the result of the query is XML.
  - Must be able to combine the data from a database to produce an infinite number of representations
  - Must be able to structure results in any required XML structure
  - Must be able to bridge the logical mismatch between relational (only unordered two-dimensional tables) and XML (only hierarchy and sequence)

The Need for XML Query Standards

- XML was designed to be platform independent.
- Java and JDBC were also designed for platform independence.
- Emerging Problems
  - Cross platform programming for XML can be done with JDBC+DOM+Java, but is tedious and difficult to maintain.
  - Proprietary XML extensions from relational vendors work on only one database, and vary in quality.
- Emerging Standards
  - SQL/XML and XQuery allow queries to create XML – they are powerful and flexible.
  - XQuery allows queries to span multiple sources, and can support XML views of relational data.

Java and Relational (JDBC)





XML Is Not Relational!

Customers

CustId	Name	Address
1	Woodworks	Baltimore
2	Software Solutions	Boston
3	Food Supplies	New York
4	Hardware Store	Washington
5	Books Inc.	New Orleans
…		

Projects

ProjId	Name	CustId
1	Medusa	1
2	Pegasus	4
3	Python	6
…	…	…
8	Typhon	4
…		

```
<?xml version="1.0" encoding="UTF-8"?>
<customers>
  <customer id="1">
    <name>Woodworks</name>
    <address>Baltimore</address>
      <projects>
        <project id="1"><name>Medusa</name></project>
      </projects>
  </customer>
…
  <customer id="4">
    <name>Hardware Store</name>
    <address>Washington</address>
      <projects>
        <project id="2"><name>Pegasus</name></project>
        <project id="8"><name>Typhon</name></project>
      </projects>
  </customer>
…
</customers>
```

XML Is Not Relational! *(Continued)*

Customers

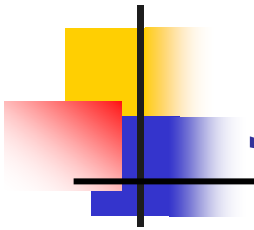
CustId	Name	Address
1	Woodworks	Baltimore
2	Software Solutions	Boston
3	Food Supplies	New York
4	Hardware Store	Washington
5	Books Inc.	New Orleans
…		

Projects

ProjId	Name	CustId
1	Medusa	1
2	Pegasus	4
3	Python	6
…	…	…
8	Typhon	4
…		

```
<customer id="4">
    <name>Hardware Store</name>
    <address>Washington</address>
      <projects>
        <project id="2"><name>Pegasus</name></project>
        <project id="8"><name>Typhon</name></project>
      </projects>
</customer>
```

- Traditional Relational
  - Two dimensional tables
  - No Hierarchy
  - No Sequence
  - Relationships represented using joins on data
- XML
  - Primary relationships
    - Hierarchy
    - Sequence



JDBC + DOM + Java

- JDBC connection to relational database
- DOM, SAX, XPath to process XML
- Java code

JDBC + DOM + Java *(Continued)*

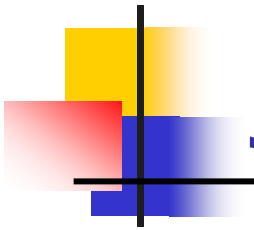
```
// Create XML document
DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
DocumentBuilder db=dbf.newDocumentBuilder();
doc=db.newdoc();

// Create the root element
Element root=doc.createElement("customers");
doc.appendChild(root);

// Create prepared statements
PreparedStatement stmt1=con.prepareStatement(
  "select c.CustId,c.Name,c.Address from Customers c");
PreparedStatement stmt2=con.prepareStatement(
  "select p.ProjId, p.Name from Projects p where p.CustId=?");

// Result set for customer information
ResultSet rs1=stmt1.executeQuery();
```

JDBC + DOM + Java *(Continued)*

```
// Create XML document
DocumentBuilderFactory dbf=DocumentBuilderFactory.newInstance();
DocumentBuilder db=dbf.newDocumentBuilder();
doc=db.newdoc();

// Create the root element
Element root=doc.createElement("customers");
doc.appendChild(root);


// Create prepared statements
PreparedStatement stmt1=con.prepareStatement(
  "select c.CustId,c.Name,c.Address from Customers c");

PreparedStatement stmt2=con.prepareStatement(
  "select p.ProjId, p.Name from Projects p where p.CustId=?");

// Result set for customer information
ResultSet rs1=stmt1.executeQuery();
```

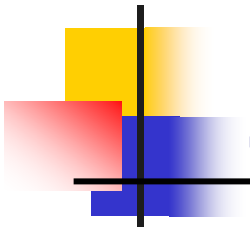
JDBC + DOM + Java *(Continued)*

```
// For all selected customers
while (rs1.next()) {
  // Retrieve and add customer information
  Element customer=addCustomer(rs1,root);

  // "projects" child container element
  Element projects=doc.createElement("projects");
  customer.appendChild(projects);

  // Create result set for project information
  stmt2.setInt(1,Integer.parseInt(customer.getAttribute("id")));
  ResultSet rs2=stmt2.executeQuery();

  // For all selected projects
  while (rs2.next()) {
    // Retrieve and add project information
    Element project=addProject(rs2,projects);
    }
  rs2.close();
  }
rs1.close();
```



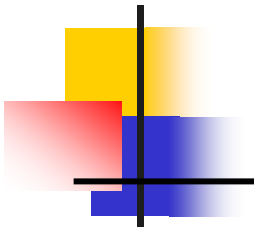
JDBC + DOM + Java *(Continued)*

```
// 'Print' DOM document (Xerces version)
OutputFormat of=new OutputFormat();
of.setIndent(2);
Writer ow=(Writer)new OutputStreamWriter(System.out);
XMLSerializer xmlS= new XMLSerializer(ow,of);
xmlS.serialize(doc);

…
```



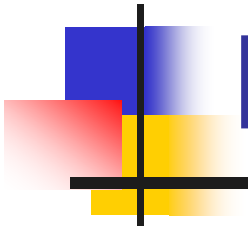
JDBC + DOM + Java *(Continued)*

- A lot of code and not trivial to maintain
- Optimizing the implementation is difficult
- Works for all RDBMS

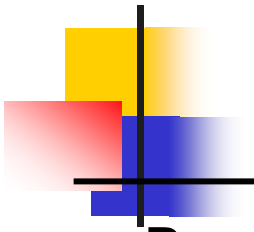


Proprietary Database Extensions

- DB2 XML Extender
- Oracle
- Microsoft
- Others – not discussed here
- Each works only for one vendor's database (and not for all releases)



DB2 XML Extender



DB2 XML Extender

- Based on DB2's generic extender architecture
- A DB2 Extender implements
  - user defined datatypes (UDT)
  - associated user defined functions (UDF)
- DB2 XML Extender UDT's
  - XMLVarchar, XMLClob and XMLFile
- UDF's to manipulate the UDT's using two modes
  - Column mode
  - Collection mode
- SOAP toolkit

DB2 XML Extender *(Continued)*

- Column mode
  - Stores complete XML documents
  - Support for (indexed) "side" columns
- Collection mode
  - Composes XML out of "normal" tabular information
  - Decomposes XML into "normal" tabular information
  - Document Access Definition (DAD) defines mapping
    - RDB node mapping
      - ✓ table/column references mapped to XML structures
    - SQL mapping
      - ✓ SQL select expressions mapped to XML structures

DB2 XML Extender *(Continued)*

DAD file (collection mode with RDB node)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DAD SYSTEM "dad.dtd">
<DAD>
  <validation>YES</validation>
  <Xcollection>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE customers SYSTEM "customers.dtd"</doctype>
    <root_node>
      <element_node name="customers">
        <element_node name="customer">
          <RDB_node>
            <table name="Customers" key="CustId" />
            <table name="Projects" key="ProjId" />
            <condition>
              Customers.CustId=Projects.CustId
            </condition>
          </RDB_node>
          <attribute_node name="id">
            <RDB_node>
              <table name="Customers" />
              <column name="CustId" type="integer"/>
            </RDB_node>
            <RDB_node>
              <table name="Projects" />
              <column name="CustId" type="integer"/>
            </RDB_node>
            </text_node>
```

DB2 XML Extender *(Continued)*

DAD file (collection mode with RDB node)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DAD SYSTEM "dad.dtd">
<DAD>
  <validation>YES</validation>
  <Xcollection>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE customers SYSTEM "customers.dtd"</doctype>
    <root_node>
      <element_node name="customers">
        <element_node name="customer">
          <RDB_node>
            <table name="Customers" key="CustId" />
            <table name="Projects" key="ProjId" />
            <condition>
              Customers.CustId=Projects.CustId
            </condition>
          </RDB_node>
          <attribute_node name="id">
            <RDB_node>
              <table name="Customers" />
              <column name="CustId" type="integer"/>
            </RDB_node>
            <RDB_node>
              <table name="Projects" />
              <column name="CustId" type="integer"/>
            </RDB_node>
          </root_node>
```

DB2 XML Extender *(Continued)*

DAD file (collection mode with RDB node)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DAD SYSTEM "dad.dtd">
<DAD>
  <validation>YES</validation>
  <Xcollection>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE customers SYSTEM "customers.dtd"</doctype>
    <root_node>
      <element_node name="customers">
        <element_node name="customer">
          <RDB_node>
            <table name="Customers" key="CustId" />
            <table name="Projects" key="ProjId" />
            <condition>
              Customers.CustId=Projects.CustId
            </condition>
          </RDB_node>
          <attribute_node name="id">
            <RDB_node>
              <table name="Customers" />
              <column name="CustId" type="integer"/>
            </RDB_node>
            <RDB_node>
              <table name="Projects" />
              <column name="CustId" type="integer"/>
            </RDB_node>
          </attribute_node>
```

DB2 XML Extender *(Continued)*

DAD file (collection mode with RDB node)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DAD SYSTEM "dad.dtd">
<DAD>
  <validation>YES</validation>
  <Xcollection>
    <prolog>?xml version="1.0"?</prolog>
    <doctype>!DOCTYPE customers SYSTEM "customers.dtd"</doctype>
    <root_node>
      <element_node name="customers">
        <element_node name="customer">
          <RDB_node>
            <table name="Customers" key="CustId" />
            <table name="Projects" key="ProjId" />
            <condition>
              Customers.CustId=Projects.CustId
            </condition>
          </RDB_node>
          <attribute_node name="id">
            <RDB_node>
              <table name="Customers" />
              <column name="CustId" type="integer"/>
            </RDB_node>
            <RDB_node>
              <table name="Projects" />
              <column name="CustId" type="integer"/>
            </RDB_node>
          </attribute_node>
```

DB2 XML Extender *(Continued)*

```
<element_node name="name" >
  <text_node>
    <RDB_node>
      <table name="Customers" />
      <column name="Name" type="varchar(50)"/>
    </RDB_node>
  </text_node>
</element_node>
<element_node name="address" >
  <text_node>
    <RDB_node>
      <table name="Customers" />
      <column name="Address" type="varchar(50)"/>
    </RDB_node>
  </text_node>
</element_node>
<element_node name="projects">
  <element_node name="project">
    <attribute_node name="id">
      <RDB_node>
        <table name="Projects" />
        <column name="ProjId" type="integer"/>
      </RDB_node>
    </attribute_node>
```

DB2 XML Extender *(Continued)*

```
<element_node name="name" >
  <text_node>
    <RDB_node>
      <table name="Customers" />
      <column name="Name" type="varchar(50)"/>
    </RDB_node>
  </text_node>
</element_node>
<element_node name="address" >
  <text_node>
    <RDB_node>
      <table name="Customers" />
      <column name="Address" type="varchar(50)"/>
    </RDB_node>
  </text_node>
</element_node>
<element_node name="projects">
  <element_node name="project">
    <attribute_node name="id">
      <RDB_node>
        <table name="Projects" />
        <column name="ProjId" type="integer"/>
      </RDB_node>
    </attribute_node>
```

DB2 XML Extender *(Continued)*

```
<element_node name="name" >
  <text_node>
    <RDB_node>
      <table name="Customers" />
      <column name="Name" type="varchar(50)"/>
    </RDB_node>
  </text_node>
</element_node>
<element_node name="address" >
  <text_node>
    <RDB_node>
      <table name="Customers" />
      <column name="Address" type="varchar(50)"/>
    </RDB_node>
  </text_node>
</element_node>
<element_node name="projects">
  <element_node name="project">
    <attribute_node name="id">
      <RDB_node>
        <table name="Projects" />
        <column name="ProjId" type="integer"/>
      </RDB_node>
    </attribute_node>
```

DB2 XML Extender *(Continued)*

```
            <element_node name="name" >
              <text_node>
                <RDB_node>
                  <table name="Projects" />
                  <column name="Name" type="varchar(50)"/>
                </RDB_node>
              </text_node>
            </element_node>
          </element_node>
        </element_node>
      </element_node>
    </root_node>
  </Xcollection>
</DAD>
```

DB2 XML Extender *(Continued)*

```
            <element_node name="name" >
              <text_node>
                <RDB_node>
                  <table name="Projects" />
                  <column name="Name" type="varchar(50)"/>
                </RDB_node>
              </text_node>
            </element_node>
          </element_node>
        </element_node>
      </element_node>
    </root_node>
  </Xcollection>
</DAD>
```

DB2 XML Extender *(Continued)*

Code

```
...
// DB2 XML Extender stored procedure
String sql="{call db2xml.dxxShredXML(?,?,?,?)}";
CallableStatement stmt=con.prepareCall(sql);

// Input parameters
stmt.setString(1,readFile("customers.dad"));
stmt.setString(2,readFile("customers.xml"));

// Register the output parameters
stmt.registerOutParameter(3,Types.INTEGER);
stmt.registerOutParameter(4,Types.VARCHAR);

// Execute DB2 XML Extender stored procedure
stmt.execute();

// User feedback
System.out.println("returnCode="+stmt.getInt(3));
System.out.println("returnMsg="+stmt.getString(4));
```

DB2 XML Extender *(Continued)*

Code

```
...
// DB2 XML Extender stored procedure
String sql="{call db2xml.dxxShredXML(?,?,?,?)}";
CallableStatement stmt=con.prepareCall(sql);

// Input parameters
stmt.setString(1,readFile("customers.dad"));
stmt.setString(2,readFile("customers.xml"));

// Register the output parameters
stmt.registerOutParameter(3,Types.INTEGER);
stmt.registerOutParameter(4,Types.VARCHAR);

// Execute DB2 XML Extender stored procedure
stmt.execute();

// User feedback
System.out.println("returnCode="+stmt.getInt(3));
System.out.println("returnMsg="+stmt.getString(4));
```

DB2 XML Extender *(Continued)*

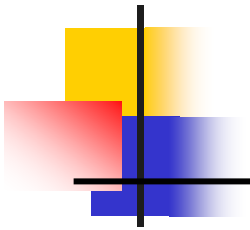
Code

```
...
// DB2 XML Extender stored procedure
String sql="{call db2xml.dxxShredXML(?,?,?,?)}";
CallableStatement stmt=con.prepareCall(sql);

// Input parameters
stmt.setString(1,readFile("customers.dad"));
stmt.setString(2,readFile("customers.xml"));

// Register the output parameters
stmt.registerOutParameter(3,Types.INTEGER);
stmt.registerOutParameter(4,Types.VARCHAR);

// Execute DB2 XML Extender stored procedure
stmt.execute();

// User feedback
System.out.println("returnCode="+stmt.getInt(3));
System.out.println("returnMsg="+stmt.getString(4));
```

DB2 XML Extender *(Continued)*

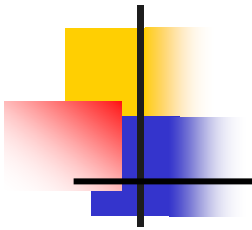
Code

```
...
// DB2 XML Extender stored procedure
String sql="{call db2xml.dxxShredXML(?,?,?,?)}";
CallableStatement stmt=con.prepareCall(sql);

// Input parameters
stmt.setString(1,readFile("customers.dad"));
stmt.setString(2,readFile("customers.xml"));

// Register the output parameters
stmt.registerOutParameter(3,Types.INTEGER);
stmt.registerOutParameter(4,Types.VARCHAR);

// Execute DB2 XML Extender stored procedure
stmt.execute();

// User feedback
System.out.println("returnCode="+stmt.getInt(3));
System.out.println("returnMsg="+stmt.getString(4));
```



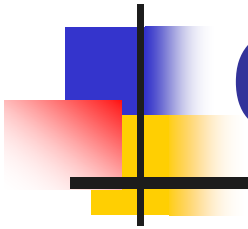
DB2 XML Extender *(Continued)*

- User Defined Functions
  - Generate XML
    - dxxGenXML, dxxRetrieveXML
  - Insert XML
    - dxxShredXML, dxxInsertXML
  - Administrative functions

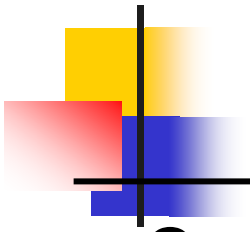


DB2 XML Extender *(Continued)*

- Flexible and symmetric mapping
- Easy to use "stored procedure approach"
- DAD file is not trivial to edit and maintain
- GUI DAD editor in WebSphere

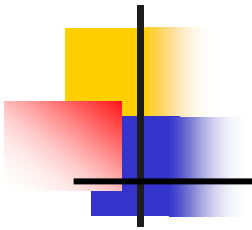


Oracle



Oracle

- Oracle 8i
  - XML Developer Kit (XDK)
    - XML Parser
    - XSLT
    - XML SQL Utility (XSU)
    - XSQL Servlet
    - SOAP
    - …
- Oracle 9i1
  - XML data type
- Oracle 9i2
  - XML database (XDB)



Oracle *(Continued)*

- XML SQL Utility
  - Part of XDK
    - "outside" database (Java,C++)
    - "inside" database (PL/SQL wrappers for Java classes)
  - "Canonical" XML-(O)RDBMS mapping
    - Transforms data retrieved from (O)RDBMS to XML
    - Inserts XML information into database
    - Updates/deletes database data with XML data
  - XSLT "integration"

Oracle *(Continued)*

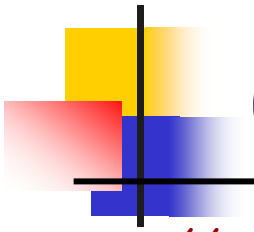
Example

```
// Build query in StringBuffer
StringBuffer sb=new StringBuffer();
sb.append("SELECT ");
sb.append("  c.CustId as \"@id\", ");
sb.append("  c.Name, ");
sb.append("  c.Address, ");
sb.append("  cursor (  ");
sb.append("    SELECT ");
sb.append("      p.ProjId as \"@id\", ");
sb.append("      p.Name");
sb.append("    FROM Projects p ");
sb.append("    WHERE p.CustId=c.CustId ) AS projects ");
sb.append(" FROM Customers c ");
```

Oracle *(Continued)*

Example

```
// Build query in StringBuffer
StringBuffer sb=new StringBuffer();
sb.append("SELECT ");
sb.append("  c.CustId as \"@id\", ");
sb.append("  c.Name, ");
sb.append("  c.Address, ");
sb.append("  cursor (  ");
sb.append("    SELECT ");
sb.append("      p.ProjId as \"@id\", ");
sb.append("      p.Name");
sb.append("    FROM Projects p ");
sb.append("    WHERE p.CustId=c.CustId ) AS projects ");
sb.append(" FROM Customers c ");
```



Oracle *(Continued)*

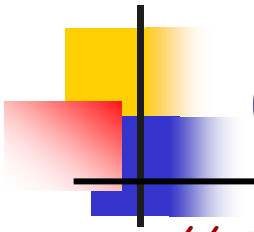
```
// XSU object
OracleXMLQuery q=new OracleXMLQuery(con,sb.toString());

// Result "customization"
q.setRowsetTag("customers");
q.setRowTag("customer");
q.useLowerCaseTagNames();
q.setRowIdAttrName("");

// Stylesheet for result "clean-up"
q.setXSLT("q.xslt",null);

// Generate result document and print
Document resDoc=q.getXMLDOM();
XMLDocument oraDoc=(XMLDocument)resDoc;
oraDoc.print(System.out);
```

Oracle *(Continued)*

```
// XSU object
OracleXMLQuery q=new OracleXMLQuery(con,sb.toString());

// Result "customization"
q.setRowsetTag("customers");
q.setRowTag("customer");
q.useLowerCaseTagNames();
q.setRowIdAttrName("");

// Stylesheet for result "clean-up"
q.setXSLT("q.xslt",null);

// Generate result document and print
Document resDoc=q.getXMLDOM();
XMLDocument oraDoc=(XMLDocument)resDoc;
oraDoc.print(System.out);
```

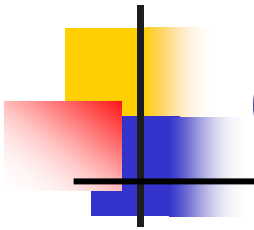
Oracle *(Continued)*

```
// XSU object
OracleXMLQuery q=new OracleXMLQuery(con,sb.toString());

// Result "customization"
q.setRowsetTag("customers");
q.setRowTag("customer");
q.useLowerCaseTagNames();
q.setRowIdAttrName("");

// Stylesheet for result "clean-up"
q.setXSLT("q.xslt",null);

// Generate result document and print
Document resDoc=q.getXMLDOM();
XMLDocument oraDoc=(XMLDocument)resDoc;
oraDoc.print(System.out);
```



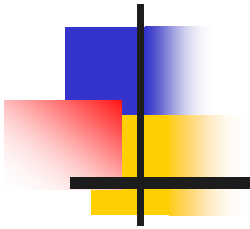
Oracle *(Continued)*

```
// XSU object
OracleXMLQuery q=new
   OracleXMLQuery(con,sb.toString());

// Result "customization"
q.setRowsetTag("customers");
q.setRowTag("customer");
q.useLowerCaseTagNames();
q.setRowIdAttrName("");

// Stylesheet for result "clean-up"
q.setXSLT("q.xslt",null);

// Generate result document and print
Document resDoc=q.getXMLDOM();
XMLDocument oraDoc=(XMLDocument)resDoc;
oraDoc.print(System.out);
```

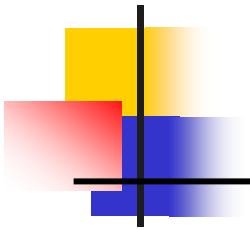


Oracle *(Continued)*

- Flexible bidirectional mappings relying on Oracle's object-relational type system
- Easy XSLT integration
- Nice accompanying toolset (*e.g.* XSQL)
- Oracle 9i2 == XML database (?)



Microsoft SQL Server



Microsoft SQL Server

- SQLXML 3.0 (SQL Server 2000 add on)
  - XML formatting (server or client) with "FOR XML"
  - XML consumption using "OPENXML"
  - XML views using annotated XSD schemas
    - XPath support
    - Diffgrams for updates
  - SOAP support
  - Note: SQLXML ≠ SQL/XML

Microsoft SQL Server *(Continued)*

- **SELECT ... FOR XML**

```
FOR XML
   [RAW|AUTO|EXPLICIT]
   [, XMLDATA]
   [, ELEMENTS]
   [, BINARY BASE64]
```

  - "RAW" and "AUTO" use canonical mapping
  - "EXPLICIT" – user defined mapping
  - "XMLDATA" – generate an XDR schema
  - "ELEMENTS" – create sub-elements iso attributes
  - "BINARY BASE64" – use base64 encoding for binary (default hex encoding)

Microsoft SQL Server *(Continued)*

- **OPENXML**
  - XML is passed as "handle" (*idoc*)
  - Creates rowset view of any XML document
  - XPath defines "rows"
  - XPath or "reference table" defines "columns"

```
OPENXML (
     idoc int [in],
     rowpattern nvarchar[in],
     [flags byte[in]] )
  [WITH(SchemaDeclaration|TableName)]
```

Microsoft SQL Server *(Continued)*

OPENXML Example

```
// Build Transact-SQL statement
StringBuffer sb=new StringBuffer();
// Variable declarations
sb.append("DECLARE @idoc int\n");
sb.append("DECLARE @doc varchar(3000)\n");

// Define XML document parameter and parse
sb.append("SET @doc=?\n");
sb.append("EXEC sp_xml_preparedocument @idoc OUTPUT,@doc\n");

// Update Customers from XML
sb.append("update Customers\n");
sb.append("  set Customers.Name=u.Name,Customers.Address=u.Address\n");
sb.append("FROM OPENXML (@idoc,'/customers/customer',2) \n");
sb.append("WITH  (CustId integer      '@id',\n");
sb.append("       Name varchar(50)    'name/text()',\n");
sb.append("       Address varchar(50) 'address/text()') u\n");
sb.append("where Customers.CustId=u.CustId\n");
```

Microsoft SQL Server *(Continued)*

OPENXML Example

```
// Build Transact-SQL statement
StringBuffer sb=new StringBuffer();
// Variable declarations
sb.append("DECLARE @idoc int\n");
sb.append("DECLARE @doc varchar(3000)\n");

// Define XML document parameter and parse
sb.append("SET @doc=?\n");
sb.append("EXEC sp_xml_preparedocument @idoc OUTPUT,@doc\n");

// Update Customers from XML
sb.append("update Customers\n");
sb.append("  set Customers.Name=u.Name,Customers.Address=u.Address\n");
sb.append("FROM OPENXML (@idoc,'/customers/customer',2) \n");
sb.append("WITH  (CustId integer      '@id',\n");
sb.append("       Name varchar(50)    'name/text()',\n");
sb.append("       Address varchar(50) 'address/text()') u\n");
sb.append("where Customers.CustId=u.CustId\n");
```

Microsoft SQL Server *(Continued)*

OPENXML Example

```
// Build Transact-SQL statement "batch"
StringBuffer sb=new StringBuffer();
// Variable declarations
sb.append("DECLARE @idoc int\n");
sb.append("DECLARE @doc varchar(3000)\n");

// Define XML document parameter and parse
sb.append("SET @doc=?\n");
sb.append("EXEC sp_xml_preparedocument @idoc OUTPUT,@doc\n");

// Update Customers from XML
sb.append("update Customers\n");
sb.append("  set Customers.Name=u.Name,Customers.Address=u.Address\n");
sb.append("FROM OPENXML (@idoc,'/customers/customer',2) \n");
sb.append("WITH  (CustId integer       '@id',\n");
sb.append("       Name varchar(50)     'name/text()',\n");
sb.append("       Address varchar(50) 'address/text()') u\n");
sb.append("where Customers.CustId=u.CustId\n");
```

Microsoft SQL Server *(Continued)*

OPENXML Example

```
// Build Transact-SQL statement "batch"
StringBuffer sb=new StringBuffer();
// Variable declarations
sb.append("DECLARE @idoc int\n");
sb.append("DECLARE @doc varchar(3000)\n");

// Define XML document parameter and parse
sb.append("SET @doc=?\n");
sb.append("EXEC sp_xml_preparedocument @idoc OUTPUT,@doc\n");

// Update Customers from XML
sb.append("update Customers\n");
sb.append("  set Customers.Name=u.Name,Customers.Address=u.Address\n");
sb.append("FROM OPENXML (@idoc,'/customers/customer',2) \n");
sb.append("WITH  (CustId integer       '@id',\n");
sb.append("       Name varchar(50)     'name/text()',\n");
sb.append("       Address varchar(50) 'address/text()') u\n");
sb.append("where Customers.CustId=u.CustId\n");
```

Microsoft SQL Server *(Continued)*

```
// Update Projects from XML
sb.append("update Projects\n");
sb.append("  set Projects.Name=u.Name\n");
sb.append("FROM OPENXML (@idoc, '/customers/customer/projects/project',2)\n");
sb.append("WITH  (CustId integer '../../@id', ProjId integer '@id', \n ");
sb.append("       Name varchar(50) 'name/text()' ) u \n");
sb.append("where Projects.ProjId=u.ProjId \n");
// Release 'parsed' XML document
sb.append("EXEC sp_xml_removedocument @idoc\n");

// Create statement and set bind marker value
PreparedStatement stmt=con.prepareStatement(sb.toString());
stmt.setString(1,readFile("example.xml"));

// Execute
stmt.execute();
int rowCount;
// Loop through results
do {
  rowCount=stmt.getUpdateCount();
  if(rowCount >= 0) {
    System.out.println("Updated:"+rowCount);
    stmt.getMoreResults();
  }
}
while( rowCount>=0 );
```

Microsoft SQL Server *(Continued)*

```
// Update Projects from XML
sb.append("update Projects\n");
sb.append("  set Projects.Name=u.Name\n");
sb.append("FROM OPENXML (@idoc, '/customers/customer/projects/project',2)\n");
sb.append("WITH  (CustId integer '../../@id', ProjId integer '@id',\n ");
sb.append("       Name varchar(50) 'name/text()' ) u\n");
sb.append("where Projects.ProjId=u.ProjId\n");
// Release 'parsed' XML document
sb.append("EXEC sp_xml_removedocument @idoc\n");

// Create statement and set bind marker value
PreparedStatement stmt=con.prepareStatement(sb.toString());
stmt.setString(1,readFile("example.xml"));

// Execute
stmt.execute();
int rowCount;
// Loop through results
do {
  rowCount=stmt.getUpdateCount();
  if(rowCount >= 0) {
    System.out.println("Updated:"+rowCount);
    stmt.getMoreResults();
  }
} while( rowCount>=0 );
```

Microsoft SQL Server *(Continued)*

```
// Update Projects from XML
sb.append("update Projects\n");
sb.append("  set Projects.Name=u.Name\n");
sb.append("FROM OPENXML (@idoc, '/customers/customer/projects/project',2)\n");
sb.append("WITH  (CustId integer '../../@id', ProjId integer '@id',\n ");
sb.append("       Name varchar(50) 'name/text()' ) u\n");
sb.append("where Projects.ProjId=u.ProjId\n");
// Release 'parsed' XML document
sb.append("EXEC sp_xml_removedocument @idoc\n");

// Create statement and set bind marker value
PreparedStatement stmt=con.prepareStatement(sb.toString());
stmt.setString(1,readFile("example.xml"));

// Execute
stmt.execute();
int rowCount;
// Loop through results
do {
  rowCount=stmt.getUpdateCount();
  if(rowCount >= 0) {
    System.out.println("Updated:"+rowCount);
    stmt.getMoreResults();
  }
} while( rowCount>=0 );
```

Microsoft SQL Server *(Continued)*

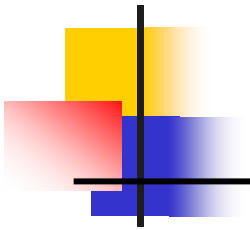
```
// Update Projects from XML
sb.append("update Projects\n");
sb.append("  set Projects.Name=u.Name\n");
sb.append("FROM OPENXML (@idoc, '/customers/customer/projects/project',2)\n");
sb.append("WITH  (CustId integer '../../@id', ProjId integer '@id',\n ");
sb.append("       Name varchar(50) 'name/text()' ) u\n");
sb.append("where Projects.ProjId=u.ProjId\n");
// Release 'parsed' XML document
sb.append("EXEC sp_xml_removedocument @idoc\n");

// Create statement and set bind marker value
PreparedStatement stmt=con.prepareStatement(sb.toString());
stmt.setString(1,readFile("example.xml"));

// Execute
stmt.execute();
int rowCount;
// Loop through results
do {
  rowCount=stmt.getUpdateCount();
  if(rowCount >= 0) {
    stmt.getMoreResults();
  }
} while(rowCount >= 0);
```

Microsoft SQL Server *(Continued)*

```
// Update Projects from XML
sb.append("update Projects\n");
sb.append("  set Projects.Name=u.Name\n");
sb.append("FROM OPENXML (@idoc,
   '/customers/customer/projects/project',2)\n");
sb.append("WITH  (CustId integer '../../@id', ProjId integer
   '@id',\n ");
sb.append("       Name varchar(50) 'name/text()' ) u\n");
sb.append("where Projects.ProjId=u.ProjId\n");
// Release 'parsed' XML document
sb.append("EXEC sp_xml_removedocument @idoc\n");

// Create statement and set bind marker value
PreparedStatement stmt=con.prepareStatement(sb.toString());
stmt.setString(1,readFile("example.xml"));
```

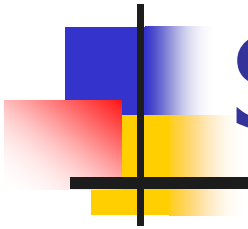
Microsoft SQL Server *(Continued)*

```
// Execute
stmt.execute();
int rowCount;
// Loop through results
do {
  rowCount=stmt.getUpdateCount();
  if(rowCount >= 0) {
    stmt.getMoreResults();
    }
  }
while(rowCount >= 0);
```

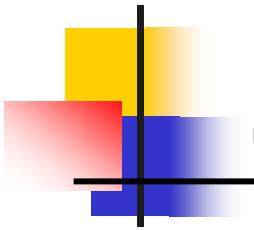


Microsoft SQL Server *(Continued)*

- XML retrieval
  - Easy to use "canonical" mapping using FOR XML RAW | AUTO
  - Difficult to use flexible mapping using FOR XML EXPLICIT
- XML "shredding"
  - Easy to use and powerful OPENXML
- Not so Java friendly
  - XML views and related diffgrams



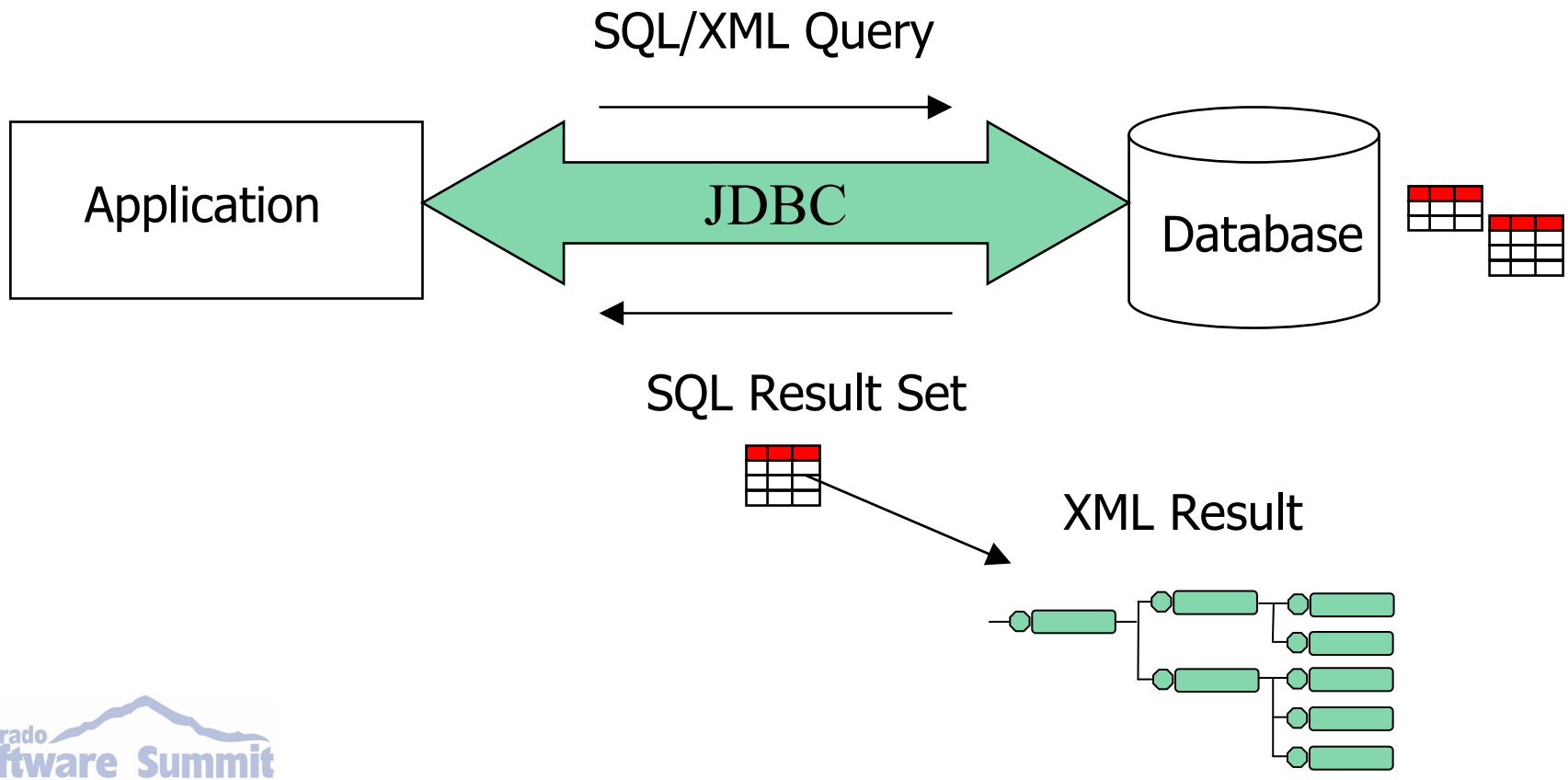
SQL/XML

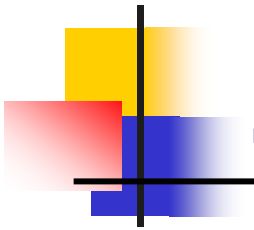


SQL/XML

- Part of SQL 2003
- Defines XML extensions for SQL
- ISO, ANSI, INCITS H2.3
- Standard expected during 2003
- Implementations from Oracle, IBM
- Friendly for SQL programmers
- Basis for XQuery views of relational data

SQL/XML and JDBC





SQL/XML

- XML data type
- Mapping rules (schema and instance)
- Extensions to the select statement syntax specify how to construct XML from a (tabular) select result set
  - xmlelement
  - xmlattributes
  - xmlconcat
  - xmlforest
  - xmlagg

SQL/XML – Mapping

Customers

CustId	Name	Address
1	Woodworks	Baltimore
2	Software Solutions	Boston
3	Food Supplies	New York
4	Hardware Store	Washington
5	Books Inc.	New Orleans
6	Photo Shop	Dallas
7	Software Solutions	Sant Jose
8	Computer Supplies	Chicago
9	Eastern Publishers	New York
10	Car Supplies	Los Angeles
11	Health Insurances	Seattle
12	Bank Lucerne	San Francisco
13	Marlington Hotels	New York
14	US Bakeries	Columbia
15	Entertainment Inc	Orlando

```
<Customers>
  <row>
    <CustId>1</CustId>
    <Name>Woodworks</Name>
    <Address>Baltimore</Address>
  </row>
  <row>
    <CustId>2</CustId>
    <Name>Software Solutions</Name>
    <Address>Boston</Address>
  </row>
 <row>
    <CustId>3</CustId>
    <Name>Food Supplies</Name>
    <Address>New York</Address>
  </row>
  ...
</Customers>
```

SQL/XML – xmlelement()

```
select
  c.CustId,
  xmlelement(name customer,
    xmlelement(name name,c.Name),
    xmlelement(name address,c.Address)) as CustInfo
from Customers c
```

CustId	CustInfo
1	<customer> <name>Woodworks</name> <address>Baltimore</address> </customer>
2	<customer> <name>Software Solutions</name> <address>Boston</address> </customer>
…	…

SQL/XML – xmlelement()

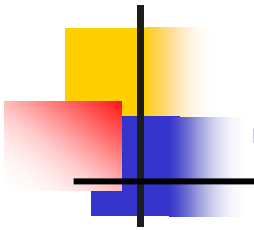
```
select
  c.CustId,
  xmlelement(name customer,
   xmlattributes(c.CustId as id,c.Name,c.Address)) as
   CustInfo
from Customers c
```

CustId	CustInfo
1	<customer id="1" Name="Woodworks" Address="Baltimore"/>
2	<customer id="2" Name="Software Solutions" Address="Boston"/>
…	…

SQL/XML – xmlelement()

```
select
  c.CustId,
  xmlconcat(
    xmlelement(name id,c.CustId),
    xmlelement(name name,c.Name),
    xmlelement(name address,c.Address)) as CustInfo
from Customers c
```

CustId	CustInfo
1	<id>1</id> <name>Woodworks</name> <address>Baltimore</address>
2	<id>2</id> <name>Software Solutions</name> <address>Boston</address>
…	…



SQL/XML

```
select
  c.CustId,
  xmlelement(name customer,
    xmlattributes(c.CustId as id,c.Name as name),
    xmlelement(name projects,
    (select xmlagg(xmlelement(name project,
              xmlattributes(p.ProjId as id,p.Name as name)))
     from Projects p
     where p.CustId=c.CustId))) as "customer-projects"
from Customers c
```

CustId	customer-projects
1	<customer id="1" name="Woodworks"> <projects> <project id="1" name="Medusa"/> </projects> </customer>
…	…
4	<customer id="4" name="Hardware Store"> <projects> <project id="2" name="Pegasus"/> <project id="8" name="Typhon"/> </projects> </customer>
…	…

SQL/XML – JDBC Example

```
...
String sqlStr=
  new String ("select c.CustId,xmlelement(name ....\n");
Statement stat=con.createStatement();

ResultSet rs=stat.executeQuery(sqlStr);

while(rs.next())
  {
  int id=rs.getInt(1);
  com.ddtek.jdbc.jxtr.XMLType xmlC=
    (com.ddtek.jdbc.jxtr.XMLType)rs.getObject(2);
  org.w3c.dom.Document doc=xmlC.getDOM();
  ...
  }
...
```

SQL/XML – JDBC Example

```
...
String sqlStr=
  new String ("select c.CustId,xmlelement(name ....\n");
Statement stat=con.createStatement();

ResultSet rs=stat.executeQuery(sqlStr);

while(rs.next())
  {
  int id=rs.getInt(1);
  com.ddtek.jdbc.jxtr.XMLType xmlC=
    (com.ddtek.jdbc.jxtr.XMLType)rs.getObject(2);
  org.w3c.dom.Document doc=xmlC.getDOM();
  ...
  }
...
```

SQL/XML – JDBC Example

```
...
String sqlStr=
  new String ("select c.CustId,xmlelement(name ....\n");
Statement stat=con.createStatement();

ResultSet rs=stat.executeQuery(sqlStr);

while(rs.next())
  {
  int id=rs.getInt(1);
  com.ddtek.jdbc.jxtr.XMLType xmlC=
    (com.ddtek.jdbc.jxtr.XMLType)rs.getObject(2);
  org.w3c.dom.Document doc=xmlC.getDOM();
  ...
  }
...
```

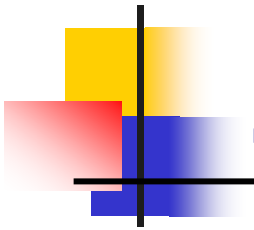
SQL/XML – JDBC Example

```
...
String sqlStr=
  new String ("select c.CustId,xmlelement(name ....\n");
Statement stat=con.createStatement();

ResultSet rs=stat.executeQuery(sqlStr);

while(rs.next())
  {
  int id=rs.getInt(1);
  com.ddtek.jdbc.jxtr.XMLType xmlC=
    (com.ddtek.jdbc.jxtr.XMLType)rs.getObject(2);
  org.w3c.dom.Document doc=xmlC.getDOM();
  ...
  }
...
```

XML Type

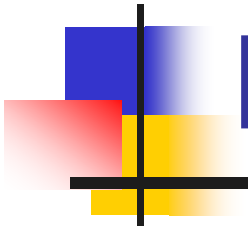
```
...
String sqlStr=
  new String ("select c.CustId,xmlelement(name ....\n");
Statement stat=con.createStatement();

ResultSet rs=stat.executeQuery(sqlStr);

while(rs.next())
   {
   int id=rs.getInt(1);
   com.ddtek.jdbc.jxtr.XMLType xmlC=
     (com.ddtek.jdbc.jxtr.XMLType)rs.getObject(2);
   org.w3c.dom.Document doc=xmlC.getDOM();
   ...
   }
...
```

The XML type is part of SQL 2003 and is anticipated in JDBC 4.0. For now, a proprietary implementation must be used.



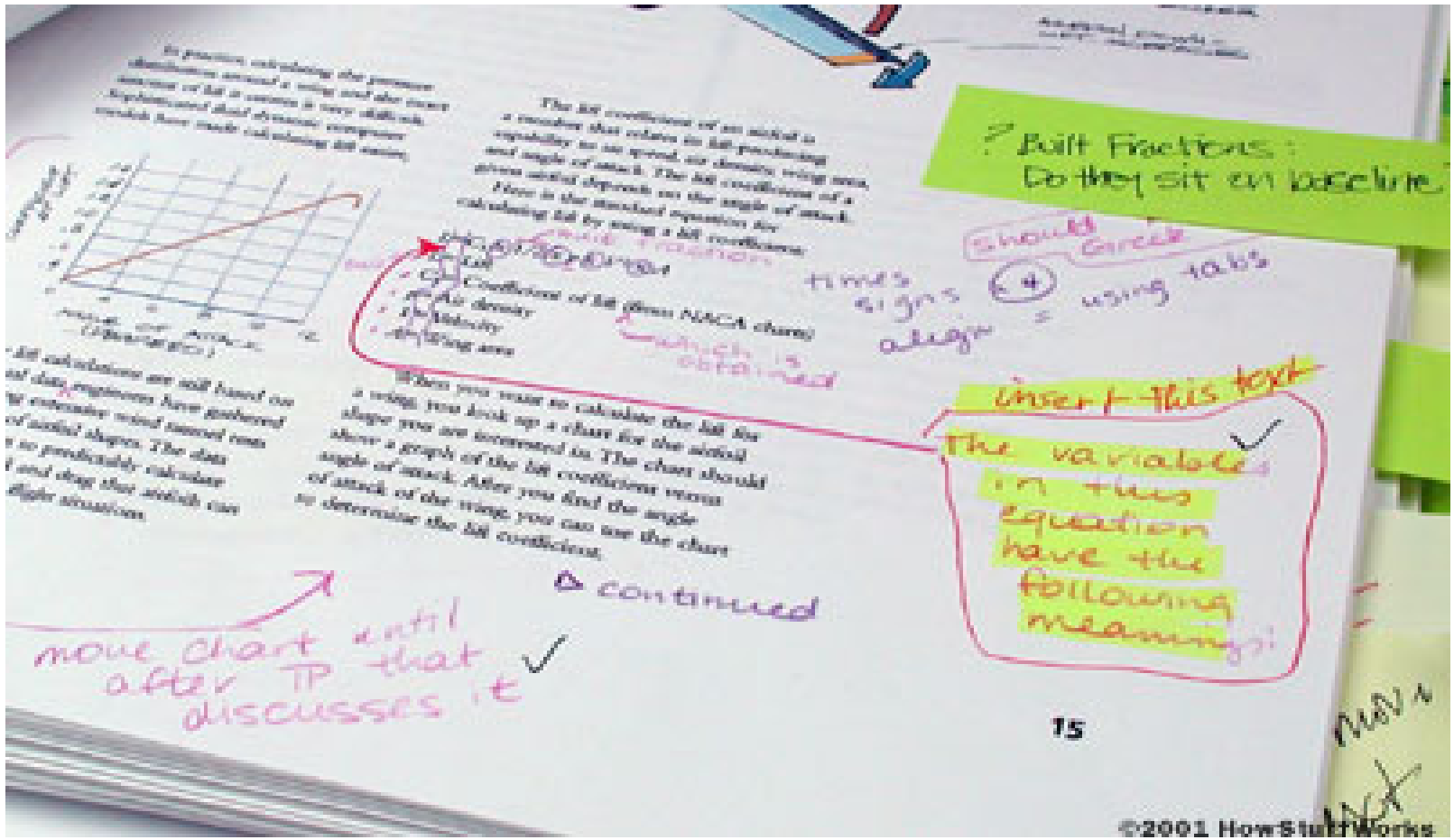
SQL/XML

- Flexible way to retrieve XML from table/column data
- Part of upcoming SQL 2003 Standard
- Support available for DB2, Oracle
- Cross-platform implementations exist
- No support for "consuming" XML to update RDBMS



Native XML Programming

Markup



From the "How Stuff Works" Web site – http://www.howstuffworks.com

Marking Up Data Structures

2003-07-12
Wile E. Coyote
New Mexico
ACME
Atomic Pogo Stick
142.50

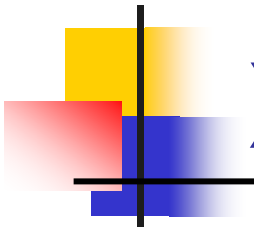
```
<invoice>
  <date>2003-07-12</date>
  <customer>
    <name>Wile E. Coyote</name>
    <state>New Mexico</state>
  </customer>
  <items>
    <product>
      <mfr>ACME</mfr>
      <name>Atomic Pogo Stick</name>
      <price>142.50</price>
    </product>
  </items>
</invoice>
```

"Data Structures Programs"

```
let $in := doc("invoice.xml")
for $i in $in/invoice
where sum($i//price) > 100
return $i/customer
```

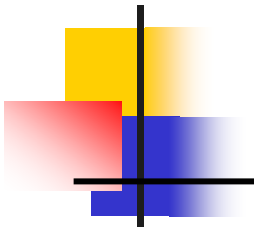
```
<invoice>
  <date>2003-07-12</date>
  <customer>
    <name>Wile E. Coyote</name>
    <state>New Mexico</state>
  </customer>
  <items>
    <product>
      <mfr>ACME</mfr>
      <name>Atomic Pogo Stick</name>
      <price>142.50</price>
    </product>
  </items>
</invoice>
```

Document-oriented XML has data structures too…



XML Is Not Objects

- XML Types
  - Document, Element, Attribute, other node types are fundamental
  - xs:integer, xs:decimal, xs:float, xs:string, other simple data types
  - Complex data types and content models
  - Hierarchy, sequence, "contains"
- Java / C# Types
  - Classes and objects
    - Encapsulation, Methods, Polymorphism, Identity, Type, Regular Structure
  - int, float, double, string, other simple data types
  - Pointers can be used to model hierarchies
  - No native support for navigating XML hierarchies



Navigation and Conversion

```
<stock>
  <name>Cindy's Snowshoes</name>
  <ticker>NASDAQ:RAKD</ticker>
  <price>20.00</price>
  <revenues>2.00</revenues>
  <expenses>1.00</expenses>
</stock>
```

- DOM

```
Tree t = ParseXML("stock.xml");
PERatio =
      number(t.getmember("/stock/price"))
     /((number(t.getmember("/stock/revenues")
      - number(t.getmember("/stock/expenses"))
```

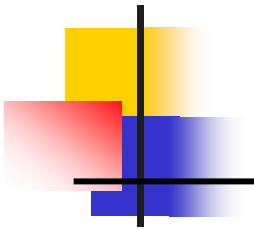
- XQuery

```
let $stock := doc('stock.xml')/stock
return $stock/price div ($stock/revenue - $stock/expenses)
```

Example taken from: "Speaking XML", Adam Bosworth
http://www.fawcette.com/xmlmag/2002_12/magazine/columns/endtag/

Logically, XML Is Not Just Text

- The text-only myth: "XML is just text – look at the XML spec!"
- Native XML programs use the "logical structure" of a document, as defined in the XML spec:
  - Each XML document has both a logical and a physical structure. […] Logically, the document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup.
- The XML BNF maps characters to logical structures:
  [39] element ::= EmptyElemTag | STag content ETag
- The XML spec is largely about these logical structures
  - Example: The element structure of an XML document may, for validation purposes, be constrained using element type and attribute-list declarations. An element type declaration constrains the element's content.



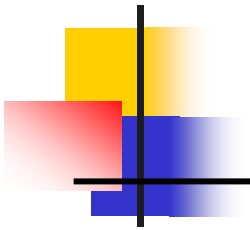
Logically, XML Is Not Just Text

- XML can be hard to process as raw text
- These documents are the same!

```
<invoice xmlns='acme.example.com'>
  <customer>Wile E. Coyote</customer>
  <items>
    <item>
      <maker>ACME</maker>
      <name>Rocket Skates</name>
      <price>532.12</price>
    </item>
</invoice>
```

```
<ac:invoice xmlns:ac='acme.example.com'>
  <ac:customer>Wile E. Coyote</ac:customer>
  <ac:items>
    <ac:item>
      <ac:maker>ACME</ac:maker>
      <ac:name>Rocket Skates</ac:name>
      <ac:price>532.12</ac:price>
    </ac:item>
</ac:invoice>
```

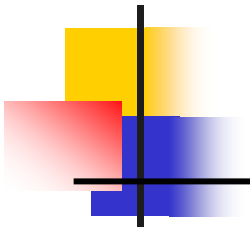
```
default element namespace='acme.example.com'
let $i := doc('invoice.xml')
return sum($i//item/price)
```



Native XML Programming

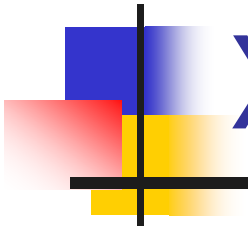
- Program logic is close to the logic of the XML structure
- Hides details of parsing and serialization
- Data model based on the logical structure of XML
- XML Schema simple datatypes

- Direct support for
  - Creating XML structures
  - Locating information in XML structures
  - Restructuring existing XML structures



Native XML Programming

- Native XML languages
  - XPath
  - XSLT
  - XQuery
  - XStatic
- Non-native languages
  - Java
  - C#
  - Perl
  - Python



XQuery and Relational Data

XML Views of Relational Data

Customers

CustId	Name	Address
1	Woodworks	Baltimore
2	Software Solutions	Boston
3	Food Supplies	New York
4	Hardware Store	Washington
5	Books Inc.	New Orleans
…		

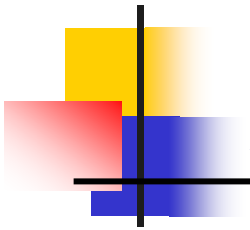
Projects

ProjId	Name	CustId
1	Medusa	1
2	Pegasus	4
3	Python	6
…	…	…
8	Typhon	4
…		

```
<Customers>
  <row>
    <CustId>4</CustId>
    <Name>Hardware Store</Name>
    <Address>Washington</Address>
  </row>
  <!-- SNIP -->
```

```
<Projects>
  <row>
    <ProjId>2</ProjId>
    <Name>Pegasus</Name>
    <CustId>4</CustId>
  </row>
  <row>
    <ProjId>8</ProjId>
    <Name>Typhon</Name>
    <CustId>4</CustId>
  </row>
  <!-- SNIP -->
```

These views are not physical – the RDB sees only SQL, the XQuery system sees only XML, and middleware mediates between the physical representations.



XQuery against Relational Views

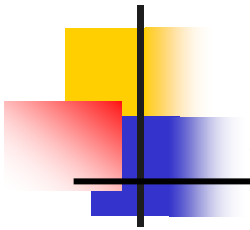
```
<Customers>
  <row>
    <CustId>4</CustId>
    <Name>Hardware Store</Name>
    <Address>Washington</Address>
  </row>
```

```
<Projects>
  <row>
    <ProjId>2</ProjId>
    <Name>Pegasus</Name>
    <CustId>4</CustId>
  </row>
  <row>
    <ProjId>8</ProjId>
    <Name>Typhon</Name>
    <CustId>4</CustId>
  </row>
```

```
for $c in $cust/row
return
  <customer>
   { $c/CustId }
   { $c/Name }
  </customer>
```

Output:

```
 <customer>
   <CustId>4</CustId>
   <Name>Hardware Store</Name>
 </customer>
```



XQuery against Relational Views

```
<Customers>
  <row>
    <CustId>4</CustId>
    <Name>Hardware Store</Name>
    <Address>Washington</Address>
  </row>
```

```
<Projects>
  <row>
    <ProjId>2</ProjId>
    <Name>Pegasus</Name>
    <CustId>4</CustId>
  </row>
  <row>
    <ProjId>8</ProjId>
    <Name>Typhon</Name>
    <CustId>4</CustId>
  </row>
```

```
for $c in $cust/row
return
  <customer id="{ $c/CustId }">
    <name>{ string($c/Name) }</name>
  </customer>
```

Output:

```
<customer id = "4">
  <name>Hardware Store</name>
</customer>
```

XQuery against Relational Views

```
<Customers>
  <row>
    <CustId>4</CustId>
    <Name>Hardware Store</Name>
    <Address>Washington</Address>
  </row>
```

```
<Projects>
  <row>
    <ProjId>2</ProjId>
    <Name>Pegasus</Name>
    <CustId>4</CustId>
  </row>
  <row>
    <ProjId>8</ProjId>
    <Name>Typhon</Name>
    <CustId>4</CustId>
  </row>
```

```
for $c in $cust/row,
    $p in $proj/row
where $c/CustId = $p/CustId
return
   <CustomerProject>
      <Customer>{ string($c/Name) }</Customer>
      <Project>{ string($p/Name) }</Project>
   </CustomerProject>
```

Output:

```
 <CustomerProject>
      <Customer>Hardware Store</Customer>
      <Project>Pegasus</Project>
  </CustomerProject>
  <CustomerProject>
      <Customer>Hardware Store</Customer>
      <Project>Typhon</Project>
  </CustomerProject>
```

XQuery against Relational Views

```
<Customers>
 <row>
  <CustId>4</CustId>
  <Name>Hardware Store</Name>
  <Address>Washington</Address>
 </row>
```

```
<Projects>
 <row>
  <ProjId>2</ProjId>
  <Name>Pegasus</Name>
  <CustId>4</CustId>
 </row>
 <row>
  <ProjId>8</ProjId>
  <Name>Typhon</Name>
  <CustId>4</CustId>
 </row>
```

```
for $c in $cust/row
return
 <customer id="{$c/CustId}">
  <name>{ string($c/Name) }</name>
  <projects>
   {
    for $p in $proj/row
    where $p/CustId = $c/CustId
    return
      <project id="{$p/ProjId}" name="{$p/Name}"/>
   }
  </projects>
</customer>
```

Output:

```
<customer id = "4">
  <name>Hardware Store</name>
  <projects>
   <project id = "2" name = "Pegasus"/>
   <project id = "8" name = "Typhon"/>
  </projects>
</customer>
```

SQL-centric *vs.* XML-centric

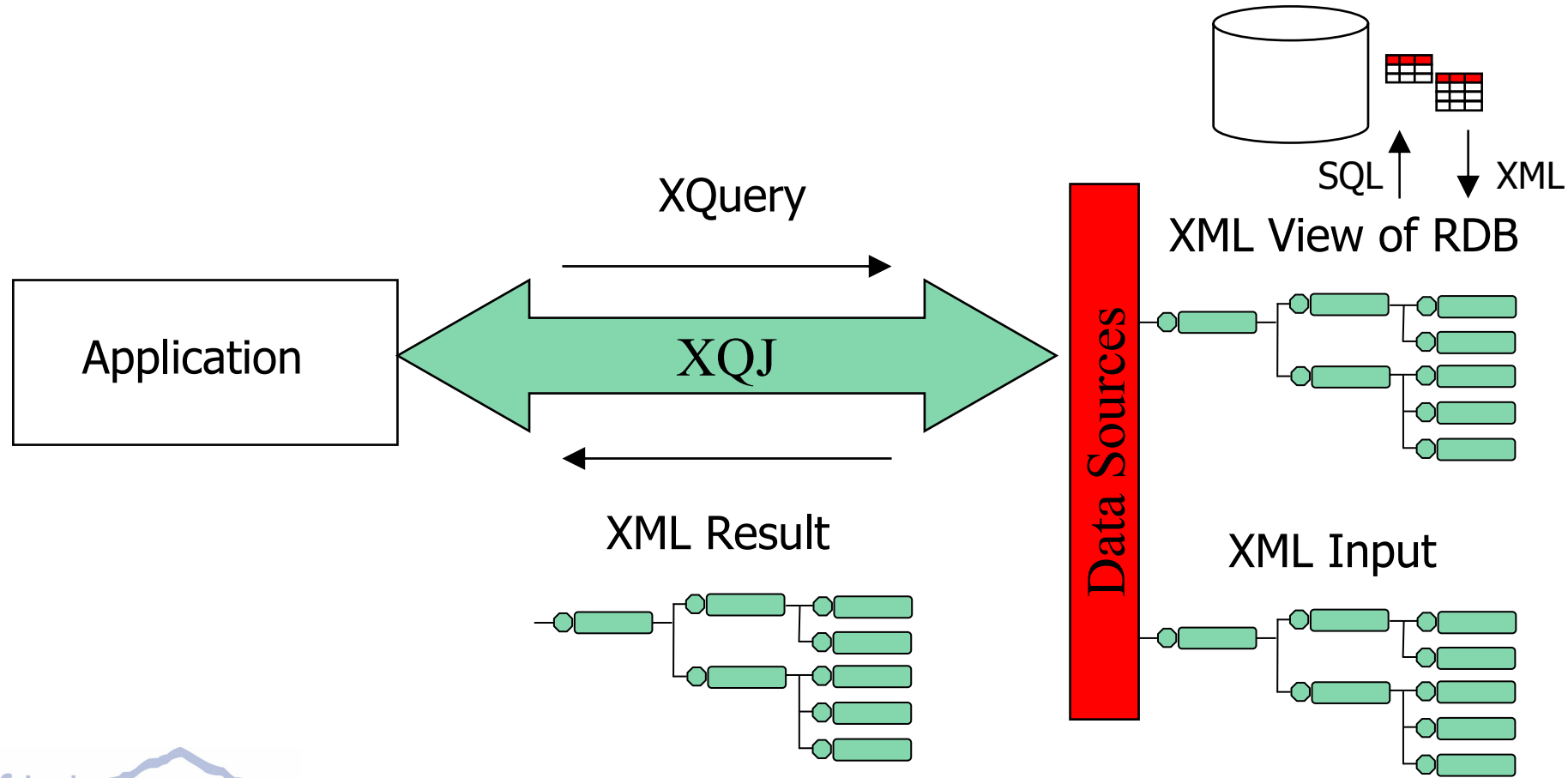
SQL/XML

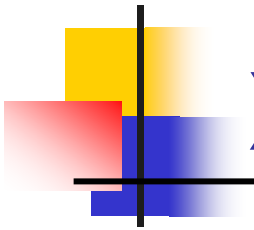
```
select
  xmlelement(name customer,
  xmlattributes(c.CustId as id,
                c.Name as name),
  xmlelement(name projects,
  (select xmlagg(xmlelement(name project,
          xmlattributes(p.ProjId as id,
                        p.Name as name)))
   from Projects p
   where p.CustId=c.CustId)))
         as "customer-projects"
from Customers c
```

XQuery

```
for $c in $cust/row
return
  <customer id="{$c/CustId}"
            name="{$c/Name}">
   <projects>
   {
    for $p in $proj/row
    where $p/CustId = $c/CustId
    return
      <project id="{$p/ProjId}"
               name="{$p/Name}"/>
   }
   </projects>
  </customer>
```

XQuery, XML, and Relational Data





XQJ: XQuery for Java (JSR 225)

- "JDBC for XQuery"
- Needs to support both relational and XML sources
- Developed under Java Community Process
- The example on the next page uses an imaginary syntax that illustrates what XQJ will be able to do

"Picture if You Will…"

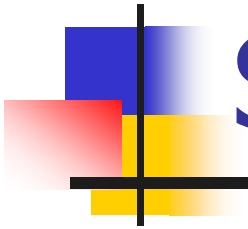
```
JDBCDataSource ds = new JDBCDataSource();
ds.setURL("jdbc:datadirect:oracle://localhost:1521;SID=ORA92");
ds.setUserName("myUserName");
ds.setPassword("myPassword");
ds.setVariableName("$db");

// Bind $db to the database
Session ses = ds.getSession();
PreparedQuery qry = ses.prepareQuery( q );

// Now call the prepared query for a given input message
Sequence msg = new SAXSequence("c:/temp/input.xml");
qry.bindVariable("$message", msg);
Sequence seq = qry.execute();

// output the results
Item i;
while ((i = seq.getNextItem()) != null) {
  System.out.println(i.getString());
}
```

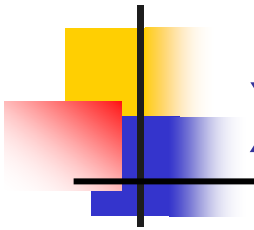
These APIs have not yet been defined, so this is very speculative. XQJ is still in the requirements phase at the time of writing.

Summary: XML and Databases

- Code it Yourself
  - Portable
  - Tedious to write, hard to maintain
  - Often used by OEMs because of need for portability
- Vendor Extensions
  - Non-portable
  - Wide variety of approaches
  - Likely to be replaced by standards
- SQL/XML
  - Standards compliant
  - SQL-centric
  - Cross-database implementations exist
  - Small extension of an established, mature language
- XQuery
  - Standards compliant
  - XML-centric
    - Native XML programming
    - XML views
  - Good for integrating relational sources with XML sources
  - Widely supported

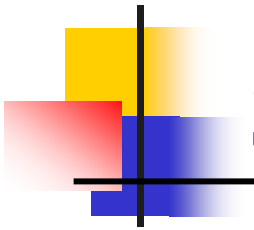
Tradeoffs – What Is Simplicity?





XML: Simple to Implement and Use

- A graduate student can write a parser in < 1 month
- Documents can be written with a simple text editor
- More sophisticated editing tools emerged quickly
- Easy to integrate into many programming environments
- W3C XML Schema makes it much more complex
  - Main payoff: datatypes
  - RELAX NG is simpler, but little traction so far
  - W3C XML Schema is established
- Popular mindset: "Beware tools you can't write yourself"
- Simple tools solve part of the problem – many applications need Java+SQL+XSLT+DOM

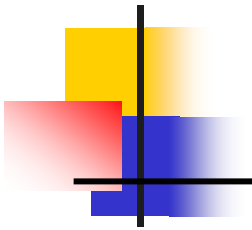


SQL/XML: Simple for SQL Users

- Leverages existing SQL implementations, tools, and APIs
- Hard to implement – requires a SQL implementation
- Not easily adapted to non-database environments
- Easy to learn for SQL programmers – a simple extension to SQL
- SQL-centric view of data
- Often replaces Java+SQL+XSLT+DOM with one extended SQL query

XQuery: Simple for XML Programs

- XQuery is designed for native XML programming
- Harder to implement than traditional XML tools
- XQuery is a language, and must be learned
- >20 full or partial implementations, available in diverse environments
- Designed to easily combine data sources
- Optimizable for large database environments
- Uses SQL/XML mappings for XML views of RDBMS (internally, queries are executed as SQL)
- Often replaces Java+SQL+XSLT+DOM with one XQuery



Questions

- Today
- Later:

  jonathan.robie@datadirect.com
- XQuery Feedback email list:

  public-qt-comments@w3.org
- Public email list:

  www-ql@w3.org