Database To Web

The Problem

- Need to present information in a database on web pages
  - want access from any browser
  - may require at least HTML 4 compatibility
- Want to separate gathering of data from formatting of data
  - allows developers with different skill sets to work independently
    - database developers (JDBC)
    - servlet/XML developers (Java servlets, DOM parsers)
    - web content developers (HTML, XML, XSLT)
  - allows formatting changes without modifying compiled code
  - allows many formatting variations using the same data gathering software

A Solution

- Steps
  - write Java classes that represent data in the database
    - these may be useful apart from translation to HTML
  - write code that queries the database and creates Java objects to hold the data
  - create DTD or XML Schema describing XML that will be generated
    - provides data format documentation and run-time validation
  - write code that generates XML from these Java objects
  - write servlet that uses the above code to generate XML and applies XSLT stylesheets to it
  - create XSLT stylesheets
  - use a web/application server to execute the servlet

could use Enterprise Java Bean (EJB) entity beans or Hibernate or JDO to accomplish both of these

Example Application

- These concepts will be presented in the context of a web application for displaying information about a music collection

Database

- The database contains the following tables
  - Artists
    - columns include ID and Name
  - Recordings
    - columns include ID, Title, ArtistID, CategoryID, Year, Format and CoverImageURL
  - Categories
    - examples are Pop, Alternative, Jazz, Country and Classical
    - columns include ID and Name
  - Tracks
    - columns include ID, Name, Time and RecordingID

Domain Classes

- These domain classes are required
  - Artist
  - Recording
  - Category
  - Track
- They are all fairly similar
- An example of one of them follows

Artist.java

```
import java.util.*;

public class Artist {
  private List recordings = new ArrayList();
  private String name;

  public Artist(String name) {
    this.name = name;
  }

  public void addRecording(Recording recording) {
    recordings.add(recording);
  }

  public String getName() {
    return name;
  }

  public List getRecordings() {
    return Collections.unmodifiableList(recordings);
  }
}
```

Java Database Connectivity (JDBC)

- Provides a portable way to query relational databases
- Supports transactions and stored procedures
- `java.sql` package contains interfaces
- These interfaces are implemented by
data source specific JDBC drivers
- An example using the JDBC-ODBC bridge follows
  - included with JDK
  - provides access to all databases that have ODBC drivers
  - driver class for bridge is `sun.jdbc.odbc.JdbcOdbcDriver`

This method is in the class `MusicCollection`.

JDBC Example

Import these packages:
- `java.sql`
- `java.util`

```
public List getArtists() {
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); // loads the JDBC driver
  String databaseURL = "jdbc:odbc:MusicCollection"; // register in ODBC control panel
  String username = "volkmann"; // can be an empty string if not needed
  String password = "funwithxml"; // can be an empty string if not needed
  Connection conn = DriverManager.getConnection(databaseURL, username, password);
  Statement stmt = conn.createStatement();

  String sql = "select * from Artists"; // can use joins
  ResultSet resultSet = stmt.executeQuery(sql);
  List artists = new ArrayList();
  while (resultSet.next()) {
    String name = resultSet.getString("Name"); // can get other data types
    artists.add(new Artist(name));
  }

  conn.close();
  return artists;
}
```

- also need to query for recordings and tracks and attach them to the data structure
- exception handling has been omitted

See similar example using the **MySQL** JDBC driver on page 23 of the Software Setup section.

Document Type Definition (DTD)

- Defines rules to which XML documents must conform
- DTD for music collection

? means 0 or 1
* means 0 or more
+ means 1 or more
none means exactly 1

```
<!ELEMENT music-collection (artist*)>
<!ELEMENT artist (name, recording*)>
<!ELEMENT recording (title, year, cover-image-url?, format, track*)>
<!ELEMENT track (name, time?)>

<!ELEMENT cover-image-url (#PCDATA)>
<!ELEMENT format (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT owner (#PCDATA)>
<!ELEMENT time (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

XML Generation

- DOM parsers can do this
  - DOM stands for Document Object Model
- Create a tree of DOM Java objects representing the content in an XML document
- An example using JAXP follows

This method is in the class MusicCollection.

DOM Example

Import these packages:
- javax.xml.parsers
- org.w3c.dom

```
public Document getMusicXML() {
  // Create an empty Document.
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  DocumentBuilder db = dbf.newDocumentBuilder();
  Document doc = db.newDocument();

  Element rootElement = doc.createElement("music-collection");
  doc.appendChild(rootElement);

  List artists = getArtists();
  Iterator iter = artists.iterator();
  while (iter.hasNext()) {
    Artist artist = (Artist) iter.next();
    Element artistElement = doc.createElement("artist");
    rootElement.appendChild(artistElement);
    Element nameElement = doc.createElement("name");
    nameElement.appendChild(doc.createTextNode(artist.getName()));
    artistElement.appendChild(nameElement);
  }

  return doc;
}
```

need to add more artist child elements and loop through recordings and tracks to create elements for them

Generated XML Document

- The real document contains multiple artists and each artist contains multiple recordings

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE music-collection SYSTEM "music-collection.dtd">
<music-collection>
  <artist>
    <name>McLachlan, Sarah</name>
    <recording>
      <title>Mirrorball</title>
      <year>1999</year>
      <cover-image-url>Mirrorball.gif</cover-image-url>
      <format>CD</format>
    </recording>
  </artist>
</music-collection>
```

using file names that are assumed to be in a certain directory instead of URLs

Java Servlet

- Typically invoked from a web browser
- Performs two steps
  - invokes code that generates XML from the database
  - applies an XSLT stylesheet to the XML
- An example using JAXP follows

Servlet Example

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.*;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;

public class MusicCollectionServlet extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
  throws IOException, ServletException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    Document doc = new MusicCollection().getMusicXML();

    try {
      ServletContext context = getServletContext();
      InputStream xsltStream = context.getResourceAsStream("recordings.xsl");
```

finds the stylesheet relative to the location of the web app.

Servlet Example (Cont’d)

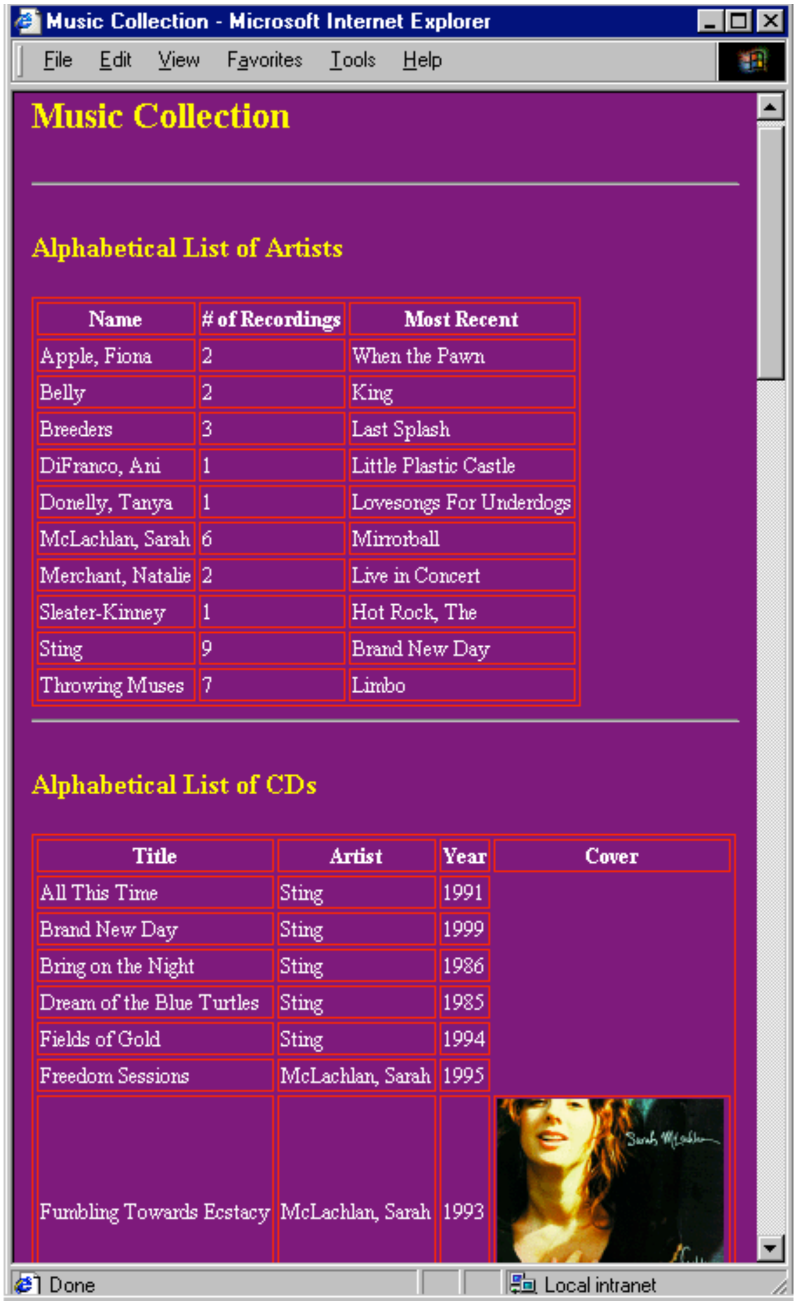
```
        // Apply XSLT stylesheet using JAXP 1.1 and Xalan 2.
        Source xsltSource = new StreamSource(xsltStream);
        TransformerFactory factory = TransformerFactory.newInstance();
        Transformer transformer = factory.newTransformer(xsltSource);
        Source xmlSource = new DOMSource(doc);
        Result result = new StreamResult(out);
        transformer.transform(xmlSource, result);

      } catch (TransformerException e) {
        out.println(e);
      }
    }
  }
```

reparses the stylesheet every time;
improve by creating a `Templates` object the first time and asking it to create a new `Transformer` object in each invocation

XSLT Stylesheet

- Transforms an XML document into
  - HTML
  - different XML
  - text
- Contains rules called templates
- Is itself an XML document
- An example follows

Desired Output



Music Collection - Microsoft Internet Explorer

Music Collection

Alphabetical List of Artists

Name	# of Recordings	Most Recent
Apple, Fiona	2	When the Pawn
Belly	2	King
Breeders	3	Last Splash
DiFranco, Ani	1	Little Plastic Castle
Donelly, Tanya	1	Lovesongs For Underdogs
McLachlan, Sarah	6	Mirrorball
Merchant, Natalie	2	Live in Concert
Sleater-Kinney	1	Hot Rock, The
Sting	9	Brand New Day
Throwing Muses	7	Limbo

Alphabetical List of CDs

Title	Artist	Year	Cover
All This Time	Sting	1991	
Brand New Day	Sting	1999	
Bring on the Night	Sting	1986	
Dream of the Blue Turtles	Sting	1985	
Fields of Gold	Sting	1994	
Freedom Sessions	McLachlan, Sarah	1995	
Fumbling Towards Ecstacy	McLachlan, Sarah	1993	

XSL Stylesheet Example

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <xsl:variable name="title">Music Collection</xsl:variable>

    <html>
      <head>
        <title><xsl:value-of select="$title"/></title>
      </head>
      <body style="background-color:purple; font-color:white">
        <h1 style="color:yellow"><xsl:value-of select="$title"/></h1>
        <xsl:apply-templates select="music-collection"/>
      </body>
    </html>
  </xsl:template>
```

XSL Stylesheet Example (Cont'd)

```
<xsl:template match="music-collection">
  <hr/>
  <h2 style="color:yellow">Alphabetical List of Artists</h2>
  <table border="1" bordercolor="red" style="color:white">
    <tr>
      <th>Name</th>
      <th># of Recordings</th>
      <th>Most Recent</th>
    </tr>
    <xsl:apply-templates select="artist">
      <xsl:sort select="name"/>
    </xsl:apply-templates>
  </table>
```

this template continues on the next page

XSL Stylesheet Example (Cont’d)

```
  <hr/>
  <h2 style="color:yellow">Alphabetical List of CDs</h2>
  <table border="1" bordercolor="red" style="color:white">
    <tr>
      <th>Title</th>
      <th>Artist</th>
      <th>Year</th>
      <th>Cover</th>
    </tr>
    <xsl:apply-templates select="artist/recording">
      <xsl:sort select="title"/>
    </xsl:apply-templates>
  </table>
</xsl:template>
```

XSL Stylesheet Example (Cont'd)

```
<xsl:template match="artist">
  <tr>
    <td><xsl:value-of select="name"/></td>
    <td><xsl:value-of select="count(recording)"/></td>
    <td>
      <xsl:for-each select="recording">
        <xsl:sort data-type="number" order="descending" select="year"/>
        <xsl:if test="position() = 1">
          <xsl:value-of select="title"/>
        </xsl:if>
      </xsl:for-each>
    </td>
  </tr>
</xsl:template>
```

getting most recent recording by this artist

XSL Stylesheet Example (Cont'd)

```
  <xsl:template match="recording">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="../name"/></td>
      <td><xsl:value-of select="year"/></td>
      <td>
        <xsl:choose>
          <xsl:when test="cover-image-url">
            <img src="/music/images/
                      {translate(../name, ' ,', '')}/
                      {cover-image-url}"/>
          </xsl:when>
          <xsl:otherwise>
            &#160; <!-- non-breaking space -->
          </xsl:otherwise>
        </xsl:choose>
      </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

image files are stored in a subdirectory of Tomcat/webapp/music/images with the same name as the artist

all of this must be on one line

removes spaces and commas from artist name

MusicCollectionServlet Setup Steps

- Use the ODBC control panel to add a data source called "MusicCollection" which maps to `MusicCollection.mdb`
  - under Windows 2000 and XP, go into the "Administrative Tools" directory of the Control Panel and double-click "Data Sources (ODBC)"
- Copy `music.war` to the Tomcat webapps directory
  - the Ant deploy target does this
- Start Tomcat
- From any web browser, enter the following URL
  - http://localhost:8080/music/servlet/MusicCollectionServlet

This is case sensitive!