Server-side
Web Programming

Lecture 13:
JDBC Database Programming

JDBC Definition

• Java Database Connectivity (JDBC): set of classes that provide methods to
  – Connect to a database through a database server (using a driver)
  – Query database using SQL syntax, getting “list” of records that match query
  – Manipulate database by executing SQL commands to modify, insert, and delete records
JDBC Components

- Major objects involved:
  - **Connection**: represents connection to a database through a server
  - **Statement**: represents SQL statement executed on database via that connection
  - **ResultSet**: represents “list” of records matching a query

```java
<%@ page import="java.sql.*" %>
<% 
    Connection connection = null;
    Statement statement = null;
    ResultSet books = null;
%>
```

Connecting to the Database Server

- Load the database driver
  - Not necessary in most recent version, but safe thing to do
  - Syntax:
    ```java
    Class.forName("driver class").newInstance();
    ```

- Name of driver class based on url of provider
  - Example: `com.mysql.jdbc.Driver`

```java
try {
    Class.forName("com.mysql.jdbc.Driver").newInstance(); // load the driver
    connection = DriverManager.getConnection("jdbc:mysql://localhost/TestDB", "john", "sesame");
} 
```
Connecting to the Database Server

• Connect to the database
  – Need to provide username and password
• Need to provide url of database
  Usual form: `jdbc:server type:url of server/database name`
  Example: `jdbc:mysql://localhost/TestDB`
• Syntax:
  ```java
  connection = DriverManager.getConnection("databaseURL", "username", "password");
  ```

Exception Handling in JDBC

• Any database-related statement may throw an `SQLException`
  – Your code must put in try/catch block
  – May also need to catch other exceptions
    • `ClassNotFoundException` for missing database driver

```java
try {
    Class.forName("com.mysql.jdbc.Driver").newInstance(); // load the driver
    connection = DriverManager.getConnection(
            "jdbc:mysql://localhost/TestDB", "john", "sesame");
}
catch (ClassNotFoundException e) { %> NO DRIVER <% }
catch (SQLException e) { %> NO CONNECTION <% }
```
Executing Queries

- Create new **statement** object using the connection
- Execute an **SQL query** using that statement
- Store results in a **ResultSet** object

Syntax:
```java
statement = connection.createStatement();
statement.executeQuery("SQL query");
```

Reading ResultSets

- Can only do simple access:
  - Read in field values from current record
  - Move to next record

- Syntax to move to next record: `ResultSetObject.next()`;
  - Returns `false` if no next record, true otherwise
  - Must execute once before reading first record
  - Usually `while` loop to read until no more records

```java
while (ResultSetObject.next()) {
  code to read in current record
}
```

```java
while (books.next()) {
  String productCode = books.getString("productCode");
  String title = books.getString("title");
}
Reading ResultSets

- Syntax to read field from current record:

```java
value = ResultSetObject.getType(fieldname);
```

Specify **type** data is to be read in as
- `varChar` → `getString`
- `int` → `getInt`
- `double` → `getDouble`

Specify **field name** used in database
Reading ResultSets

• Once ResultSet read in, can use in own code
  – Display in JSP
  – Store in array for future use, etc.

```
while (books != null && books.next()) {
    String productCode = books.getString("productCode");
    String title = books.getString("title");
    double price = books.getDouble("price");

    <tr align="top">
    <td>&lt;%= title %&gt;</td>
    <td>&lt;%= price %&gt;</td>
    <td>
        <form action="&lt;%= response.encodeURL("/jdoc/cart/cartServlet?quantity=1") %&gt;"
             method = "get">
            <input type="hidden" name="productCode" value="&lt;%= productCode %&gt;">
            <input type="submit" value="Add To Cart">
        </form>
    </td>
    </tr>
```
Executing Update Statements

- Syntax:
  ```java
  Statement statement = connection.createStatement();
  statement.executeUpdate("SQL statement");
  ```

- Example:
  ```java
  statement.executeUpdate("INSERT INTO books
  (productCode, title, price)
  VALUES ('0004', 'Green Eggs and Ham', 9.95)";
  ```

```
mysql> select * from books;
+---------+--------------------------+------+
<table>
<thead>
<tr>
<th>productCode</th>
<th>title</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Murach's Java Servlets and JSP</td>
<td>31.19</td>
</tr>
<tr>
<td>0002</td>
<td>Murach's ASP.NET 2.0 Web Programming with VB 2005</td>
<td>33.08</td>
</tr>
<tr>
<td>0003</td>
<td>HTML and XHTML Pocket Reference</td>
<td>10.39</td>
</tr>
<tr>
<td>0004</td>
<td>Green Eggs and Ham</td>
<td>9.95</td>
</tr>
</tbody>
</table>
+---------+--------------------------+------+
4 rows in set (0.00 sec)
```
Inserting Parameter Values

• Read in parameter values
  – Validate if necessary

protected void processRequest(HttpServletRequest request, HttpServlet throws ServletException, IOException {

  // Get parameters to create new record
  String productCode = request.getParameter("productCode");
  String title = request.getParameter("title");
  double price = Double.parseDouble(request.getParameter("price"));

  // Create query to put new record into database
  try {
    String sql = "INSERT INTO books (productCode, title, price) VALUES (", productCode, ", " + title + ", " + price + ");

    statement.executeUpdate(sql);
  }

  // Insert into SQL statement
  – Will need to use + to append values into SQL statement string
  – Note that string values must be inside ‘‘ to avoid syntax errors

  Creates string of form:

  INSERT INTO books (productCode, title, price) VALUES (’0004’, ’Green Eggs and Ham’, 9.95)
Validation and Updates

- Usually need to validate update with database to avoid problems
  - Don’t add item if already in database
  - Don’t update or remove if not in database
    - Won’t cause database error, but probably want to inform user
- Checking whether item in database involves query
  - Create query for item in question
  - If no results then not in database
  - Key idea: use statement of form `if (ResultSetObject.next())`
    - True if at least one result
    - False if no matches

Example:
Validating that no book with given productCode exists in database before adding it
- Query for book with that productCode
- Redirect to error page if `books.next()` is true

```java
// First check whether the given product code already exists
// by creating a query for it.
try {
    statement = connection.createStatement();
    books = statement.executeQuery("SELECT * FROM books WHERE productCode = "+productCode+");
    if (books.next()) {
        // At least one record with this product code
        // Redirect to the error JSP
        RequestDispatcher dispatcher =
            getServletContext().getRequestDispatcher("/AddError.jsp");
        dispatcher.forward(request, response);
        return;
    }
```