Three-Tier Architecture

Students example: Create a jdbc-examples database in your db, then create this schema

```
CREATE TABLE students (  
  ID SERIAL PRIMARY KEY,  
  pid INTEGER,  
  first_name TEXT,  
  middle_name TEXT,  
  last_name TEXT  
);

INSERT INTO students (pid, first_name, middle_name, last_name)  
VALUES (77777777, 'Mary', '', 'Doe');

INSERT INTO students (pid, first_name, middle_name, last_name)  
values (88888888, 'John', 'T', 'Smith');
```
import java.sql.*;

class JdbcTest {
    public static void main (String args [])
        throws SQLException {
            // Registering Postgresql JDBC driver
            Class.forName("org.postgresql.Driver");

            // Open a connection to the database
            Connection conn = DriverManager.getConnection("jdbc:postgresql://localhost/jdbc-examples?" + "user=postgres&password=postgres");

            // Query the student PIDs
            Statement stmt = conn.createStatement();
            ResultSet rset = stmt.executeQuery("SELECT pid FROM students");

            // Print out the PID (1st attribute)
            while (rset.next ())
                System.out.println (rset.getInt(1));

            //close the result set, statement, and connection
            rset.close();
            stmt.close();
            conn.close();
        }
}

// Updates
stmt.executeUpdate("UPDATE students SET first_name = 'John' WHERE ID = 1");
PreparedStatement Object

• If you want to execute a Statement many times + defend against SQL injection attack use a PreparedStatement object instead:

```java
PreparedStatement pstmt = conn.prepareStatement(
    "UPDATE students SET first_name = ? " +
    "WHERE ID = ?");
pstmt.setString(1, "John");
pstmt.setInt(2, 1);
pstmt.executeUpdate();
```

PreparedStatement Object (cont’d)

• The following two code fragments accomplish the same thing:

```java
String updateStr =
    "UPDATE students SET first_name = 'John' " +
    "WHERE ID = 1";
stmt.executeUpdate(updateStr);
```

```java
PreparedStatement pstmt = conn.prepareStatement(
    "UPDATE students SET first_name = ? " +
    "WHERE ID = ?");
pstmt.setString(1, "John");
pstmt.setInt(2, 1);
updateSales.executeUpdate();
```

ResultSet Object

• `int getInt(int columnIndex)`
  Retrieves the value of the designated column in the current row of a ResultSet object as an int Java type

• `int getInt(String columnName)`

• `String getString(int columnIndex)`

• `String getString(String columnName)`
Using Transactions

- When a connection is created, it is in AutoCommit mode, that is, each individual SQL statement is treated as a transaction and will be automatically committed right after it is executed.

```java
conn.setAutoCommit(false);
...
</transaction>
...
conn.commit();
conn.setAutoCommit(true);
```

You can omit if you do not wish to switch back to autocommit mode. Then this point becomes the start of a new transaction.

Why use transactions – Example

Requirement:
Students 2 and 3 either enroll in the class 2 together or they do not enroll in it at all.

```java
conn.setAutoCommit(false);
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO enrollment (student, class) VALUES (?, ?)");
pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
pstmt.setInt(1, 3);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
conn.commit();
```

Why use transactions? Automatic Recovery protects from Crash Effects

```java
conn.setAutoCommit(false);
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO enrollment (student, class) VALUES (?, ?)";
pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
pstmt.setInt(1, 3);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
conn.commit();
```

Crash
Covered and not-Covered Crash Cases

Covered (the typical crash):
• CPU halt
• Loss of RAM memory state
• Crash of database server

Not Covered
• Loss of disk state

Recovery of non-committed transactions guarantees that the executed part has no effect

```java
conn.setAutoCommit(false);
PreparedStatement pstmt = conn.prepareStatement(
    "INSERT INTO enrollment (student, class)
    VALUES (?, ?)"
); pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
pstmt.setInt(1, 3);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
conn.commit();
```

Recovery of committed transactions guarantees that the executed part is definitely in the database

```java
conn.setAutoCommit(false);
PreparedStatement pstmt = conn.prepareStatement(
    "INSERT INTO enrollment (student, class)
    VALUES (?, ?)"
); pstmt.setInt(1, 2);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
pstmt.setInt(1, 3);
pstmt.setInt(2, 2);
pstmt.executeUpdate();
conn.commit();
```
Why use transactions?

Bugs due to Concurrent Processes

Classes have enrollment limit of 100 students. See what can happen by two processes running in parallel, without using concurrency control: First process enrolls student 1 in class 2. Second process enrolls student 2 in class 2.

Concurrent Control guarantees that the concurrent computation is limited to computations equivalent to a serial execution of the transactions

Retrieving Exceptions

- JDBC lets you catch the exceptions generated by your DBMS
Retrieving Exceptions (cont’d)

* Release resources even if an exception if thrown
try:
  ...
} catch (SQLException e) {
  ...
} finally {
  // release resources in reverse-order of creation
  if (rs != null) {
    try { rs.close(); } catch (SQLException e) { // Ignore
    rs = null;
  }
  if (pstmt != null) {...}
  if (conn != null) {...}
}
1st Attempt; Split apart menus, advertisements, etc

Menu HTML Code

```html
<b>Data Entry Menu</b>
<ul>
  <li><a href="students.jsp">Students</a></li>
  <li><a href="classes.jsp">Classes</a></li>
  <li><a href="enrollment.jsp">Enrollment</a></li>
</ul>
```

1st Attempt

JSP Code

```html
<html>
<body>
<table>
<tr>
  <td><jsp:include page="menu.html"/></td>
  <td><Open Connection Code>
      <Statement Code>
      <Presentation Code>
      <Close Connection Code>
  </td>
</tr>
</table>
</body>
</html>
```

1st Attempt

Open Connection Code

```jsp
<%@ page import="java.sql.*" %>
<%-- Import the java.sql package --%>
<%-- variable declarations and initializations--%>
try {
  // Registering Postgresql JDBC driver
  Class.forName("org.postgresql.Driver");
  // Open a connection to the database
  conn = DriverManager.getConnection("jdbc:postgresql://localhost/jdbc-examples?" + "user=postgres&password=postgres");
}%
```
Statement Code

```java
// Create the statement
stmt = conn.createStatement();

// Use the statement to SELECT the students
// FROM the students table.
rs = stmt.executeQuery("SELECT * FROM students");
```
1st Attempt

Close Connection Code

```java
<% // Close the ResultSet
    rs.close();

    // Close the Statement
    statement.close();

    // Close the Connection
    conn.close();

} catch (SQLException e) { <Exception handling> }
%>
```

2nd Attempt: Insertion form included

![Image of a form](http://.../students.jsp?action=insert&...)

"Model 1" programming

![Diagram of database accessing code and HTML-producing part of JSP](http://.../students.jsp?action=insert&...)

![Diagram of database accessing code and HTML-producing part of JSP](http://.../students.jsp?action=update&...)

![Diagram of database accessing code and HTML-producing part of JSP](http://.../students.jsp?action=delete&...)
### JSP Code

```html
<html>
<body>
<table>
<tr>
<td>
<jsp:include page="menu.html"/>
</td>
<td>
<Open Connection Code>
<Insertion Code>
<Statement Code>
<Presentation Code>
</td>
</tr>
</table>
</body>
</html>
```

### Presentation Code

```html
<table>
<tr>
<th>ID</th>
<th>PID</th>
<th>First Name</th>
<th>Middle Name</th>
<th>Last Name</th>
</tr>
<%-- Iterate over the ResultSet -->%>
<% while ( rs.next() ) { %>
<Iteration Code>
<% } %>
</table>
```

### Insert Form Code

```html
<form action="students.jsp" method="POST">
   <input type="hidden" name="action" value="insert"/>
   <th>&nbsp;</th>
   <th><input value="" name="pid" size="10"/></th>
   <th><input value="" name="first" size="15"/></th>
   <th><input value="" name="middle" size="15"/></th>
   <th><input value="" name="last" size="15"/></th>
   <th><input type="submit" value="Insert"/></th>
</form>
```
**2nd Attempt**

**Insertion Code**

// Check if an insertion is requested
String action = request.getParameter("action");
if (action != null && action.equals("insert")) {
    // Create the prepared statement to INSERT student values
    pstmt = conn.prepareStatement("INSERT INTO students (pid, first_name, middle_name, last_name) VALUES (?, ?, ?, ?)");
    pstmt.setInt(1, Integer.parseInt(request.getParameter("pid")));
    ... int rowCount = pstmt.executeUpdate();
    ...
}

**3rd Attempt**

**JSP Code**

```html
<html><body><table>
  <tr>
    <td><jsp:include page="menu.html"/></td>
    <td>
      <Open Connection Code>
      <Insertion Code>
      <Update Code>
      <Delete Code>
      <Statement Code>
      <Presentation Code>
      <Close Connection Code>
    </td>
  </tr>
</table></body></html>
```
3rd Attempt

Presentation Code

```xml
<table>
<tr>
 <th>ID</th>
 <th>PID</th>
 <th>First Name</th>
 <th>Middle Name</th>
 <th>Last Name</th>
</tr>
<!-- Insert Form Code -->
<!-- Iterate over the ResultSet -->
<!-- while ( rs.next() ) { -->
<!-- Iteration Code -->
<!-- } -->
</table>
```

3rd Attempt

Iteration Code

```xml
<form action="students.jsp" method="POST">
 <input type="hidden" name="action" value="update"/>
 <input type="hidden" name="id" value="" />
...</form>
```

```java
Get the pid
```

```java
<td>
 <input value="" name="pid"/>
</td>
```

```java
<td>
 <input type="submit" value="Update">
</td>
```

```java
</tr>
```

3rd Attempt

Delete Code

```java
// Check if a delete is requested
if (action != null && action.equals("delete")) {
 ...
 // Create the prepared statement to DELETE students
 pstmt = conn.prepareStatement(  
 "DELETE FROM Students WHERE id = ?");
 pstmt.setInt(1,  
 Integer.parseInt(request.getParameter("id")));  
 int rowCount = pstmt.executeUpdate();
...
```

13
Update Code

// Check if an update is requested
if (action != null && action.equals("update")) {
    ...
    // Create the prepared statement to UPDATE student values
    pstmt = conn.prepareStatement("UPDATE students
        SET pid = ?, first_name = ?,
        middle_name = ?, last_name = ?
        WHERE ID = ?");
    pstmt.setInt(1, Integer.parseInt(request.getParameter("pid")));
    ... int rowCount = pstmt.executeUpdate();
    ...
}