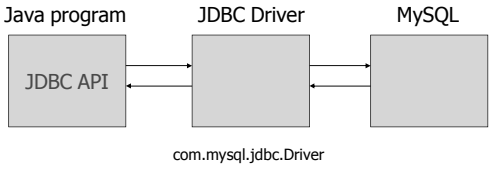
CS320 Web and Internet Programming
JDBC and JSTL SQL

Chengyu Sun
California State University, Los Angeles

JDBC

- An interface between Java programs and SQL databases



Example: HelloJDBC.java

- Where is the database?
  - The `items` table
- Where is the JDBC driver?
- How to connect to the database?
- Display the content of the `items` table

JDBC Basics …

- import java.sql.*;
- Load driver
  - Class.forName("com.mysql.jdbc.Driver")
- Create connection
  - URL
    - jdbc:mysql://[host:port]/[database][?user=cs320stu31&password=abcd]
  - DriverManager.getConnection(URL)
  - DriverManager.getConnection(URL, user, pass)

… JDBC Basics

- Create statement
  - Statement stmt = c.createStatement();
    - stmt.executeQuery(String sql)
    - stmt.executeUpdate(String sql)
- Get result back
  - ResultSet rs

http://download.oracle.com/javase/1.5.0/docs/guide/jdbc/

Example: GuestBook (MVC) – Display

- Create a `guest_book` table
- Retrieve the entries in a servlet
- Display the entries in a JSP

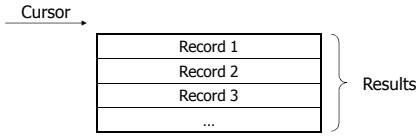
DB Query Results

- In a program, we want to
  - Access each record
  - Access each attribute in a record
  - Access the name of each attribute

select * from guest_book;

name	comment
John	Hello!
Jane	Nice site.

JDBC ResultSet – Row Access



- `next()` – move cursor down one row
  - Cursor starts from *before the 1st record*
  - `true` if the current record is valid
  - `false` if no more records

Common Code for Processing ResultSet

- Process each row
  - `while(rs.next()) {…}`
- Check whether a result set is empty
  - `if(rs.next()) {…}`

JDBC ResultSet – Column Access

- Access the columns of *current row*
- getXxx(String columnName)
  - E.g. getString("user");
- getXxx(int columnIndex)
  - columnIndex starts from 1
  - E.g. getString(1);

JDBC ResultSet – Access Column Names

```
ResultSetMetaData meta = rs.getMetaData();
```

- ResultSetMetaData
  - getColumnName(columnIndex)
    - Column name
  - getColumnLabel(columnIndex)
    - Column title for display or printout

JDBC ResultSet – Size

- No size() method?
- Something about *FetchSize*
  - getFetchSize()
  - setFetchSize(int nrows)

Example: GuestBook (MVC) – Add

- Save new guest book entries to the database
  - executeQuery() vs. executeUpdate()
- Potential problems of handing user input
  - Special characters
  - SQL injection attack

Example: SQL Injection Attack

- User input should NOT be trusted
- Regular user input
  - Username: `cysun`
  - Password: `abcd`
- Malicious user input
  - Username: `cysun`
  - Password: `abcd' or username <> 'cysun`
- *Prevent SQL injection attack?*

Prepared Statements

- Statements with parameters

```
String sql = "insert into items values (? ? ?)";

PreparedStatement pstmt =c.prepareStatement(sql);

pstmt.setString(1, "orange");
pstmt.setBigDecimal(2, 0.59);
pstmt.setInt(3, 4);

pstmt.executeUpdate();
```

Benefits of Prepared Statements

- Special characters are properly handled
- Secure if the SQL statement is constructed from user input
- The SQL statement is more readable
- Better performance (maybe)

JSTL SQL

- sql:transaction
- sql:query
- sql:update
- sql:param
- sql:dateParam
- sql:setDataSource

http://download.oracle.com/docs/cd/E17802_01/products/products/jsp/jstl/1.1/docs/tlddocs/index.html

Example: HelloSQL.jsp

- Data source
- Query
- Results display

sql:setDataSource

- `var` – data source name. Only needed when you have multiple db sources.
- `scope` – scope of the data source
- `driver` – "com.mysql.jdbc.Driver"
- `url`
- `user`
- `password`
- `dataSource`

sql:query

- `var` – name of the result set
- `scope` – scope of the result set
- `sql` – query statement
- `dateSource` – name of the data source
- `startRow`
- `maxRows` – max number of rows in the result set

sql:query Result Set

- javax.servlet.jsp.jstl.sql.Result
  - SortedMap[] getRows()
  - Object[][] getRowsByIndex()
  - String[] getColumnNames()
  - int getRowCount()
  - boolean isLimitedByMaxRows()

http://download.oracle.com/docs/cd/E17802_01/products/products/jsp/jstl/1.1/docs/api/

sql:query example 1

```
<sql:query var="results" sql="select * from items"/>

<table>
 <c:forEach items="${results.rows}" var="row">
  <c:forEach items="${row}" var="col">
   <tr>
    <td>${col.key}</td><td>${col.value}</td>
   </tr>
  </c:forEach>
 </c:forEach>
</table>
```

sql:query example 2

```
<sql:query var="results">
 select * from items where price > 2.00
</sql:query>

<table>
 <c:forEach items="${results.rowsByIndex}" var="row">
 <tr>
  <c:forEach items="${row}" var="col">
     <td>${col}</td>
  </c:forEach>
 </tr>
 </c:forEach>
</table>
```

sql:query example 3

- Place holder and `<sql:param>`

```
<sql:query var="results">

   select * from items where
      price < ? and quantity > ?

 <sql:param value="2.00"/>
 <sql:param value="2"/>

</sql:query>
```

More About sql:setDataSource

- The data source without a name (i.e. without the `var` attribute) is the *default* data source
- The default scope of a data source is the page scope
- How to reference a data source in <sql:query> and <sql:update>

Example: GuestBook (Model 1) – Display

- GuestBook.jsp

sql:update

- `var` – name of the result variable. int
  - number of rows affected by the update
  - 0 if the update statement doesn't return anything
- `scope`
- `sql`
- `dateSource` – name of the data source

sql:update example

```
<c:if test="${! empty param.setPrice}">

   <sql:update var="r">
      update items set price = ? where name = ?
      <sql:param value="${param.price}"/>
      <sql:param value="${param.name}"/>
   </sql:update>

</c:if>
```

Example: GuestBook (Model 1) – Add

- AddComment.jsp

JSTL SQL vs. JDBC

- JSTL SQL
  - Simple applications
  - Small relations
  - Straight-forward operations

⇩

Model 1

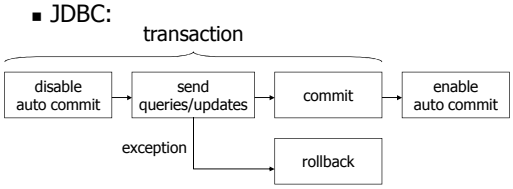
- JDBC
  - Everything else

⇩

MVC

Beyond Basics ...

- ACID
- Transaction
  - <sql:transaction>
  - JDBC:



... Beyond Basics ...

- It's rather expensive to open a db connection
  - So how about once we open a connection, we leave it open forever??
- Connection pooling
  - Max number of connections
  - Max number of idle connections
  - Abandoned connection timeout
  - http://tomcat.apache.org/tomcat-7.0-doc/jndi-datasource-examples-howto.html

... Beyond Basics

- Mismatch between an OO design and a relational design
- Object-relational mapping
  - hibernate - http://www.hibernate.org/