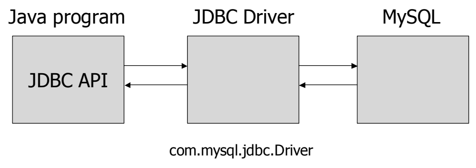
CS320 Web and Internet Programming
JDBC and JSTL SQL

Chengyu Sun
California State University, Los Angeles

JDBC

- An interface between Java programs and SQL databases



JDBC Basics …

- import java.sql.*;
- Load driver
  - Class.forName("com.mysql.jdbc.Driver")
- Create connection
  - Connection c = DriverManager.getConnection(URL);
    - jdbc:mysql://[hostname]/[dbname][?user=cs320stu31&password=abcd]
  - Connection c = 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://java.sun.com/j2se/1.3/docs/guide/jdbc/

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 items;

name	**price**	**quantity**
milk	3.89	2
beer	6.99	1

JDBC ResultSet – Row Access

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

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)

Prepared Statements

- Statements with parameters

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

PreparedStatement stmt =c.prepareStatement(sql);

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

stmt.executeUpdate();
```

Benefits of Using Prepared Statements

- Easier to create the query string
- Much more secure if part of the query string is provided by user
- Better performance (maybe)

```
// without PreparedStatement, you need to worry
// about quotations
String sql = "select salary from employees where " +
             "username ='" + username + "'";

// and somebody may try to pass a username like
// "cysun' or username <> 'cysun"
```

JSTL SQL

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

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` – "jdbc:mysql:///dbname"
- `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://java.sun.com/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>
```

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

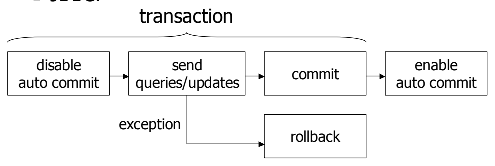
```
<c:if test="${! empty param.setPrice}">

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

</c:if>
```

Using JSTL SQL

- Use JSTL SQL
  - simple application
  - small relation
  - straight-forward operations
  - In the final
- Don’t use JSTL SQL
  - Other

Beyond Basics ...

- Some queries and/or updates should complete as a whole or fail as a whole
- 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://jakarta.apache.org/tomcat/tomcat-5.0-doc/jndi-datasource-examples-howto.html

... Beyond Basics

- OO → relational
- Why do we care about relational model anyway? We just need *persistent objects*.
- Object-relational mapping
  - hibernate - http://www.hibernate.org/