Database Programming (JDBC)

Lecture 5

Outline

- Java DB Connectivity (JDBC) overview
- JDBC API

Reading:
Embedded SQL

- Direct SQL (= ad-hoc SQL) is rarely used
- In practice: SQL is embedded in some application code
  - user interaction, devices, programming logic
- SQL code is embedded into a host language using special syntax

JDBC (Java DB Connectivity)

Java application
{ ...
"SELECT ... FROM ... WHERE"
... }

DBMS
xDBC

• JDBC: *standard* for Java language
• ODBC: Open Data Base Connectivity
  – Language bindings for C/C++

JDBC in Practise

![Diagram showing JDBC in Practise](image)
**JDBC Drivers**

![Diagram showing JDBC Drivers]

**Running a JDBC Application**

<table>
<thead>
<tr>
<th>Phase</th>
<th>Task</th>
<th>Relevant java.sql classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialisation</td>
<td>Load driver Create connection</td>
<td>DriverManager Connection</td>
</tr>
<tr>
<td>Processing</td>
<td>Generate SQL statements Process result data</td>
<td>Statement ResultSet etc.</td>
</tr>
<tr>
<td>Termination</td>
<td>Terminate connection Release data structures</td>
<td>Connection Statement etc.</td>
</tr>
</tbody>
</table>
Loading of Driver

- Create an instance of the driver
- Register driver in the driver manager
- *Explicit* loading

`Class.forName("com.mysql.jdbc.Driver")`

- Several drivers can be loaded and registered
Example: Oracle and JDBC

```java
Class.forName("oracle.jdbc.driver.OracleDriver")

Connection con = DriverManager.getConnection(
    "jdbc:oracle:thin:@dbaprod1:1544:SHR1_PRD",
    username, passwd);
```

Implicit Driver Loading

- Setting system property: `jdbc.drivers`
  - A colon-separated list of driver classnames.

- Can be set when starting the application
  ```
  java -Djdbc.drivers=org.postgresql.Driver application
  ```

- Can also be set from within the Java application
  ```java
  Properties prp = System.getProperties();
  prp.put("jdbc.drivers",
  System.setProperties(prp);
  ```

- The `DriverManager` class attempts to load all the classes specified in `jdbc.drivers` when the `DriverManager` class is initialized.
Addressing Database

• A connection is a session with one database
• Databases are addressed using a URL of the form "jdbc:<subprotocol>:<subname>"

Examples
  jdbc:mysql:database
  jdbc:mysql://host/database
  jdbc:mysql://host:port/database

• Defaults: host=localhost, port=3306

Connecting to Database

• Connection is established
  Connection con = DriverManager.getConnection(URL,USERID,PWD);

• Connection properties (class Properties)

• Close the connection
  con.close();
Simple SQL Statements

• **Statement** object for invocation

```java
stmt = con.createStatement();
ResultSet rset = stmt.executeQuery("SELECT address,script,type FROM worklist");
```

• **ResultSet** object for result processing

Impedance Mismatch

• Example: SQL in Java:
  – Java uses int, char[..], objects, etc
  – SQL uses tables
• **Impedance mismatch** = incompatible types
• Why not use only one language?
  – SQL cannot do everything that the host language can do
• Solution: use cursors
Using Cursors

- Access to tuples
  - ResultSet object manages a cursor for tuple access
  - Example

```
Statement stmt=con.createStatement();
ResultSet rset=stmt.executeQuery("SELECT ...");
while (rset.next()) {
...
}
rset.close();
```

Accessing Attributes (Columns)

- Access to columns of a tuple
  - Using column index or column name
  - Example

```
while (rset.next()) {
    // return the value of the first column as a String
    String address = rset.getString(1);
    // return the value of the column “type” as a String
    String type = rset.getString("type")
    ...
}
```
More on Cursors

• Cursors can also modify a relation
  rset.updateString("script", "ebay");
  rset.updateRow(); // updates the row in the data source

• The cursor can be a scrolling one: can go forward, backward
  first(), last(), next(), previous(), absolute(5)

• We can determine the order in which the cursor will get tuples by the ORDER BY clause in the SQL query

Inserting a row with Cursors

rs.moveToInsertRow(); // moves cursor to the insert row
rs.updateString(1, "Lausanne"); // updates the first column of
  // the insert row to be Lausanne
rs.updateInt(2, 35); // updates the second column to be 35
rs.updateBoolean(3, true); // updates the third column to true
rs.insertRow();
rs.moveToCurrentRow();
Dynamic JDBC Statements

• Variables within SQL statement
• *Precompiled* once, multiple executions
  – Faster execution
• *PreparedStatement* for invocation

```java
PreparedStatement stmt = con.prepareStatement(
    "SELECT * FROM data WHERE date = ?");
stmt.setDate (1, j_date);
ResultSet rset = stmt.executeQuery();
```

SQL Data Types

• For passing parameters to prepared statements specific SQL data types are needed

• Example
  ```java
  java.util.Date jd = new java.util.Date();
  java.sql.Date j_date = new java.sql.Date(jd.getTime());
  ```
Update Statements

• Updates have no result set
  int result = stmt.executeUpdate("delete from worklist");
• Return value of executeUpdate
  – DDL-statement: always 0
  – DML-statement: number of tuples

Error Handling

• Each SQL statement can generate errors
  – Thus, each SQL method should be put into a try-block
• Exceptions are reported through exceptions of class SQLException
import java.sql.*;
public class JdbcDemo {
    public static void main(String[] args) {
        try {
            Class.forName(com.pointbase.jdbc.jdbcUniversalDriver);
        } catch (ClassNotFoundException exc) {
            System.out.println(exc.getMessage());
        }
        try {
            Connection con =
                DriverManager.getConnection("jdbc:jdbc:demo",
                        "tux","penguin");
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM data");
            while (rs.next()) {
                // ... process result tuples ...
            }
        } catch (SQLException exc) {
            System.out.println("SQLException: " +
                        exc.getMessage());
        }
    }
}
ResultSet Metadata

- `java.sql.ResultSetMetaData` describes the structure of a result set object
- Information about a ResultSet object
  - Names, types and access properties of columns

Database Metadata

- `java.sql.DatabaseMetaData` provides information about the database (schema etc.)
- Information about the database
  - Name of database
  - Version of database
  - List of all tables
  - List of supported SQL types
  - Support of transactions
Example

```java
ResultSet rset = stmt.executeQuery("SELECT * FROM data");
ResultSetMetaData rsmeta = rset.getMetaData();
int numCols = rsmeta.getColumnCount();

for (int i=1; i<=numCols; i++) {
    int ct = rsmeta.getColumnType(i);
    String cn = rsmeta.getColumnName(i);
    String ctn = rsmeta.getColumnTypeName(i);
    System.out.println("Column #" + i + ": " + cn + " of type " + ctn + " (JDBC type: " + ct + ")");
}
```

Summary

- JDBC is a powerful way to connect to a Relational Database
- All we learned is how SQL can be used within the programming language
- Independent of any RDBMS implementation