Introduction to JDBC

Michelle Lee, Ye Wu & Jeff Offutt

http://www.cs.gmu.edu/~offutt/

SWE 642
Software Engineering for the World Wide Web

sources: Java for the Web with Servlets, JSP and EJB, Kurniawan, New Riders
Professional Java Server Programming, Patzer, Wrox

JDBC

• JDBC API allows Java programs to connect to DBs

• Provides cross-vendor connectivity and data access across relational databases from different vendors

• Classes and interfaces allow users to access the database in a standard way

• The JVM uses the JDBC driver to translate generalized JDBC calls into vendor specific database calls
JDBC=Java Data Base Connectivity?

• Most people believe that JDBC stands for Java Data Base Connectivity
• But not quite—it used to be, but now is a trademarked name
• Excerpt:
  – “JDBC (TM) is a Java (TM) API for executing SQL statements. (As a point of interest, JDBC is a trademarked name and is not an acronym; nevertheless, JDBC is often thought of as standing for ‘Java Database Connectivity’.)”

JDBC Drivers

1. JDBC-ODBC bridge
2. Part Java, Part Native Driver
3. Intermediate DAccess Server
4. Pure Java Drivers
1. JDBC-ODBC Bridge

- ODBC (Open Database Connectivity)
  - A set of APIs for database access
  - Originally only for Windows platforms, later extended to non-Windows platforms
- Originally C interfaces
- Hard to learn
- The standard JDK includes classes for the JDBC-ODBC bridge (sun.jdbc.odbc.JdbcOdbcDriver)
- There is no need for additional installation, apart from having to configure the ODBC driver by creating data source names
2. Part Java, Part Native Driver

- A mixture of Java implementation and vendor-specific native APIs for data access

- This is similar to type 1 except that these have one less layer to go through and so is faster

- The native JDBC driver (part Java, part native code) must be installed on each client along with the vendor-specific native language API
3. Intermediate Database Access Server

- Based on intermediate (middleware) database servers

- Connect to various database servers via an intermediate server that acts like a gateway for multiple database servers

- The intermediate server can abstract details to connections to database servers

3. Intermediate Database Access Server (2)

```
<table>
<thead>
<tr>
<th>Java Application</th>
<th>JDBC API</th>
<th>JDBC Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Data Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Native Driver</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>JDBC Driver Server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
```
4. Pure Java Driver

- Convert the JDBC API calls to direct network calls
  - Using vendor-specific networking protocols
  - Making direct socket connections with the database
- It is the most efficient method to access database, both in performance and development time
- It is the simplest to deploy
- All major database vendors provide Type 4 JDBC drivers for their databases
  - Also available from third party vendors
- A list of JDBC drivers:
  - http://www.java2s.com/Tutorial/java/0340_Database/AListofJDBCDriversconnectionstringdrivername.htm

4. Pure Java Driver (2)
Typical JDBC Programming Procedure

1. Load the database driver
2. Obtain a connection
3. Create and execute statements
4. Use result sets to navigate through the results
5. Close the connection

Driver Manager

• The purpose of the java.sql.DriverManager class in JDBC is to provide a common access layer on top of different database drivers used in an application

• DriverManager requires that each driver required by the application must be registered before use

• Load the database driver using ClassLoader:
  – Class.forName("oracle.jdbc.driver.OracleDriver");
Connecting to a Database

• Type 4 JDBC Driver
  – Oracle Server
    `Class.forName("oracle.jdbc.driver.OracleDriver");
    con = DriverManager.getConnection(
      "jdbc:oracle:thin:@apollo.vse.gmu.edu:1521:ite10g",
      "your_username", "your_oracle_password");
  – “your_username” is the same as your Mason ID
  – “your_oracle_password” is created when you activate
    your Oracle account at [https://access.vse.gmu.edu/](https://access.vse.gmu.edu/)

  – MySQL Server
    `Class.forName("org.gjt.mm.mysql.Driver");
    con = DriverManager.getConnection
      ("jdbc:mysql://localhost/databasename", uid, passwd);

Creating and Executing SQL Statements

• Statement object
  – `Statement statement = con.createStatement();`
  – `statement.executeUpdate("CREATE TABLE
    STUDENT");`
  – `statement.executeQuery("SELECT * FROM
    STUDENT");`
SQL Statements

- Create Tables
  
  ```sql
  CREATE TABLE table (field type [(size)] [NOT NULL] 
  [index1], ..., [CONSTRAINT multifieldindex ...)
  ```

- Create Index
  
  ```sql
  Create [UNIQUE] INDEX index ON table (field
  [ASC|DESC],...) [WITH {PRIMARY|DISALLOW NULL|IGNORE NULL}]
  ```

- Drop
  
  ```sql
  Drop TABLE table
  Drop INDEX index ON table
  ```

SQL Statements (2)

- Alter Table
  
  ```sql
  ALTER TABLE table {ADD {COLUMN field type...} | 
  DROP {COLUMN field|CONSTRAINT indexname}}
  ```

- DELETE
  
  ```sql
  DELETE FROM table WHERE criteria
  ```

- Insert
  
  ```sql
  INSERT INTO target [(field1, ...)] VALUES (value1,...)
  ```

- Update
  
  ```sql
  UPDATE table SET newvalue WHERE criteria
  ```
SQL Statements (3)

- Select
  
  ```sql
  SELECT {*
  | table.*
  | [table].field
  FROM tableexpression
  WHERE {criteria
  | [NOT] [IN]
  | (value1...)}
  [GROUP BY] [HAVING] [ORDER BY] ....
  ```

- Select into
  
  ```sql
  SELECT fields INTO newtable FROM SOURCE
  ```

- Select subquery
  
  ```sql
  SELECT selectstatement
  (SELECT selectstatement (SELECT selectatement))
  ```

Creating Tables Examples

- Creating a Coffee table
  
  ```sql
  CREATE TABLE COFFEES (COF_NAME VARCHAR(32), SUP_ID INTEGER, PRICE FLOAT, SALES INTEGER, TOTAL INTEGER)
  ```

- Creating JDBC statements
  
  ```java
  Statement stmt = con.createStatement();
  ```

- Execute a statement
  
  ```java
  stmt.executeUpdate("CREATE TABLE COFFEES " + "(COF_NAME VARCHAR(32), SUP_ID INTEGER, PRICE FLOAT," + "SALES INTEGER, TOTAL INTEGER)");
  ```
Checking to See if a Table Exists

Use metadata about the table

```java
DatabaseMetaData dmd = con.getMetaData();
ResultSet rs = dmd.getTables (null, null, "COFFEES", null);
if (rs == null)
{
    // table does not exist, create it.
}
```

Execute Statements

- This uses executeUpdate because the SQL statement contained in createTableCoffees is a data definition language (DDL) statement
- DDL statements are executed with executeUpdate
  - Create a table
  - Alter a table
  - Drop a table
- executeUpdate is also used to execute SQL statements that update a table
Execute Statements

• `executeUpdate` is used far more often to update tables than to create them
  – We create a table once but update it many times

• The method used most often for executing SQL statements is `executeQuery`

• `executeQuery` is used to execute SELECT statements
  – SELECT statements are the most common SQL statements

Entering Data to a Table

```java
Statement stmt = con.createStatement();
stmt.executeUpdate("INSERT INTO COFFEES " + "VALUES ('Colombian', 101, 7.99, 0, 0)" );
stmt.executeUpdate("INSERT INTO COFFEES " + "VALUES ('French_Roast', 49, 8.99, 0, 0)" );
stmt.executeUpdate("INSERT INTO COFFEES " + "VALUES ('Espresso', 150, 9.99, 0, 0)" );
stmt.executeUpdate("INSERT INTO COFFEES " + "VALUES ('Colombian_Decaf', 101, 8.99, 0, 0)" );
stmt.executeUpdate("INSERT INTO COFFEES " + "VALUES ('French_Roast_Decaf', 49, 9.99, 0, 0)" );
```
**Prepared Statement**

- A PreparedStatement object can hold precompiled SQL statements.
- If the same SQL statement is executed many times with different parameters, it is more efficient to use a PreparedStatement object.
- Examples:
  ```java
  PreparedStatement pstat = connection.prepareStatement("Insert into student (title, ...) values (?, ?, ..)");
  pstat.setString(1, name);
  pstat.executeUpdate();
  ```

**Getting and Using Data From Tables**

```java
ResultSet rs = stmt.executeQuery("SELECT COF_NAME, PRICE FROM COFFEES");
while (rs.next())
{
    String s = rs.getString("COF_NAME");
    float n = rs.getFloat("PRICE");
    System.out.println(s + " " + n);
}
```
# Navigating Result Sets

## JDBC Types Mapped to Java Types

<table>
<thead>
<tr>
<th>JDBC Types</th>
<th>Java Types</th>
<th>JDBC Types</th>
<th>Java Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>String</td>
<td>DATE</td>
<td><code>java.sql.Date</code></td>
</tr>
<tr>
<td>VARCHAR</td>
<td>String</td>
<td>TIME</td>
<td><code>java.sql.Time</code></td>
</tr>
<tr>
<td>LONGVARCHAR</td>
<td>String</td>
<td>TIMESTAMP</td>
<td><code>java.sql.Timestamp</code></td>
</tr>
<tr>
<td>TINYINT</td>
<td>short</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>Int</td>
<td>JAVAOBJECT</td>
<td>Object</td>
</tr>
<tr>
<td>BIGINT</td>
<td>Long</td>
<td>BLOB</td>
<td><code>java.sql.Blob</code></td>
</tr>
<tr>
<td>REAL</td>
<td>Float</td>
<td>CLOB</td>
<td><code>java.sql.Clob</code></td>
</tr>
<tr>
<td>FLOAT</td>
<td>Double</td>
<td>ARRAY</td>
<td><code>java.sql.Array</code></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Double</td>
<td>STRUCT</td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>boolean</td>
<td>REF</td>
<td></td>
</tr>
</tbody>
</table>

```java
ResultSet rs = statement.executeQuery("select * from student");
while (rs.next())
{
    System.out.print(rs.getString("name") + "\n");
}

ResultSetMetaData metaData = rs.getMetaData();
metaData.getColumnName(I);
metaData.getColumnType(I);
```

---

---
Navigating Result Sets

- A default ResultSet object is read-only (not updateable) and has a cursor that moves forward only (next()).

- Scrollable result sets have more operations
  - First, last, absolute, relative, next, previous,
  - beforeFirst, afterLast, isFirst isBeforeFirst, isLast, isAfterLast

Batch Update

- What is batch update?
  - Send multiple update statement in a single request to the database

- Why batch update?
  - Better performance

- How do we perform batch update?
  - Statement.addBatch (sqlString);
  - Statement.executeBatch();
Transaction Support

• A database transaction is a work unit treated in a coherent and reliable way independent of other transactions
• If a transaction starts, all pieces must complete
• Two main purposes:
  – Allow recovery from errors when operations do not complete
  – Provide concurrency control
• A database transaction must be ACID:
  – Atomic
  – Consistent
  – Isolated
  – Durable

Transaction Support (2)

• By default, a connection is in auto-commit mode
  – Each individual SQL statement is treated as a transaction
• To turn off auto-commit mode:
  – connection.setAutoCommit(false);
• Commit and Rollback
  – Commit – a call to commit() will commit everything that was done since the last commit was issued
  – Rollback – a call to rollback() will undo any changes since the last commit
Exception Handling

• Objectives
  – Handle exceptions gracefully
  – Maintain the integrity of the database

• Exception handling with a rollback:
  try {
    con.setAutoCommit (false);
    statement.executeUpdate ("...");
    statement.executeUpdate ("...");
    commit(); con.setAutoCommit (true);
  }
  catch (SQLException ex) {
    con.rollback();
  }

• SQL warnings
  ResultSet rs = statement.executeQuery ("Select * from student");
  SQLWarning warn = statement.getWarnings();
  if (warn != null)
    System.out.println (warn.getMessage());

JDBC Summary

• Most web applications use databases to store data in a persistent way

• The techniques for accessing databases from Java programs are identical in web applications and in stand-alone Java programs

• This lecture does not teach how to set up or use a database or the details of constructing SQL queries
  – INFS 614, Database Management
  – INFS 740, Database Programming for the World Wide Web