JDBC Concepts

Kyle Brown
Knowledge Systems Corporation
Outline

- What is the JDBC?
- What other persistence mechanisms are available?
- What facilities does it offer?
- How is it used?
What is the JDBC

- JDBC is the Java DataBase Connectivity specification
- A platform independent API for relational databases
- The Java counterpart to ODBC
JDBC Architecture

- **Database-independent JDBC API**
  - single programming interface for all interactions
  - hides complexity from the Java programmer

- **Database-specific Drivers**
  - convert generic API calls to native database calls
  - JDBC-ODBC Bridge available from Sun
  - Database vendors offer their own native drivers
JDBC Architecture

Java Application

JDBC

JDBC-ODBC Bridge

ODBC

MS-Access, etc.

Native Driver

DB2, Oracle, ...
JDBC Competitors

- Other Relational Database connectivity packages
- Object Databases
  - GemStone, Object Design, O2
What does JDBC provide?

- A generic way to query and update relational tables and translate the results into Java datatypes.

- A few basic concepts
  - Driver
  - Connection
  - Statement
  - ResultSet
Step 0: Loading a driver

- Each Driver is explicitly loaded by your application code.
  
  ```java
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  ```

- You can have more than one Driver loaded at once

- Only needs to happen once (good candidate for static initializer)

- Loading registers the Driver with the DriverManager
Step 1: Connecting to a database

- JDBC specifies individual databases with a URL format
  \[jdbc:subprotocol:subname\]
  
  - subprotocol
    - driver name (db2, odbc, etc.)
  
  - subname
    - Database name
Database Connection

- The DriverManager searches for a registered driver that can use this protocol and returns a Connection.

  ```java
  Connection conn = DriverManager.getConnection(url, login, password);
  ```
Connections

- A Connection represents a session with a particular database
- It can provide information about the database and its tables
- It can create Statements, commit, and rollback transactions.
Connection methods

- Create a Statement object
  ```java
  public abstract Statement createStatement() throws SQLException
  ```

- Create a PreparedStatement object
  ```java
  public abstract PreparedStatement prepareStatement(String sql) throws SQLException
  ```
Connection methods

- Commit a transaction
  public abstract void commit() throws SQLException

- Rollback a transaction
  public abstract void rollback() throws SQLException

- Change auto-commit mode
  public abstract void setAutoCommit(boolean autoCommit) throws SQLException
A simple example

// Load a driver and create a connection

Class.forName
("sun.jdbc.odbc.JdbcOdbcDriver");

Connection currentConnection =
    DriverManager.getConnection
("jdbc:odbc:MS Access 7.0", "Sample", "Sample");

// Connections auto-commit by default
Step 2: Creating Statements

- A Statement object is used for executing a dynamic SQL statement and obtaining the results produced by it.
- Statements return ResultSets as the result of executing SQL.
Statement methods

- Execute a SQL SELECT statement that returns a single ResultSet
  
  public abstract ResultSet executeQuery(String sql) throws SQLException

- Execute a SQL DELETE, UPDATE or INSERT statement
  
  public abstract int executeUpdate(String sql) throws SQLException
Example continued

// Load a driver and create a connection

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection currentConnection = DriverManager.getConnection("jdbc:odbc:MS Access 7.0", "Sample", "Sample");

// Create a statement and execute a query

Statement query = currentConnection.createStatement();
ResultSet result = query.executeQuery("SELECT * FROM Samples");
PreparedStatements

- A SQL statement can be pre-compiled and stored in a PreparedStatement object.
- This object can then be used to efficiently execute this statement multiple times.
- You simply change the variable parameters and re-execute the statement.
PreparedStatement methods

● Setting Parameter Values
  » `setInt(int, int)`, `setDate(int, Date)`, etc.
  » One `setXXX` method per datatype

● `executeQuery()`

● `executeUpdate()`
PreparedStatement example

// Load a driver and create a connection as previous

// Create a preparedStatement

PreparedStatement insert = currentConnection.createPreparedStatement(""""""INSERT INTO Samples (Name, Company, ContactDate) VALUES (? ? ?)""");

// The following code would be executed many times
insert.setString(1, "John Williams");
insert.setString(2, "CP&L");
insert.setDate(3, new Date());
insert.executeUpdate();
Step 3: Processing a ResultSet

- A ResultSet provides access to a table of data generated by executing a Statement.
- A ResultSet maintains a cursor pointing to its current row of data.
- Initially the cursor is positioned before the first row.
- The `next()` method moves the cursor to the next row. The table rows are retrieved in sequence.
Processing a ResultSet

Underlying Relational Rows

"ABC" "XYZ" 123 Jan 17, 1997
"ABC" "XYZ" 123 Jan 17, 1997
"ABC" "XYZ" 123 Jan 17, 1997
"ABC" "XYZ" 123 Jan 17, 1997
"ABC" "XYZ" 123 Jan 17, 1997

next() advances the ResultSet
ResultSet column values

- ResultSet methods retrieve column values for the current row.
- Within a row its column values can be accessed in any order.
- You can retrieve values either using the index number of the column, or by using the name of the column.
### ResultSet methods

<table>
<thead>
<tr>
<th>String getString(int)</th>
<th>String getString(String)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date getDate(int)</td>
<td>Date getDate(String)</td>
</tr>
<tr>
<td>Float getFloat(int)</td>
<td>Float getFloat(String)</td>
</tr>
<tr>
<td>int getInt(int)</td>
<td>int getInt(String)</td>
</tr>
<tr>
<td>byte[] getBytes(int)</td>
<td>byte[] getBytes(String)</td>
</tr>
<tr>
<td>InputStream getAsciiStream(int)</td>
<td>InputStream getAsciiStream(String)</td>
</tr>
<tr>
<td>InputStream getBinaryStream(int)</td>
<td>InputStream getBinaryStream(String)</td>
</tr>
</tbody>
</table>

also for Shorts, Doubles, etc.
ResultSet column values

Underlying Relational Row

<table>
<thead>
<tr>
<th>Name</th>
<th>Company</th>
<th>Contact Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>“ABC”</td>
<td>“XYZ”</td>
<td>Jan 17, 1997</td>
</tr>
</tbody>
</table>

aResultSet.getString(“Name”) -- returns “ABC”

aResultSet.getDate(3) -- returns a Date object
Simple Example

// Need the previous driver load & connection

Statement query = currentConnection.createStatement();
ResultSet result = query.executeQuery("SELECT * FROM Samples");

// iterate through result and print out

while (result.next()) {
    System.out.println(result.getString("Name"));
    System.out.println(result.getString("Company"));
    System.out.println(
        result.getDate("Contact Date")
    );
}
Special Considerations

● Exception Handling
  » Most of the methods in Statement and Connection may throw a SQLException

● Must be trapped and handled in one of two ways
  » recover locally (works for a Warning)
  » pass another exception up the chain
JDBC Class Diagram

- Statement
  - creates
  - results in
  - PreparedStatement
    - creates
    - CallableStatement
      - interface
      - class

- Connection
  - creates
  - ResultSet
    - returns
    - ResultSetMetaData
      - returns
      - DatabaseMetaData
        - contains
        - DriverManager
          - contains
          - Driver
Summary

- We’ve seen what JDBC is
  » Portable architecture for RDB connectivity
- We’ve seen what JDBC provides
  » How Drivers are loaded into Java
  » How Connections are obtained
  » How Statements are created and executed
  » How ResultSets are processed
- We’ve worked through a short example
  » retrieving and processing rows from a query