Application Programming for Relational Databases

Cyrus Shahabi
Computer Science Department
University of Southern California
shahabi@usc.edu
Overview

JDBC Package

Connecting to databases with JDBC

Executing select queries

Executing update queries
Overview

- **Role of an application**: Update databases, extract info, through:
  - User interfaces
  - Non-interactive programs

- **Development tools** (Access, Oracle):
  - For user Interfaces

- **Programming languages** (C, C++, Java, ...):
  - User Interfaces
  - Non-Interactive programs
  - More professional
Client server architecture

- **Database client:**
  - Connects to DB to manipulate data:
    - Software package
    - Application (incorporates software package)

- **Client software:**
  - Provide general and specific capabilities
  - Oracle provides different capabilities as Sybase (its own methods, ... )
Client server architecture

- Client-Server architectures:
  - 2 tier
  - 3 tier

- Layer 1:
  - user interface

- Layer 2:
  - Middleware

- Layer 3:
  - DB server

- Middleware:
  - Server for client
  - Client for DB

FIGURE 8.1
A variety of client-server architectures for information systems
Client server architecture

Example: Web interaction with DB

- Layer 1: web browser
- Layer 2: web server + cgi program
- Layer 3: DB server

FIGURE 8.2
Architecture of a Web site supported by databases
Client server architecture

- Application layer (1):
  - User interfaces
  - Other utilities (report generator, …)
  - Connect to middleware
  - Can connect to DB too
  - Can have more than one connection
  - Can issue SQL, or invoke methods in lower layers.

- Middleware layer (2):
  - More reliable than user applications
Database interaction in Access

- Direct interaction with DB
  - For implementing applications
  - Not professional
- Developer edition:
  - Generates stand alone application
- Access application:
  - GUI + “Visual Basic for Applications” code
Database interaction in Access

- **Connection to DB through:**
  - Microsoft Jet database engine
    - Support SQL access
    - Different file formats
  - Other Database Connectivity (ODBC)
    - Support SQL DBs
    - Requires driver for each DB server
      - Driver allows the program to become a client for DB
    - Client behaves Independent of DB server
Database interaction in Access

Making data source available to ODBC application:

- Install ODBC driver manager
- Install specific driver for a DB server
- Database should be registered for ODBC manager

How application works with data source:

- Contacts driver manager to request for specific data source
- Manager finds appropriate driver for the source

FIGURE 8.3
The ODBC architecture for database access
Database interaction in Java

- **Includes:**
  - **Java.sql package**
    - Set of classes
    - Supports JDBC (java database connectivity?) strategy, independent of the DB server

- **Difference between JDBC and ODBC:**
  - JDBC driver manager is part of the application
Database interaction in Embedded SQL

- Extension of a language (C++, C) with new commands:
  - Void addEmployee( char *ssn, char *lastname,
    char *firstname) {
    • Exec SQL
      - Insert into customer( ssn, lastname, firstname )
        values( :ssn, :lastname, :firstname )
  }

- Not legal language
- Compilation precedes by a translation preprocessor from embedded SQL into legal C

- Advantages: ???
- Disadvantages:
  - Not portable between database systems
  - Difficult debugging
JDBC: Architecture

- Four Architectural Components:
  - Application (initiates and terminates connections, submits SQL statements)
  - Driver manager (load JDBC driver)
  - Driver (connects to data source, transmits requests and returns/translates results and error codes)
  - Data source (processes SQL statements)
JDBC Architecture (Contd.)

Four types of drivers:

**Bridge:**
- Translates SQL commands into non-native API. Example: JDBC-ODBC bridge. Code for ODBC and JDBC driver needs to be available on each client.

**Direct translation to native API, non-Java driver:**
- Translates SQL commands to native API of data source. Need OS-specific binary on each client.

**Network bridge:**
- Send commands over the network to a middleware server that talks to the data source. Needs only small JDBC driver at each client.

**Direction translation to native API via Java driver:**
- Converts JDBC calls directly to network protocol used by DBMS. Needs DBMS-specific Java driver at each client.
**JDBC package**

- **Collection of interfaces and classes:**
  - **DriverManager:** Loads the driver
  - **Driver:** creates a connection
  - **Connection:** represents a collection
  - **DatabaseMetaData:** information about the DB server
  - **Statement:** executing queries
  - **PreparedStatement:** precompiled and stored query
  - **CallableStatement:** execute SQL stored procedures
  - **ResultSet:** results of execution of queries
  - **ResultSetMetaData:** meta data for ResultSet

- **Reminder:** Each JDBC package implements the interfaces for specific DB server
JDBC, different strategies

- Strategies to USE JDBC
  - JDBC-ODBC bridge
    - Con: ODBC must be installed
  - JDBC database client
    - Con: JDBC driver for each server must be available
  - JDBC middleware client
    - Pro: Only one JDBC driver is required
    - Application does not need direct connection to DB (e.g., applet)
Connecting with JDBC

- Database connection needs two pieces
  - JDBC package driver class name
    - Package driver provide connection to DB
  - URL of the database
    - JDBC package designator
    - Location of the server
    - Database designator, in form of:
      - Server name, Database name, Username, password, ...
      - Properties
Connecting to DB with JDBC

Step 1: Find, open and load appropriate driver

1. Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
2. Class.forName(“oracle.thin.Driver”);
3. Class.forName(“symantec.dbAnywhere.driver”);

Or:

4. DriverManager.registerDriver(your jdbc driver);

Informs availability of the driver to “DriverManager” (registers the driver with DriverManager)

(Example 1)
Connecting to DB with JDBC

- **Step 2: Make connection to the DB**
  - `Connection conn = DriverManager(URL, Properties);`
    - Properties: specific to the driver
  - `URL = Protocol + user`
    - Protocol= jdbc:<subprotocol>:@<subname>
      - E.g.: jdbc:odbc:mydatabase
      - E.g.: jdbc:oracle:thin://oracle.cs.fsu.edu/bighit

- (Example 1)
Connecting to DB with JDBC

- **Step 3: Make Statement object**
  - Used to send SQL to DB
    - `executeQuery()`: SQL that returns table
    - `executeUpdate()`: SQL that doesn’t return table
    - `Execute()`: SQL that may return both, or different thing

- **Step 4: obtain metadata (optional)**
  - `DatabaseMetaData` object
    - `getTimeDatefunctions`: all date and time functions
    - ....

  - *(Example 2)*
Executing select queries

- Step 5: issue select queries
  - Queries that return table as result
  - Using statement object
  - Uses executeQuery() method
  - Return the results as ResultSet object
    - Meta data in ResultSetMetaData object
  - Every call to executeQuery() deletes previous results

  (Example 2)
Executing select queries

- Step 6: retrieve the results of select queries
  - Using ResultSet object
    - Returns results as a set of rows
    - Accesses values by column name or column number
    - Uses a cursor to move between the results
    - Supported methods:
      - JDBC 1: scroll forward
      - JDBC 2: scroll forward/backward, absolute/relative positioning, updating results.
      - JDBC 2: supports SQL99 data types (blob, clob, ...)
  - Meta data in ResultSetMetaData:
    - Number of columns, Column names, column type name,
  - (Example 2)
## Matching Java and SQL Data Types

<table>
<thead>
<tr>
<th>SQL Type</th>
<th>Java class</th>
<th>ResultSet get method</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>Boolean</td>
<td>getBoolean()</td>
</tr>
<tr>
<td>CHAR</td>
<td>String</td>
<td>getString()</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>String</td>
<td>getString()</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>Double</td>
<td>getDouble()</td>
</tr>
<tr>
<td>FLOAT</td>
<td>Double</td>
<td>getDouble()</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer</td>
<td>getInt()</td>
</tr>
<tr>
<td>REAL</td>
<td>Double</td>
<td>getFloat()</td>
</tr>
<tr>
<td>DATE</td>
<td>java.sql.Date</td>
<td>getDate()</td>
</tr>
<tr>
<td>TIME</td>
<td>java.sql.Time</td>
<td>getTime()</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>java.sql.TimeStamp</td>
<td>getTimestamp()</td>
</tr>
</tbody>
</table>
Executing update queries

- Step 7: issue update queries
  - Queries that return a row count (integer) as result
    - Number of rows affected by the query
    - -1 if error
  - Using statement object
  - Uses executeUpdate() method

- Meta data in ResultSetMetaData object
  - (Example 3)
Executing update queries

- Step 8: More Advanced
  - Use PreparedStatement
    - faster than regular Statement
    - (Example 4)

- Cursors
  - forward, backward, absolute/relative positions
  - (Example 5)
Mapping Objects

- To read attributes that are retrieved as objects:
  - Example: Spatial data types
    - (Example 6: it is for point, line and other types are similar)
    - Read “Oracle Spatial – User’s Guide and Reference”
      - Chapter 2 for geometry types
      - Chapter 9-14 for geometry functions
    - Read “Oracle Spatial API Document” for reading geometry types in Java