Database Application Development
JDBC and SQLJ

CS430/630
Lecture 14

Slides based on “Database Management Systems” 3rd ed., Ramakrishnan and Gehrke

Outline
- Embedded SQL
- Dynamic SQL
- JDBC (API)
- SQLJ (Embedded)
- Stored procedures

APIs: Alternative to Embedding
- Use library that implements API of DBMS calls
- No need to modify compilation process
- API: standardized interface with objects and procedures
- Pass SQL strings from the programming language
- API returns result sets in language-friendly form
- DBMS API for Java is Sun’s JDBC
  - It is mainly a specification
  - DBMS-neutral
  - Each DBMS vendor can implement its own version
  - JDBC driver traps calls, translates them into DBMS-specific code
  - Packages java.sql.*, javax.sql.*
  - Collection of classes and interfaces

JDBC: Architecture
- Application initiates/terminates connections; submits SQL statements
- Driver Manager loads JDBC driver
- JDBC Driver (Oracle) connects to data source; transmits requests and returns/converts results and error codes
- JDBC Driver (MSSQL) processes SQL statements

Driver Types
- Bridge
  - Translates SQL commands into non-native API
  - Example: JDBC-ODBC bridge
- Direct translation to native API via non-Java driver
  - Translates SQL commands to native API of data source
  - Need OS-specific binary on each client
- Direct 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
- Network bridge
  - Send commands over the network to middleware server
  - Needs only small JDBC driver at each client
Using JDBC

- 3 steps to submit a database query:
  1. Load the JDBC driver
  2. Connect to the data source
  3. Execute SQL statements

JDBC Driver Management

- All drivers are managed by the DriverManager class
- Loading a JDBC driver:
  - From inside the Java code:
    Class.forName("oracle.jdbc.driver.OracleDriver");
  - When starting the Java VM:
    -Djdbc.drivers=oracle/jdbc.driver

Connections in JDBC

- Interaction with data source through sessions
  - A connection identifies a logical session
  - JDBC URL: jdbc:<protocol>:<otherParameters>
  - Example:
    String url="jdbc:oracle:www.bookstore.com:3083";
    Connection conn;
    try{
      conn = DriverManager.getConnection(url, "user", "password");
    } catch SQLException e {…}
  - Many other forms: check Java API
  - Properties of connection: autocommit, connection pooling, etc.

Executing SQL Statements

- Statement class
  - 2 subclasses:
    PreparedStatement (semi-static SQL statements)
    CallableStatement (stored procedures)
  - PreparedStatement class:
    - Precompiled, parametrized SQL statements
    - Structure is fixed
    - Values of parameters are determined at run-time

Example

```java
/* local variables */
int sid=10;
String sname="Yuppy";
int rating = 5;
float age = 40.0;

/* creating the statement object */
String sql="INSERT INTO Sailors VALUES(?,?,?,?)";
PreparedStatement pstmt=conn.prepareStatement(sql);
```

Example (contd.)

```java
/* initialize parameters */
pstmt.clearParameters();
pstmt.setInt(1,sid);
pstmt.setString(2,sname);
pstmt.setInt(3, rating);
pstmt.setFloat(4,age);

/* no results will be returned, use executeUpdate() method */
int numRows = pstmt.executeUpdate();

executeUpdate() returns the number of affected records
```
Retrieving Data: ResultSet class

- `Statement.executeQuery` returns data
  - encapsulated in a ResultSet object (a cursor)
- `PreparedStatement` can also be used for this purpose
- Retrieval by attribute name or position

```java
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT sname FROM Sailors WHERE rating = " + rating);
```

// rs is now a cursor
while (rs.next()) {// process the data
    String name = rs.getString("sname"); // rs.getString(1);
}

```

ResultSet

- ResultSet is a very powerful cursor:
  - `next()`, `previous()`, `first()`, `last()`
  - `absolute(int num)`: moves to the row with the specified number
  - `relative (int num)`: moves forward or backward

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>getFloat()</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer</td>
<td>getLong()</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>

JDBC: Exceptions and Warnings

- Most of `java.sql` methods throw `SQLException`
- `SQLWarning` is a subclass of `SQLException`
  - not as severe (their existence has to be explicitly tested)
    ```java
    try {
        stmt=conn.createStatement();
        ... SQLWarning warning=conn.getWarnings();
        while(warning != null) {
            // handle SQLWarnings;
            warning = warning.getNextWarning();
        }
        conn.clearWarnings();
    } catch( SQLException SQLe) {
        // handle the exception
    }
    ```

Examining Database Metadata

- `DatabaseMetaData` object gives catalog information

```java
DatabaseMetaData md=conn.getMetaData();
ResultSet trs=md.getTables(null,null,null,null);
while(trs.next()) {
    String tableName = trs.getString("TABLE_NAME");
    System.out.println("Table: " + tableName);
    ResultSet crs = md.getColumns(null,null,tableName, null);
    while (crs.next()) {
        System.out.println(crs.getString("COLUMN_NAME"));
    }
}
```
SQLJ complements JDBC with a (semi-)static query model

Compiler can perform syntax checks, type checking, schema/query consistency

```sql
cursor_name = {
    SELECT name, rating INTO :name, :rating
    FROM Books WHERE sid = :sid;
}
```

Compare to JDBC:
```java
sid = rs.getInt(1);
if (sid == 1) { name = rs.getString(2); }
else { name2 = rs.getString(2); }
```