More Database Programming

CS157A

Chris Pollett

Nov. 2, 2005.

Outline

- JDBC
- SQLJ

Introduction

- Last day we went over some JDBC and SQLJ code examples from prior classes.
- Today, we will discuss JDBC and SQLJ in more detail

JDBC Connecting to a Database

- JDBC (Java Database Connectivety) is an API that enable a Java program to send SQL commands to a database.
- The first step in writing a Java program using JDBC is to import the JDBC classes:

  import java.sql.*;
- Then one needs to load the JDBC driver:

  Class.forName("oracle.jdbc.driver.OracleDriver");
- Then one create a connection to the database:

  Connection conn = DriverManager.getConnection(url);
- Here url has the format:

  jdbc:oracle:drivertype:user/password@database

  where drivertype is one of oci7, oci8, or thin(use thin!!) and database can be the name of a tnsnames.ora file entry or of the form host:port:sid. For example,

  `sigma.cs.sjsu.edu:1521:cs157a`

Using Connections

- Once a connection is established, it can be used to send SQL statements to the server.
- There are three main classes of interest to do this: Statement, PreparedStatement, and CallableStatement.

```
Statement stmt = conn.createStatement();
PreparedStatement pstmt = conn.createStatement(sqlString);
CallableStatement cstmt = conn.createStatement(sqlString);
```

- These commands might throw an SQLException. In both cases above, sqlString is an String containing an SQL command which might have one or more ? slots to be filled in later using pstmt or cstmt setXXX (String, Int, etc) method.
- We won't talk about CallableStatement but it can be used to invoke PL/SQL procedures.
- Once we are done with our statements we can call the conn.close().
- conn also has a setAutoCommit method. If this is set to false we might also call conn.commit() or conn.rollback().
- One last useful method is conn.getMetaData()

Executing SQL Queries

- stmt above could be used in one of two kinds of ways:
  - To execute DML or DDL commands:

    String ssn = “123456789”;

    stmt.executeUpdate(“delete from EMPLOYEE where ssn=”+ssn);
  - To execute queries:

    ResultSet rs = stmt.executeQuery(“select * from EMPLOYEE”);
  - Using these commands may generate more than one message back from the database SQLException’s. Let’s say we get such an exception named e. We can use e.getMessage() to get the first message. e.getnextException(), to cycle to the next exception.

Still More on Queries

- It is slightly more convenient to get ResultSet’s using PrepareStatement’s rather than Statement’s

```
PrepararedStatement pstmt =
   conn.prepareStatement(“select * from
   EMPLOYEE where fname=? and salary=?”);
pstmt.setString(1, “bob”);
pstmt.setInt(2, 123);
ResultSet rs = pstmt.executeQuery();
```

Using a ResultSet

- rs is a cursor to a row in the result of the query.
- rs.next() moves the cursor to the next row. The first time next() is called, it moves to the first row. It becomes null when there are no more rows.
- rs.close() releases the cursor and any associated database resources.
- rs.getXXX(columnNum) where XXX is String, Int, etc retrieves the value in the current row of query results for that column. (rs.getXXX(columnName) also works)
- rs.findColumn(columnName) takes a String name for a column and returns the int number of that column.

Metadata

- MetaData can be had both about a ResultSet and a Connection:

```
ResultSetMetaData rsMeta = rs.getMetaData();
int colCount = rsMeta.getColumnCount();
/* other methods include getColumnDisplaySize(),
   getColumnLabel(), getColumnName(), getPrecision(),
   getScale(), getTableName(), getColumnType(),
   getColumnTypeName() */
DatabaseMetaData dbMeta = conn.getMetaData();
String s = dbMeta.getDatabaseProductName();
/* other methods include getDriverName(), getTables(),
   getMaxColumnsInTable() */
```

Error and Warnings

- We already seen an SQLException supports getMessage(), and getNextMessage() methods.
- It also supports getSQLState() and getErrorCode(). The first returns the state of the database according to the X/Open SQL spec. The second returns a vendor specific error code.
- SQLException has a subclass SQLWarning which in addition to the above methods has a getNextWarning methods.
- Unlike exceptions to find out warnings you need to either call conn.getWarnings() or stmt.getWarnings() or rs.getWarning() to see them.

Scrollable ResultSets

- createStatement also has an additional overloaded form:
  Statement stmt = conn.createStatement(resultSetType, resultSetConcurrency);
- The type is of the form ResultSet.XXX where XXX is TYPE_FORWARD_ONLY (default), TYPE_SCROLL_INSENSITIVE (forward, backward random order changes not seen by Java), TYPE_SCROLL_SENSITIVE (forward, backward random order changes seen by Java)
- The concurrency is one of ResultSet.XXX where XXX is CONCUR_READ_ONLY or COCUR_UPDATABLE.
- Depending on the type one might be able to use the following additional methods with ResultSet's:
  - absolute(row), relative(row), first(), last(), previous(), next(), beforeFirst(), afterLast(), isFirst(), isLast(), isBeforeFirst(), isBeforeLast(), getRow()

SQLJ

- Recall from last day we said SQLJ was a standard for embedding SQL into a Java program.
- A SQLJ program is compiled and run in a three step process:

```
sqlj myprog.sqlj
javac myprog.java
java myprog
```

- The only thing that is new over a standard Java program is the first step. This step checks the syntax of the SQL code embedded in your Java and converts this code to JDBC calls.

Writing a SQLJ program

- Need to import the necessary classes:
```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;
import java.io.*;
import oracle.sqlj.runtime.*;
```
- Register the driver:
```
DriverManager.registerDriver( new oracle.jdbc.OracleDriver());
```
- Connect to the database:
```
DefaultContext cx = Oracle.getConnection(connect, login, password,
    autocommit);
DefaultContext.setDefaultContext(cx);
```
- Embed sqlj commands #sql{<sql_statement};

Host Variables

- Variables from the Java language can be used in the SQLJ commands by preceding them with a ‘:’.

```
#sql{select a into :a from t where b = :b1};
```

- One also specify if the Java variable is being written to/from or both using the keywords IN, OUT or, INOUT

```
#sql {select a into :OUT a1 from t where b = :IN b1};
```

- One can also use host variables in expresseion provided they are parenthesized:

```
#sql{ update member set cash_balance = :(x+100) where
    mid = :y};
```

Iterators

- To perform queries in SQLJ where the results might be multiple rows, iterators are used.
- The format of a iterator declaration is

  #sql iterator myIter (colType1 colName1,…, colTypeN, colNameN);

  For example,

  #sql iterator EmpName(String FName, String NName);
- Once defined an iterator can be declared and set equal to the result of a query:

```
 EmpName e = null;
 #sql e = {select FName, LName from EMPLOYEE};
 while(e.next())
 {
     System.out.println("F:"+e.FName() + " L:"+e.LName());
 }
 e.close();
```