SQL and Java

Database Systems Lecture 19
Natasha Alechina
In this Lecture

• SQL in Java
  • SQL from within other Languages
  • SQL, Java, and JDBC

• For More Information
  • Sun Java tutorial: http://java.sun.com/docs/books/tutorial/jdbc
  • Connolly and Begg 29.7
SQL and Other Languages

• Combining SQL and another language
  • Use SQL to run queries on the database
  • Use another language (Java, C, etc) to do the rest of the work: e.g. user interface, or complicated processing
  • Need an interface between the two

• ODBC (Open DB Connectivity) is a common standard
  • Provides an API which is widely supported
  • Allows you to pass queries to a database, and return the results to a program
JDBC

• JDBC is a Java API for database connectivity
  • It is not the same as ODBC but implements a similar specification
  • JDBC enables programmers to write Java applications that
    • Connect to a database
    • Send queries and update statements to the database
    • Retrieve and process the results received from the database in answer to the query
JDBC

SQL and Java
JDBC

- JDBC consists of:
  - The JDBC™ API proper: interfaces, classes and methods for executing SQL statements, retrieving results, and propagating changes back to the database
  - JDBC Driver Manager: a class that defines objects which can connect Java applications to a JDBC driver.
  - JDBC Test Suite
  - JDBC-ODBC Bridge
Using JDBC

- Basic steps when using JDBC
  - Register a database driver
  - Open a connection
  - Pass some queries to the database
  - Process the results as needed
  - Close the connection
  - Deal with any errors

- Preamble: import java.sql.*;
Register a Driver

• We need to register an appropriate driver with the DriverManager
  • There is a different driver for each DBMS
  • We’ll need to use the driver for Oracle:

```java
DriverManager.registerDriver(
    new oracle.jdbc.driver.OracleDriver()
);
```
Open a Connection

- Next we open a connection to the database from the DriverManager
  - We give the address of the database, a username and a password

```java
Connection conn = DriverManager.getConnection(
    "jdbc:oracle:thin:@oracle.cs.nott.ac.uk:1521:maindb",
    "xxx06u", "somepassword");
```

Your username
Your sqlplus password

SQL and Java
Passing Queries to the DB

• Now we can send queries to the DB
  • We do this through a Statement object
  • Each Statement can deal with one query at a time
  • A single Connection can have several statements open at any time

• Statement objects
  • Are created from a Connection
  • The executeUpdate() method runs a query that doesn’t return any results (UPDATE, CREATE TABLE, etc)
  • executeQuery() is used when a result is expected
Creating a table and inserting a record:

```java
Statement sttable = conn.createStatement();
sttable.executeUpdate(
    "CREATE TABLE Fruit(Name VARCHAR(10), Amount INT)"
);
sttable.close();

Statement stinsert1 = conn.createStatement();
stinsert1.executeUpdate(
    "INSERT INTO Fruit VALUES('Apple', 5)"
);
stinsert1.close();
```
Passing Queries to the DB

```java
Statement stinsert2 = conn.createStatement();
stinsert2.executeUpdate("INSERT INTO Fruit VALUES(‘Pumpkin’, 1)“);
stinsert2.close();
```
Processing Query Results

- When a query returns a result
  - We use the Statement object’s `executeQuery` method
  - The results are put in a `ResultSet` object
  - Each Statement can deal with a single `ResultSet` at any one time

- The `ResultSet` object
  - Is essentially a table
  - Has a cursor that points to the current row of data
  - Initially the cursor is positioned *before* the first row
  - The `next()` method moves to the next row, and returns false if there isn’t one
Statement stresult = conn.createStatement();
ResultSet fruit = stresult.executeQuery("SELECT * FROM Fruit");
while(fruit.next()) {
    System.out.println(fruit.getString("Name") + ", " + fruit.getInt("Amount");
}
fruit.close();

SQL and Java

Processing Query Results
Working with ResultSets

• We get values from the ResultSet with
  • `getInt()`
  • `getString()`
  • `getDouble()`
  • `etc.`

• Each takes either
  • The name of the column as a String, or
  • The index of the column as an integer
Advanced ResultSets

• By default a ResultSet
  • Allows you to go over the results once, from start to finish
  • Allows you to read, but not change, the information in the result

• We can change this behaviour so that
  • We can move forward and backwards
  • We can update existing rows
  • We can add rows
  • This is decided when we create the Statement object from the Connection
Creating Statements Again

```java
conn.createStatement(<scroll>, <update>);
```

- `<scroll>` is one of
  - `ResultSet.TYPE_FORWARD_ONLY`
  - `ResultSet.TYPE_SCROLL_SENSITIVE`
  - `ResultSet.TYPE_SCROLL_INSENSITIVE`

- `<update>` is one of
  - `ResultSet.CONCUR_READ_ONLY`
  - `ResultSet.CONCUR_UPDATABLE`
Scrollable ResultSets

- If we use the option `TYPE_SCROLL_SENSITIVE` or `TYPE_SCROLL_INSENSITIVE`:
  - We can move around the ResultSets made from that statement.
  - There are a lot of options available for this.
  - For a result set called `rs`...

```java
rs.first(); rs.absolute(1)
rs.absolute(2)
rs.absolute(3)
rs.relative(-2)
rs.previous(); rs.relative(-1)
rs.next(); rs.relative(1)
rs.relative(2)
rs.absolute(-3)
rs.absolute(-2)
rs.absolute(-1)
rs.last(); rs.absolute(1)
rs.absolute(2)
rs.absolute(3)
```
Updating ResultSets

• If we use the option `CONCUR_UPDATABLE`
  • We can update the values in the result set or add a new row
  • In Oracle you can’t have an updatable forward-only result set
  • Also in Oracle you have to explicitly specify the columns in your `SELECT` statement if you want to update it (no `SELECT *...`)
Updating a Row

// Make an updatable Statement
Statement result2 = conn.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
ResultSet rset2 = result2.executeQuery("SELECT Name, Amount FROM Fruit");
rset2.absolute(2);// set current row to second
rset2.updateInt("Amount", 3); //
rset2.updateRow(); // updates the second row
Inserting a Row

// rset2 is set up as in the previous example
// Get ready to insert a row
rset2.moveToInsertRow();
// Put the values of the new row in each column
rset2.updateString("Name", "Orange");
rset2.updateInt("Amount", 7);
// Add this row
rset2.insertRow();
// Go back to the row we were at before inserting
rset2.moveToCurrentRow();
Dealing with Errors

- Things can go wrong with all of this
  - Incorrect SQL statements
  - DBMS might not be available
  - DBMS might not support some features
- If something goes wrong then an SQLException occurs

- If an exception is thrown:
  - We need to deal with it as best we can
  - Make sure any database objects are closed
  - If a connection is left open it can consume resources and might interfere with later use of the database
Exception Handling

// Declaration of any database objects
try {
    // Some database code
} catch (Exception e) {
    // Error reporting etc.
} finally {
    // Make sure all database objects are
    // closed and cleaned up
}
Closing Objects

- To make sure the object is closed
  - See if the object exists
  - If it does, call its close method
  - This might throw an exception itself, which needs to be caught
  - At some stage we have to stop handling the exceptions

```java
Connection conn;
try {
    ...
} finally {
    if (conn != null) {
        try {
            conn.close();
        } catch (...) {
            // what to do?
        }
    }
}
```
That’s it

• If you have revision questions, please contact me.