جامعة الأهرام الكندية
AHRAM CANADIAN UNIVERSITY

Lecture (07)
Database connectivity (II)

Dr. Ahmed ElShafee

Agenda

- Connecting DB

Connecting DB (cont,..)

The most basic kind of database access involves writing

1) updates- INSERT, UPDATE, or DELETE

2) queries – SELECT

- With these you know ahead of time the type of statements you are sendingto the database.
- generating implementation of java.sql.Statement tied to the database
- with the Statement object you can execute updates and queries
- The result of executing queries and update is java.sql.ResultSet
- ResultSet provides you with access to the data retrieved by a query.

Connecting DB (cont,..)

Statement class represents SQL statements.

- It has three generic forms of statement execution methods:

1) executeQuery(String query)

- Usage: for any SQL calls that expect to return data from database

2) executeUpdate(String query)

- Usage: when SQL calls are not expected to return data from database
- It returns the number of row affected by query

Connecting DB (cont,..)

3) execute()

- Usage: when you cannot determine whether SQL is an update or query
- return true if row is returned, use getResultset() to get the row otherwise returns false

Insert01.java

```
String url = "jdbc:mysql://localhost/testDB";
Connection conn = null;
String update= "insert into license (id, name, sex, date,licenseType) values
    (125,\"Sayed Mahmoud\", \"male\", \"17.03.2011\", \"Export\")";
try {
Class.forName ("com.mysql.jdbc.Driver").newInstance();
conn = DriverManager.getConnection (url, userName, password);
Statement stmt = conn.createStatement();
int update_count = stmt.executeUpdate(update);
System.out.print("\n"+update_count+" rows updated.");
conn.close();
} //end try
```

Connecting DB (cont,..)

PreparedStatement

- Prepared Statement is a precompiled SQL statement.
- It is more efficient than calling the same SQL statement over and over.
- The PreparedStatement class extends the Statement class by adding the capability of setting parameters inside of a statement.

Connecting DB (cont,..)

Ex:

```
String query= "update license set name=?, sex=? where id=125";
conn = DriverManager.getConnection (url, userName, password);
PreparedStatement  prest=conn.prepareStatement(query);
prest.setString(1,"Sara Mohamed");
prest.setString(2,"Female");
int update_count=prest.executeUpdate();
```

Update01.java

```
String url = "jdbc:mysql://localhost/testDB";
        Connection conn = null;
        String query= "update license set name=?, sex=? where id=125";
        PreparedStatement prest=null;
try
    {
    Class.forName ("com.mysql.jdbc.Driver").newInstance();
    conn = DriverManager.getConnection (url, userName, password);
    prest=conn.prepareStatement(query);
    prest.setString(1,"Sara Mohamed");
    prest.setString(2,"Female");
    int update_count=prest.executeUpdate();
    System.out.print("\n"+update_count+" rows updated.");
    conn.close();} //end try
```


Connecting DB (cont,..)

Transaction Management

- A transaction is a set of one or more statements that are executed together as a unit.
- Either all of the statements are executed, or none of the statements is executed.
- There are times when you do not want one statement to take effect unless another one also succeeds.
- This is achieved through the setAutoCommit() method of Connection object.
- The method takes a boolean value as a parameter.

Connecting DB (cont,..)

- When a connection is created, it is in auto-commit mode.
- Each individual SQL statement is treated as a transaction and will be automatically committed right after it is executed.
- The way to allow two or more statements to be grouped into a transaction is to disable auto-commit mode.

Example:

```
con.setAutoCommit(false);
```


Connecting DB (cont,..)

- Once auto-commit mode is disabled, no SQL statements will be committed until you call the method commit explicitly.
- This is achieved through the commit() method of connection objects.
- All statements executed after the previous call to the commit() method will be included in the current transaction and will be committed together as a unit.
- If you are trying to execute one or more statements in a transaction and get an SQLException , you should call the rollback() method to abort the transaction and start the transaction all over again.

Transaction01.java

```
String url = "jdbc:mysql://localhost/testDB";
Connection con = null;
try {
    Class.forName ("com.mysql.jdbc.Driver").newInstance();
    con = DriverManager.getConnection (url, userName, password);
    con.setAutoCommit(false);
    PreparedStatement updateName =null;
    String query = null;
    query="UPDATE license SET name = ? WHERE id = 125";
    updateName= con.prepareStatement(query);
    updateName.setString(1, "Zaki mohamed");
    updateName.executeUpdate();
```


Transaction01.java (cont,..)

```
    PreparedStatement updateSex = null;
     query = "UPDATE license SET sex =?";
    updateSex = con.prepareStatement(query);
    updateSex.setString(1, "Male");
    updateSex.executeUpdate();
    con.commit();
    con.setAutoCommit(true);
    con.close();
    } //end try
```

Connecting DB (cont,..)

ResultSet

- A ResultSet is one or more rows of data returned by a database query.
- The class simply provides a series of methods for retrieving columns from the results of a database query
- ResultSet class handles only a single row from the database at any given time.
- The class provides the next( ) method for making it reference the next row of a result set.
- If next() returns true, you have another row to process and any subsequent calls you make to the ResultSet object will be in reference to that next row.
- If there are no rows left, it returns false.

ResultSet01.java

```
String url = "jdbc:mysql://localhost/testDB";
Connection conn = null;
String query = "Select * FROM license";
try {
    Class.forName ("com.mysql.jdbc.Driver").newInstance();
    conn = DriverManager.getConnection (url, userName, password);
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    System.out.print("\nID\t\tName\t\t\tLicenseType");
    while (rs.next())
        {
        id=rs.getInt("id");
        if(rs.wasNull()) break;
```

ResultSet01.java

```
            System.out.print("\n"+id);
            name=rs.getString("name");
            if(rs.wasNull()) break;
            System.out.print("\t\t"+name);
            license=rs.getString("licenseType");
            if(rs.wasNull()) break;
            System.out.print("\t\t"+license);
            } //end while
    conn.close();
} //end try
```


Connecting DB (cont,..)

SQL Null Versus Java null

- SQL and Java have a serious mismatch in handling null values.
- Java ResultSet has no way of representing a SQL NULL value for any numeric SQL column.
- After retrieving a value from a ResultSet, it is therefore necessary to ask the ResultSet if the retrieved value represents a SQL NULL.
- To avoid running into database oddities, however, it is recommended that you always check for SQL NULL.
- Checking for SQL NULL involves a single call to the wasNull( ) method in your ResultSet after you retrieve a value.

Connecting DB (cont,..)

Scrollable ResultSet

- The single most visible addition to the JDBC API in its 2.0 specification is support for scrollable result sets.
- Using scrollable result sets starts with the way in which you create statements.
- The Connection class actually has two versions of createStatement()

1) the zero parameter version

Example:

```
Statement stm = con.createStatement();
```


Connecting DB (cont,..)

2) a two parameter version that supports the creation of Statement instances that generate scrollable ResultSet objects.

```
createStatement(int rsType,int rSConcurrency)
```

Parameters:

rsType - a result set type; one of
ResultSet.TYPE_FORWARD_ONLY,

- ResultSet.TYPE_SCROLL_INSENSITIVE, or
- ResultSet.TYPE_SCROLL_SENSITIVE

rsConcurrency - a concurrency type; one of

- ResultSet.CONCUR_READ_ONLY or
- ResultSet.CONCUR_UPDATABLE

Connecting DB (cont,..)

JDBC defines three types of result sets:

1) TYPE_FORWARD_ONLY
2) TYPE_SCROLL_SENSITIVE
3) TYPE_SCROLL_INSENSITIVE

Out of these three TYPE_FORWARD_ONLY is the only type that is not scrollable.

The other two types are distinguished by how they reflect changes made to them.

TYPE_SCROLL_INSENSITIVE ResultSet is unaware of in-place edits made to modifiable instances.

TYPE_SCROLL_SENSITIVE, on the other hand, means that you can see changes made to the results if you scroll back to the modified row at a later time.

Connecting DB (cont,..)

Navigating ResultSet

- When ResultSet is first created, it is considered to be positioned before the first row.
- Positioning methods such as next() point a ResultSet to actual rows.
- Your first call to next( ), for example, positions the cursor on the first row.
- Subsequent calls to next() move the ResultSet ahead one row at a time.
- With a scrollable ResultSet, however, a call to next() is not the only way to position a result set.

Connecting DB (cont,..)

- The method previous( ) works in an almost identical fashion to next( ).
- While next() moves one row forward, previous() moves one row backward.
- If it moves back beyond the first row, it returns false. Otherwise, it returns true.
- Because a ResultSet is initially positioned before the first row, you need to move the ResultSet using some other method before you can call previous().

ResultSet02.java

```
String url = "jdbc:mysql://localhost/testDB";
Connection conn = null;
String query = "Select * FROM license";
try
    {
    Class.forName ("com.mysql.jdbc.Driver").newInstance();
    conn = DriverManager.getConnection (url, userName, password);
    Statement stmt =
    conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CON
    CUR_READ_ONLY);
    ResultSet rs = stmt.executeQuery(query);
    System.out.println("Got results:");
    rs.afterLast( );
    System.out.print("\nID\t\tName");
```

Connecting DB (cont,..)

```
while(rs.previous( ))
        {
        int a;
        String str;
        a = rs.getInt("id");
        a = rs.wasNull( ) ? -1 : a;
        str = rs.getString("name");
        str = rs.wasNull( ) ? null : str;
        System.out.print("\n" + a);
        System.out.println("\t\t"+str);
        }
System.out.println("Done.");
}
```


Connecting DB (cont,..)

Other Navigation Methods

- JDBC 2.0 provides new methods to navigate around rows in result sets:
  1) beforeFirst( )
  2) first( )
  3) last( )
  4) isBeforeFirst( )
  5) isFirst( )
  6) isLast( )
  7) isAfterLast()
  8) getRow( )
  9) relative( )
  10) absolute( )

Connecting DB (cont,..)

- Except for absolute() and relative(), the names of the methods say exactly what they do.
- Each take integer arguments.

Example:

- A call to absolute(5) moves the ResultSet to row 5 unless there are four or fewer rows in the ResultSet.
- A call to absolute() with a row number beyond the last row is therefore identical to a call to afterLast()

Connecting DB (cont,..)

- You can also pass negative numbers to absolute().
- A negative number specifies absolute navigation backwards from the last row

Example:

- absolute(1) is identical to first(), absolute(-1) is identical to last()
- Similarly, absolute(-3) is the third to last row in the ResultSet. If there are fewer than three rows in the ResultSet.

Connecting DB (cont,..)

- The relative() method handles relative navigation through a ResultSet.
- In other words, it tells the ResultSet how many rows to move forward or backward.

Example:

- A value of 1 behaves just like next() and a value of -1 just like previous().

Delete01.java

```
String url = "jdbc:mysql://localhost/testDB";
Connection conn = null;
String sql= "delete from license where id=125;";
Try {
Class.forName ("com.mysql.jdbc.Driver").newInstance();
conn = DriverManager.getConnection (url, userName, password);
Statement stmt = conn.createStatement();
int col=stmt.executeUpdate(sql);
System.out.print("\nnumber of updates="+col);
conn.rollback();
conn.close();
    } //end try
```

Select02.java

```
String url = "jdbc:mysql://localhost/testDB";
Connection conn = null;
String query = "Select * FROM license";
try {
    Class.forName ("com.mysql.jdbc.Driver").newInstance();
    conn = DriverManager.getConnection (url, userName, password);
    Statement stmt = conn.createStatement();
    if (stmt.execute(query)==true){
        ResultSet rs=stmt.getResultSet();
```


Select02.java (cont,..)

```
        while (rs.next()) {
                dbtime = rs.getString(1);
                System.out.print("\n"+dbtime);
                dbtime = rs.getString(2);
                System.out.print("\t"+dbtime);
                dbtime = rs.getString(3);
                System.out.print("\t"+dbtime);
                dbtime = rs.getString(4);
                System.out.print("\t"+dbtime);
                dbtime = rs.getString(5);
                System.out.print("\t"+dbtime);} //end while
                } conn.close();} //end try
```

Connecting DB (cont,..)

CallableStatement

- CallableStatement allows non-SQL statements (such as stored procedures) to be executed against the database.
- CallableStatement class extends the PreparedStatement class, which provides the methods for setting IN parameters.
- Methods for retrieving multiple results with a stored Procedure are supported with the Statement.getMoreResults() method.

StoredProcedure01.java

```
String dbrs;
String url = "jdbc:mysql://localhost/testDB";
Connection conn = null;
try {
    Class.forName ("com.mysql.jdbc.Driver").newInstance();
    conn = DriverManager.getConnection (url, userName, password);
    CallableStatement cs = conn.prepareCall("{call number_of_license}");
    ResultSet rs = cs.executeQuery();
    while (rs.next())
            {
        dbrs = rs.getString(1);
          System.out.print("\n"+dbrs);
          } //end while
conn.close();} //end try
```

Connecting DB (cont,..)

CleanUP

- The Connection, Statement, and ResultSet classes all have close( ).
- It is always a good idea to close any instance of these objects when you are done with them.
- It is useful to remember that closing a Connection implicitly closes all
- Statement instances associated with the Connection.
- Similarly, closing a Statement implicitly closes ResultSet instances associated with it.

Thanks,

See you next Week, isA