Chapter Table of Contents

Chapter 5.2

Setting up a Java Database Connectivity

Aim

To provide students with basics concepts and set up procedures for Java Database Connectivity

Instructional Objectives

After completing this chapter, you should be able to:

- Demonstrate JDBC setup
- Illustrate the ways to handle SQL exceptions
- Describe the ways to access result sets
- Demonstrate an example JDBC application

Learning Outcomes

At the end of this chapter, you are expected to:

- Describe the ways to create and execute JDBC statements
- Describe exception handling in JDBC
- Identify different types of result sets
- Justify whether a statement and its resultSet will be closed on a commit or rollback
- Connect excel spreadsheet using JDBC in Java

5.2.1 Setting up JDBC

(i) Loading a Driver

Before you can connect to a database in Java, you need to load the driver. There are two methods possible and it depends on the code used. The methods are as follows:

1. First method

To access a database in Java application, the driver needs to be installed initially with the program. It can be done using static registerDriver( ) method of java.sql.Drivermanager, this class manages JDBC drivers for basic services. In this method, it takes the "driver" class as input, *i.e.,* a class that used to implement java.sql.Driver interface (in this case: OracleDriver)

Note: forName( ) can be used alternatively for java.lang.class to load JDBC drivers directly. ***For example,*** Class.forName ("oracle.jdbc.OracleDriver");

2. Second method

This process transfers the driver as a parameter to the Java Virtual Machine (JVM) as it begins, practising - D argument.

For example, java-Djdbc.drivers=org.postgresql.Driverexample.PictureViewer

In this example, the JVM will try to load the driver as part of its beginning. Once done, the PictureViewer will start to load.

Now, this method is the better one to use because it allows code to be used by different type of database packages without recompiling the program code.

One last thing in loading driver is when program code then tries to open a Connection and the developer gets a No driver ready SQLException being thrown by it; this is reasonably effected by the driver not moving in the classpath, or the value of the parameter not being correct.

(ii) Connecting to a Database

To get started and for purity's sake, we'll use a terminal or console window to find out the results/output from a database.

a) To start a new project:

File>New Project from the NetBeans menu onto the window

Create a Java Application.Call the project database_console and the Main class

b) When you click on Finish button onto the window, your code must look like this:

```
Package database_console
public class DBConnect
{
   public static void main(String [] args)
}
```

Connecting to the Database: Developer needs a connection object to connect to a database. This connection object uses a Driver Manager. This Driver Manager passes in developer's database username and the location of the database.

c) Developer must use these three critical statements to the top of the code:

Import java.sql.Connection;

Import java.sql.DriveManager;

Import java.sql.SqlException;

To setup up a link with a database, the code will be:

Connection conn=DriverManager.getConnection(host,username,password);

So the DriverManager has a method named getConnection. It needs a hostname, username, password. If a connection is successful, a connection object will be create called con.

The address of the highlighted database overhead is:

JDBC:derby://localhost:1234/Employee

Jdbc:derby://localhost is the database type and server that developer is using. The 1234 is the port number and Employees is a Database name. This can all go in String Variable:

String host ="Jdbc:derby://localhost:1234/Employee";

d) You can add two or more strings for the username and the password:

String urName="My_Username_Here";

String urPass="My_Password_Here";

After adding these three connection string your code will look like this:

```
packagedatabase_console
Import java.sql.Connection;
Import java.sql.DriveManager;
Import java.sql.SqlException;
public class DBConnect
{
   public static void main(String [] args)
   {
      String host ="Jdbc:derby://localhost:1234/Employee";
      String urName="My_Username_Here";
      String urPass="My_Password_Here";
      Connection
      con=DriverManager.getConnection(host,username,password);
   }
}
  //But sometimes a SqlException Error may occur in "Connection
con=DriverManager.getConnection(host,username,password)".//
```

The developer needs to write code to deal with this potential error.

In the code below, just tapping the error in catch part of the try catch statement.

```
Try{}catch(SqlExceptioner)
{
   System.out.println(er.getMessage());
}
Add this above try..catch block code with the main code, then your
code will be look like this:
packagedatabase_console
Import java.sql.Connection;
Import java.sql.DriveManager;
Import java.sql.SqlException;
public class DBConnect
{
   public static void main(String [] args)
   {
      try
      {
         String host ="Jdbc:derby://localhost:1234/Employee";
         String urName="My_Username_Here";
         String urPass="My_Password_Here";
         Connection
         con=DriverManager.getConnection(host,username,password);
      }
      catch(SqlExceptioner)
      {
```

```
            System.out.println(er.getMessage());
        }
    }
}
```

Now connect your database with Database Server: NetBeans>Services>Databases>Start Server

Once your server started, run the program. If developer found error related with driver then use these steps:

When to click on Add JAR, the file developer is looking for "derbyclient.jar". Click on it to open and the file will be added to your project library.

Now that you have a Client driver added to your project run your program again. You should now be error free.

(iii) Creating and Executing JDBC Statements

In generally, to process any SQL statement with JDBC, just follow these steps:

STEP 1: Establishing a connection

STEP 2: Create a statement

STEP 3: Execute the query

STEP 4: Process the ResultSet object

STEP 5: Close the connection

Creating Statements

A Statement is an interface that represents an SQL statement. Developer executes declarations of objects and they create ResultSet objects, which is a table of data designing a database ResultSet. A Developer needs a Connection object to create a Statement object.

For example,

HousesTables.viewTable generates a Statement object with the following code:

stmt=con.createStatement( );

There are three types of statements in creating reports:

1. **Statement:** Used to execute simplistic SQL statements with no parameters in the code.

2. **PreparedStatement:** Used for re-compiling SQL statements that might hold input parameters. See Using Prepared Statements for further knowledge or information. It is also called Extends Statement.

   For example, String sql = "select * from people where id=?";

3. **PreparedStatement preparedStatment = connection.preparedStatment(sql); CallableStatement:** (Extends PreparedStatement.) Used to perform stored methods that may contain both input/output parameters.

   create or replace procedure "INSERTR"

   (id IN NUMBER, name IN VARCHAR)

   is

   begin

   insert into user420 values(id,name);

   end;

Executing Queries

To perform a query, call an execute method from Statement such as the following statements:

- **Execute:** Returns true if the initial object that the question declares is a ResultSet object. Use a execute( ) method if the query could return one or more ResultSet objects. Recover the ResultSet objects delivered from the query by frequently calling Statement.getResultSet in code.

- **executeQuery:** Returns one ResultSet object at least.

- **executeUpdate:** Returns an integer describing the number of rows affected by the SQL statement. Use this type of method if you are using INSERT, DELETE, or UPDATESQL declarations in code.

  For example, HousesTables.viewTable executed a Statement object with the following code:

  ResultSetrs = stmt.executeQuery(query);

Self-assessment Questions

1) Which of the following is/are true about DriverManager JDBC's class?
   a) DriverManager of JDBC is a class that maintains a list of database drivers
   b) It matches connection requests from the Java application with the proper database driver using communication subprotocol
   c) Both of the above
   d) It matches a connection request from database driver using command protocol

2) Identify the following type of JDBC driver to be used when a type 3 or type 4 driver is not available yet for your database?
   a) Type 1 b) Type 2
   c) Type 3 d) Type 4

3) The JDBC-ODBC Bridge support more than one concurrent open statements per connection.
   a) True b) False

4) Which of the following is used to call stored procedures in the database?
   a) Statement b) PreparedStatement
   c) CallableStatement d) None of the above

5) How does JDBC examine the data types of Java and database?
   a) The JDBC driver converts the Java data type to the appropriate JDBC type before sending it to the database
   b) It uses a fault mapping for most data types
   c) Both of the above
   d) None of the above

6) Out of byte[] or a java.sql.Blob, which has the best performance when used to manipulate data from the database?
   a) byte[] b) java.sql.Blob

7) A Java program cannot directly communicate with an ODBC driver because JDBC
   a) ODBC wrote in C language b) ODBC wrote in C# language
   c) ODBC wrote in C++ language d) ODBC wrote in Basic language

8) Which of the following driver uses JDBC-ODBC Bridge driver turned the JDBC API over ODBC API?

a) JDBC drivers in Java
b) ODBC drivers in Java
c) a and b both
d) None a and b

9) The_____ package holds classes that help in joining/connecting to a database, sending SQL reports to the database and processing the query results.

a) connection.sql package
b) db.sql package
c) pkg.sql package
d) java.sql package

10) The_____ method carries out a simple query and delivers a single ResultSet object.

a) executeUpdate( )
b) executeQuery( )
c) execute( )
d) noexecute( )

11) The connection object can be initialised using the_____ method of the Driver Manager class.

a) putConnection( )
b) setConnection( )
c) Connection( )
d) getConnetion( )

12) Which of the following is a Metadata interface of JDBC?

a) DatabaseMetaData
b) ResultSetMetaData
c) Both of the above
d) None of the above

5.2.2 Handling SQL Exceptions

When JDBC encounters a failure during communication with a source of Data, it throws an instance of SQL Exception as differentiating it from the normal Exception.

The SQL Exception case contains the resulting information that can help you to find out the cause of the mistake:

- Retrieves the String object that contains this description by calling the method SQLException.getMessage.

  For Example:

  ```
  public Batch add(String sqlLines) {
      notNull("SQL commands", sqlLines);
      try {
          statement.addBatch(sqlLines);
      } catch (SQLException e) {
          JdbcUtils.closeStatement(statement);
          throw new RuntimeException(e.getMessage(), e);}
  return this;}
  ```

  This code gets the JDBC driver's error message for an error, handled by the driver or gets the Oracle error number and message for a database error.

- An SQLState code. Code including their respective meanings has been standardised by ISO/ANSI and Open Group (X/Open), although some codes have been a reserve for database hucksters to define for themselves. This String object consists of five alphanumeric figures. Recover this code by calling the method SQLException.getSQLState.

  For Example:

  ```
  public String getSQLState()
  Retrieves the SQLState for this SQLException object.

  Returns: It will return the SQLState value.
  ```

- An error code. An integer value was classifying the error that caused the SQLException instance to be thrown by an integer. It is worth and application are implementation-specific and might be the exact error code returned by the underlying source of data. Recover the error by calling the method SQLException.getErrorCode.

  For Example:

  ```
  public int getErrorCode()
  ```

```
Retrieves the vendor-specific exception code for
this SQLException object.

Returns: This public method will return the vendor's error code
```

- Cause. An SQL Exception instance might have a causal connection, which consists of one or more Throwable objects that caused the SQLException instance to be thrown by objects in this section. To operate this chain of circumstances, recursively call the method SQLException.getCause until the query returns a null value.

  For Example:

```
try {...
} catch (SQLException sqle) {
  for(Throwable t : sqle) {
    System.out.println("Throwable: " + t);
    Throwable cause = t.getCause();
    while (cause != null) {
      System.out.println("Cause: " + cause);
      cause = cause.getCause();
    } }}
```

  This code allows to check for cause and not just loop through all the exception

- A source to any chained exceptions. If more than one error happens, the exceptions are mentioned through this chain. Retrieve these exceptions by calling the method SQLException.getNextException on the exception that threw away.

  For Example,

```
public SQLException getNextException()

Retrieves the exception chained to this SQLException object by
setNextException(SQLException ex).

Returns: the next SQLException object in the chain; null if there are
none
```

Retrieving Exceptions:

The following method - JDBCUtilities.printSQLExecution outputs the SQLState,error code, error description and cause contained in the SQLException as well as any other error chained to it :

```
public static void printSQLException(SQLException ex)
{
  for (Throwable e: exx)
```

```
  {
    if (e instanceofSQLException)
    {
      if (ignoreSQLException(((SQLException)e).getSQLState()) == false)
      {
        e.printStackTrace(System.er);
        System.er.println("SQLState: " +((SQLException)e).getSQLState());
        System.er.println("Error Code: " +((SQLException)e).getErrorCode());
        System.er.println("Message is: " + e.getMessage());
        Throwable p = exx.getCause();
        while(p != null)
        {
          System.out.println("Cause: " + p);
          p = p.getCause();
        }
      }
    }
  }
}
```

For example, if you call the method HousesTable.dropTables with java DB as your DBMS. The table Houses does not exist and you remove the call to JDBCUtilities.printSQLExecution.ignogeSQLException, the output will be similar to the following:

SQLState: 42Y55

ErroeCode: 3000

Message 'DROP TABLE' cannot be performed on

'TESTDB'.HOUSES' because it does not exist.

Self-assessment Questions

13) Find the following blocks are used for error handling in SQL Server?

a) try……catch
b) try……final
c) try……end
d) catch….try

14) Which of the following records can be checked for Errors?

a) CREATE
b) DROP
c) DELETE
d) INSERT

15) Exception handling is probably in SQL Server using _____.

a) THROW
b) FINAL
c) FINALLY
d) Both A and B

16) Which of the following is an Error function used within CATCH block?

a) ERROR_STATE( )
b) ERROR_STATUS( )
c) ERROR_MSG( )
d) ERROR_ERR( )

17) ERROR_SEVERITY( ) method returns the _____level of the error.

a) State number
b) Full text
c) Severity
d) Half text

18) Which of the following is global variable for error handling?

a) @@ERRORS
b) @@ERROR
c) @@ERR
d) @ERRORS

5.2.3 Accessing ResultSets

The SQL statements that are read from a database query, returns its data in the form of result set. The standard way to select rows from a database, return the data in a ResultSet .The SELECT statement does standard way to select the rows from the database and view them in ResultSet.

Type of ResultSets:

The responsiveness of a ResultSet object is defined by one of the three different ResultSet types:

1. TYPE_FORWARD_ONLY

The sequence set cannot be scrolled; its cursor moves ahead only, from before the first row to after the last row. The rows comprised of the ResultSet depend on whereby the underlying database creates the results.

That is, it holds the rows that provide the query at unless the time

The query is executed or as the rows are retrieved.

2. TYPE_SCROLL_INSENSITIVE

The result can be writing; its cursor can move both forward and backwards

About the contemporary position and it can move to an arbitrary position.

The ResultSet is indifferent to changes made to the underlying data origin while it is open. It holds the rows that provide the query at either the time the query is executed or as the rows are retrieved.

3. TYPE_SCROLL_SENSITIVE:

The result can be documented; its cursor can move both forward and backwards

About the current state and it can move to an arbitrary position.

The ResultSet displays changes made to the underlying data source while the result set remains open.

Note: The default ResultSet type is TYPE_FORWARD_ONLY.

For example,

```
try
{
   Statement stmt = conn.createStatment(
   ResultSet.TYPE_FORWARD_ONLY,
   ResultSet.CONCUR_READ_ONLY);
}
catch(Exception ex)
{
         …..
}
finally
{
       …..
}
```

Methods of ResultSet interface:

The method of ResultSets can be divided into three sections:

1. **Navigational Methods:** This method is used to move the cursor around.
2. **Get methods:** Used to view the data in the columns in the contemporary row being designated.
3. **Update methods:** Columns of the current row are updated with data in this method. The updates can then be updated in the underlying database as well.

Self-assessment Questions

19) Which one of the following is/are default ResultSet type?

a) TYPE_SCROLL_INSENSITIVE b) TYPE_FORWARD_ONLY

c) TYPE_SCROLL_SENSITIVE d) All of above

20) Accessing Column Values- String, int, long, double, BigDecimal, etc. all take the name of the column to obtain the column value for, as a parameter.

a) TRUE b) FALSE

21) The type of a ResultSet object fixes the constant of its functionality in two areas: the ways in which the cursor can be manipulated and how the ResultSet object returns simultaneous changes made to the underlying data source.

a) TRUE b) FALSE

22) Methods of the ResultSet interface can be broken down into _____ categories.

a) Four b) Two

c) Six d) Three

5.2.4 An Example JDBC Application to Query a Database

This topic gives an example on how to fetch documents from a database table using JDBC application. Before performing the following example, make sure you have the following in place:

To implement the following example, you can replace the username and password with user's actual username, password. MySQL or any other preferred database developer is using.

Required Steps are:

The following steps are required to create a new Database using JDBC application:

a) **Import the packages:** Requires that developer inserts the packets comprising the classes of JDBC needed for the database code.

b) **Register the JDBC driver:** You initialises a driver so you can open a communications channel with the database.

c) **Connection free:** Requires applying the DriverManager.getConnection( ) system to generate a Connection object, which represents a real connection with a database server.

d) **Execute a query:** An object of type Statement for construction and submitting an SQL statement to select/fetch records from a table.

e) **Extract Data from Table:** Once SQL query is executed, you can fetch records from the table.

f) **Clean up the conditions/environment:** Requires explicitly closing all database resources versus relying on the JVM's garbage collection. The following example describes the JDBC application

```
//STEP 1. Import required packages
import java.sql.*;
public class JDBCExample
{
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/";
   //  Database credentials
```

```
static final String USER = "username";
static final String PASS = "password";
public static void main(String[] args)
{
   Connection conn = null;
   Statement stmt = null;
   try
   {
      //STEP 2: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");
      //STEP 3: Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);
      //STEP 4: Execute a query
      System.out.println("Creating database...");
      stmt = conn.createStatement();
      String sql = "CREATE DATABASE STUDENTS";
      stmt.executeUpdate(sql);
      System.out.println("Database created successfully...");
   }
   catch(SQLException se)
   {
      //Handle errors for JDBC
      se.printStackTrace();
   }
   catch(Exception e)
   {
      //Handle errors for Class.forName
      e.printStackTrace();
   }
   Finally
   {
      //finally block used to close resources
      try
      {
         if(stmt!=null)
         stmt.close();
      }
      catch(SQLException se2)
      {}// nothing we can do
      try
      {
         if(conn!=null)
         conn.close();
      }
      catch(SQLException se)
      {
         se.printStackTrace();
      }
      //end finally try
   }
```

```
      //end try
      System.out.println("Goodbye!");
   }
   //end main
}
//end JDBCExample
```

Self-assessment Question

23) Viewing a Result Set is a get method for each of the possible data types and each get method has two versions –

- One that takes in a column name.
- One that takes in a column index.

a) True b) False

Summary

- A Java virtual machine, which provides the fundamental basis for platform independence.
- The database provides support for generating and storing, applications of Java. Database introduces the Java language to Oracle PL/SQL users, who are accustomed to developing server-side applications that are integrated with SQL data. The developer can develop server-side Java applications that take resources of the scalability and performance of Oracle Database.
- It's a functionality that JDBC provides more than on implementations for existing and be used by the same application. The Application Programming Interface (API) presents a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager.
- The Driver Manager is using as a connection factory for creating JDBC connections.
- Database Connectivity is an API for the programming language Java, which defines how a customer/client may access a database.
- When a Java application requires a database connection, DriverManager. getConnection( ) processes is used to generate a JDBC connection. The Link used is dependent upon the appropriate database and JDBC driver. It will always begin with the "JDBC:" protocol, but the rest is up to the appropriate vendor.

Terminal Questions

1. Demonstrate JDBC setup
2. Illustrate the ways to handle SQL exceptions
3. Describe the ways to access result sets
4. Demonstrate an example JDBC application

Answer Keys

Self-assessment Questions	
Question No.	Answer
1	c
2	b
3	b
4	c
5	c
6	b
7	a
8	b
9	d
10	b
11	d
12	c
13	a
14	d
15	a
16	a
17	c
18	b
19	b
20	a
21	a
22	d
23	a

Activity

Activity Type: Offline

Duration: 60 Minutes

Description:

Create a login form in java and connect it with database using JDBC.

Bibliography

e-References

- (2016). Retrieved 13 June 2016, from • http://www.tutorialspoint.com/jdbc/pdf/jdbc-environment-setup.pdf
- (2016). Retrieved 13 June 2016, from http://www.cbcb.umd.edu/confcour/Spring2011/CMSC424/Simple%20tutorial%20for%20using%20JDBC.pdf

External Resources

- Database Programming with JDBC & Java, 2nd Edition by George Reese, Publisher: O'Reilly Media

Video Links

Topic	Link
Connecting to a Database	http://www.javatpoint.com/steps-to-connect-to-the-database-in-java
Result Set	http://tutorials.jenkov.com/jdbc/resultset.html
Handling SQL	https://www.youtube.com/watch?v=bfLT2ZSPJa8
Connections JDBC	http://www.tutorialspoint.com/jdbc/jdbc-db-connections.html

Notes: