Lab 5. JDBC introduction

This lab introduces the basics of JDBC and how it can be used to connect Java programme with an existing database.

JDBC stands for **Java Database Connectivity**, which is a Java API to connect with the database and execute queries. The goal of JDBC is to allow connecting an application program to almost any database system. With JDBC, we can handle a database using Java programme and perform the following activities:

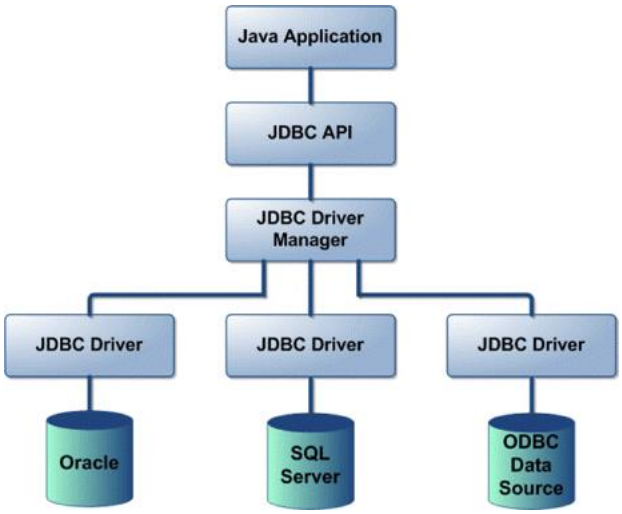
1. Connect to the database.
2. Execute queries and update statements to the database.
3. Retrieve the result received from the database.

Applications using JDBC are DBMS independent on both source code and executable code level (providing that the SQL standard is satisfied).

JDBC Architecture

A typical JDBC architecture comprises of 4 elements:

1. An application (programme): initiates a connection with a database, acquires locks, submits SQL statements, receives data, processes data, processes error messages, and disconnects from the database to terminate a session. Also, it can set transaction boundaries and decide whether to commit or roll-back a transaction.
2. A driver manager: loads the drivers needed, passes JDBC function calls from the application to the correct driver, handles JDBC initialization from the application, and performs some rudimentary error checking.
3. DBMS specific drivers: establish connection with a database, submit data manipulation requests, accept returning results, translate DBMS specific data types into Java data types, and translate error messages.
4. Databases: process data manipulation commands and return results.



Connect to MySQL Database with Java

1. Preparation work:

- Make sure you have MySQL database installed – http://dev.mysql.com/downloads/
- Download MySQL JDBC Driver – http://dev.mysql.com/downloads/connector/j/. For Windows system, choose "Platform Independent". For instructions on installation of Connector/J, please see https://dev.mysql.com/doc/connector-j/en/connector-j-binary-installation.html.
- Create a Java project in Eclipse.

2. Connect with MySQL:

When you are using JDBC outside of an application server, the *DriverManager* class manages the establishment of connections using `Class.forName("com.mysql.cj.jdbc.Driver").`

After the driver has been registered with the DriverManager, you can obtain a Connection instance that is connected to a particular database by calling `DriverManager.getConnection().`

Code example:

Note: Make sure you have connector (Connector/J) installed correctly and the CLASSPATH set up.

(If you have problems setting the CLASSPATH, the easiest way is to put the "mysql-connector-j-8.4.0.jar" file in the same directory as your java file, and specify the classpath in the -cp argument when you run java, e.g. `java -cp .;mysql-connector-j-8.4.0.jar JavaFileName`)

```
package jdbcdemo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class LoadDriver {

	public static void main(String[] args) {

		Connection conn = null;

		try {

			Class.forName("com.mysql.cj.jdbc.Driver");
			System.out.println("Driver loaded");

			conn =
		DriverManager.getConnection("jdbc:mysql://localhost:3306/DreamH
		ome", "root", "password");
			System.out.println("Connection established!");

		}
		catch (ClassNotFoundException e) {
			System.out.println("Exception:" + e.getMessage());
		}
		catch (SQLException e) {
```

```
                System.out.println("SQLException:" + e.getMessage());
            }
        }

}
```

Once a Connection is established, it can be used to create `Statement` and `PreparedStatement` objects, as well as retrieve metadata about the database.

3. A general development process (Make queries):

1. Get connection.
2. Submit SQL query.
3. Process result set.

Let's take a look at a code snippet example of reading all names from Staff relation in the DreamHome database.

1. Get a connection to database.

A session with a database is started by the creation of a Connection object. The format is `DriverManager.getConnection(url, userId, password).`

```
Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost://3306/database_na
me","userId","password");
```

2. Submit SQL query:

- Create a SQL statement:

```
Statement myStmt = conn.createStatement();
```

- Execute SQL query.

```
ResultSet myRs = myStmt.executeQuery("select * from Staff");
```

If the SQL statement is one of CREATE, INSERT, DELETE, UPDATE, or SET type, apply `executeUpdate()` method on a `Statement` object.

Else (the SQL statement is of the SELECT type), create a `ResultSet` object and feed into the `ResultSet` object the return value by applying `executeQuery();`

3. Process the result set.

```
While (myRs.next()) {
      System.out.println(myRs.getString("fName") + "," +
      myRs.getString("lName"));
}
```

Full code example:

```
package jdbcdemo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;

public class LoadDriver {

	public static void main(String[] args) {

		try {
			// 1. Get a connection to database
			Connection conn =
			  DriverManager.getConnection
			  ("jdbc:mysql://localhost:3306/DreamHome", "root",
			   "password");
			// 2. Create a statement
			Statement  myStmt = conn.createStatement();
			// 3. Execute SQL query
			ResultSet myRs = myStmt.executeQuery("select * from
			Staff");
			// 4. Process the result set
			while (myRs.next()) {
			   System.out.println(myRs.getString("fName")
			        +","+myRs.getString("lName"));
			}
		}
		catch (SQLException e){
			 System.out.println("SQLException: " + e.getMessage());
		}
	}
}
```

Notes:

- The `executeQuery()` method returns an object of the type set.
- This set object should be assigned to an object of the `ResultSet` class.
- The `ResultSet` class has the `next()` method that allows traversing the set in a tuple at a time fashion.
- Initially, the `ResultSet` object is positioned before the first tuple of the result.
- The method `next()` returns true if there is a next tuple in the result, otherwise false.
- After executing `next()`, the `ResultSet` object contains the pointer to the current tuple.

4. Controlling transaction behaviour:

DBMSs use transactions to maintain data in a consistent state (data consistency) while allowing more than one user to access the database at the same time (data concurrency). A transaction is a set of one or more SQL statements that make up a logical unit of work. A transaction ends with either a commit or a rollback, depending on whether there are any problems with data consistency or data concurrency.

- Start:

By default, a Connection automatically commits changes after executing each SQL statement. The method: `public abstract void setAutoCommit( boolean autoCommit) throws SQLException` is applied on a `Connection` object to designate the start of a transaction (BEGIN point) by assigning a value false to `autoCommit`.

```
myConn.setAutoCommit(false);
```

- End:

A transaction is terminated using either: `public abstract void commit() throws SQLException` or `public abstract void rollback() throws SQLException`. And (after any of them) `myConn.setAutoCommit(true)` on the Connection object.

Closing a connection:

Before exiting from an application program all connections acquired should be closed by applying `public abstract void close() throws SQLException` method on each of them.

Exceptions:

Most of the methods in `java.sql` can throw an exception of the type `SQLException` if an error occurs. In addition to inherited `getMessage()` method, `SQLException` class has two additional methods for providing error information:

- `public String getSQLState()` that returns an SQL state identifier according to SQL:1999 standard,
- `public int getErrorCode()` that retrieves a vendor specific error code.

Each JDBC method that throws an exception has to be placed inside a `try` block followed by a `catch` block.

Example:

```
try{
     /* Code that could generate an exception
     goes here. If an exception is generated, the catch
     block below will print out information about it*/
}
catch (SQLException ex){
     System.out.println(ex.getMessage());
     System.out.println(ex.getSQLState());
     System.out.println(ex.getErrorCode());
}
```

Exercises:

1. Write your own Java programme to connect with MySQL database DreamHome.
2. Write a Java programme to retrieve all staff details and print out the results.
3. Write a Java programme to increase all managers' salary by 10% and print out the new salaries.

4. Write a Java programme to delete the branch B003 tuple in *Branch* relation.
5. Use try and catch statements to catch an exception and print out the error code. For example, trying to write a SQL statement with incorrect keywords.