UNIT : 8

Reference :-Marty Hall, Larry Brown, "Core Servlets and JavaServer Pages Volume – 1", Pearson Education, 2nd ed.(2004)
Chapter :- 17 : Accessing Database with JDBC

What is JDBC?

- JDBC provides a standard library for accessing relational databases.
- By using the JDBC API, you can access a wide variety of SQL databases with exactly the same Java syntax.
- Officially, JDBC is not an acronym and thus does not stand for anything. Unofficially, "Java DataBase Connectivity" is commonly used as the long form of the name.

Using JDBC

- Seven standard steps for querying databases

1. Load the driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement object
5. Execute a query or update
6. Process the results
7. Close the connection

JDBC steps in brief

- **Step 1 → Load Driver**
  Specify the classname of the database driver in the Class.forName method.
  By doing so, it automatically creates a driver instance and register it with the JDBC driver manager.
- **Step 2 → Define the Connection URL**
  In JDBC, a connection URL specifies the server host, port, and database name with which to establish a connection.

JDBC steps in brief

- **Step 3 → Establish the connection**
  With the connection URL, username, and password, a network connection to the database can be established.
  Once the connection is established, database queries can be performed until the connection is closed.
- **Step 4 → Create a statement object**
  Creating a Statement object enables you to send queries and commands to the database.

JDBC steps in brief

- **Step 5 → Execute query or update**
  Given a Statement object, you can send SQL statements to the database by using the execute, executeQuery, executeUpdate, or executeBatch methods.
- **Step 6 → Process the results.**
  When a database query is executed, a ResultSet is returned. The ResultSet represents a set of rows and columns that you can process by calls to next and various getXxx methods.
- **Step 7 → Close the connection.**
  When you are finished performing queries and processing results, you should close the connection, releasing resources to the database.

Step 1 → Load JDBC Driver

- The driver is the piece of software that knows how to talk to the actual database server.
- To load the driver, you just load the appropriate class; a static block in the driver class itself automatically makes a driver instance and registers it with the JDBC driver manager.
- how do you load a class without making an instance of it?
- how can you refer to a class whose name isn't known when the code is compiled?
- The answer to both questions is to use Class.forName

Step 1 → Load JDBC Driver

- Class.forName :

  This method takes a string representing a fully qualified classname (i.e., one that includes package names) and loads the corresponding class.

  This call could throw a ClassNotFoundException, so it should be inside a try/catch block as shown below.

```
try {
Class.forName("connect.microsoft.MicrosoftDriver");
Class.forName("oracle.jdbc.driver.OracleDriver");
Class.forName("com.sybase.jdbc.SybDriver");
}          catch(ClassNotFoundException          cnfe)          {
System.err.println("Error loading driver: " + cnfe); }
```

Step 1 → Load JDBC Driver

- The JDBC driver (which is on the client) translates calls written in the Java programming language into the native format required by the server, database server doesn't require any changes.
- So obtain a JDBC driver specific to the database you are using and that you will need to check the vendor's documentation for the fully qualified class name to use.
- use Class.forName for any class in your CLASSPATH.
- Most JDBC driver vendors distribute their drivers inside JAR files. So, during development be sure to include the path to the driver JAR file in your CLASSPATH setting

Step 1 → Load JDBC Driver

- For deployment on a Web server, put the JAR file in the WEB-INF/lib directory of your Web application.
- if multiple Web applications are using the same database drivers, the administrator will place the JAR file in a common directory used by the server. For example, in Apache Tomcat, JAR files common to multiple applications can be placed in 1 install_dir/common/lib.
- Two common JDBC driver implementations :

Step 1 → Load JDBC Driver

1) JDBC-ODBC bridge,

A driver that uses the JDBC-ODBC bridge approach is known as a Type I driver.

Since many databases support Open DataBase Connectivity (ODBC) access, the JDK includes a JDBC-ODBC bridge to connect to databases

2) pure Java implementation.

Use the vendor's pure Java driver, if available, because the JDBC-ODBC driver implementation is slower than a pure Java implementation. Pure Java drivers are known as Type IV.

Figure : JDBC Driver

The JDBC specification defines two other driver types, Type II and Type III; however, they are less common

Step 2 → Define Connection URL

- After loading JDBC Driver, specify the location of the database server.
- URLs referring to databases use the jdbc: protocol and embed the server host, port, and database name (or reference) within the URL.

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String oracleURL = "jdbc:oracle:thin:@" + host + ":" + port + ":" +
   dbName; String sybaseURL = "jdbc:sybase:Tds:" + host + ":" +
   port + ":" + "?SERVICENAME=" + dbName;
String msAccessURL = "jdbc:odbc:" + dbName;
```

Step 3 → Establish Connection

- To make the actual network connection, pass the URL, database username, and database password to the getConnection method of the DriverManager class, throws an SQLException, so you need to use a try/catch block.

```
String username = "jay_debesee";
String password = "secret";
Connection connection = DriverManager.
getConnection(oracleURL, username, password);
```

Step 3 → Establish Connection

- Connection class methods :
  1) prepareStatement : Creates precompiled queries for submission to the database
  2) prepareCall : Accesses stored procedures in the database.
  3) rollback/commit: Controls transaction management.
  4) close : Terminates the open connection.
  5) isClosed : Determines whether the connection timed out or was explicitly closed.

Step 3 → Establish Connection

- Optional part of establishing the connection is to look up information about the database with the getMetaData method.
- This method returns a DatabaseMetaData object that has methods with which you can discover
  - the name and version of the database itself (getDatabaseProductName, getDatabaseProductVersion) or
  - of the JDBC driver (getDriverName, getDriverVersion)

Step 3 → Establish Connection

```
DatabaseMetaData dbMetaData = connection.getMetaData();
String productName =
dbMetaData.getDatabaseProductName();
System.out.println("Database: " + productName);
String productVersion =
dbMetaData.getDatabaseProductVersion();
System.out.println("Version: " + productVersion);
```

Step 4 → Create a statement object

- A Statement object is used to send queries and commands to the database.
- It is created from the Connection using createStatement :

```
Statement statement = connection.createStatement();
```

- Most, but not all, database drivers permit multiple concurrent Statement objects to be open on the same connection.

Step 5 → Execute Query / Update

- A Statement object, you can use it to send SQL queries by using the executeQuery method, which returns an object of type ResultSet.

```
String query = "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet = statement.executeQuery(query);
```

Step 5 → Execute Query / Update

- Statement class example :

  1) executeQuery :

  Executes an SQL query and returns the data in a ResultSet.

  The ResultSet may be empty, but never null.

  2) executeUpdate :

  Used for UPDATE, INSERT, or DELETE commands. Returns the number of rows affected, which could be zero. Also provides support for Data Definition Language (DDL) commands, for example, CREATE TABLE, DROP TABLE, and ALTER TABLE.

Step 5 → Execute Query / Update

3) executeBatch :

Executes a group of commands as a unit, returning an array with the update counts for each command.

Use addBatch to add a command to the batch group. Note that vendors are not required to implement this method in their driver to be JDBC compliant.

4) setQueryTimeout :

Specifies the amount of time a driver waits for the result before throwing an SQLException.

Step 5 → Execute Query / Update

5) getMaxRows / setMaxRows

Determines the number of rows a ResultSet may contain. Excess rows are silently dropped. The default is zero for no limit.

you can use a Statement object to create parameterized queries by which values are supplied to a precompiled fixed-format query

Step 6 → Process the Results

- next method of ResultSet to move through the table a row at a time.
- Within a row, ResultSet provides various getXxx methods that take a column name or column index as an argument and return the result.
- use getInt if the value should be an integer, getString for a String, and so on for most other data types.
- If you just want to display the results, you can use getString for most of the column types.

Step 6 → Process the Results

- if you use the version of getXxx that takes a column index (rather than a column name), note that columns are indexed starting at 1 (following the SQL convention), not at 0 as with arrays, vectors, and most other data structures in the Java programming language.
- The first column in a ResultSet row has index 1, not 0.

```
while(resultSet.next()) {
System.out.println(resultSet.getString(1) + " " +
resultSet.getString(2) + " " + resultSet.getString("firstname") +
" " resultSet.getString("lastname"));
}
```

Step 6 → Process the Results

- use the column name instead of the column index. That way, if the column structure of the table changes, the code interacting with the ResultSet will be less likely to fail.
- In JDBC 1.0, you can only move forward in the ResultSet; however, in JDBC 2.0, you can move forward (next) and backward (previous) in the ResultSet as well as move to a particular row (relative, absolute).
- Be aware that neither JDBC 1.0 nor JDBC 2.0 provides a direct mechanism to determine the JDBC version of the driver.

Step 6 → Process the Results

- In JDBC 3.0, this problem is resolved by the addition of getJDBCMajorVersion and getJDBCMinorVersion methods to the DatabaseMetaData class.

Step 6 → Process the Results

- ResultSet methods:
- 1) next / previous : Moves the cursor to the next (any JDBC version) or previous (JDBC version 2.0 or later) row in the ResultSet, respectively.
- 2) relative / absolute : The relative method moves the cursor a relative number of rows, either positive (up) or negative (down). The absolute method moves the cursor to the given row number. If the absolute value is negative, the cursor is positioned relative to the end of the ResultSet (JDBC 2.0).

Step 6 → Process the Results

- 3) getXXX : Returns the value from the column specified by the column name or column index as an Xxx Java type (see java.sql.Types). Can return 0 or null if the value is an SQL NULL.
- 4) wasNull : Checks whether the last getXxx read was an SQL NULL. This check is important if the column type is a primitive (int, float, etc.) and the value in the database is 0. A zero value would be indistinguishable from a database value of NULL, which is also returned as a 0. If the column type is an object (String, Date, etc.), you can simply compare the return value to null.

Step 6 → Process the Results

- 5) findColumn : Returns the index in the ResultSet corresponding to the specified column name.
- 6) getRow : Returns the current row number, with the first row starting at 1 (JDBC 2.0).
- 7) getMetaData : Returns a ResultSetMetaData object describing the ResultSet. ResultSetMetaData gives the number of columns and the column names

Step 6 → Process the Results

- To be able to dynamically discover high-level information about the result. That is the role of the ResultSetMetaData class: it lets you determine the number, names, and types of the columns in the ResultSet.
- ResulteSetMetaData methods :
- 1) getColumnCount : Returns the number of columns in the ResultSet.
- 2) getColumnName : Returns the database name of a column (indexed starting at 1).

Step 6 → Process the Results

- 3) getColumnType : Returns the SQL type, to compare with entries in java.sql.Types.
- 4) isReadOnly : Indicates whether the entry is a read-only value.
- 5) isSearchable : Indicates whether the column can be used in a WHERE clause.
- 6) isNullable : Indicates whether storing NULL is legal for the column.

Step 6 → Process the Results

- ResultSetMetaData does not include information about the number of rows; however, if your driver complies with JDBC 2.0, you can call last on the ResultSet to move the cursor to the last row and then call getRow to retrieve the current row number

Step 7 → Close the connection

- To close the connection explicitly :
  connection.close();
- Closing the connection also closes the corresponding Statement and ResultSet objects.
- Reusing existing connections is such an important optimization that the JDBC 2.0 API defines a ConnectionPoolDataSource interface for obtaining pooled connections.