**JDBC – Java Database Connectivity**

* **What is JDBC?**
  + In the IT world, the center of attraction is DATA. When we build software or any application, we actually process data.
  + To process data, first we need to store it in the database. There are several SQL and NO-SQL databases available in the market.
    - List of SQL-DBs: RDBMS
      * MySQL
      * PostgreSQL
      * Microsoft SQL Server
      * Oracle Database
      * SQL Lite
      * Amazon RDS
    - List of No-SQL DBs:
      * MongoDB
      * RavenDB
      * Amazon DocumentDB
  + To work with DBs, we have to fire SQL Queries. In the real world, user will not execute SQL queries manually but will communicate with DB via GUI or from Frontend. That means we must connect our Front-end application to the backend Database, and it requires connectivity layer between them and in java connectivity layer between Java-Frontend application to Backend database called JDBC.
  + So, whenever we want to connect java application with database will require JDBC.
  + To make connectivity java application with DB it requires driver and in Java we have 4 different types of drivers,
    - Native driver 🡪 ODBC(Open Database Connectivity) bridge 🡪 Use for MS-Access DB
    - Native API:
      * Oracle: Requires Oracle client installed
      * DB2: Uses IBM DB2 native libraries
      * Sybase: Older driver, requires native components
    - Network Protocol Driver:
      * IDS Server: Used for Internet databases
      * IBM WebSphere: Used with AS/400
    - Pure JAVA Driver:
      * MySQL: Pure Java, widely used
      * PostgreSQL: Robust open-source support
      * SQL Server: Official Microsoft driver
      * Oracle (Thin): Thin client, pure Java
      * MariaDB: Lightweight, modern
      * MongoDB (BI Connector): JDBC-to-Mongo translation
* **Steps to connect Java Application with Database and Perform DB operations: (Here we are using MySQL DB)**
  + **Step 1: Import the package**
    - For MySQL we have to import the package below in the Java file:
      * **import java.sql.\*;**
  + **Step 2: Load and Register the Driver**
    - **Step 2.1: Load Driver**
      * Loading the Driver is dependent on which database we are working with.
      * Each database, such as MySQL, Oracle, or PostgreSQL, uses a different JDBC driver. Here, we are using MySQL DB so for that use driver below:
        + **com.mysql.jdbc.Driver**
      * **To load a driver, we have to download a JAR file, a library from the internet**
        + **Library name:** mysql-connector
        + In newer versions of JDBC (JDBC 4.0+), the driver is auto-loaded if in the classpath, so this step is sometimes optional.
    - **Step 2.2: Register the Driver**
      * To register the Driver, we need to use method called ***“forName()”*** in the class file which will load your driver. Below is the Syntax for the method:
        + Class.forName("com.mysql.cj.jdbc.Driver");
  + **Step 3: Connecting or establishing the connection**
    - To establish the connection, we have to instantiate ***“Connection”*** interface and use ***“getConnection()”*** static method which belongs to **DriverManager** class.
    - ***“getConnection()”*** method will take 3 different parameters,
      * **URL**: Database URL
      * **Username**: Database Username
      * **Password**: Database Password
    - **Syntax:**

Connection connectionObject = DriverManager.getConnection(“DB\_URL”, “USER\_NAME”, “PASSWORD”);

* + **Step 4: Create the statement**
    - To write a statement, we need to use **“*Statement*”** interface and call ***“createStatement()”*** method with the object of the Connection interface which we have created in the above step.
    - **Syntax:**

Connection connectionObject = DriverManager.getConnection(“DB\_URL”, “USER\_NAME”, “PASSWORD”);

Statement statementObject = connectionObject.createStatement();

* + - **There are 3 types of statements:**
      * **Normal Statement:**
        + When we want to execute a Query, we can do it with the help of Statement class object.
        + It only uses for Simple static queries.
        + **The drawback of Statement is it does not accept any input parameters and it is not SQL Injection safe.**
        + **Syntax:**

Statement statementObject = conn.createStatement();

ResultSet resultObject = statementObject.executeQuery("ADD QUERY");

**executeQuery(“ADD QUERY”)** is used for SELECT

**executeUpdate(“ADD QUERY”)** is used for INSERT, UPDATE, DELETE

* + - * **Prepared Statement:**
        + If we have pre-defined queries or in-built but if we want to pass different values, then prepared statement is the best option.
        + **It is useful for** Parameterized queries (with placeholders ?). For example, **String sql = "SELECT \* FROM users WHERE id = ?";**
        + **It is safe against SQL Injections.**
        + You use ? as a placeholder in SQL and then set values using setDataType() methods (e.g., setInt, setString).
        + **Syntax:**

String sqlVariable= "ADD QUERY WITH = ?";

PreparedStatement preparedObject = conn.prepareStatement(sqlVariable);

preparedObject.setDataType(value1, value2,…);

ResultSet resultObject = preparedObject.executeQuery();

* + - * + **Example:**

String sql = "SELECT \* FROM users WHERE id = ?";

PreparedStatement pstmt = conn.prepareStatement(sql);

pstmt.setInt(1, 101);

ResultSet rs = pstmt.executeQuery();

* + - * **Callable Statement:**
        + If we want to use PL-SQL in the script or use store procedure the Callable statement is the best option.
        + It is useful to executing logic or Enterprise/business logic on the database side.
        + It allows input, output and INOUT parameters.
        + It is also safe against SQL injections.
        + **Syntax:**

CallableStatement callableObject = conn.prepareCall("{call ADD PROCEDURE NAME(?)}");

callableObject.setDataType(value1, value2,…);

ResultSet resultObject = callableObject.executeQuery();

* + - * + **Example:**

**MySQL Store Procedure:**

CREATE PROCEDURE getUserById(IN userId INT)

BEGIN

SELECT \* FROM users WHERE id = userId;

END;

**Java Code:**

CallableStatement cstmt = conn.prepareCall("{call getUserById(?)}");

cstmt.setInt(1, 101);

ResultSet rs = cstmt.executeQuery();

* + **Step 5: Execute the query**
    - Once we have prepared the query, we need to execute the query. For execute the query we can call appropriate execution methods:
      * executeQuery(“Add QUERY”) for **SELECT**
      * executeUpdate(“Add QUERY”) for **INSERT**, **UPDATE**, **DELETE**
    - In SQL we have different types of queries:
      * **DDL: Data Definition Language** 
        + In DDL, whenever we want to create structure of a table or a database
        + DDL commands are used to define and manage database schema and structure. They create or modify tables, databases, indexes, and other schema-related elements. Below operations we do in DDL:

**CREATE**: Creates a new table, database, view, etc.

**ALTER:** Modifies an existing table structure (e.g., add column, remove column).

**DROP:** Deletes a table or database permanently.

**TRUNCATE:** Removes all records from a table (structure remains).

**RENAME:** Renames a table or column.

* + - * **DML: Data Manipulation Language**
        + In DML, whenever we want to delete, update or insert a value.
        + DML commands are used to **manipulate data** stored in database tables. These commands let you insert, update, or delete rows. Below operations we do in DML:

**INSERT:** Adds new records to a table.

**UPDATE:** Modifies existing data.

**DELETE:** Removes data from a table.

* + - * **DQL: Data Quary Language**
        + In DQL, whenever we want to fetch the value.
        + DQL is used to **retrieve data** from the database. It focuses only on querying data, usually with the SELECT statement. Below operations we do in DQL:

**SELECT:** Extracts data from one or more tables.

* + - * **TCL: Transaction Control Language**
        + TCL commands manage transactions, ensuring data consistency and integrity. They are used to commit, roll back, or save changes made during a transaction. Below operations we do in TCL:

**COMMIT:** Saves all changes made during the current transaction.

**ROLLBACK:** Undoes changes made in the current transaction.

**SAVEPOINT:** Sets a point in a transaction to roll back to later.

**SET TRANSACTION:** Sets properties for the transaction.

**Ex.:** UPDATE users SET age = 31 WHERE id = 2;

* + - See Step 4 for full code example.
    - To execute the query, we need to call execute methods, executeQuery() or executeUpdate() as per the operation needs to perform with using Statement interface object.
    - The result of the query can be store in the ***“ResultSet”***  interface object.
    - **Syntax:**

Connection connectionObject = DriverManager.getConnection(“DB\_URL”, “USER\_NAME”, “PASSWORD”);

Statement statementObject = connectionObject.createStatement();

ResultSet resultObject = statementObject.executeQuery(“Add QUERY”); // For Select operation

ResultSet resultObject = statementObject.executeUpdate(“Add QUERY”); // For Insert, Update, Delete operation

* + **Step 6: Process result or Receive the result based on above query** 
    - Once we execute the query, either we get the table structure, or it return the number of rows affected if we execute INSERT query.
    - To process the result we can use iteration statements such as forLoop, whileLoops, etc. and visualize the data in the table format. See the example below:

**ResultSet rs = stmt.executeQuery("SELECT \* FROM users");**

**while (rs.next()) {**

**int id = rs.getInt("id");**

**String name = rs.getString("name");**

**System.out.println("ID: " + id + ", Name: " + name);**

**}**

* + - To process the result and display which we have stored in the ResultSet interface object, we need to use below methods for it:
      * **next() method:**
        + While we are fetching the results from the table, our pointer will be int the column name row.

Initial Pointer Position While Fetching Data From The Table

|  |  |
| --- | --- |
| Roll No. | Name |
| 1 | Harsh |
| 2 | Ashik |

* + - * + So, to shift pointer to the very first row of the data, we need to use next() method with the object of ResultSet interface in the while loop as a starting position.
        + This method will perform 2 operations,

First, it will shift the pointer to the next line

Second, it will check if there is next line available or not

* + - * + Ex.: while (rs.next()) {…}
      * **getDataType(ColumnNumber) method:**
        + Our database have multiple types of data stored in the single table with the multiple columns for that we need to use getDataType() method.
        + In getDataType() method, replace ***“DataType”*** keyword with the actual datatype class name and add column number for particular column and while we use it in while loop, we need to pass column name in the parameter with double quotes. Such as, for,

Integer 🡪 getInt(ColumnNumber) or getInt(“Column Name”)

String 🡪 getString(ColumnNumber) or getString(“Column Name”)

* + - **Syntax:**

Connection connectionObject = DriverManager.getConnection(“DB\_URL”, “USER\_NAME”, “PASSWORD”);

Statement statementObject = connectionObject.createStatement();

ResultSet resultObject = statementObject.executeQuery(“Add QUERY”); // For Select operation

while(resultObject.next()){

}

* + **Step 7: Close the connection** 
    - Once all the task done for the db, make sure we always close the connection or resources to avoid memory leaks.
    - To close the connection, we use ***“close()”*** method.
    - We can add close connections method to the finally block of try…catch…finally block. That will release or close the resources even code will generate the exceptions.
    - **Ex.:**  
      rs.close();
* **Implement above steps in the Java code:**

**// Step 1: Import the package**

import java.sql.\*;

public class DatabaseExample{

public static void main(String args[]){

String url = "jdbc:mysql://localhost:3306/testdb";

String user = “root”;

String password = “password”;

try {

**// Step 2.2: Load Driver**

Class.forName(“com.mysql.cj.jdbc.Driver”);

**// Step 3: Establish connection**

Connection conn = DriverManager.getConnection(url, user, password);

**// Step 4: Create the Statement (use any type of statement among Normal Statement, Prepared Statement or Callable Statement)**

Statement stmt = conn.createStatement();

**// Step 5: Execute query**

ResultSte rs = stmt. executeQuery("SELECT \* FROM users");

**// Step 6: Process results**

while (rs.next()) {

System.out.println(rs.getInt("id") + ": " + rs.getString("name"));

}

}

catch (Exception e){

e.printStackTrace();

}

finally{

**// Step 7: Close Connection**

rs.close();

stmt.close();

conn.close();

}

}

}