JDBC
(Java Database Connectivity)

➔ It is an API, as the name implies it helps to achieve the connectivity between java program & database.

➔ JDBC helps us to interact with database & in the world of java. JDBC is the one and only API that helps to interact only with RDBMS application

➔ JDBC is DB independent using JDBC we can interact with any RDBMS application exists in the world

Note:

MongoDB & Hadoop Distributed File System (HDFS/Hadoop) application are notRDBMS applications.

JDBC prerequirements: -

install an RDBMS application(MySQL)
create a database(schema) by name
"spark"

create table by name student in the above database

insert some data in the above table

SQL Queries for MySQL RDBMS application

create database spark

```
create table student
(regno int(10) not null,
first_name varchar(20),
middle_name varchar(20),
last_name varchar(20),
primary key(regno));
```

There two path two create jdbc app

1. Jar file
2. maven

Necessary Steps to work with JDBC

1. Load the Driver
2. Get the DB connection via Driver
3. Issue SL queries via Connection
4. Process the results returned by SQL Queries

5. Close all the JDBC objects

Note:
1. All the above steps are mandatory
2. All these steps are interdependent.

Note:
→ "java.sql.*" is the package representation of JDBC
→ Any class/ Interface belongs to this package means its part of JDBC

JAR (JAVA ARCHIVE) File

→ It is a collection of .class files + other necessary resources(Text file,XML, property files etc..)
→ JAR File helps us to transfer the java files/.class files/java application from one location to another location
→ JAR file will have .jar file extension and functionality wise its similar to zip files.

Steps to create JAR file: -

→ Right click on the java project which we want to transfer, select "Export".
→ Select "JAR file" option present under "java" and click on "Next".
→ Provide the "destination & file name", click on "finish".

Steps to make use of JAR file(external way): -

1. Right click on the java project, where we want to make use of jar file, select "build path" and click on "Add External jars".
2. Select jar file and click on open.
3. We see JAR under "referenced libraries".

Steps to make use of JAR file(internal way):-

1. Create a folder named lib in the project
2. Go to the location of jar file and copy it
3. Now in the lib folder paste the jar file
4. Right click on the jar file click on ‘add to build path’
5. We see JAR under “referenced libraries”.

Driver: -

→ Driver is a additional software Component required by JDBC to interact with RDBMS application.
→ Drivers are provided by DB vendor and they are DB dependent.
→ Using MySQL Driver we can only interact with MySQL RDBMS application and using DB2 Driver we can only interact with DB2 RDBMS application.
→Driver as the name implies, acts like a Bridge between java application and RDBMS application.

```
java.sql.Driver d = new com.mysql.jdbc.Driver();
```

→ Driver class is a concrete class, present in driver JAR file, is one thar implements the java.sql.Driver interface.
→ This interface is present in JDBC API and every JDBC driver provider has to implement this interface.
→ By referring Driver manual we can get the "Driver class" information

steps to load the "Driver class" into the program: -

There are 2 ways to load the Driver class
1st way: -
→ By invoking “registerDriver()” method present in “java.sql.DriverManager” class by passing an instance of “Driver class”.

Syntax

public void DriverManager.registerDriver(java.sql.Driver d) throws SQLException

code for MySQL Driver

```
java.sql.Driver ref = new com.mysql.jdbc.Driver();
DriverManager.registerDriver(ref);
```

2nd way: -

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

→ This is the most common approach to register a Driver class which helps us to pass Driver class Name at Runtime.
→ But if we create an instance of driver class using new operator, then driver class name can’t be passed at runtime

Driver Types

There are 4 types of Driver

1.Type 1 : JDBC ODBC Bridge

2.Type 2: Native - API Driver

3.Type 3: Network protocol Driver

4.Type 4: Native protocol Driver

ODBC: open database connectivity

DB URL (DataBase Uniform Resource Locator)

→ As the name implies, it uniquely identifies Database or a RDBMS application in the network.

→ The structure of DB URL is

<protocol>:<sub-protocol>:<sub-name>

1.Protocol

> it is a mandatory information.

>in case of java, protocol is "jdbc"

2.Sub-protocol

→ this information is provided by DB vendor and we have to refer the Driver manual to get this information

→It's a mandatory information

→it identifies the "DB connectivity mechanism" used to connect to DB

→In case of MySQL, sub-protocol is "mysql" but in case of Oracle or DB2 it is different.

3. Subname: -

→ it is a mandatory information.

→ it consists of,

1.Host name(computer name/ IP address)

2.Port number

3.DataBase name/schema name

4.Username & password

→ Arrangement of subname varies from driver to driver, we have to refer the driver manual and arrange accordingly.

port number:-

→It uniquely identifies an application in os.

→In case of db url, it uniquely identifies a RDBMS application.

JDBC DBURL of different database vendor: -

Oracle:
jdbc:oracle:thin:myuser/mypassword@myhost:1521:myDB
MySQL

jdbc:mysql://myhost:3306/myDB?user=myuser&password=mypassword
MS-SQL

jdbc:microsoft:sqlserver://myhost:1433;DatabaseName=myDB;user=myuser;password=mypassword

java.sql.DriverManager: -

→ DriverManager is a concrete class present in JDBC API and as the name implies it manages the driver
→ it helps java program to establish the connection to DB and for that it requires following two critical
information.
1) Driver class
2) DB URL
→ By invoking "registerDriver()" method on DriverManager we can provide an Object of Driver class.

→ By invoking "getConnection()" method on DriverManager we can provide "DB URL"

→ DriverManager getConnection() helps us to establish the connection with database or 228 returns on Object of Connection if it is able to establish the connection to DB

→ java.sql.Connection is an interface and it is an object representation of physical database connection, that is used by the java program to communicate with DB.
→ DriverManager consists of only one constructor which is private default in nature.
→ Hence it cannot be inherited or instantiated so whatever methods it exposes to the outside world, they should be
"public static" in nature.
→ There are 3 overloaded variants of getConnection() method in DriverManager class 235

1. Connection con = DriverManager.getConnection(dburl) throws SQLException;
String dburl = "jdbc:mysql://localhost:3306/myDB?user=root&password=root";

2. Connection con = DriverManager.getConnection(dburl, user,password) throws SQLException;

String dburl = "jdbc:mysql://localhost:3306/myDB";
String user = "root";
String password = "root";

3. Connection con = DriverManager.getConnection(dburl, properties);//properties is a object reference of a class java.util.Properties

Results of RDBMS application

→ Whenever we issue "SELECT" SQL queries to DB it returns DB Results.
→ Whenever we issue "other than SELECT" SQL Queries it returns number of rows affected

JDBC statements

→ JDBC statements send SQL queries to RDBMS and retrieve the data from RDBMS application.
→ There are 2 different types of JDBC statements

1. java.sql.Statement
2. java.sql.PreparedStatement

→ Once we create JDBC statement object (any of the above type) then we must invoke any one of the below method to issue SQL queries to the DB.

int executeUpdate() throws SQLException

→ This method is used to execute "Other then Select" SQL Queries.

This method returns "no. of rows affected count", in the form of integer.
ResultSet executeQuery() throws SQLException

→ This method is used to execute "only SELECT" SQL Queries

This method returns "DB results" in the form of "ResultSet" object.

java.sql.Statement

→ Its an interface & an object of statement is used to execute statement SQL queries.

→ Statement object can be created by invoking createStatement() method on Connection object.

Syntax

Statement Connection.createStatement() throws SQLException

Example

Statement stmt = con.createStatement();

→ where con is the object reference of java.sql.Connection object

java.sql.PreparedStatement

→ Its an interface & an object of PreparedStatement is used to execute "Dynamic SQL Queries"
→ PreparedStatement object can be created by invoking "prepareStatement()" method on "Connection" object.

Syntax

PreparedStatement Connection.prepareStatement(String query) throws SQLException

Example

String query = "SELECT * FROM students_info WHERE regno=?";

PreparedStatement pstmt = con.prepareStatement(query);

→where "con" is the object reference of "java.sql.Connection" object 313
→ PreparedStatements must be used with ? & then ? needs to be set using proper setXXX() method before executing dynamic SQL query.

Syntax

void setXXX(position of ? as int value, corresponding runtime value) throws SQLException

→ where XXX = java data type corresponding to DB column data type
→ Also, "Runtime value" Data type should be same as "XXX Data type".
→ PreparedStatement are also known as "preCompiledStatement" & they helps us to acheive high performance.

Processing the Results returned by SQL Queries

Whenever we issue sql query to RDBMS application via JDBC. there are two kinds of result executed out of RDBMS application

1. Number of rows affected count as "integer value".
2. "DB results" in the form of ResultSet Object. 332

java.sql.ResultSet

→It is an interface an object of ResultSet is an "object representation of the DB result" produced by "select" sql query.

→ResultSet object is produced by invoking executeQuery() on Statement or PreparedStatement objects.

Example:

```
ResultSet rs = stmt.executeQuery("Sql query");
```

Where, stmt is an object reference of Statement interface

```
ResultSet rs = pstmt.executeQuery();
```

→ Where, pstmt is an object reference of PreparedStatement interface

→ ResultSet contains N no of rows with each row containing N no of columns

→no of rows and column in ResultSet are directly depends on "where" condition and "column list" respectively in select SQL query.

→ ResultSet object may consists of zero/more or zero/one rows

→ if ResultSet consists of zero/more of data then we must use while loop

→ if ResultSet consists of zero/one row of data then we can either use while loop or if block(Preferred).

→Once the ResultSet is produced, data from ResultSet can be extracted as follows

1. Move to desired row by calling necessary ResultSet methods
   for eg: next(), first(), last() etc...
2. Retrieve the desired column value using
   get***(position of the column in SQL Query as integer value);
   (OR)
   get***(name of the column in SQL Query as String value);

Where, *** => java datatype corresponding to DB table column datatype

Note: getXXX() are the only method to retrieve data from ResultSet object.

why we need to close Necessary JDBC objects

Jdbc objects such as **Connection**, **Statement**, **PreparedStatement**, **ResultSet** makes use of memory.

→ In case of Connection object, further RDBMS application resources are consumed

→ Also memory consumed by ResultSet object is comparatively more compared to other JDBC Objects.

→ Hence forgetting to close any of the JDBC objects will "heavily impact java aswell as RDBMS application performance" and Garbage Collection should not be relied upon.

→ So it is important to close any of the JDBC object as soon as their job is done.

→To close any of the JDBC object invoke close() method.

public void close() throws SQLException

→ Commonly used JDBC objects are

1. java.sql.DriverManager
2. java.sql.Connection
3. java.sql.Statement
4. java.sql.PreparedStatement
5. java.sql.ResultSet
6. java.sql.SQLException

➔ Apart from DriverManager and SQLException rest of them are "interfaces" whereas DriverManager and SQLException are "Concrete class".

➔ SQLException is a concrete class which extends java.lang.Exception and its a checked exception.

```
try
{
//step 1
//step 2
//step 3
//step 4

}
catch(_____)
{


}
finally
{
    //step 5
}
```

→Step 1 to 4 will be in try block and step 5 will be in finally block
→While making use of JDBC, we must follow 5 steps and out of 5, which happens only once are
1. We need to load the Driver
2. We have to get DB connection
5. We have to close JDBC objects
> Step 3 and 4 (Issuing Sql query and processing results) can happen N no of times depending on ones need.

Code Example

```
package org.example;

import com.sun.org.apache.xerces.internal.impl.io.UCSReader;

import java.sql.*;
import java.util.concurrent.Executor;

/**
 * Hello world!
 *
 */
public class EmployeeManagement
{

    //jdbc basic cre
    private static final String JDBC_URL="jdbc:mysql://localhost:3306/spark";
    private static final String USER_NAME="root";
    private static final String PASSWORD="root";


    private static Connection connection;
    private static PreparedStatement preparedStatement;
    private static ResultSet resultSet;

    public static void main( String[] args )
    {
        try {
            //making database connectivity
            connection = DriverManager.getConnection(JDBC_URL, USER_NAME,
PASSWORD);

            //create table
           // createEmployeeTable();

            //insert data into table
            insertDataIntoDb("rahul",100,"rahul@gmail.com");



        }
```

```
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }

    private static void insertDataIntoDb(String name, int age, String email)
throws SQLException {
        String insertSQL = "INSERT INTO Employee (name,age,email) VALUES (? ,
?, ?)";
        preparedStatement =connection.prepareStatement(insertSQL);
        preparedStatement.setString(1,name);
        preparedStatement.setInt(2,age);
        preparedStatement.setString(3,email);

        preparedStatement.executeUpdate();
        System.out.println("Data Inserted ....");

    }

    private static void createEmployeeTable() throws SQLException {

        String createTable="CREATE TABLE IF NOT EXISTS Employee ( id INT
AUTO_INCREMENT PRIMARY KEY," +
                " name VARCHAR(100) ,"
                +"age INT ,"
                +"email VARCHAR(100)"
                +")";
        preparedStatement=connection.prepareStatement(createTable);
        preparedStatement.execute();
        System.out.println("Table created ....");

    }
}
```

BLOB Example - FETCH

```
package org.example;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.time.LocalDateTime;

public class BlobFetchDemo {
```

```
    private static final String JDBC_URL="jdbc:mysql://localhost:3306/spark";
    private static final String USER_NAME="root";
    private static final String PASSWORD="root";


    private static Connection connection;
    private static PreparedStatement preparedStatement;
    private static ResultSet resultSet;

    public static void main(String[] args) {

        try {
            connection = DriverManager.getConnection(JDBC_URL, USER_NAME,
PASSWORD);
            preparedStatement=connection.prepareStatement("Select image from
blobdemo where id = ?");
            preparedStatement.setInt(1,2);

            resultSet=preparedStatement.executeQuery();
            if(resultSet.next())
            {
                //stream
                InputStream is =resultSet.getBinaryStream("image");
                        //location
                FileOutputStream fis = new FileOutputStream("D:\\Kafka-
Learning\\Jdbc2nd\\src\\main\\java\\res\\rahul.jpg");

                byte[] buffer = new byte[1024];
                while (is.read(buffer)>0)
                {
                    fis.write(buffer);
                }

                fis.close();
                is.close();
                System.out.println("Data Fetched .....");
            }
            else {
                System.out.println("No Image found.........");
            }


        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

BLOB – WRITE

```
package org.example;

import java.io.File;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.time.LocalDateTime;

public class BlobWriteDemo {

    private static final String JDBC_URL="jdbc:mysql://localhost:3306/spark";
    private static final String USER_NAME="root";
    private static final String PASSWORD="root";


    private static Connection connection;
    private static PreparedStatement preparedStatement;
    private static ResultSet resultSet;

    public static void main(String[] args) {

        try {
            connection = DriverManager.getConnection(JDBC_URL, USER_NAME,
PASSWORD);
            File f=new File("D:\\Kafka-
Learning\\Jdbc2nd\\src\\main\\java\\res\\IMG_4035.jpg");
            FileInputStream fis = new FileInputStream(f);
            preparedStatement=connection.prepareStatement("insert into blobdemo
values(?,?)");
            preparedStatement.setInt(1,3);
            preparedStatement.setBinaryStream(2,fis,(int)f.length());
            LocalDateTime now = LocalDateTime.now();
            System.out.println(now);
            preparedStatement.executeUpdate();
            LocalDateTime now1 = LocalDateTime.now();
            System.out.println(now1);
            System.out.println("Data Inserted....");
            connection.close();

        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```