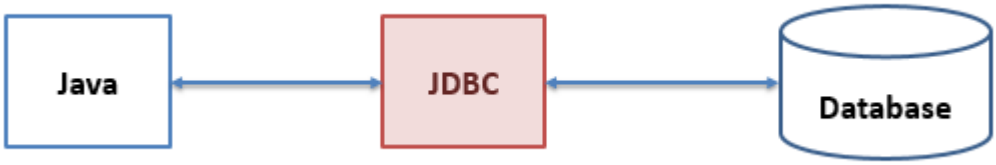
JDBC

Java DataBase Connectivity (JDBC)

- Java Data Base Connectivity is an API, as the name implies, it helps to achieve the connectivity between Java Programs & Database.
- Note: Servlet's & JSPs are also Java Programs.
- If we have a Web Application & if it has a DB, then it needs to interact with DB to read / modify the data
- JDBC helps to do this & in the world of Java, JDBC is the "One & Only API" that helps to interact ONLY with RDBMS (DB) Application
- Also JDBC is "DB Independent" i.e. using JDBC we can interact with any RDBMS Applications exist in the world (like Oracle, MySQL, DB2, MS-SQL, SyBase, etc.,)



NOTE:

- MongoDB
- Cassandra
- Hadoop Distributed File System (HDFS/Hadoop)

They are Applications to store the data but they are not RDBMS Applications.

Some Useful Queries:

1. To connect to database:- use hejm13_db;
2. To get the list of databases:- show databases;
3. To get the list of tables:- show tables;
4. To know the table structure:- describe table_name; Example: describe students_info;

JDBC Pre-requirements:

- Install an RDBMS Application (MySQL 5.5)
- Create a "Database (Schema)" by name "hejm15_db"
- Create a table by name "STUDENTS_INFO" in the above "Database"
- Insert some data into the above table

- **create database heja8_db;**
- **create table students_info(regno int(10) not null, firstname varchar(50),middlename varchar(50), lastname varchar(50), primary key(regno));**

- **insert into students_infovalues (1, 'Sunil', 'Kumar', 'C');**

"java.sql.*" and "javax.sql.*" is the Package Representation of JDBC i.e. Any Class / Interface belongs to this package means, it's part of JDBC.

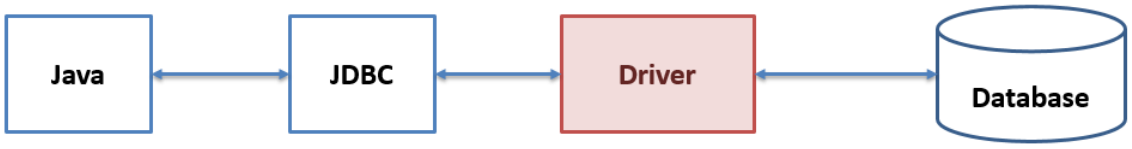
Necessary Steps to Work with JDBC

1. Load the "Driver"
2. Get the "DB Connection" via "Driver"
3. Issue "SQL Queries" via "Connection"
4. "Process the DB Results" returned by "SQL Queries".
5. Close All "JDBC Objects".

Note:

- All steps are mandatory
- All steps are interdependent

Drivers:



- Drivers are external software component required by JDBC to interact with RDBMS Application.
- Drivers are provided by "DB Vendor" & they are "DB Dependent".
- i.e. Using My-SQL Driver we can ONLY interact with My-SQL RDBMS Application & Using DB2 Driver we can ONLY interact with DB2 RDBMS Application.
- "Drivers", as the name implies, acts like a "Bridge" between Java Application and RDBMS Application.
- DB Vendor providers Driver Software in the form of a "JAR File".

JAR (Java ARchive) File:

- It's a Collection of Packages & ".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 an another location.
- Hence JAR File represents a "Java Application"
- JAR File will have ".jar" file extension & functionality wise it's similar to "ZIP" file.

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" & click on "Next".
- Provide the "Destination & File Name", click on "Finish".

Steps to Make Use of JAR File :-

- Right Click on the Java Project, where we want to make use of JAR File, select "Build Path"& click on "Add External Archives..."
- Select the "JAR File" & Click on "Open"
- We see JAR File under "Referenced Libraries"

Driver Class

- "Driver Class" is a Concrete Class, present in driver JAR file, is the one that implements the "java.sql.Driver" interface
- This interface is present in JDBC API & 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:-

- By invoking "registerDriver()" method present in "java.sql.DriverManager" Class by passing an instance of "Driver Class".

Syntax :

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

Code for MySQL Driver :

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

        DriverManager.registerDriver(ref);
```

- Second approach to Load the Driver Class is with the help of Java's "Class.forName()" Method

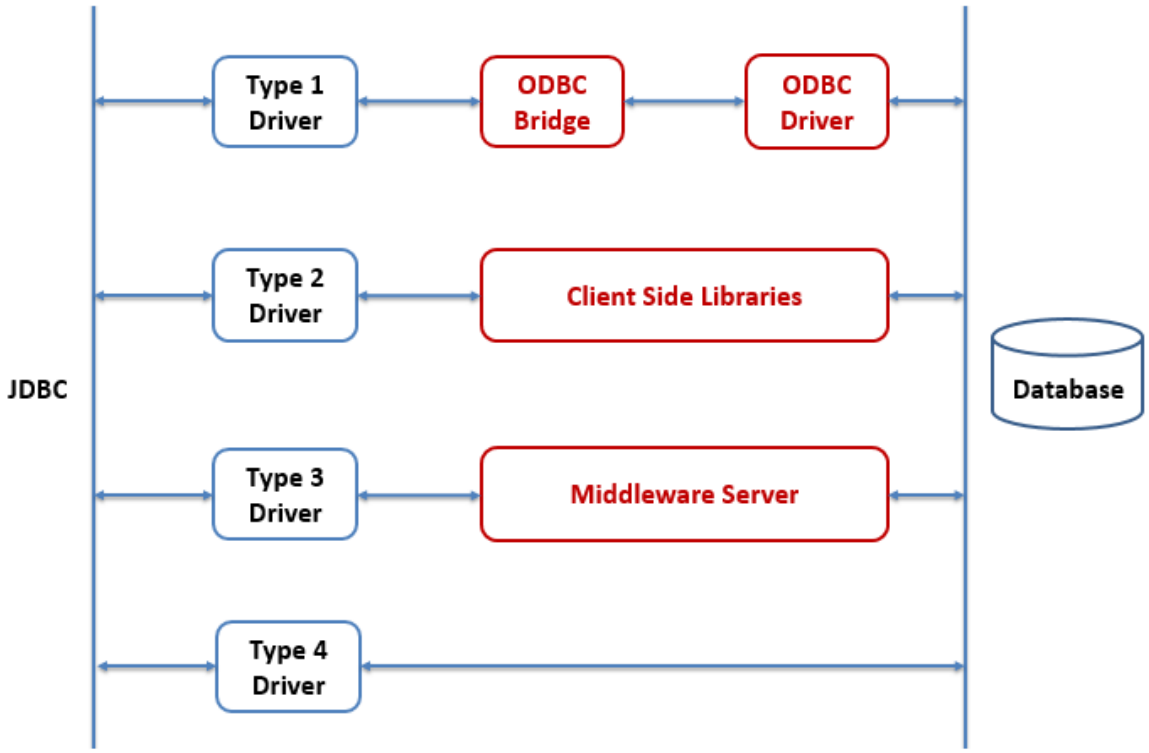
Example for MySQL Driver :

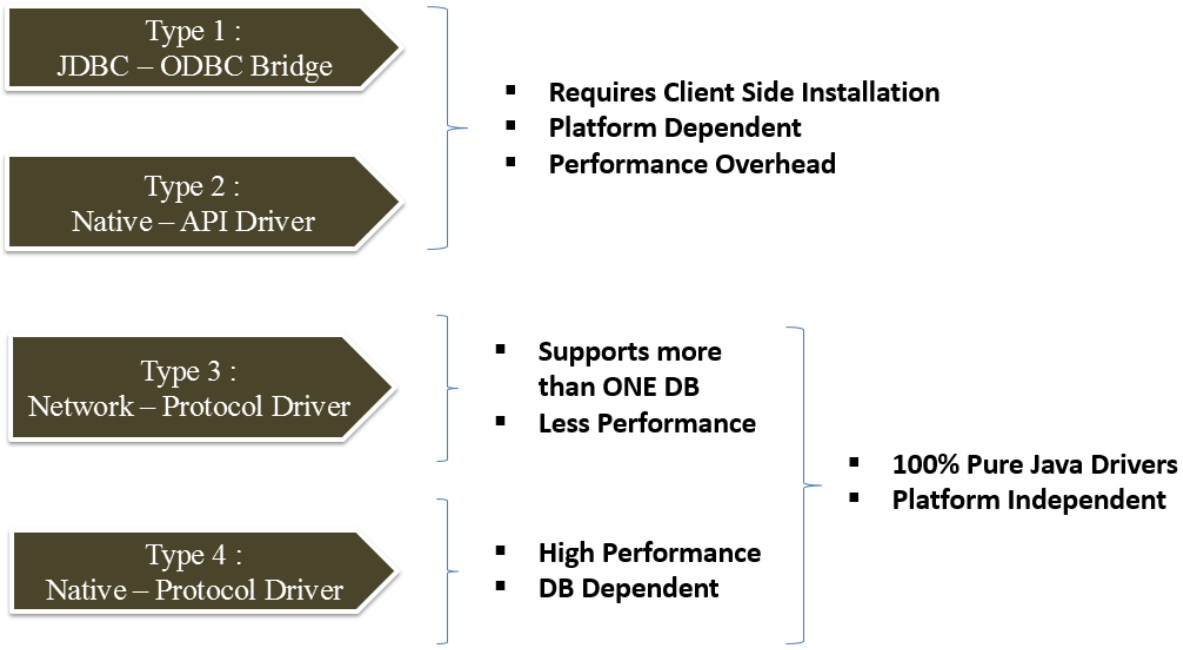
```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

- 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 Drivers





Note: ODBC = Open DataBase Connectivity

Uniform Resource Locator (URL)

- As the name implies, it uniquely identifies a "Resource" or an "Application" in a Network
- There are "many" types of URL's & couple of them are
    1. DB URL
    2. Web URL

DataBase Uniform Resource Locator (DB URL)

- DB URL, as the name implies, it uniquely identifies Database OR a RDBMS Application in the Network.
- The structure of DB URL is **<Protocol>:<Subprotocol>:<Subname>**

Protocol

- It's a Mandatory Information
- > In case of Java / Java EE, Protocol is always "jdbc"
- Protocol is "Case In-sensiti

Subprotocol

- It's a Mandatory Information

- It identifies the "DB Connectivity Mechanisam" used to Connect to DB
- This information is provided by DB Vendor & we have to refer the Driver Manual to get this info
- In case of MySQL, Subprotocol is "mysql" but, incase of Oracle or DB2 its different
- Subprotocol is also "Case In-sensitive".

Subname

- It's a Mandatory Information
- It Consists of,
    - Host Name (Computer Name/IP Address and Case In-sensitive)
    - Port Number (should be Digits)
    - Database Name / Schema Name (Case In-sensitive)
- User Name & Password (Optional and Case Sensitive)
- Arrangement of Subname varies from driver to driver, we have to refer the manual & arrange accordingly

Note:

- **Port Number:**
    a. It uniquely identifies an application in an Operating System
    b. In case of DB URL, it uniquely identifies a RDBMS Application.
- Apart from "user name & password" rest all are Case In-sensitive

JDBC URL - Few Examples:-

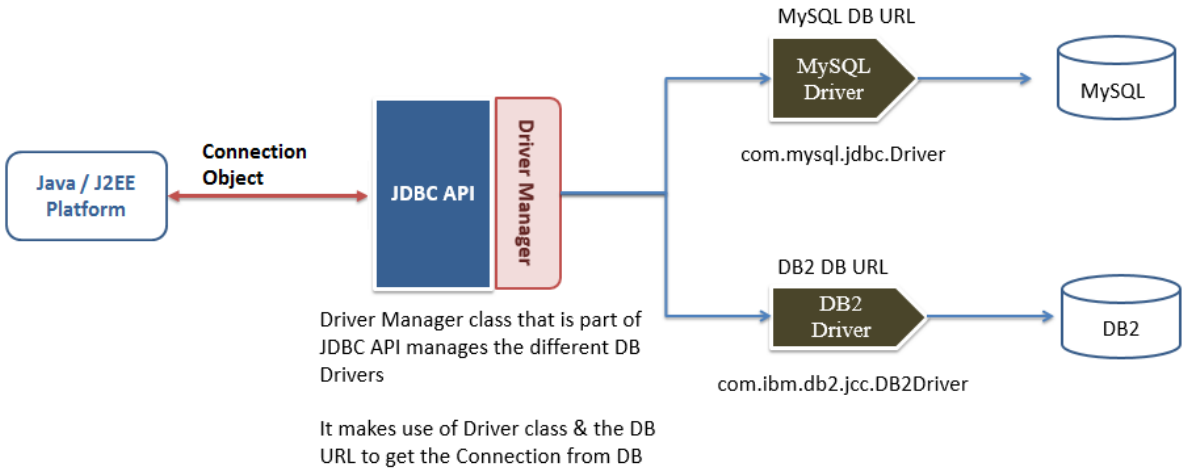
Oracle: jdbc:oracle:thin:myUser/myPassword@myHost:1521:myDB

MySQL: jdbc:mysql://myHost:3306/myDB?user=myUser&password=myPassword

MS-SQL Server:

jdbc:microsoft:sqlserver://myHost:1433;DatabaseName=myDB;user=myUser;password=myPassword

java.sql.DriverManager



- DriverManager is a "Concrete Class" present in JDBC API & as the name implies, it manages the drivers
- It helps Java Program to establish the connection to DB & for that it requires following information
  - Driver Class
  - 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's getConnection() method helps us to establish the connection to the DB. This method
  - throws "**SQLException**" if it is unable to establish the connection to DB
- OR
  - returns an object of "**Connection**" if it is able
- to establish the connection to DB
- "**java.sql.Connection**" is an interface & It's an "Object representation of Physical DB Connection" that is used by Java program to communicate with DB
- DriverManager consist of only **one constructor** which is "**Private Default**" in nature
- Hence it cannot be **inherited** or **instantiated**. So whatever the methods it exposes to outside world, they "should be public static" in nature.
- DriverManager has overloaded versions of **getConnection()** methods. They are,

1. Connection getConnection(String dbUrl) throws SQLException

```
String dbUrl =
"jdbc:mysql://localhost:3306/BECM19_DB?user=root&password=root";
 Connection con = DriverManager.getConnection(dbUrl);
```

2. Connection getConnection(String dbUrl, String userNM, String password)

throws SQLException

```
String dbUrl = "jdbc:mysql://localhost:3306/j2ee";
String userNM = "root";
String pass   = "1234";
Connection con   = DriverManager.getConnection(dbUrl, userNM, pass);
```

3. Connection getConnection (String url, Properties info)

throws SQLException

```
String dbUrl="jdbc:mysql://localhost:3306/BECM19_DB";
String filePath = "C:\\configure.properties";
FileReader reader = new FileReader(filePath);
Properties props = new Properties();
props.load(reader);
Connection con = DriverManager.getConnection(dbUrl, props);
```

Data Present in "configure.properties" File is:-

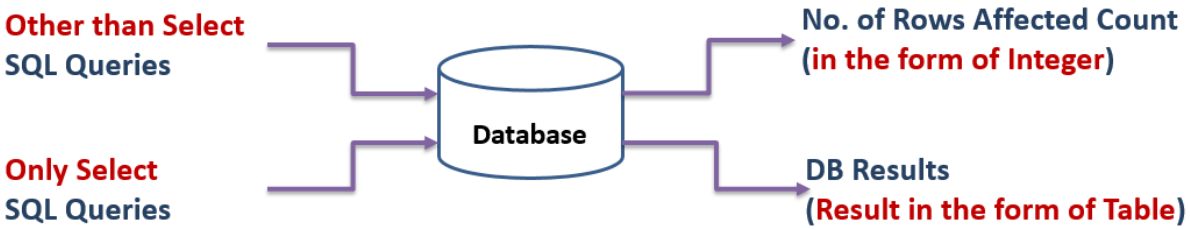
```
#DB Credentials
user = root
password = root
```

NOTE:-

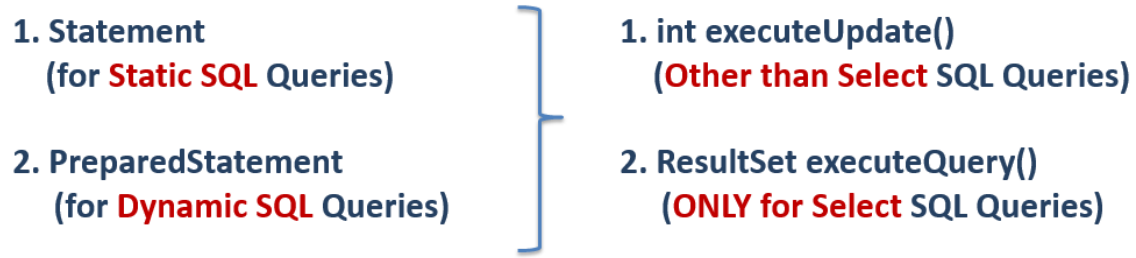
- We can make use of any version of "getConnection()" method to establish connection to RDBMS application
- But "getConnection (String url, Properties info)" helps us to take out the hardcoded credentials from program & keep it outside of the application
- Hence this method is widely used because it helps us to "easily maintain the application" whenever there is change in DB credentials

Results of RDBMS Application

- Whenever we issue "Select SQL Queries" to DB it returns DB Results
- Whenever we issue "Other than Select SQL Queries" to DB then it returns "No. of Rows Affected Count" in the form of Integer
- Hence w.r.t results we can group SQL Queries into Two Groups
    1. Select SQL Query
    2. Other Than Select SQL Query



JDBC Statements



Static SQL Queries

ANY SQL queries "without conditions" OR "ALL Conditions with harcoded values" are called as "Static SQL Queries"

Example :

1. select * from ABC;
2. create database DB_NAME;

3. select * from ABC where X = 1;
4. insert into ABC values (1, 'sunil');

Note : ABC = Table Name

Dynamic SQL Queries

ANY SQL Queries which has conditions AND One/More conditions value get decided at runtime are known as "Dynamic SQL Queries"

Examples:

1. select * from ABC where X=? and Y=?;
2. select * from ABC where X=1 and Y=?;
3. insert into ABC values (?, 'sunil');

JDBC Statements

- JDBC Statements send SQL queries to RDBMS and retrieve the data from RDBMS Application
- There are 3 different types of JDBC Statements
  1. java.sql.Statement
  2. java.sql.PreparedStatement
  3. java.sql.CallableStatement
- 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 DB

1. int executeUpdate() throws SQLException

- This method is used to execute "Other than SELECT" SQL Queries
- This method returns "No. of Rows Affected Count" in the form of Integer.

2. 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

3. boolean execute() throws SQLException

- This method is used to execute "ANY SQL QUERIES"

- This method returns boolean "true" when we execute "SELECT" type of query and returns boolean "false" when it is "other than SELECT" sql query.

java.sql.Statement

- Its an interface & an Object of Statement is used to execute "Static SQL Queries"
- Statement Object can be created by invoking "createStatement()" method on "Connection" Object

  Syntax: public Statement 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: public PreparedStatement 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

- PreparedStatements MUST be used with ?(place holders) & these "?" needs to be set using proper setXXX() method before executing dynamic SQL query.

  Syntax: void setXXX(Postiton of ? as Int Value, 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 DataType"

- PreparedStatements are also known as “Precompiled Statements” & they helps us to achieve "high performance".

Processing the Results returned by SQL Queries :-

- Whenever we issue SQL Queries to RDBMS Application via JDBC there are two kinds of results expected out of RDBMS Application
  1. No. of Rows Affected Count
  2. DB Results
- JDBC returns
  a. "No. of Rows Affected Count" as "Integer Value"
  b. "DB Results" in the form of "ResultSet Object"

java.sql.ResultSet

- Its an interface & an Object of ResultSet is an "Object representation of DB Results" produced by Select SQL Query
- ResultSet object is produced by invoking "executeQuery()" Method on Statement OR PreparedStatement Objects

Example:-

1. **ResultSet rs = stmt.executeQuery(query);**

   Where, "stmt" is a Object reference of Statement .

   "query" is a variable consisting of sql query in String format.

2. **ResultSet rs = pstmt.executeQuery();**

   where "pstmt" is a Object reference of PreparedStatement

- ResultSet consists of N number of Rows with each row containing N number of Columns
- Number of rows and columns in Resultset directly depends on "where condition" & "column list" respectively in "Select SQL Query"
- ResultSet object may consist of "Zero/More" OR Zero/One" rows
- ResultSet consists of "Zero/One" row if where condition is on Primary Key with "equal to (=)"
- for rest of the cases it consists of "Zero/More" rows
- If ResultSet consist of zero/more rows of data thenwe must use "while loop"
- If ResultSet consist of zero/one row of data then we can use either "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 Ex : next(), first(), last(), etc.

   Syntax: public boolean next() throws SQLException

2. Retrieve the desired column value using any one ofthe below getXXX() methods
   a. public XXX getXXX(String columnName) throws SQLException
   b. public XXX getXXX(int columnIndex) throws SQLException

   where XXX = Java Data Type corresponding to DB Table column data type

NOTE : getXXX() methods are the ONLY way to retrieve data from ResultSet object.

Why we need to Close Necessary JDBC Objects:-

- JDBC Objects such as
  i. Connection
  ii. Statement, PreparedStatement and
  iii. ResultSet make 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 as well as RDBMS application performance" & Garbage Collection should not be relied upon
- So it's important to close any of the JDBC Object as soon as their job is done
- To close any of the JDBC Objects invoke "close()" method

  Syntax: public void close() throws SQLException

JDBC Imp Points to Remember:

- SQLException is a Concrete Class which extends "java.lang.Exception" & its a "Checked Exception"
- JDBC Steps 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, ONLY Once
    - We need to load the Driver (Step 1)
    - We have to get the DB Connection (Step 2)
    - We have to Close JDBC Objects (Step 5)
- But, Step 3 & 4 (i.e. Issuing SQL Queries & Processing Results) can happen "N" number of times depending on our need (Min. "1" & Max."N number")
- JDBC is "DB Independent"(i.e it can work with ANY RDBMS Application) because "Driver JAR" is DB Dependent
- Though JDBC claims that it is "DB Independent" but, because of "SQL Queries (step 3)", it is actually "DB Dependent"

Note: SQL is "DB Dependent". For example, below are the different SQL queries used to fetch first 5 records of the table depending on RDBMS application

MySQL & PostgreSQL: SELECT * FROM ABC LIMIT 5;

Oracle: SELECT * FROM ABC WHERE rownum <=5;

MSSQL Server: SELECT TOP 5 * FROM ABC;

****************************END of JDBC **************************************