**JDBC**

**Why JDBC?**

As we know we use core java to develop stand alone or desktop application, but as of now, we know that technology  
has grown so much that only standalone application will not be enough, i.e., for example if we take WhatsApp, will we use WhatsApp if we are not able to send any data, no right, so that’s why connecting the java application to the database becomes essential. that’s why we need Advance java that is J2EE-java enterprise edition which is used to develop web application. We know that every web application is connected with database.

What is database?

Database is a centralized place which is used to store data, and we can access that data by using SQL.   
Here JDBC comes into picture.

**What is JDBC?**

JDBC is an acronym for Java Database Connectivity, it is an API which helps to connect a java program or application to the database.

What is API?

API stands for Application Programming Interface. It is nothing but an intermediate software that allows two applications to talk to each other.

**Basic Components of JDBC**

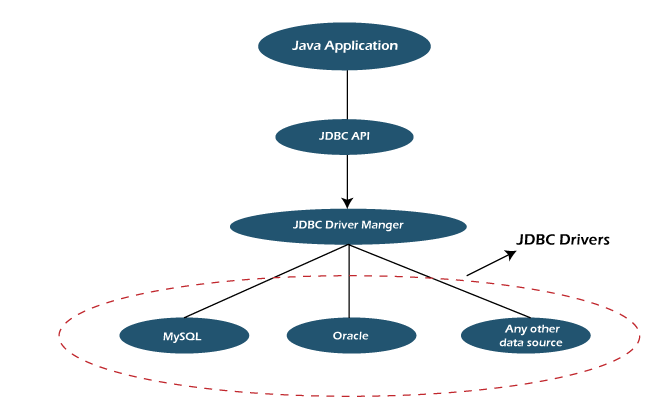
Driver: It converts java instructions into Database specific / understandable instructions and vice versa.

Connection: It is an interface, a network route / socket is required to connect java application to database and vice versa. So, a connection has to be established hence we require connection object.

Statement: To send our queries to database and receive result from database to java application we need something like transportation medium, hence statement is used. It acts as a transportation medium.

Result Set: Once the query is sent to database it is processed, executed and result is produced, that result need to be stored. Hence, we create an object of result set. The result set holds the set of results of SQL queries.

**JDBC Architecture**



Java Application: It is a java applet or a servlet that communicates with a data source.

JDBC API: The JDBC API allows Java programs to execute SQL statements and retrieve results.  
Some of the important classes and interfaces defined in JDBC API are as follows:

JDBC Driver Manager: It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.

JDBC drivers: To communicate with a data source through JDBC, we need a JDBC driver that intelligently communicates with the respective data source.

**Types of JDBC**

There are mainly four types of JDBC Drivers

\* JDBC-ODBC bridge driver

\* Native API driver

\* Network protocol driver

\* Thin driver

JDBC-ODBC bridge driver: The JDBC-ODBC bridge driver uses ODBC driver to connect to the database.  
The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.   
This is now discouraged because of thin driver. Oracle does not support the JDBC-ODBC Bridge from Java 8.   
Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.  
Advantages:  
\* Easy to use.  
\* Can be easily connected to any database.  
Disadvantages:  
\* Performance degraded because JDBC method call is converted into the ODBC function calls.  
\* The ODBC driver needs to be installed on the client machine.

Native-API driver: The Native API driver uses the client-side libraries of the database.  
The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.  
Advantage:  
\* Performance upgraded compared to JDBC-ODBC bridge driver.  
Disadvantage:  
\* The Native driver needs to be installed on each client machine.  
\* The Vendor client library needs to be installed on client machine.

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.  
Advantage:  
\* No client-side library is required because of application server that can perform many tasks like auditing,  
load balancing, logging etc.  
Disadvantages:  
\* Network support is required on client machine.  
\* Requires database-specific coding to be done in the middle tier.  
\* Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

The Thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.  
Advantage:  
Better performance than all other drivers.  
No software is required at client side or server side.  
Disadvantage:   
Drivers depend on the Database.

**Steps to achieve connection**

To connect java to database there are seven main steps that are

1.Import packages

2.Load and register driver

3.Establish connection to database

4.Create statement

5.Execute Query

6.Process the result

7.Close connection

Explanation  
1.**Import packages** - we know in java by default only java.lang packages we will be imported but to perform operations on database we need some of the classes and interfaces which belongs to java.sql package so first we need to import that by using **import java.sql.\*** statement.

2.**Load and register driver** - we will be using drivers to communicate to database. Depending on the type of database we are using we have to download corresponding jar file or we can create maven project and add dependencies, after that we have to use **Class.forName("com.mysql.cj.jdbc.Driver");**  
i.e., Class class has a method forName which will load the driver based on the path. Once after loading and registering now we have to establish connection.

3.**Establishing connection** - To establish connection between java and database we use Connection interface.  
i.e., **Connection connection=DriverManager.getconnection("URL","UN","PWD");**  
Since we cannot create an object for an interface so instead, we can provide implementation to the methods of interface and initialize reference variable with that class object. It is already done by someone and they have stored it in DriverManager class, it has a method getConnection which accepts "URL","UN" and "PWD" as the parameters. After this we will get connection with the database, then we have to create statement.

4.**Create statement** – here there are mainly three types of statement   
1. Statement   
2. Prepared Statement  
3. Callable statement  
All three are interfaces out of which we will be learning two i.e., statement and prepared statement.  
Statement is used whenever we want to execute multiple queries.  
Prepared statement is used in the case of dynamic input or when we want to execute same query 'n' number of times,  
to create statement we will write, **Statement statement=connection.createStatement();**  
it will not take parameters whereas prepared statement will take Query as the parameter.  
i.e., **PreparedStatement preparedStatement=connection.preparedstatement("Query");**

5.After preparing the statement we have to **Execute Query** for that we have three methods   
1. Execute - return type- boolean, tells if changes done or not, used in the case of DDL i.e., any changes done to table.  
2. ExecuteUpdate - return type- int, tells number of rows affected, used in the case of DML i.e., any changes in values or table data.  
3. ExecuteQuery - return type- Resultset i.e., in the tabular form, used in the case of DQL i.e., fetching values.

6.Once after query execution we have to **fetch the result / resultset** - for that we use rs.next() and then we will store the value of column in a variable, depending on datatype we will create variable and store the values then we can print that variable.

7.Once after the communication with the database done, we have to **close the connection** in order to avoid any data leakage or we have to close connection because without closing existing connection new connection can’t be created. Also, if we don’t close connection performance of the application will also decrease.

**Programs**

CRUD operations using JDBA

Creating Database

package jdbcPackage;

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

public class CreatingDatabase  
{  
public static void main(String[] args) throws ClassNotFoundException, SQLException  
{  
 Class.forName("com.mysql.cj.jdbc.Driver");  
 Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/","root","root");  
 Statement st = con.createStatement();  
 st.execute("Create database Jspiders");  
 System.out.println("DataBase created");  
 con.close();  
}  
}

Creating Table

package jdbcPackage;

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

public class CreatingTable  
{  
public static void main(String[] args) throws ClassNotFoundException, SQLException  
{  
 Class.forName("com.mysql.cj.jdbc.Driver");  
 Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/Jspiders","root","root");  
 Statement st = con.createStatement();

st.execute("Create Table Students(Stud\_Id Int,Stud\_Name Varchar(20),Mock\_Rating Varchar(5))");  
 System.out.println("Table created");  
 con.close();

}  
}

All CRUD Operations

package jdbcPackage;

import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.Scanner;

public class CrudOperation

{  
static Scanner scanner=new Scanner(System.in);

public static void main(String[] args) throws ClassNotFoundException, SQLException

{

Class.forName("com.mysql.cj.jdbc.Driver");

Connection con=DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/jspiders","root","root");

boolean b =true;

while(b)

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Select any one option :");

System.out.println("1.Insert 2.Retrive All 3.Retrive Based on ID 4.Delete record 5.Update record 6.Exit");

int val = sc.nextInt();

switch(val)

{

case 1:

{

System.out.println("Enter Student ID,Name,Rating");

PreparedStatement prep = con.prepareStatement("insert into students values(?,?,?)");

int id=sc.nextInt();

String name=sc.next();

String rate=sc.next();

prep.setInt(1, id);

prep.setString(2, name);

prep.setString(3, rate);

prep.execute();

System.out.println("Values Inserted...");

break;

}

case 2:

{

Statement st=con.createStatement();

ResultSet rs = st.executeQuery("select \* from students");

while (rs.next())

{

int Stud\_id = rs.getInt(1);

String stud\_name=rs.getString(2);

String stud\_rate=rs.getString(3);

System.out.println(Stud\_id+" "+stud\_name+" "+stud\_rate);

}

break;

}

case 3:

{

PreparedStatement prep = con.prepareStatement("select \* from students where stud\_id=?");

System.out.println("Enter student id");

int a=sc.nextInt();

prep.setInt(1, a);

ResultSet rs = prep.executeQuery();

while(rs.next())

{

int Stud\_id = rs.getInt(1);

String stud\_name=rs.getString(2);

String stud\_rate=rs.getString(3);

System.out.println("The details are:");

System.out.println(Stud\_id+" "+stud\_name+" "+stud\_rate);

}

break;

}

case 4:

{

PreparedStatement st=con.prepareStatement("delete from students where stud\_id=?");

System.out.println("Enter Student Id:");

int a=sc.nextInt();

st.setInt(1,a);

st.execute();

System.out.println("Values Deleted...");

break;

}

case 5:

{

PreparedStatement st=

con.prepareStatement("update students set stud\_id=?,stud\_name=?,Mock\_rating=? where stud\_id=?");

System.out.println("Enter Id of row to be updated ");

int a=sc.nextInt();

System.out.println("Enter updated Student ID,Name,Rating");

st.setInt(4, a);

int id=sc.nextInt();

String name=sc.next();

String rate=sc.next();

st.setInt(1, id);

st.setString(2,name);

st.setString(3,rate);

st.execute();

System.out.println("Values Updated...");

break;

}

case 6:

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*THANK YOU\*\*\*\*\*\*\*\*\*\*\*\*\*");

b=false;

break;

}

default :

{

System.out.println("Select proper Option...");

}

}

}

}

}