JDBC:-

Basically all Application divided into two parts:-

1. **Standalone-Application.**
2. **Web-Application**

Standalone-Application:-The Application which running on single machine such types of application known as Standalone-Application.
Ex:-Notepad, calculator, MS-Office, VLC etc.

Web-Application:- The application which providing services over the web such type of application known as Web-Application.
Ex:-google, Facebook, twitter etc.

Either we developed Standalone-Application or Web-Application, application required some storage area where application can store and access data like username, password, image, mp3 etc.

All Storage Area divided into two parts:-

1. **Temporary Storage Areas.**
2. **Permanent Storage Areas.**

Temporary Storage Areas:-This type of memory system where we can store data temporary.
Like RAM level Memory System, All JVM Memory (heap,stack,method etc).
The store data which store in temporary memory system we lost when shutdown/restart the system, shutdown the JVM, close the application.

Permanent/Persistence Storage area:-This type of memory system where we can store data permanent.
Once we store data in permanent memory system, we never lost either we shutdown/restart the system, shutdown the JVM, close the application.

Types of Permanent/Persistence Storage area:-

1. **File System.**
2. **DBMS**
3. **Data Warehouses.**

File System:-This is one type of permanent memory system, which is provided by local operating system.
It is best suitable for use when we store very less amount of data.

Limitation of File System:-

1. It is database dependent because this is provided by local operating system.
2. We can't store huge amount of data.
3. There is no security mechanism is available for store data.
4. There is no mechanism available to prevent duplicate data.
5. There is no support of query language.

To overcome above limitation, we should go for database:-

1. This is one type of permanent storage system, which is provided by database vendor like oracle, mysql, mssql, Sybase etc.
2. It is best suitable for use when we store huge amount of data.
3. Security mechanism is available, without enter user id and password we can't access stored data.
4. There is multiple mechanism is available to prevent duplicate like primary key, unique key.

Limitation of database:-

1. We can't store very huge amount of data like tera bytes.
2. Database provide only support of structure[Tabular data or Relational] but not provide support of Semi structured data like xml and unstructured data like video file, audio file, image etc.

To overcome above limitation, we should go for **Data Warehouses**.

In process of development java Application, Application requirement to interact with database and perform data storage and access operation, for this requirement we need a technology and that technology is JDBC.

JDBC:-

1. JDBC is a technology which can be used to interact with database from java application.
2. JDBC is the part of J2SE.
3. JDBC is an API which contains specification that is defined by java vendor(SUN MICRO SYSTEM) and implemented by Database vendor.
4. Database vendor provided implementation of JDBC specification and this implementation is known as "Driver Software".

JDBC Features:-

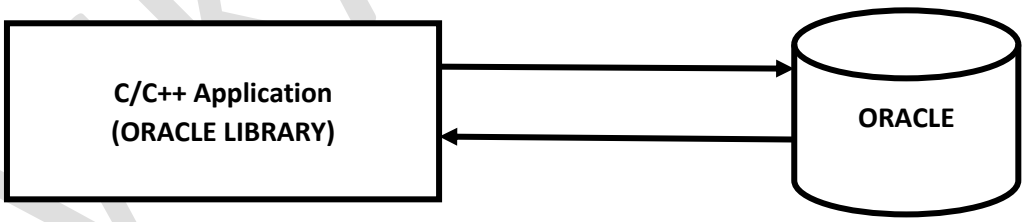
1. JDBC is a standard API, with the help of JDBC API we can communicate with any database without rewrite our application. Therefore, java JDBC API is database independent.
2. JDBC API developed in java and JDBC API applicable for all platform. So JDBC concept is platform independent API.
3. By using JDBC API we can perform all types of structure query operation.

JDBC version:-

JDBC VERSION	JAVA VERSION
JDBC 1.2	JDK 1.1
JDBC 2.1	JDK 1.2
JDBC 3.0	JDK 1.4
JDBC 4.0	JAVA SE 6
JDBC 4.1	JAVA SE 7
JDBC 4.2	JAVA SE 8
JDBC 4.3	JAVA SE 9

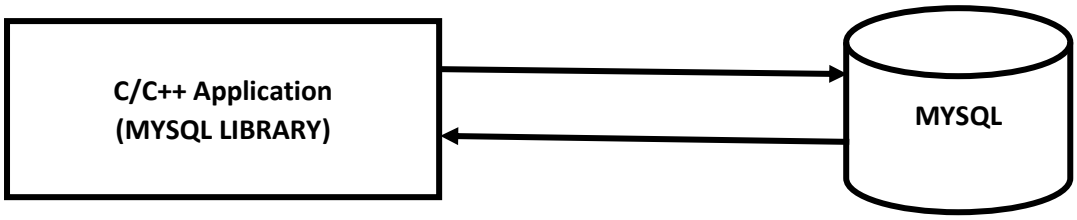
Evaluation of JDBC:-

- If we want communicate with database by using C/C++ application, then database specific libraries must be available in our application directory.

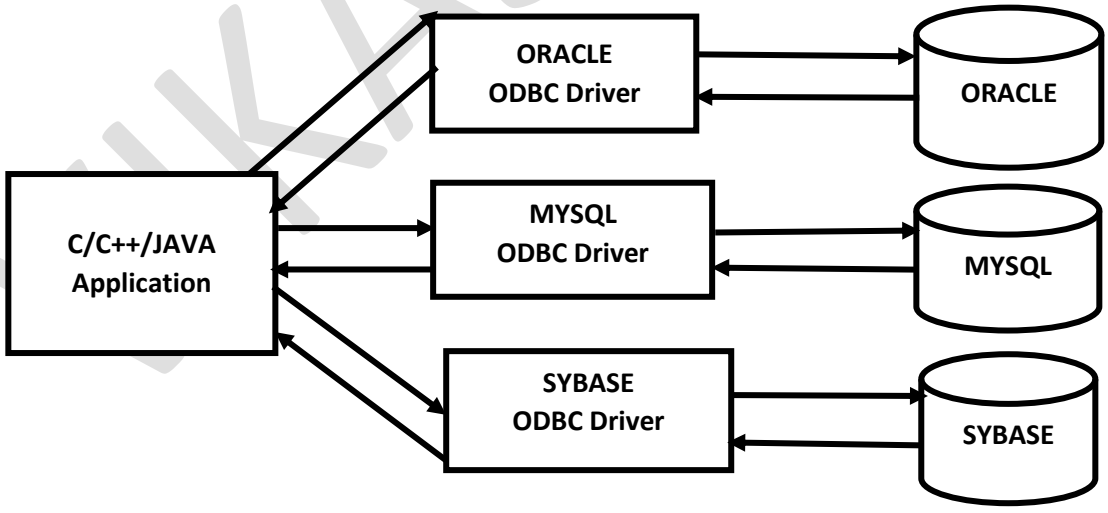


- In this diagram C/C++ application interact with ORACLE database then ORACLE database specific library must be available in application directory.
- Suppose same application interact with other database like MySql then MySql specific library must be available in application directory.

So, problem with this approach is, if we migrate one database to another database then we have to write entire application once again by using new database specific library.



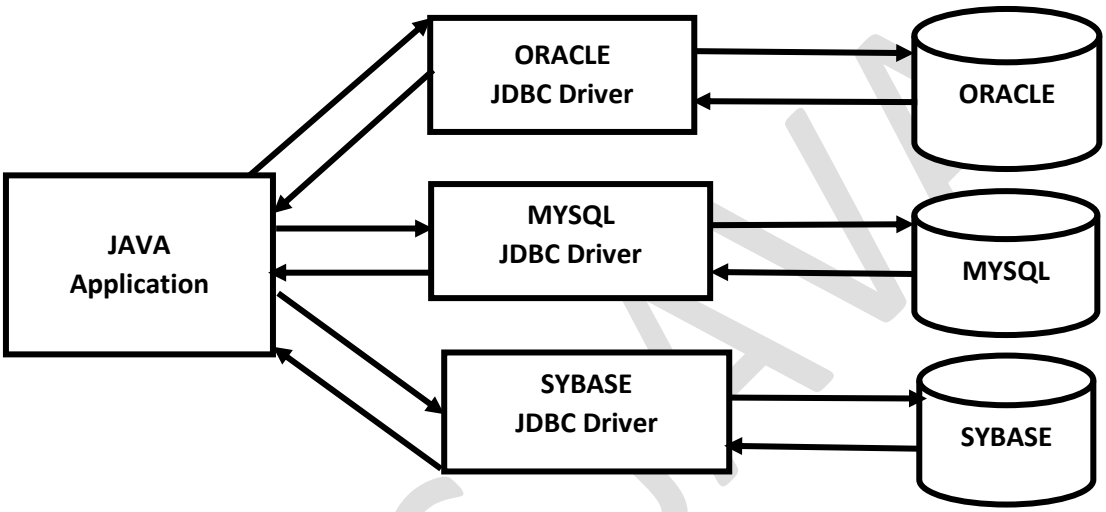
- This approach makes our application database dependent.
- Overcome these problem Microsoft introduce “ODBC”(Open Database Connectivity) concept in 1992.
- It is database independent API.
- In this approach, application can interact with any database just by selecting corresponding ODBC driver and we have not required use database specific library in application directory; Hence, our application is database independent.



Limitation with ODBC:-

- ODBC is platform dependent and it work only on window machine.
- ODBC driver are implemented in C language, if we used ODBC driver with java application, then application performance will go down because internal conversion required java to C and C to java.

- Overcome these problems sun-micro-system introduce JDBC concept in 19th feb 1997.
- JDBC concept application for any platform and it is platform independent technology.
- JDBC driver are implemented in java, if we used JDBC driver with java application then internal conversion is not required and performance of the application will not go down.



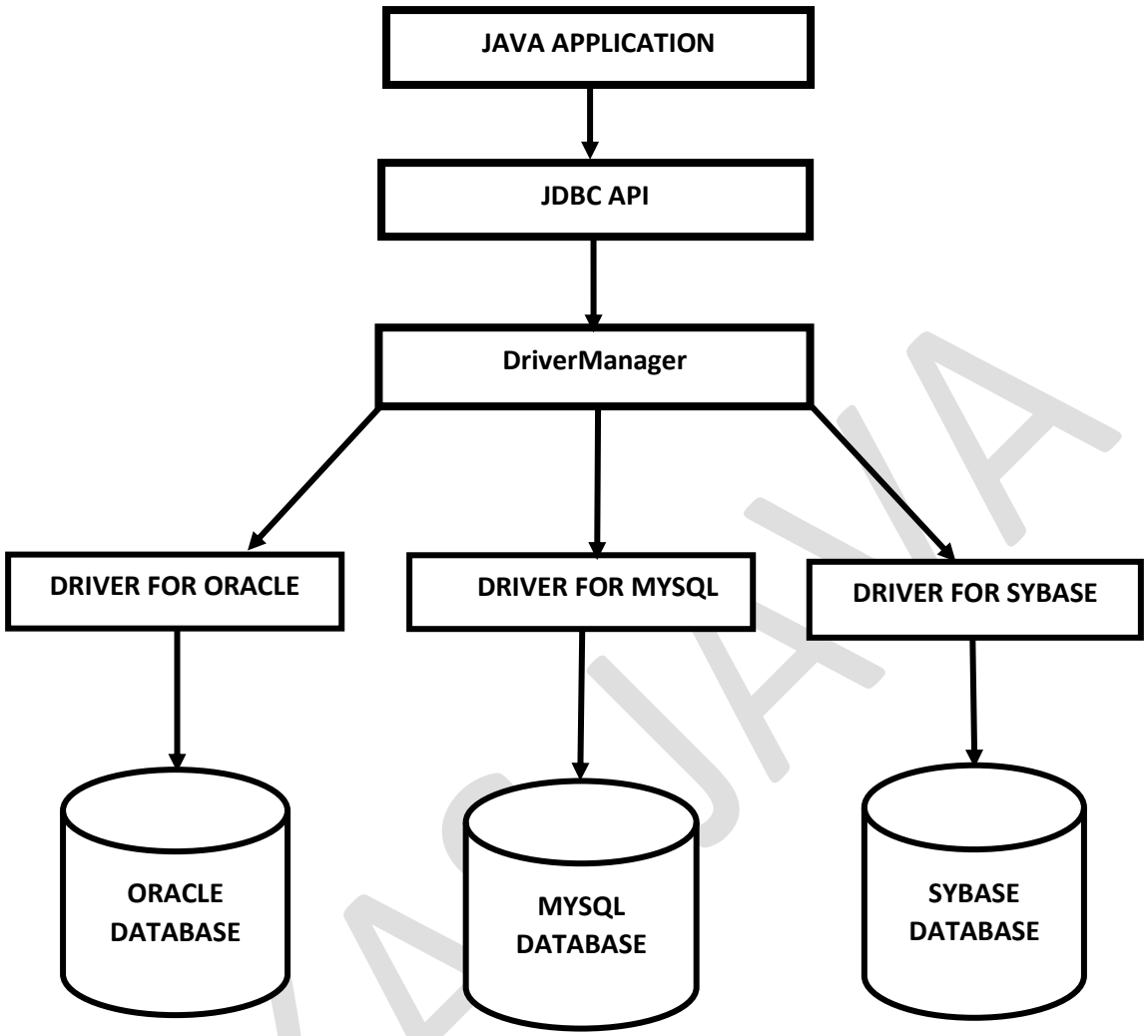
Note:- ODBC Concept is applicable for any Database and for any Language, but only for Windows Platform.

Note:-JDBC Concept is Applicable for any Platform and for any Database, but only for Java Language.

Difference between ODBC and JDBC:-

ODBC	**JDBC**
1).ODBC full name Open Database Connectivity.	1).JDBC there is no official full name but we called Java Database Connectivity.
2).ODBC is database independent and platform dependent applicable only for window machine .	2).JDBC is database independent and platform independent applicable for all machine.
3).We can use ODBC for any language like C, C++, Java etc	3).We can use JDBC only for java language.

JDBC Architecture:-



- JDBC API provide DriverManager to our Java Application.
- Java application can communicate with database with the help of DriverManager class and database specific Driver.

DriverManager:-

- DriverManager is a class present in java.sql package.
- DriverManager is very important component of JDBC architecture.
- DriverManager is responsible to manage all types database driver available in our System.
- DriverManager is responsible to register and unregister database driver.
  DriverManager.registerDriver(driver);
  DriverManager.unregisterDriver(driver);

- DriverManager is responsible to establish connection between java application and database with the help of database specific driver.
  Connection con=DriverManager.getConnection(String url, String uname, String upwd);

Database Driver:-

- Driver is a software which available in from of jar file, which is part of database.
- It is collection of implementation classes of various interface present in JDBC API.
- Database driver is very important component of JDBC Architecture, without driver software we can't touch database.
- Database driver work as translator/bridge between java application and database, driver convert java specific call to database specific call and database specific call to java specific call.

Note:-Java is platform independent and JVM is platform dependent. Similarly, Java application is database independent and Driver software is database dependent.

JDBC API:-

- JDBC API is collection of classes and interface, by using this classes and interface in our java application we can communicate with database.
- Database vendor use JDBC API to developed Driver Software.
- JDBC API contains two packages.
  1. **java.sql package.**
  2. **javax.sql package.**

java.sql package:-This package contains basic classes and interface which can be used in almost all JDBC application.

Classes present in java.sql package:-

1. **DriverManager**
2. **Date**
3. **Time**
4. **TimeStamp**
5. **Types**
6. **SQLException**

Interface present in java.sql package:-

1. **Driver**
2. **Connection**
3. **Statement**
4. **PreparedStatement**
5. **CallableStatement**
6. **ResultSet**
7. **ResultSetMetaData**
8. **DataBaseMetaData**

javax.sql.package:-This package contains advanced level class and interface, which can be used in advance level JDBC application.

javax.sql package contains sub package also.

1. **javax.sql.rowset**
2. **javax.sql.rowset.serial**
3. **javax.sql.rowset.spi**

classes present in javax.sql package

1. **ConnectionEvent**
2. **StatementEvent**
3. **RowSetEvent...etc**

Interface present in javax.sql package.

1. **Savepoint**
2. **DataSource**
3. **RowSet**
4. **ConnectionEventListener**
5. **StatementEventListener**
6. **RowSetListener**

- JDBC API contain lots of classes and interface. Generally, as a programmer we are not responsible to provide implementation of these interface.
  But most of the time Database vendor is responsible to provide implementation of these interface as a part of Driver software. So, Driver software is collection of implementing class of JDBC API interface.

- Suppose java application communicate with ORACLE database with the help of oracle driver. So, oracle driver software is collection of implementing classes of JDBC API interface.
- Every software identifies by some special class name, similarly driver software is also identified by some special class name.
- Driver is a implementation class of Driver interface present in java.sql package.

Example:-
Type-1 driver or JDBC-ODBC bridge driver:-This driver class name is **sun.jdbc.odbc.JdbcOdbcDriver**
sun.jdbc.odbc:-package name.
JdbcOdbcDriver:-Driver class name.

Type-4 driver or pure java driver or thin driver:-This driver class name is **oracle.jdbc.OracleDriver**
oracle.jdbc:-package name.
OracleDriver:-Driver class name.

What is the difference between Driver interface, Driver class and Driver Software?

Driver interface:-
Driver interface act as requirement specification present in java.sql package and driver software are responsible to provide implementation of this interface.

Driver class:-
Driver class in a implementation class which provide the implements of Driver interface present in JDBC API of java.sql package, Driver class is present in Driver software.

Driver Software:-
Driver Software is very important component of JDBC architecture, without Driver Software we can't communicate with Database.

It is collection of implementing classes of various interface present in JDBC API.

Database driver work as translator or bridge between JDBC application and database, Driver software convert java call to Database specific call and Database specific call to java call.

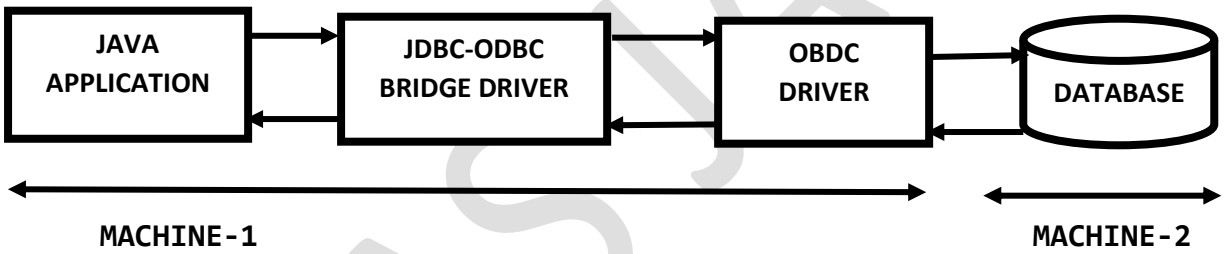
Driver software is present in form jar file like:-ojdbc14.jar, ojbc6.jar, mysql-connector.jar and it is the part of database.

Types of Driver Software:-

There are 180+ number of driver is available but based on database vendor, properties and characteristic all driver are divided into four parts:-

1. **Type-1 Driver**
2. **Type-2 Driver**
3. **Type-3 Driver**
4. **Type-4 Driver**

Type-1 Driver(JDBC-ODBC Bridge Driver OR Bridge Driver):-



- **Type-1** driver is also known as **JDBC-ODBC Bridge Driver or Bridge Driver**.
- Type-1 driver is developed by **sun-micro-system** as a part of JDK but this support is not available from **java 1.8** version.
- Type-1 driver is internally taking the support of Microsoft ODBC driver to communicate with Database.
- Type-1 driver convert java call to ODBC call and ODBC driver convert ODBC call into database call.
- Type-1 driver work as bridge between java application and ODBC driver.

Advantage of Type-1 driver:-

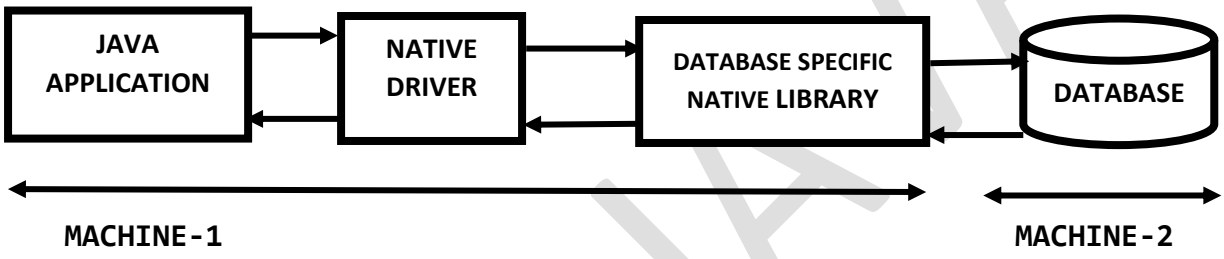
- Type-1 driver is easy to use and easy to maintain.
- When we use Type-1 driver to communicate with database, then we have not required to install any software because type-1 driver is part of JDK.

- This driver is not directly communicating with database, this driver communicates with database with the help of Microsoft ODBC driver, so it is database independent.

Disadvantage of Type-1 driver:-

- Type-1 is slowest driver because two level conversion required, java call to ODBC call and ODBC call to database call.
- It is less portable driver and platform dependent because ODBC driver is only on Microsoft system.
- This driver is not support from java 1.8 version.

Type-2 Driver(Native-Driver or Native-API-Partly-java driver):-



- Type-2 driver is also known as **Native-Driver or Native-API-Partly-java driver.**
- Type-2 driver is same as Type-1 driver just only one change in place ODBC driver we used DATABASE SPECIFIC NATIVE LIBRARY.
- Database Native Library is set of function written in non-java(like C/C++).
- Type-2 driver internally take the support of Database Specific native library to communicate with database.
- Type-2 driver convert java call to database specific native library call which is directly understandable to Database.

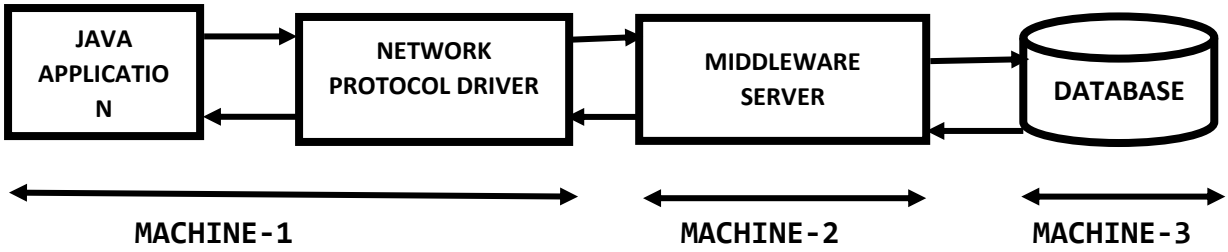
Advantage of Type-2 Driver:-

- Performance wise this driver is fast compare to type-1 driver because in this this only level conversion is required java call to database specific native library call.

Disadvantage of Type-2 driver:-

- This driver is Database dependent because database specific native library database dependent.
- This driver is also platform dependent because database specific native library developed in native language/non-java like C/C++.

Type-3 Driver(Middleware Driver or Network Protocol Driver or All Java Net Protocol Driver.):-



- Type-3 driver is also known as Middleware Driver or Network Protocol Driver or All Java Net Protocol Driver.
- Type-3 Driver internally take the support of Middleware server to communicate with Database.
- Type-3 Driver is converted java call Middleware server call and Middleware server can convert Middleware server specific call into database specific call.

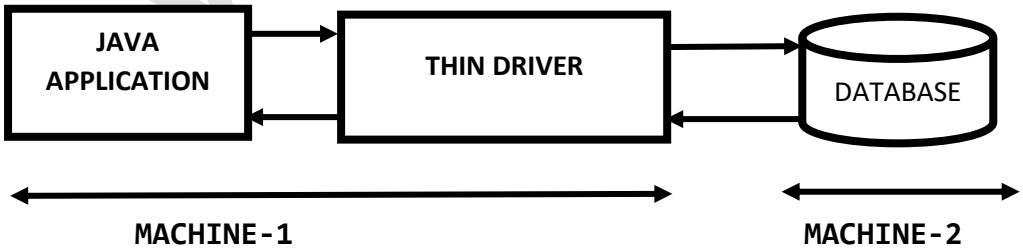
Advantage of Type-3 driver:-

- Type-3 driver is not directly communicating with database; hence it is database independent.
- This is also platform independent.
- This driver provides very good environment to communicate with multiple database.

Disadvantage of Type-3 driver:-

- Type-3 driver is dependent on Middleware server to communicate database and we have to need purchase Middleware server, so this is expensive way of communication to database.
  Example:-IDS Middleware server(Internet Database Access Server).

Type-4 Driver(Pure-Java-Driver or Thin-Driver or All-Java-Native-Protocol-Driver):-



- Type-4 driver is also known as Pure-Java-Driver or Thin-Driver or All-Java-Native-Protocol-Driver.

- Type-4 driver is directly communicating with database without help of any other component like ODBC or Database Specific native library or Middleware server, therefore this driver is known as thin driver.
- Type-4 driver is also known as Pure-Java-Driver because this driver is developed 100% in java.
- Type-4 driver is also known as All-Java-Native-Protocol-Driver because this driver uses database specific native protocol to communicate with database.

Advantage of Type-4 driver:-

- Type-4 driver is light-weight.
- It is platform independent.
- Type-4 driver is more secure because it is using database vendor specific native protocol.

Disadvantage of Type-4 driver:-

- It is database dependent driver.

Driver Software is provided by following vendors:-

1. **Java vendor:-**This types driver is provided by sun micro system itself like.
   JDBC-ODBC Bridge Driver

2. **Database vendor:-**This types driver is provided by database vendor like
   Oracle Database-------ojdbc6.jar
   Mysql Database--------mysql-connector.jar

3. **Third Party vendor:-**This types driver is provided by third party enterprise.
   Like:-
   Oracle Database---------Inet Oraxo
   Microsoft SQL Server----Inet Merlia.
   Sybase Database---------Inet Sybelux.

Note:-It is highly recommended to use Database vendor provided driver software.

There is multiple technology are used by database vendor to developed Driver software.

- If a Driver software is completely developed in java technology such types of Driver software are known as pure java driver.
- If a Driver software is developed in java and other technology like C,C++ then such type of Driver are known as Partial java Driver.

Uses of these drivers based on our requirement: -

Type-4 driver:-When we developed standalone application or practising JDBC application then it is highly recommended to use Type-4 driver.

Type-3 driver:-When we developed large application like enterprise application then it is highly recommended to use type-3 driver.

Type-2 driver:-If above two driver is not available then we used type-2 driver.

Type-1 driver:-If there is no driver available then only we should use type-1 driver.

Tier-Architecture of Driver:-Tier is count value, which defined how many System or Machine involved in the application development, If one system/machine is involved in an application development then we can say this application follows 1-Tier architecture or If two system involved then we say this application follows 2-tier architecture.
Similarly based on number of systems involved in JDBC communication driver divided into different tier.
Type-1, Type-2 and Type-4 driver is come under 2-Tier architecture.
Type-3 driver is come under 3-Tier Architecture.

Steps to developed JDBC Application:-

1. Load and Register the driver.
2. Establish the connection between Java Application and Database.
3. Prepare Statement object.
4. Write and execute SQL query.
5. Process Result from ResultSet.
6. Close the connection.

Load and Register the driver:-

- JDBC API contains set of interface and database vendor is responsible to provide implementation of these interface in the form of Driver software. So, Driver software should be available in our application for this we have to place driver software corresponding jar file in class path.
- Oracle Type-4 driver jar file name is ojdbc6.jar and we place this jar file in class path and this jar file is present in following directory.
  D:\app\vikas123\product\11.2.0\dbhome_1\jdbc\lib\ojdbc6.jar
- Once we place Driver jar file in class path now, we can load and register driver by using forName() method of Class class.

  Here we load and register oracle type-4 driver.
  oracle type-4 driver class name is:-"oracle.jdbc.OracleDriver".

  Example:-

```
Class.forName("oracle.jdbc.OracleDriver");
```

  When we load Driver class then loaded Driver class static block will be executed automatically.

  In above we loaded "OracleDriver" oracle database Driver class. So, "OracleDriver" class static block will be executed automatically.

```
public class OracleDriver
{
static
{
OracleDriver driver=new OracleDriver();
DriverManager.registerDriver(driver);
}
}
```

- Due to execution of this static block loaded driver automatically register with DriverManager class implicitly, hence we have not required to perform Driver registration explicitly.

- If we want to register driver explicitly with forName() method then we perform following task explicitly in java application.

```
OracleDriver driver=new OracleDriver();
```

DriverManager.registerDriver(driver);

- From JDBC 4.0 version(JAVA 1.6v) onward Driver call will load automatically from the class path and we have not need to load Driver class explicitly.

Establish the connection between Java Application and Database:-

- Once we load and register driver, now we required to establish connection to database by using **getConnection()** method of DriverManager class.
  public static Connection getConnection(String jdbcurl, String usename, String upwd)

  Example:-
  Connection con=
  DriverManager.getConnection(jdbcurl,username,upwd);

  Where jdbcurl represent location or address of database.

  Where username represent username of database.

  Where upwd represent password of database.

  JDBC URL Syntax:-
  <main_protocol>:<sub_protocol>:<sub_name>

  <main_protocol>:-it always jdbc.

  <sub_protocol>:--Driver protocol odbc,oracle,mysql.

  <sub_name>:------Driver connection details.

  JDBC url:- jdbc:oracle:thin:@localhost:1521:XE
  main_protocol:-jdbc
  driver protocol:-oracle.
  Driver connection Details:- thin:@localhost:1521:XE

  Driver connection Details contains four properties:-

  1. Driver name----------thin
  2. Machine on which database is running---@localhost

3. Port number of database----------1521
4. System ID----------XE

How to get Connection class object because Connection is an interface of JDBC API:-
We are not getting Connection interface object we getting implementation class object which is part of Driver Software.

```
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:or
cl","SYSTEM","vikas123");
System.out.println(con.getClass().getName());
```

In this program we try to get con variable class name then we result:- **oracle.jdbc.driver.T4CConnection/OracleConnection**

If we use implementation class name in place of Connection interface like:-

```
OracleConnection
con=(OracleConnection)DriverManager.getConnection("jdbc:oracle:thin:
@localhost:1521:orcl","SYSTEM","vikas123");
System.out.println(con.getClass().getName());
```

Result:- oracle.jdbc.driver.T4CConnection/OracleConnection
This is hardcoded and it applicable only for oracle type-4 driver we can't change the driver URL name, if we want try to change the driver name then we write entire program once again. Overcome this problem we used Connection interface in place of implementation class.

Create Statement object:-

- After established the connection between java application and database, some component/vehicle must require through which we send SQL query to the database.
- Statement object is that component/vehicle through we send SQL query to the database.
- Create Statement object by using createStatement() method of Connection interface.
  public Statement createStatement()
  Statement st=con.createStatement();

Write and execute SQL query:-
Basically, SQL query divided into two section.

1. **Select Query Operation.**
2. **Non-Select Query Operation.**

Select Query Operation:-In this query operation we retrieve single or group of result in form of ResultSet object.

Example:-select * from emp;

Non-Select Query Operation:-In this query operation we performed update, insert and delete operation and get result numeric value that represents how many is row affected.
Example:-update emp set esal=esal+50000 where esal<100000;

Once we create/write Sql query now we can execute sql query by using following method of Statement interface.

1. **executeQuery()**
2. **executeUpdate()**
3. **execute()**

executeQuery() method:-By using this method we execute select query and get single or group of result in form ResultSet object.

public ResultSet executeQuery(String SelectSqlQuery)throws SQLException

Example:-ResultSet rs=st.executeQuery("select * from emp");

executeUpdate() method:-By using this method we execute non-select query and get result numeric value, that numeric value represent how many row affected.

public int executeUpdate(String NonSelectQuery)throws SQLException

Example:-
int count=st.executeUpdate("update emp set esal=esal+50000 where esal<100000");

execute() method:-By using this method we execute select-query and non-select-query.
If we don't know types of query at beginning and we get Sql query dynamically then we used execute() method.

public boolean execute()

This method returns boolean value if sql query is select then return true value and if Sql query non-select then return value is false.

Process Result from ResultSet:-When select query will execute we get result in form of ResultSet object.

ResultSet object is hold single or group of Result.

ResultSet is a cursor which always locate before first record.

To check whether next record is available or not by using next() method of ResultSet interface.

public boolean next()

If next record is available then we get record from ResultSet object by using following method of ResultSet interface.

1. **getXXX(String column_name)**
2. **getXXX(int column_index)**

Where XXX value may Int, Double, Object....etc.

```
while(rs.next())
{
System.out.println(rs.getInt(1));
}
```

Note:-It is highly recommended to use column index value in **getXXX()** method because comparing number is very easy then comparing Strig value.

close the connection:-Whatever resources we have open in database operation, it is highly recommended to close in reverse order of opening by using close() method.

public void close()

rs.close():-After close ResultSet we not allowed further processing of ResultSet.

st.close():-After close statement then not allowed sending further query to database.

con.close():-After close Connection then not allowed further communication to database.

Example:-1 Create a table in oracle database by using type-4 driver.
TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC
{
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
//Step-1 load and register driver.
Class.forName("oracle.jdbc.OracleDriver");
//Step-2 Establish the connection between Java Application and
Database.
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
//Step-3 Prepare Statement object.
Statement statement=connection.createStatement();
//Step-4 Write SQL query
String sqlquery="create table emp(eid number(5),ename
varchar2(10),esal float(10),eaddr varchar2(10))";
//Step-5 Execute SQL query
statement.executeUpdate(sqlquery);
System.out.println("Table Created Successfully");
//close connection
statement.close();
connection.close();
}
}
```

Steps to Execute First JDBC program on command prompt:-
Step:-1

Install Database Software based on our requirement like oracle mysql...etc.
For above JDBC program execution we install oracle11g database.

Step:-2 To find required Driver jar file in following directory
D:\app\vikas123\product\11.2.0\dbhome_1\jdbc\lib\ojdbc6.jar
and place oracle6.jar file in class path by following command on command prompt.
set
classpath=D:\app\vikas123\product\11.2.0\dbhome_1\jdbc\lib\ojdbc6.jar;.;

Step:-3 Compile and run JDBC program by using following command.
javac TestJDBC.java
java TestJDBC

Create a table in Oracle Database by using type-1 Driver:-

- We are not creating table in Oracle Database by using type-1 driver because type-1 driver is deprecated and not provide support from JDK-8.
- Microsoft Window Operating System 8, 10, 11 not provide proper support of type-1 driver.

Create a table in Oracle Database by using type-2 Driver:-

- Type-2 driver for oracle name is **OCI driver** (ORACLE CALL INTERFACE) OCI Driver internally use database native library. OCI Driver present in oracle database in following jar file.
  Oracle14.jar------Oracle 10g(internally oracle uses JDK 1.4 v)
  Oracle6.jar-------Oracle 11g(internally oracle uses JDK 6 v)
  Oracle7.jar-------Oracle 12c(internally oracle uses JDK 7 v)

- To keep oracle6.jar in class path then driver software is available to our program.
- Load and Register driver for this we required driver class name and Oracle type-2 driver class name is **oracle.jdbc.driver.OracleDriver or oracle.jdbc.OracleDriver.**
- To establish connection between java application to Database we required JDBC URL.
  JDBC URL:- jdbc:oracle:oci8:@orcl(Oracle Version 8)

jdbc:oracle:oci:@orcl(Oracle Version 9)

Main-protocol:-jdbc
Sub-protocol:-oracle
Driver-Connection-Details:- oci:@orcl
oci is database specific native library name.
orcl is System ID
orcl:-orcl System ID is unique ID for every database, system id is varies from database to database or database version to version.

To find System id there are two ways:-

1. Execute following SQL command on SQL command prompt.
   select * from global_name;
2. Go in following directory:-
   D:\app\vikas123\product\11.2.0\dbhome_1\NETWORK\ADMIN\tnsnames.ora
   And find SERVICE_NAME name value like:(SERVICE_NAME = orcl)

Example:2
TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
//Step-1 load and register driver.
Class.forName("oracle.jdbc.OracleDriver");
//Step-2 Establish the connection between Java Application and Database.
Connection
connection=DriverManager.getConnection("jdbc:oracle:oci:@orcl","SYSTEM","vikas123");
//Step-3 Prepare Statement object.
Statement statement=connection.createStatement();
//Step-4 Write SQL query
String sqlquery="create table emp(eid number(5),ename
varchar2(10),esal float(10),eaddr varchar2(10))";
//Step-5 Execute SQL query
```

```
statement.executeUpdate(sqlquery);
System.out.println("Table Created Successfully");
//close connection
statement.close();
connection.close();
}
}
```

Create a table in Oracle Database by using type-3 Driver:-

Create a table in Oracle Database by using type-4 Driver:-

A java application communicates with oracle database by using type-4 driver three components required.

1. Driver Jar file.
2. Driver class name.
3. JDBC url.

Driver jar file:-Same driver jar file we will used that jar file we had uses in type-2 driver.
Jar file is jdbc14.jar, jdbc6.jar, jdbc7.jar.

To keep oracle6.jar in class path then driver software is available to our program.
Driver class name:- oracle.jdbc.driver.OracleDriver or oracle.jdbc.OracleDriver same class of type-2 driver.

JDBC URL:-"jdbc:oracle:thin:@localhost:1521:orcl".

Example:-3

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
//Step-1 load and register driver.
Class.forName("oracle.jdbc.OracleDriver");
```

//Step-2 Establish the connection between Java Application and Database.

```
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
//Step-3 Prepare Statement object.
Statement statement=connection.createStatement();
//Step-4 Write SQL query
String sqlquery="create table emp(eid number(5),ename
varchar2(10),esal float(10),eaddr varchar2(10))";
//Step-5 Execute SQL query
statement.executeUpdate(sqlquery);
System.out.println("Table Created Successfully");
//close connection
statement.close();
connection.close();
}
}
```

Steps to Developed JDBC Application by using ECLIPSE IDE(Integrated Development Environment).

1:-Installation Process:

1) download eclipse IDE(Integrated Development Environment).
2) extract eclipse zip file at the required location.
3) After unzip file, we will find "eclipse" folder.
4) Double click on eclipse icon.
5) Select workplace where we want to manage all application.

2.Prepare Java Project

1) Select File icon or right click on project area ".
2) Select "New"
3) select "Java Project"
4) Provide Project Name:"app1"
5) Click on "Next" button.
6) Click on "Finish" button.

3. Add Database Driver JAR file to Java Application

1) Right Click on "Project".
2) Select "Properties".
3) Select "java build path".
4) Select "Library".
5) Click on “Add External Jar…”.
6) Select the required Jar file.
7) Click on "OK" button.
8) Click on "Finish" button.
9) Click on "Apply" and "OK" button.

4. Prepare JDBC Application

1) Right click on src of Created Java Project.
2) Select "New" button.
3) Select "Class".
4) Provide package Name.
5) Provide Class Name.
6) Select main method if required.
7) Run JDBC by clicking on Run icon.

Example:-4 Insert record into database table.

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
int rowcount=statement.executeUpdate("insert into emp
values(1111,'abc',40000,'dli')");
System.out.println("Number of Record Inserted::"+rowcount);
statement.close();
connection.close();
}
}
```

Example:-5 Retrieve all record into database table.

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC
{
public static void main(String[] args)throws SQLException,
ClassNotFoundException
```

```
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
ResultSet resultSet=statement.executeQuery("select * from emp");
System.out.println("EID\tENAME\tESAL\tEADDR");
while(resultSet.next())
{
System.out.print(resultSet.getInt(1)+"\t"+resultSet.getString(2)+"\t
"+resultSet.getFloat(3)+"\t"+resultSet.getString(4));
System.out.println();
}
statement.close();
connection.close();
}
}
```

Execute NON-SELECT sql query execute by executeQuery() method:-We try to execute non-select query by executeQuery() method like **ResultSet resultSet=statement.executeQuery("insert into emp values(1111,'abc',60000,'dli')");**

Then we can't expect what the exact result, it varies from driver to driver, if driver is type-4, we get empty ResultSet object and perform update operation in respective database table, when we try to retrieve result from empty ResultSet object then we will get exception.

Example:-5 In this JDBC application we insert the record into database table on executeQuery() method.
TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC
{
public static void main(String[] args)throws SQLException,
ClassNotFoundException
```

```
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
ResultSet resultSet=statement.executeQuery("insert into emp
values(5678,'fghr',567000,'gulf')");
System.out.println("EID\tENAME\tESAL\tEADDR");
while(resultSet.next())
{
System.out.print(resultSet.getInt(1)+"\t"+resultSet.getString(2)+"\t
"+resultSet.getFloat(3)+"\t"+resultSet.getString(4));
System.out.println();
}
statement.close();
connection.close();
}
}
```

Execute SELECT query by executeUpdate() method:-We try to execute select query by executeUpdate() method like-

```
int rowcount=statement.executeUpdate("select * from emp");
```

Then we can't expect what the exact result, it varies from driver to driver if driver is type-4, then we will get result number of rows selected.

Example:-6 In this JDBC application we perform select operation on executeUpdate() method.
TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
```

```
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
int rowcount=statement.executeUpdate("select * from emp");
System.out.println(rowcount);
statement.close();
connection.close();
}
}
```

Execute DDL queries by executeUpdate() method:- We try to execute DDL queries like create table, drop table, alter table etc by executeUpdate() method then we get return integer value and return value is varies from driver to driver if driver is type-4 then then we get return values is 0.

Example:-7 Create table by using executeUpdate().
TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
int rowcount=statement.executeUpdate("create table emp1(eid
number)");
System.out.println(rowcount);
statement.close();
connection.close();
}
}
```

Example:-8 In this JDBC example we perform drop a database table.
TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
int rowcount=statement.executeUpdate("drop table emp1");
if(rowcount==0)
{
System.out.println("Table deleted successfully");
}
statement.close();
connection.close();
}
}
```

Example:-9 In this JDBC application we Insert record into data base table dynamically.

TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
Scanner sc=new Scanner(System.in);
```

```
while(true)
{
System.out.println("Enter Employee ID");
int eid=sc.nextInt();
System.out.println("Enter Employee Name");
String ename=sc.next();
System.out.println("Enter Employee Salary");
float esal=sc.nextFloat();
System.out.println("Enter Employee Address");
String eaddr=sc.next();
statement.executeUpdate("insert into emp
values("+eid+",'"+ename+"','"+esal+"','"+eaddr+"')");
System.out.println("Record Inserted Successfully");
System.out.println("Do you want insert more record[YES/NO] ");
String option=sc.next();
if(option.equalsIgnoreCase("no"))
{
break;
}
}
sc.close();
statement.close();
connection.close();
}
}
```

Example:-9 In JDBC application we perform Update record in database table.

TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
```

```
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
int updatecount=statement.executeUpdate("update emp set
esal=esal+20000 where esal>60000");
System.out.println("Number of Row Updated::"+updatecount);
statement.close();
connection.close();
}
}
```

Example:-10 In JDBC application we perform update multiple record in database dynamically.

TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
Scanner sc=new Scanner(System.in);
while (true)
{
System.out.println("Enter Salary Increment");
float incsal=sc.nextFloat();
System.out.println("Enter Salary Range");
float salrange=sc.nextFloat();
String sqlquery=String.format("update emp set esal=esal+%f where
esal>%f",incsal,salrange);
int updatecount=statement.executeUpdate(sqlquery);
System.out.println("Do you want more updation[YES/NO]");
String option=sc.next();
if(option.equalsIgnoreCase("no"))
{
break;
}
System.out.println("Number of Row Updated::"+updatecount);
```

```
}
sc.close();
statement.close();
connection.close();
}
}
```

Example:-11 In this JDBC application we perform Delete Record from database table.

TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
int updatecount=statement.executeUpdate("delete from emp where
ename='amit'");
System.out.println("Number of Row Updated::"+updatecount);
statement.close();
connection.close();
}
}
```

Example:-12 In this JDBC application we delete multiple record from database table based on dynamic input.

TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
```

```
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
Scanner sc=new Scanner(System.in);
System.out.println("Enter Salary Range");
float salrange=sc.nextFloat();
String sqlquery="delete from emp where esal>'"+salrange+"'";
int updatecount=statement.executeUpdate(sqlquery);
System.out.println("Number of Row Updated::"+updatecount);
sc.close();
statement.close();
connection.close();
}
}
```

Retrieve record from database table:- Order of column and number of column of Database table not need to same as order of column and number of column of ResultSet it may be same or difference. So, we retrieve all or particular column from database table based on order of ResultSet not Database.

EMP table:-

EID	ENAME	ESAL	EADDR
1234	ABCD	70000	DLI
1235	EFGH	80000	NOI

Case:-1

```
ResultSet rs=statement.executeQuery("select * from emp");
```

- Column order of database table is EID,ENAME,ESAL,EADDR.
- Number of columns of database table is 4.
- Column order of ResultSet is EID,ENAME,ESAL,EADDR.
- Number of columns of ResultSet is 4.
- So, we retrieve record from database table base as following

```
while(rs.next())
{
System.out.print(rs.getInt(1)+"  "+rs.getString(2)+"
"+rs.getFloat(3)+" "+rs.getString(4));
}
```

Case:-2

```
ResultSet rs=statement.executeQuery("select ename, esal, eaddr, eid from emp");
```

- Column order of database table is EID,ENAME,ESAL,EADDR.
- Number of columns of database table is 4.
- Column order of ResultSet is ENAME,ESAL,EADDR, EID.
- Number of columns of ResultSet is 4.
- So, we retrieve record from database table-based column order of ResultSet as following.

```
while(rs.next())
{
System.out.print(rs.getString(1)+"  "+rs.getFloat(2)+"
"+rs.getString(3)+" "+rs.getInt(4));
}
```

Case:-3

```
ResultSet rs=statement.executeQuery("select ename, eid from emp");
```

- Column order of database table is EID,ENAME,ESAL,EADDR.
- Number of columns of database table is 4.
- Column order of ResultSet is ENAME, EID.
- Number of columns of ResultSet is 2.
- So, we retrieve record from database table based column order of ResultSet as following

```
while(rs.next())
{
System.out.print(rs.getString(1)+"  "+rs.getInt(2));
}
```

Example:-13 In this JDBC application we retrieve particular record from database table.

TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
```

```
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
String sqlquery="select ename,esal from emp";
System.out.println("ENAME\tESAL");
ResultSet rs=statement.executeQuery(sqlquery);
while(rs.next())
{
System.out.print(rs.getString(1)+"\t"+rs.getFloat(2));
System.out.println();
}
statement.close();
connection.close();
}
}
```

Example:-14 In JDBC application retrieve record from database table based on initial name of employee.
TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
Scanner sc=new Scanner(System.in);
System.out.println("Enter Initial character of Employee");
```

```
String intial_char=sc.next()+"%";
String sqlquery="select * from emp where ename like
'"+intial_char+"'";
boolean flag=false;
System.out.println("EID\tENAME\tESAL\tEADDR");
ResultSet rs=statement.executeQuery(sqlquery);
while(rs.next())
{
flag=true;
System.out.print(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(
3)+"\t"+rs.getString(4));
System.out.println();
}
if(flag==false)
{
System.out.println("Record is not available");
}
sc.close();
statement.close();
connection.close();
}
}
```

Example:-15 In this JDBC application we retrieve all record from database table based on ascending or descending order.
TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
String sqlquery="select * from emp order by esal ASC";
```

```
//for descending 'DESC' we can use.
System.out.println("EID\tENAME\tESAL\tEADDR");
ResultSet rs=statement.executeQuery(sqlquery);
while(rs.next())
{
System.out.print(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(
3)+"\t"+rs.getString(4));
System.out.println();
}
statement.close();
connection.close();
}
}
```

Aggregate Function:-In oracle database multiple aggregate function are available by using this aggregate function we get summary of result like.

count(*):-This aggregate function return number of records available in oracle database table.

max(String column_name):-This aggregate function return maximum value of a particular column.

min(String column_name):-This aggregate function return minimum value of a particular column.

Example:-16 In this application we use get the result based on Aggregate Function.

TestJDBC.java

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestJDBC {
public static void main(String[] args)throws SQLException,
ClassNotFoundException
{
Class.forName("oracle.jdbc.OracleDriver");
Connection
connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:
1521:orcl","SYSTEM","vikas123");
Statement statement=connection.createStatement();
```

```
//----------ueses count(*) aggregate function------------
String sqlquery1="select count(*) from emp";
ResultSet resultSet1=statement.executeQuery(sqlquery1);
if(resultSet1.next())
{
System.out.print("Total Number Of Record Aailable=
"+resultSet1.getInt(1));
System.out.println();
}
//-------------ueses max() aggregate function--------------
String sqlquery2="select * from emp where esal in(select max(esal)
from emp)";
ResultSet resultSet2=statement.executeQuery(sqlquery2);
if(resultSet2.next())
{
System.out.println("Highest Salary of Employee Information");
System.out.println("EID\tENAME\tESAL\tEADDR");
System.out.print(resultSet2.getInt(1)+"\t"+resultSet2.getString(2)+"
\t"+resultSet2.getFloat(3)+"\t"+resultSet2.getString(4));
System.out.println();
}
//-------------ueses min() aggregate function--------------
String sqlquery3="select * from emp where esal in(select min(esal)
from emp)";
ResultSet resultSet3=statement.executeQuery(sqlquery3);
if(resultSet3.next())
{
System.out.println("Lowest Salary of Employee Information");
System.out.println("EID\tENAME\tESAL\tEADDR");
System.out.print(resultSet3.getInt(1)+"\t"+resultSet3.getString(2)+"
\t"+resultSet3.getFloat(3)+"\t"+resultSet3.getString(4));
}
statement.close();
connection.close();
}
}
```