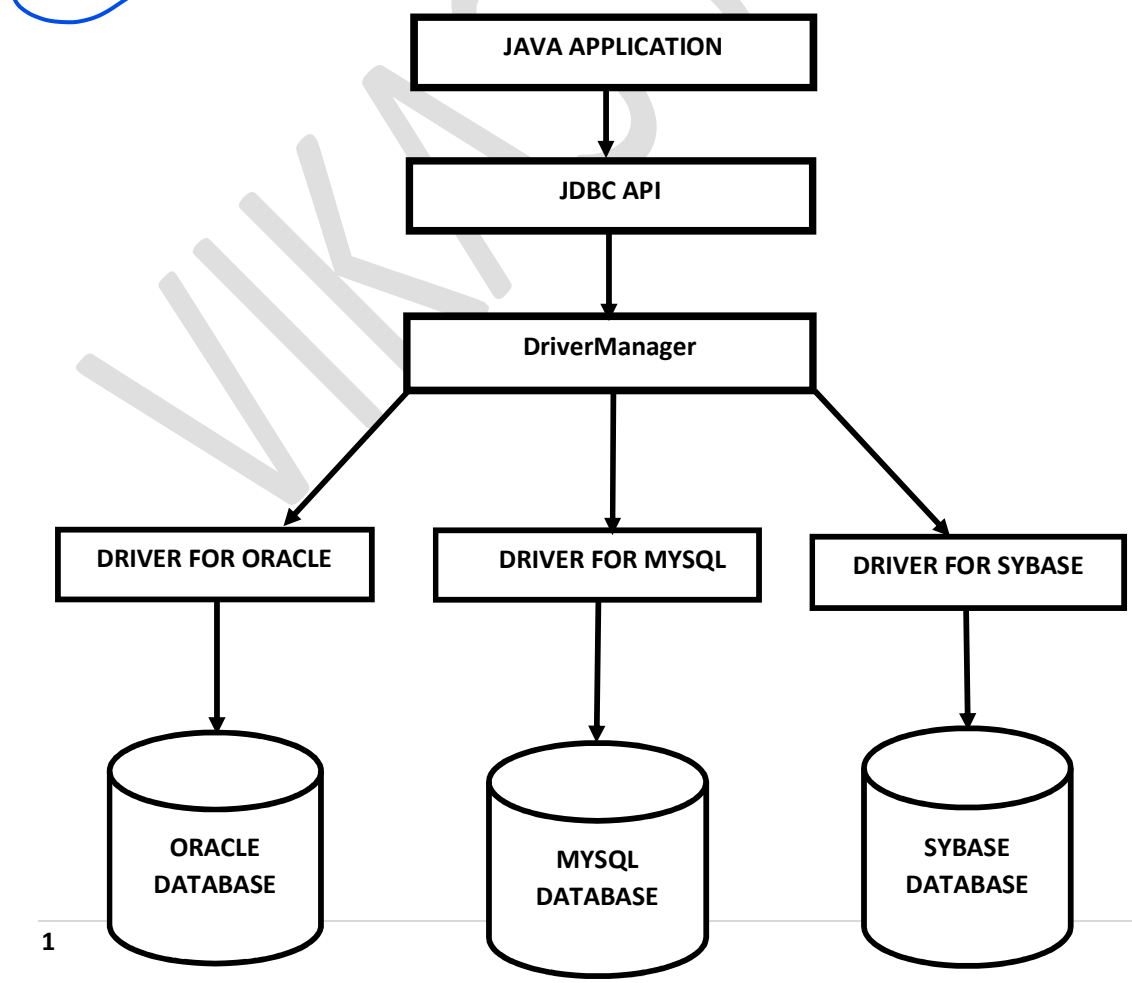
JDBC:-

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 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**

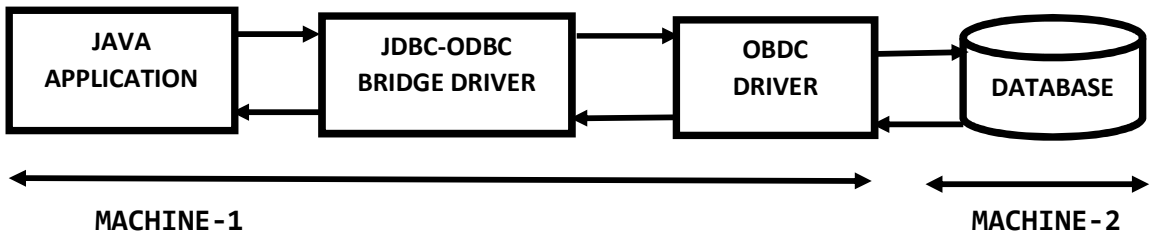
- JDBC API contain lots of classes and interface. Generally, as a programmer we are not responsible to provide implementation of these interface.But most of the time Database vendor is responsible to provide implementation of these interface as a part of Driver software. So, Driver software is collection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

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.

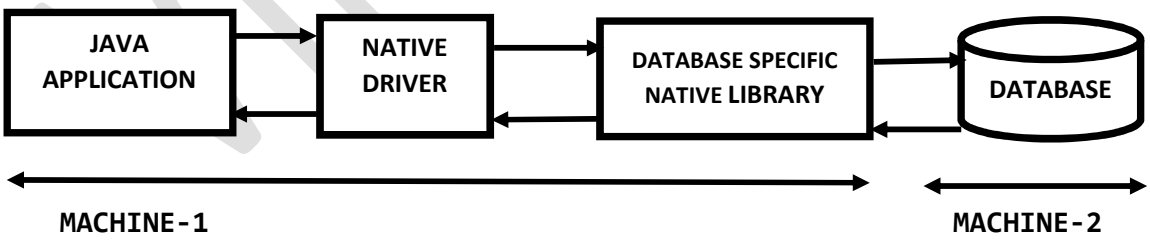
Advantage of Type-1 driver:-

- 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.

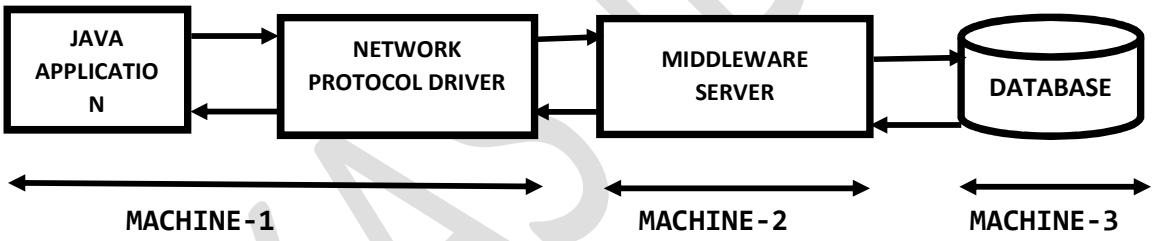
Advantage of Type-2 Driver:-

- 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.

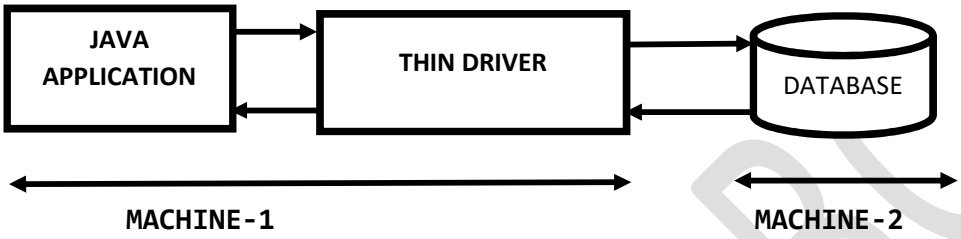
Advantage of Type-3 driver:-

- 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.

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.

Example:- 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.ja
r;.;
```

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

javac TestJDBC.java

java TestJDBC

Example:- 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:- 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();
}
}
```

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

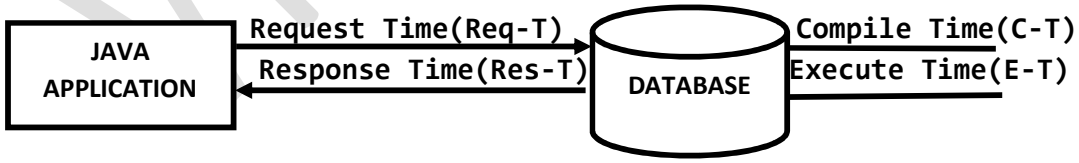
TestJDBC.java

```
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();
}
}
```

PreparedStatement(Interface):-

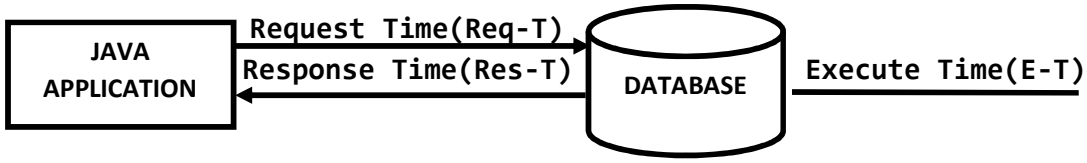
- It is child interface of Statement interface.
- Whenever we send SQL query request to database by Statement object then database engine will compile that SQL query every time for that SQL query execution, so it creates performance problem.



Total time for per query=(Req-T)+(C-T)+(E-T)+(Res-T)
1ns +1ns +1ns +1ns
Total time for 1000 Queries=4*1000ns=4000ns.

- Overcome this problem we used PreparedStatement whenever we send SQL query to database by PreparedStatement object then database engine will compile that SQL query only once even though we

execute that SQL query multiple times, so performance of the application will improve.



Total time for per query=(Req-T)+(E-T)+(Res-T)
1ns+1ns+1ns
Total time for 1000 Queries=3*1000ns=3000ns.

- Create PreparedStatement object by using prepareStatement() method of Connection interface.
  public PreparedStatement prepareStatement(String sql_query) throws SQLException
  Ex:-
  PreparedStatement pstatement = connection.prepareStatement(sql_query);

  When JVM encounter this line of the code JVM will pick provided sql_query and send to the database, database engine will compile sql_query and stored into database.

  This precompiled sql_query we get in the form of PreparedStatement object.

Steps to create JDBC application by using PreparedStatement:-

1. Create sql query.
2. Create PreparedStatement object with above SQL query.
3. Set Parameter values.
4. Execute SQL query.

Create sql query:-
There are two approaches to create SQL query.

A. SQL query without parameter.
B. SQL query with parameter.

SQL query without parameter:-In this SQL query we insert the value of columns directly.
Like:-
String sqlquery=”insert into emp values(1111,’ABC’,7000,’dli’)”;
This is also known as static query.

SQL query with parameter:-In this SQL query we insert value of the columns by parameter.
Like
String sqlquery=”insert into emp values(?,?,?,?)”;
? keyword represent positional parameter/IN parameter/place holder.

Create PreparedStatement object with above SQL query:-
PreparedStatement pstatement =connection.prepareStatement(sqlquery);

Set Parameter values:-
Once PreparedStatement object is created we set parameter of parameterized query by using corresponding setter method.
pstatement.setXXX(index_value,value);
pstatement.setInt(1,1111);
pstatement.setString(2,”ABC”);
pstatement.setFloat(3,70000);
pstatement.setString(4,”dli”);

Execute SQL query:-
We execute SQL query by using following methods based on SQL query.

a. **executeQuery()**
b. **executeUpdate()**
c. **execute()**

Example:-

```
package com.vikas.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
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");
//-------SQL query without parameter--------
String sqlquery1="insert into emp values(1240,'GYAN',100000,'BTI')";
PreparedStatement
pstatement1=connection.prepareStatement(sqlquery1);
int rowcount1=pstatement1.executeUpdate();
```

```
System.out.println("Number of Row Affected by This Updation=
"+rowcount1);
//-------SQL query with parameter--------
String sqlquery2="insert into emp values(?,?,?,?)";
PreparedStatement
pstatement2=connection.prepareStatement(sqlquery2);
pstatement2.setInt(1,1241);
pstatement2.setString(2,"rishi");
pstatement2.setFloat(3,55555);
pstatement2.setString(4,"gbz");
int rowcount2=pstatement2.executeUpdate();
System.out.println("Number of Row Affected by This Updation=
"+rowcount2);
pstatement1.close();
pstatement2.close();
connection.close();
}
}
```

Advantage of PreparedStatement:-

- Performance wise it is recommended.
- We are not required to provide input value at the beginning, we can provide dynamically so execute same SQL query with different set of values.
- We provide input value in java supported format not database specific format.
- Best suitable for insert operation.
- It prevents SQL injection attack.
- PreparedStatement we can used both without parameter type SQL queries and with parameter type SQL queries

Disadvantage of PreparedStatement:-

- A PreparedStatement object applicable only for one sql query which had we provided at the beginning but a Statement object is applicable for multiple sql query.