Chapter :- 8 Java Database Connectivity (JDBC)

JDBC

- Java Database connectivity is a JAVA API to connect and execute query with a database.
- It enables us to write java program to access any kind of tabular data stored in a Relational database.
- It is a part of JavaSE (Standard Edition)

JDBC enables us to :-

- Make connection to database.
- Create SQL and MySQL statement
- Execute queries in database.
- View and Modify results.

Java Database connectivity

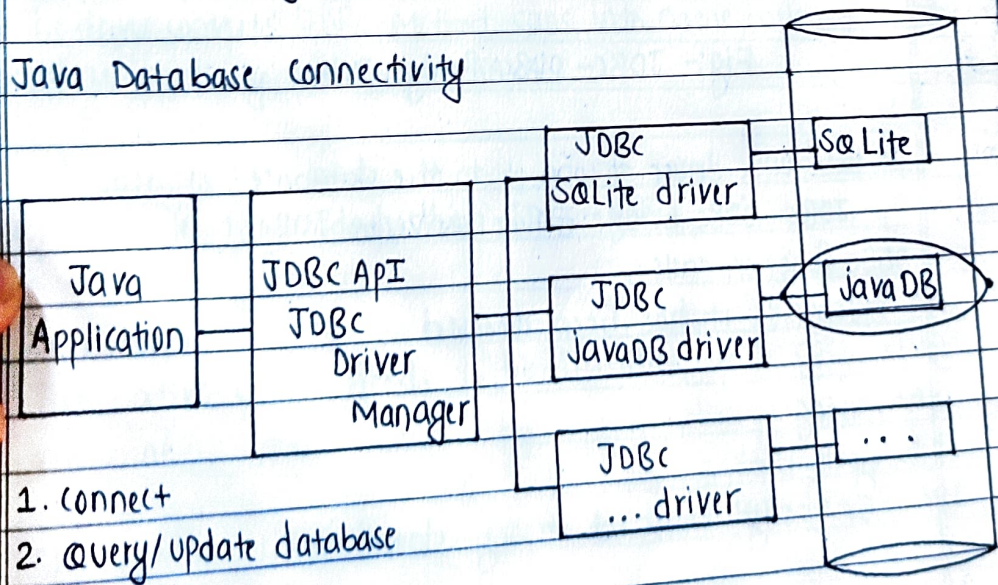


Fig:- JDBC

JDBC Drivers

i) JDBC - ODBC Bridge Driver
ii) Native Driver
iii) Network Protocol Driver
iv) Thin Driver

i) JDBC - ODBC Bridge Driver

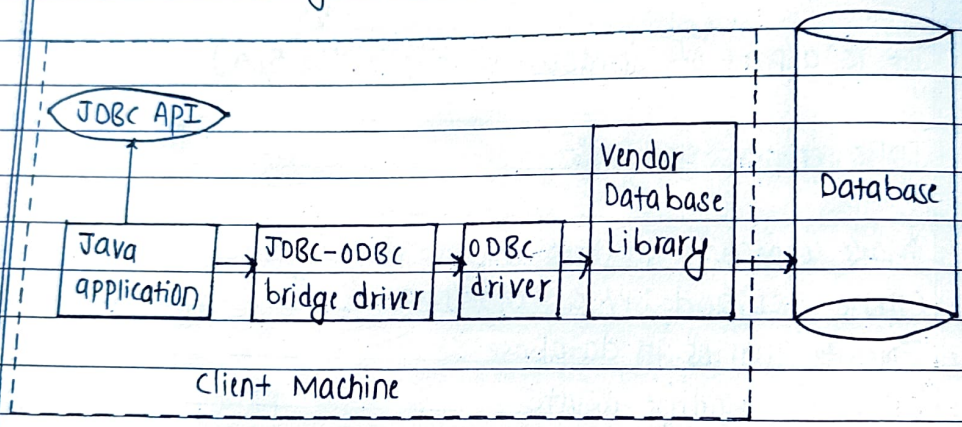


Fig:- JDBC - ODBC Bridge Driver

- It uses ODBC driver to connect to the database.
- The JDBC - ODBC bridge driver converts JDBC Method calls into ODBC function calls.
- In Java 8, it has been removed.

Advantages.
- Easy to use
- can be easily connected to any database.

Disadvantages
- Performance degrades because JDBC Method calls are converted into ODBC function calls.

- The ODBC driver needs to be installed on the client Machine.

ii) Native - API Driver

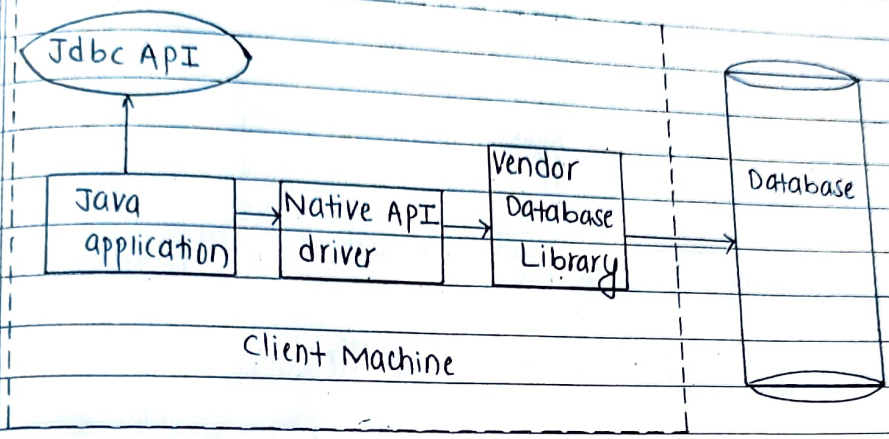


Fig:- Native - API Driver

- It uses client side libraries of the database
- The driver converts JDBC Method calls into native calls of database.
- It is partially written in JAVA.

Advantages

- performance upgraded than JDBC - ODBC

Disadvantages

- The native driver needs to be installed into each client Machine.

- The vendor client library Must be installed into client Machine.

~~18546~~ 10/20/81 Page 10

iii) Network protocol driver

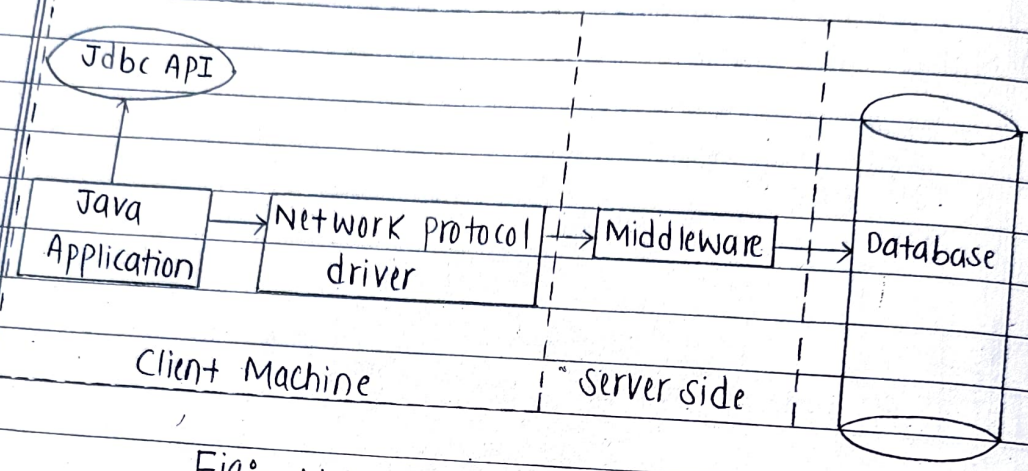


Fig:- Network protocol driver

- The Network protocol Driver uses Middleware that converts JDBC Method calls directly or indirectly into a vendor specifies database protocol.
- It is completely written in java.

Advantages

- 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 becomes costly.

iv) Thin driver

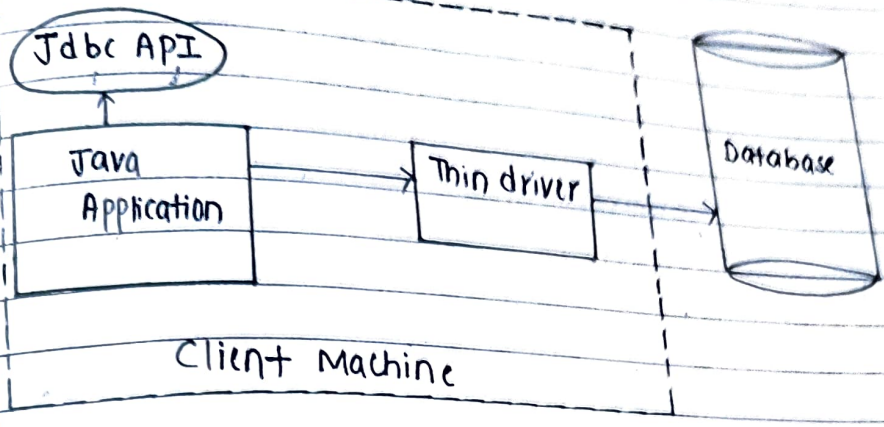


Fig :- Thin Driver.

- Thin Driver converts JDBC Method calls directly into vendor specific protocols.
- It is completely written in Java.

Advantages:-

- Best performance among all other drivers
- No software is required at client side or server side.

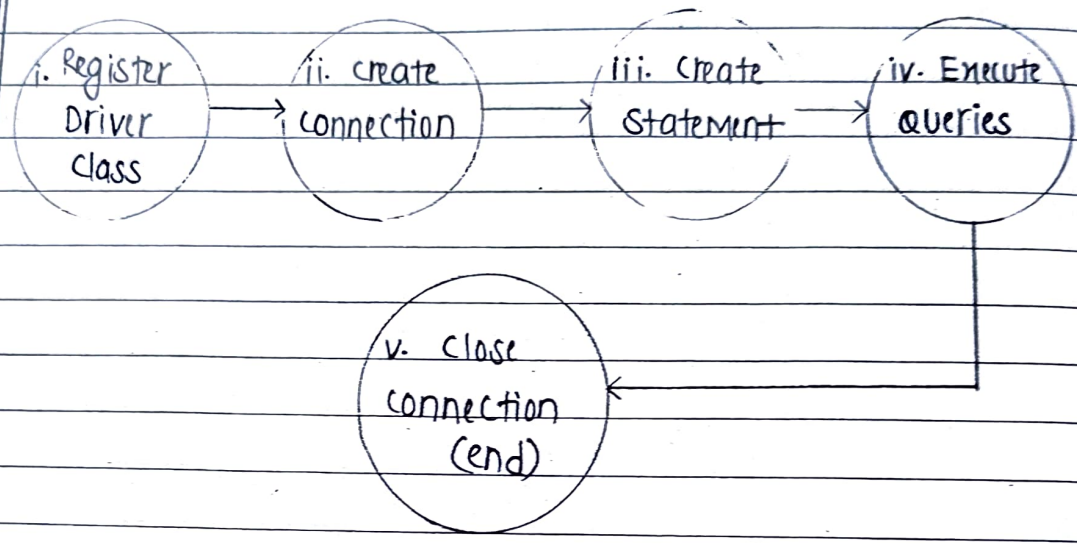
Disadvantages

- Driver depends on the database.

Define JDBC Driver.

- A JDBC driver is a software component enabling a java application to interact with a database.

Steps to connect to a Database.



S.N	Steps	code snippets
i)	Register Driver class	class.forName("com.mysql.jdbc.Driver")
ii)	Create connection	Connection conxn = DriverManager.getConnection("jdbc:mysql://localhost/dbaseName", "username", "pwd")
iii)	create statement	Statement stmt = con.createStatement()
iv)	Execute query	ResultSet rs = stmt.executeQuery("select * from table")
v)	close connection	con.close()

11. Types of SQL commands

1) Data Defination Language (DDL)
2) Data Manipulation Language (DML)
3) Data query Language (DQL)
4) Data control Language (DCL)
5) Transaction control Language (TCL)

1) (DDL) Data Defination Language
- To perform changes on the physical structure of any table in a database. Eg:- CREATE, DROP, ALTER, TRUNCATE

i) CREATE :- To create a table or database
eg:- CREATE table student

ii) ALTER :- Modify values in the table.
eg:- ALTER table student
ADD column roll_no int

iii) DROp :- To delete table from the database.
eg:- DROp table student.

iv) TRUNCATE :- To delete the data inside a table.
eg:- TRUNCATE table student.

2) (DML) Data Manipulation Language.

i) INSERT :- To insert rows in the table.
eg:- INSERT into student (roll, name) values (2, 'THOMAS')

ii) DELETE :- DELETE a row or the entire table.

UPDATE :- Update values of the existing rows of the table.
eg:- UPDATE Student SET name = 'Bob' WHERE roll=1.

3) Data query Language (DQL)

SELECT :- To fetch data from tables / database
eg:- SELECT * FROM student.

// To fetch and display all record from the database table.

```
import java.sql.*;

class Demo{

    public static void main(String[] args){

    String URL = "jdbc:mysql://localhost:3306/student";
    String username = "root";
    String pwd = "potpot";

try{

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

Connection conxn = DriverManager.getConnection(URL, username,
    pwd);

Statement stmt = conxn.createStatement();
```

```
ResultSet rs = stmt.executeQuery("SELECT * FROM
software WHERE roll<2");
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2)+
" "+rs.getString(3));
}
    conxn.close();
}catch(Exception e){
    System.out.println(e.getMessage());
}
}
}
```

// Change the salary to 100000 for all record with post Manager.

// update software set faculty = "BECE" where roll=1;

```
import java.sql.*;
public class UpdateDemo{
public static void main(String[] args){
try{
```

```
Class.forName("com.mysql.cj.jdbc.Drivers");

Connection conxn = DriverManager.getConnection("jdbc:
mysql://localhost:3306/student", "root", "abcabc");

Statement stmt = conxn.createStatement();

int r = stmt.executeUpdate("UPDATE Software SEt
faculty = 'BECE' WHERE roll=1");

System.out.println("no of rows affected=" +r);

    conxn.close();

} catch(Exception e){

    System.out.println(e.getMessage());
  }
}
}

// Dynamic approach using a prepraedStatement

import java.sql.*;

class PreparedStmt {

   public static void main(string[], args) {
try{
Class.forName("com.mysql.cj.jdbc.Driver");
```

```
Connection conxn = DriverManager.getConnection ("jdbc:
mysql://localhost:3306/student","root","abcabc");
preparedStatement stMt = conxn.prepareStatement ("
    INSERT into software values (?,?,?)");
    stMt.setInt (1,200);
    stMt.setString (2,"Thomas");
    stMt.setString (3,"BEIT");
  int i = stMt.executeUpdate();

  system.out.println ("Number of rows added="+i);

        conxn.close();
  } catch (Exception e) {

        system.out.println (e.getMessage());
  }
  }
  }
```

ResultSet

- A ResultSet object is a table of data representing a database result set, which is usually generated by executing a statement that queries the database.

- The object of Resultset maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

- A default ResultSet object is not updatable and has a cursor that moves forward only.

Old is gold solution of chapter-8

2013-Fall

6 a) What are the steps involved for Making a connection with a database. Write a java program to extract and display the information in console from java program to display from ABC table of MS-access with suitable values. The ABC table has AAA and BBB fields.

Ans:-

First part is from copy.

```
import java.sql.*;
(import java.sql.*;) ⇒//This is for JDBC.

class Example{
  public static void Main(String[], args){
try{  String dbase = "student.mdb";
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

String url= "jdbc:odbc:Driver={Microsoft Access
Driver (*.mdb)};
   DBQ="+database+"; DriverID=22; READONLY=true";

Connection c = DriverManager.getConnection(url);

Statement st= c.createStatement();

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

While(rs.next()){
```

```
System.out.println (rs.getString(1));
}
        c.close();
}catch (Exception e)
  {  system.out.println(e); }
  }
}
```

b) Differences between JDBC and ODBC.

JDBC	ODBC
• JDBC stands for java database connectivity.	• ODBC stands for open Database connectivity.
• Introduced by SUN Micro systems in 1997.	• Introduced by Microsoft in 1992.
• We can use JDBC only for java language.	• We can use ODBC for any language like C, C++, Java etc.
• We can use JDBC in any platform.	• We can use ODBC only in windows platform.
• JDBC is object oriented.	• ODBC is procedural.
• Code is easy to understand	• code is complex.
• The JDBC driver is a bridge between java application and database software.	• ODBC driver is a bridge between applications developed in any programming language and database software.

2013-spring

6a) Write a program to connect to a database using JDBC. Assume that database name is test Db and it has table named employee with 2 records.

Ans:-

```
import java.sql.*;

class Databases{

public static void main (string[], args) {
try{
      int nrows = 0;
class.forName ("com.mysql.jdbc.Driver");

connection conxn = DriverManager.getConnection ("jdbc:
      Mysql://localhost:3306/testDb"; "root",
                  "potpot");

Statement stmt = conxn.createStatement();

ResultSet rs = stmt.executeQuery ("select * from employee")

While (rs.next()) {
            nrows++;
System.out.println("no of rows" +nrows);
}   conxn.close();
}catch (Exception e)
  {
    System.out.println(e);
}
}
```

6b) Repeated

2014 fall

6a) Repeated

6b) Write the steps to insert data in following table.

Table name: Student

Column	Datatype
Id	Number
Name	Varchar
Roll	number

#// program

```
import java.sql.*;

Class Example{
public static void Main(String[], args){

try{
    Class.forName("com.mysql.jdbc.Driver");

Connection conxn = DriverManager.getConnection("jdbc:mysql://
localhost:3306/student", "root", "abcabc");

PreparedStatement stmt = conxn.PrepareStatement("INSERT
into student values(?,?,?)");

    stmt.setInt(1, 02);
    stmt.setString(2, "BEIT");
    stmt.setInt(3, 181546);

int i = stmt.executeUpdate();
```

```
System.out.println("No of rows =", +i);
    conxn.close();
}catch(Exception e)
{
    system.out.println(e.getMessage());
}
}
}
#}=> closing
```

2014- spring

6a) Repeated

6b) Write a simple program to connect ms access database and insert data in the table named "student" which have four feilds named "id", "name", "address", "DOB" and class.

Ans:-

```
import java.sql.*;

Class Example{
public static void Main(String[], args){
try{
    String database = "student.mdb";
class.forName("sun.jdbc.odbc.JdbcOdbcDriver"));
```

```
String url = "jdbc:odbc:Driver={Microsoft Access
Driver (*.mdb)};
    DBQ="+database+";DriverID=22;READONLY=False";

Connection conxn = DriverManager.getConnection(url);

PreparedStatement stMt = conxn.prepareStatement("Insert
into student values (?,?,?,?,?,?,?)");

    stMt.setInt(1,23);
    stMt.setString(2,"Thomas");
    stMt.setString(3,"KTM");
    stMt.SetInt(4,30);
    stMt.setInt(5,9);

// for date of birth  stMt.setDate(4, new Date
    (623445533000L));

// int i = stMt.executeQuery();

    int i = stMt.executeUpdate();

System.out.println("No of rows=",+i);

    conxn.close();
} catch (Exception e){

    System.out.println(e.getMessage());
}
}
}
```

2015 spring

6a) Difference between simple statements and prepared statement with suitable example.

Simple statement

- It is used when SQL query is to be executed only once.
- We cannot pass parameters at runtime.
- It is only used for DDL statements
- It's performance is very low.
- It is base interface
- It is used to execute normal SQL queries.
- We cannot use statement for reading binary data.
- No binary protocol is used for communication.

eg:-

```
// creating the statement object
Statement stmt = con.createStatement();
// Executing the statement
stmt.executeUpdate("CREATE TABLE student(
ID NUMBER NOT NULL, NAME VARCHAR)");
```

PreparedStatement

- It is used when sql queries is to be executed multiple times.
- We can pass parameters at runtime.
- Performance is better than simple statement.
- It extends statement interface.
- It is used to execute dynamic sql queries.
- It is used for any sql queries.
- We can use prepared statement for writing binary data.
- Binary protocol is used for communication.

eg:-

```
// creating the prepared statement object

PreparedStatement stmt = con.prepareStatement("update
 student set name=? where ID=?");

  //setting values

    stmt.setString(1, "Ram");
    stmt.setInt(2, 512);

  // Executing preparedstatement
    stmt.executeUpdate();
```

6b) Write a simple java program to connect database and read data in the table named "student" which have four feilds named "id", "name", "address", "DOB" and "class". Assuming the "id" field as simple number, display the data of the students with id less than 100.

```
import java.sql.*;

  Class Java{
public static void main(string[], args){
try{

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

connection conxn = DriverManager.getConnection("jdbc:MySQL:
// localhost:3306/student", "root", "abc35");

Statement stmt = conxn.createStatement();

ResultSet rs = stmt.executeQuery("select * from student
  where id<100");

While(rs.next()){
    system.out.println(rs.getInt(1)+" "+ rs.getstring(2)
+" "+ rs.getString(3)+" "+ rs.getInt(4));
 }

    conxn.close();
}catch(Exception e){

  System.out.println(e);}}}
```

2016-Fall

6a) What is ResultSetMetadata. provide a simple example to illustrate.

Ans:-

- The ResultSetMetaData provides information about the obtained Resultset object like, the number of columns, names of columns, datatypes of columns, name of the table etc...
- ResultSetMetaData interface is useful because it provides methods to get Metadata from the Resultset object.
- The following some Methods of ResultSetMetaData class are:-

i) getcolumnCount() :- It retrives the number of columns in the current resultset object.

ii) getColumnLabel() :- It retrives the suggested name of the column for use.

iii) getColumnName() :- Retrives the name of the column.

iv) getTableName() :- It retrives the name of the table.

- The getMetaData() Method of Resultset interface returns the object of ResultSetMetaData.
- SYNTAX:-

public ResultSetMetaData getMetaData() throws SQL Exception.

eg:-

```
import java.sql.*;

class Rsmd{

Public static void Main (String[], args){

try{
    class.forName ("com.MysqL.jdbc.Driver");

connection con = DriverManager.getConnection ("jdbc:MysqL:
//localhost:3306/student", "root" , "pwdsts");

PreparedstateMent stMt = con.PrepareStateMent ("select *
   from employee");

ResultSet rs = stMt.executeQuery();

 ResultSetMetaData rsmd = rs.getMetaData();

 system.out.println ("Total columns =" + rsmd.getColumn
  Count());

system.out.println ("column name of 1st column =" +
  rsmd.getColumnName(1));

system.out.println ("Table name =" + rsmd.getTableName());

    con.close();
}catch (Exception e){
    system.out.println(e.getMessage()); }

}
}
```

6b repeated.

2016 spring 6b repeated

2017 Fall 6b) Repeated

2017 spring

6a) Write a program to update data on following table.

Table : student

Column :	Name Type
id	number
name	varchar
age	number.

program

```
import java.sql.*;

Class Example {

public static void main (string[], args) {
try {

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

connection conxn = DriverManager.getConnection ("jdbc:mysql
://localhost:3306/students", "root", "pwdsst");

  Statement stmt = conxn.createStatement();

int r = stmt.eneluteUpdate ("update student set
  Column = 'address' WHERE id = 2");
```

```
System.out.println("no of rows affected = ", +r);
    conxn.close();
}catch(Exception e){
    system.out.println(e.getMessage());
  }
}
}
```

2018 Fall. Repeated

2018 - Spring

6a) What is the benefit of using prepared statement in java? What is JDBC database connection pool? How to setup in java?

⇒ Repeated.

6b) A database 'testdb' contains table 'employee' with some records having id, name, post, salary. Write a program to update the salary to 50,000 whose post is Manager.

As:- Ans

```
import java.sql.*;

class Demo{

public static void main(String[], args){
```

```
try {
    Class.forName("com.mysql.jdbc.Drivers");

Connection conxn = DriverManager.getConnection("jdbc:
 mysql://localhost/3306/testdb", "root", "pdyst");

Statement Stmt = conxn.createStatement();

int r = Stmt.executeUpdate("update employee set
 salary = 50000 where post = 'manager' ");

System.out.println("rows affected = ", +r);

    conxn.close();
} catch (Exception e)
    {
      System.out.println (e.getMessage());
    }
}
}
```

2019 – Fall

6a Repeated

6b) Write a program to display only those records whose salary is more than 25000 from a table that contains id, name, post and salary of some employee.

Ans:-

```
import java.sql.*;

class DEMO{
public static void main (String[], args) {
try{
    Class.forName("com.mysql.jdbc.drivers");

Connection conxn = DriverManager.getConnection("jdbc:mysql://
  localhost/3306/Database", "root" "pass35");


// Statement stmt = conxn.execute

Statement stmt = conxn.createStatement();

ResultSet rs = stmt.executeQuery("select * from
 employees where salary>25000 ");

While(rs.next()){

System.out.println(rs.getInt(1) + " " + rs.getString(2)
 + " "+ rs.getString(3) + " " + rs.getFloat(4));
   }

  conxn.close();
} catch(Exception e)
  {
    system.out.println(e.getMessage());
  }
}
}
```