Page No.:
Date: 14 3 202

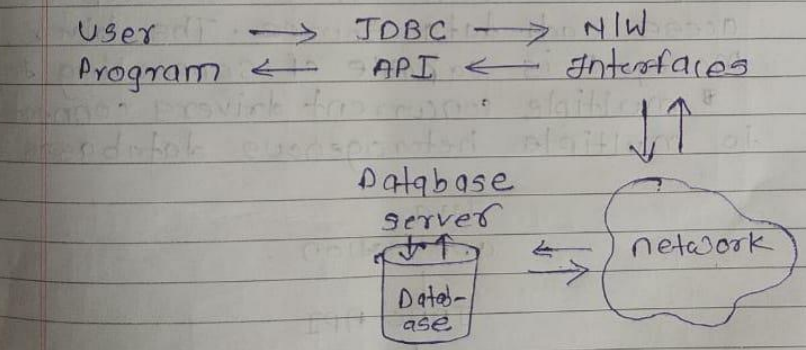
3. Database programming.

* The design of JDBC
* Types of drivers
* Executing SQL statements, ~~set~~ query execution
* Scrollable & updatable result set

JDBC

JDBC stands for Java database connectivity.

JDBC is standerd API for database independent connectivity bet^n the Java programming language & a databases such as postgresql, mysql, oracle, Access



JDBC Arechitecture :-

SM

JDBC API supports both two-tier & three-tier processing model for database acess

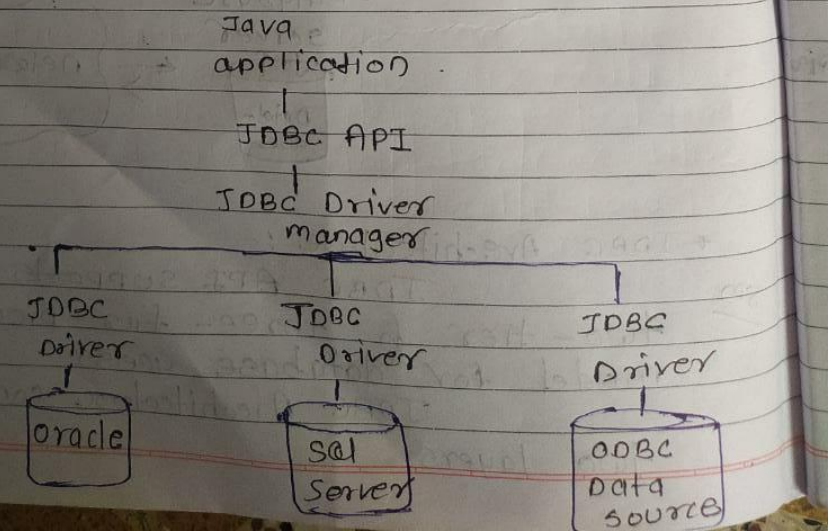
JDBC Arechitecture consist of two layers.

1) JDBC API :-
This provide application to JDBC manager connection.

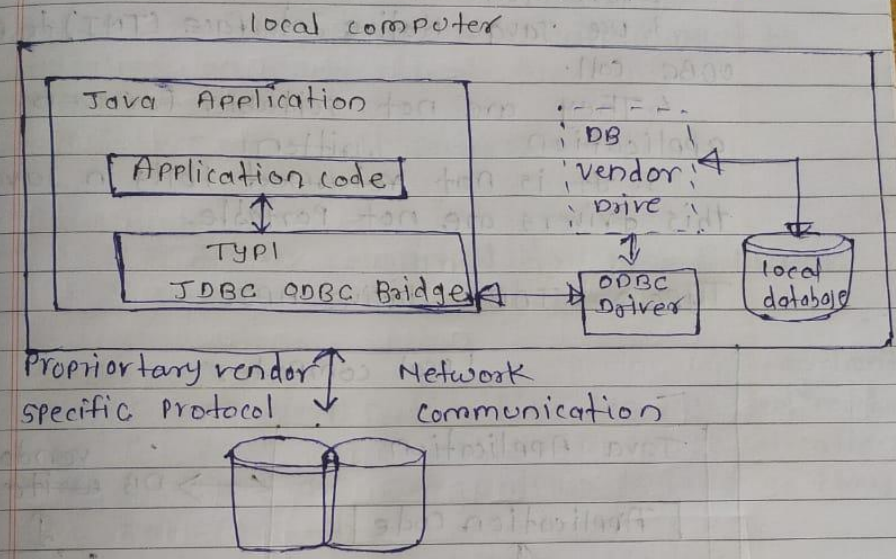
2) JDBC Driver API :-
This Support JDBC manager to driver connection.

3) JDBC API uses driver manager & database specific driver. to provide transparent connectivity to heterogenous database

4) The JDBC driver manager ensures that the correct driver is used to acess each data source. The driver manager is capable of sopporting to multiple concurrent drivers connected to multiple heterogenous databases



Type 1 : JDBC-ODBC Bridge driver.



In type 1 driver A JDBC Bridge is used to access ODBC driver installed on each client machine

using ODBC, requires configuring on your system a data source name (DSN) that represent target database.

Advantage

1. easy to use
2. allow easy connectivity to all database supported by DDBC driver.

JDBC drivers created by DBMS manufacture have to

i) establish A connection bet^n java & component
ii) Translate sql command into a form that can be understood by both dabase & java components.
iii) return any kind of error message that confirmed to JDBC driver ~~to the JDBC driver.~~
iv) It should be closed connection bet^n dbms & java components

JDBC drivers :-

JDBC driver is required to establish relationship bet^n application & database
It also helps sql process request & generating result
there are 4 type of JDBC drivers.

1. Type-1 Driver or JDBC-ODBC bridge
2. Type-2 Driver or Native API Partly Java driver
3. Type-3 Driver or Network Protocol driver
4. Type-4 Driver or Thin driver or Native Protocol ~~dr~~ pure Java Driver.

Page No.:
Date:

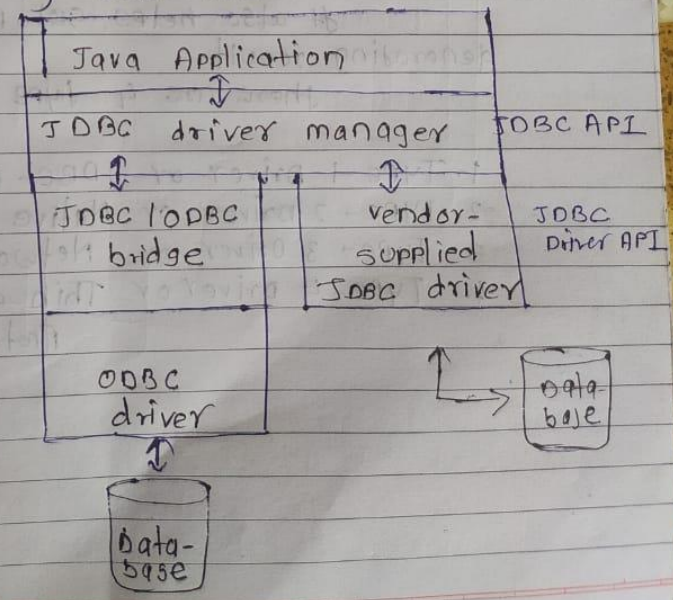
* Common JDBC components :-

① Driver manager
② Driver interface
③ Connection interface
④ Statement
⑤ Result set.

* Design of JDBC :-

There are many databases avilable like mysql, db2, postgresql, oracle etc.
Sun microsystem create JDBC drivers & JDBC API.

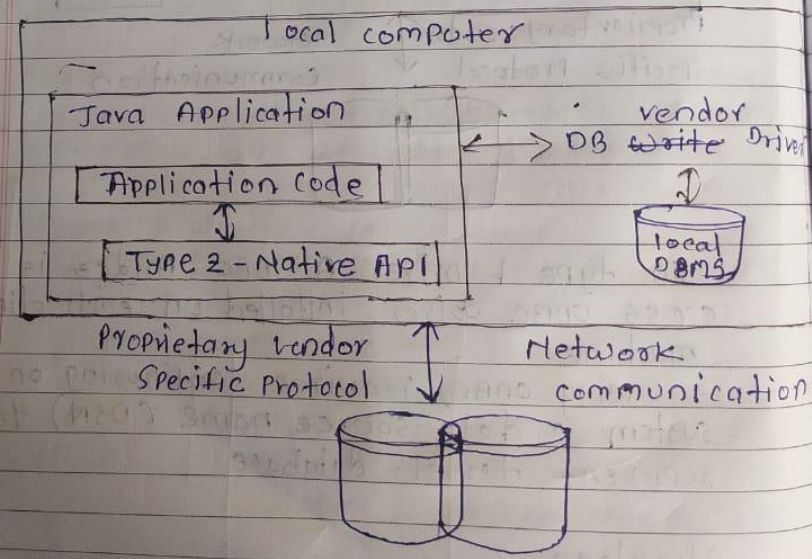
JDBC drivers are required. It should translate sql quiries into a language understood by JDBC API



Page No.:
Date.:

Disadvantages
1. Slow execution time
2. Dependent on ODBC Driver
3. Use Java Native Interface (JNI) to make ODBC call.
4. They are not suitable for web application
5. It is not ~~returned~~ written fully in Java, so this drivers are not portable.

Type 2 - JDBC - Native API



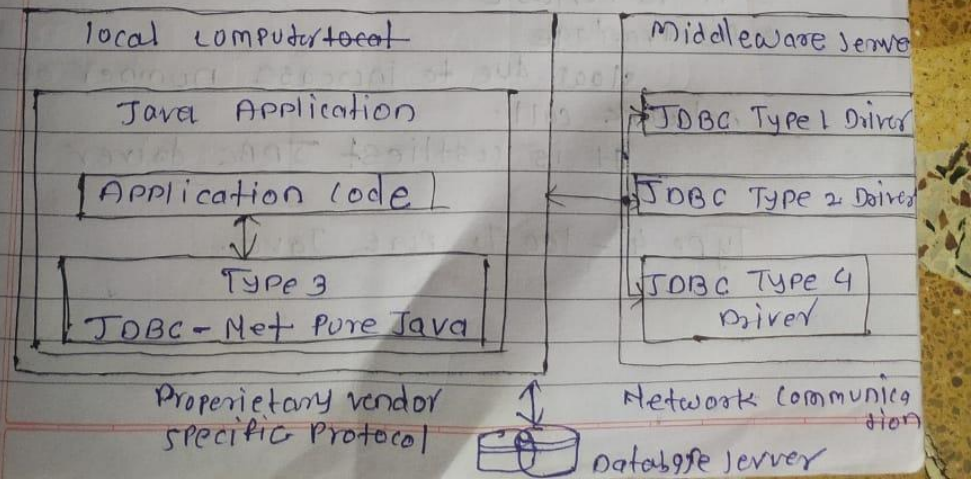
It also refferd as Native API Partially Java drivers
In type 2 driver JDBC API calls converted into native C, C++, API calls

which are unic tool the database
This drivers are typcially provided by Vender.
The vender specific drivers must be installed on each client machine.
eg
oracle call interface (OCI)

Advantages :-
1. faster as compared to Type-1 Driver.
2. contains additional features.

Disadvantages :-
1. requried native libaries installed on client machin
2. ~~JAS~~ not suitable for web application
3 They are not written in java so they are not Portable
4. Increased cost of application

Type 3- JDBC Net pure Java.

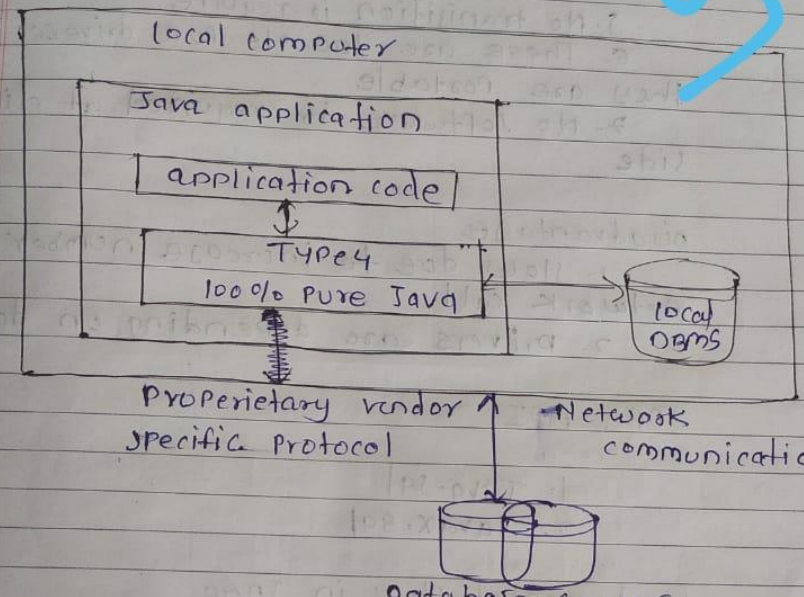


In type 3 drive a three ~~drie~~ to acess database
JDBAC client use stand network socket to communiate with middle-ware application server
The software into is translated by middleware application server into the call format required by dBms. & forwarded to database seever.
This kind of Driver is extermly flexibal

* Advantage
1. Does not require any native library to be installed on client
2. Database Independancy
3. Provid facility to switch over from 1 database to another database
4. They are suitable for Web application
5. This is Pure Java driver therfore it is Portable

* Disadvantage
slow due to increase number of network call.
It is costliest JDBC driver

Type 4 - 100% Pure Java.



In type 4- driver a pure Java based driver communicates directly with vendor database through socket connection

This is highest Performace driver apilable for database

e.g. MySQL collector / J driver is Type 4 driv

Advantage

Native

1. Does not required any library
2. Does not require any middleware server.
3. Best Performance than other driver
4. Better Performance than other driver

5. No tranisition is require
6. These are 100 olo pure drivers. Then they are portable
7. No software is required at client side

oisadvantage
1. slow due to increase number of network call
2. Drivers are depending on database

JDBC API

1. Java.sql
2. Javax.sql

fundamental steps in JDBC

1. import JDBC Package
2. load & register the JDBC driver.
3. open a connection to the database.
4. create a statement object to perform query
5. execute a statement object & return a query
6. Process the resultset
7. close the resultset & statement objects
8. close the connection.

```
import java.sql.*;
class Student
  {
    Public Static void main (String arg[])
    {
        connection con = null;
       Statement stmt = null;
       ResultSet rs = null;

     try
       {
         // load driver
        Class.forName ("org.postgresql.Driver");

        // establish a connection
con = DriverManager.getConnection ("jdbc:
postgresql://localhost/postgress","postgress","");
          //create a statement & execute queries

        stmt = con.createStatement ();
        rs = stmt.executeQuery ("select *
                          from student");
        while (rs.next())
        {
        System.out.println (rs.getInt(1)+"\t"
        + rs.getString ("sname")+"\t" +
          rs.getfloat ("per") );
        }
    // close resultset, Statement & connet
                                    object
          rs.close();
         stmt.close();
         con.close();
```

```
}//try
catch (Exception e)
{
    System.out.println(e);
}
}//main
}//class
```

1. Register JDBC Driver :-
   using Class.forName() :-

This is most common method to register a driver
This method is used to dynamically load drivers class file into memory. Which automatically registers it.

Syntax :-

```
try {
    class.forName("org.postgresql.Driver
    }
    catch (ClassNotFoundException ex)
    {
    System.out.println("Error: unable
to load driver class");
    System.exit(1);
}
```

2. DriverManager.registerDriver()
This is the second phase to register a driver
registerDriver() method is static method
Syntax :-

```
try {
Driver myDriver = new org.postgresql.Driver();
DriverManger.registerDriver (myDriver);
}
catch (ClassNotFoundException ex)
{
System.out.println ("Error: unable to
load driver class!");
System.exit()
}
```

3. Open connection to a database / establish Connection

DriverManager.getConnection() method is used to establish a connection
Syntax :-

```
getConnection (String url)
getConnection (String url, Properties Prop)
getConnection (String url, String user,
String password)
```

A database url is an address that points to a user database

RDMS	JDBC Driver name	URL format
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://hostname/databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@hostname:portNumber/databaseName
DB2	com.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:hostname:portNumber/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:hostname:portNumber/databaseName
Postgresql	~~com~~.org.postgresql.Driver	jdbc:postgresql://localhost/databasename

DriverManager.getConnection() to create a connection object.

e.g.

```
Connection conn = DriverManager.getConnection("jdbc:postgresql://localhost/temp",
                  "Postgres", "");
```

closing JDBC connections:-
to close the open connection, call close() method as follows.

```
conn.close();
```

4. Create a statement object to perform a query

the jdbc Statement, callableStatement, and PreparedStatement interfaces define the methods & properties that enables to send SQl command & received data form the database.

	Interface	Recommeded use
1.	Statement	- use this for general purpose acess to your database. usefull for static sql statement. The statement interface cannot accept parameter.

2. Creating Statement object

```
Statement Stmt = null;
try {
    Stmt = conn.crcateStatement()

}
catch (SQLException e)
{
}
```

finally
{
...
}.

after creating statement object to execute a sql statement with one of its 3 execute method

i). boolean execute (String SQL) :-
Returns a boolean value of true if a resultset object can be retrive otherwise its return false. use this method SQL, DDL stat.

ii) int executeUpdate (String SQL) :-
Returns the number of rows affected by the execution of the SQl statement.
e.g. INSERT, UPDATE, DELETE

iii) ResultSet executeQuery (String SQL) :-
Returns a ResultSet object. use this method when you expect to get result set as you would with a SELECT statement.

* closing Statement object
Stmt.close()

* Prepared_Statement
Use this when to use the sql statement many times
The Prepared Statement interface accept input parameter at runtime

PreparedStatement interface extends the ~~Statment~~ Statement interface

This Statement use the flexibility of supplying arguments dynamically

```
PreparedStatement pstmt = null;
    try {
        String SQL = "update empolyees SET
                      age = ? WHERE id = ?";

        Pstmt = conn. prepareStatement(SQL);
        ...
        }
        catch (SQLException e)
        {
        ...
        }
        Finally {
        ...
        }
```

all parameter in jdbc are resprssented by ? symbol which is known as parameter marker.

You must supply values for every parameter before executing the sql statemen

The setxxx() methods bind values to the parameters. where xxx represent the java datatype of the value to bind to the input parameter.

e.g setInt(), setFloat(), setString() etc

each parameter marker is reffered by its ordinal position (placed holder)
The first marker ~~returned~~ represented position, next position so on

~~The~~ for executing sql queries. Prepared Statement, object uses following method

execute(), executeQuery(), executeUpdate()

```
PreparedStatement pstmt = null;

try
 {
  String sql = "update Employee SET
               age = ? WHERE id = ?";

  Pstmt = conn.PrepareStatement(SQL);
  ...
 }
 catch (SQLException e)
  {
  ...
 }
 finally {
    Pstmt.close();
```