AGENDA

- INTRODUCTION TO JDBC
- JDBC API
- JDBC ARCHITECTURE
- JDBC DRIVER

Introduction to JDBC

- JDBC stands for java database connectivity. JDBC is a java API to connect and execute the query with the database. It is a part of javase (java standard edition). JDBC API uses JDBC drivers to connect with the database.
- We can use jdbc api to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like open database connectivity (ODBC) provided by microsoft.

JDBC API

- Due to JDBC API technology, user can also access other tabular data sources like spreadsheets or flat files even in the a heterogeneous environment.
- JDBC API is a part of the java platform that have included java standard edition (java SE) and the java enterprise edition (java EE) in itself.
- The JDBC API has four main interface:
- The latest version of JDBC 4.0 API is divided into two packages.
  i-) java.sql
  ii-) javax.sql.
- The current version of JDBC is 4.3. It is the stable release since 21st september, 2017. It is based on the X/open SQL call level interface. The **java.sql** package contains classes and interfaces for JDBC API.

JDBC API CONTI….

❖A list of popular *interfaces* of JDBC API are given below:

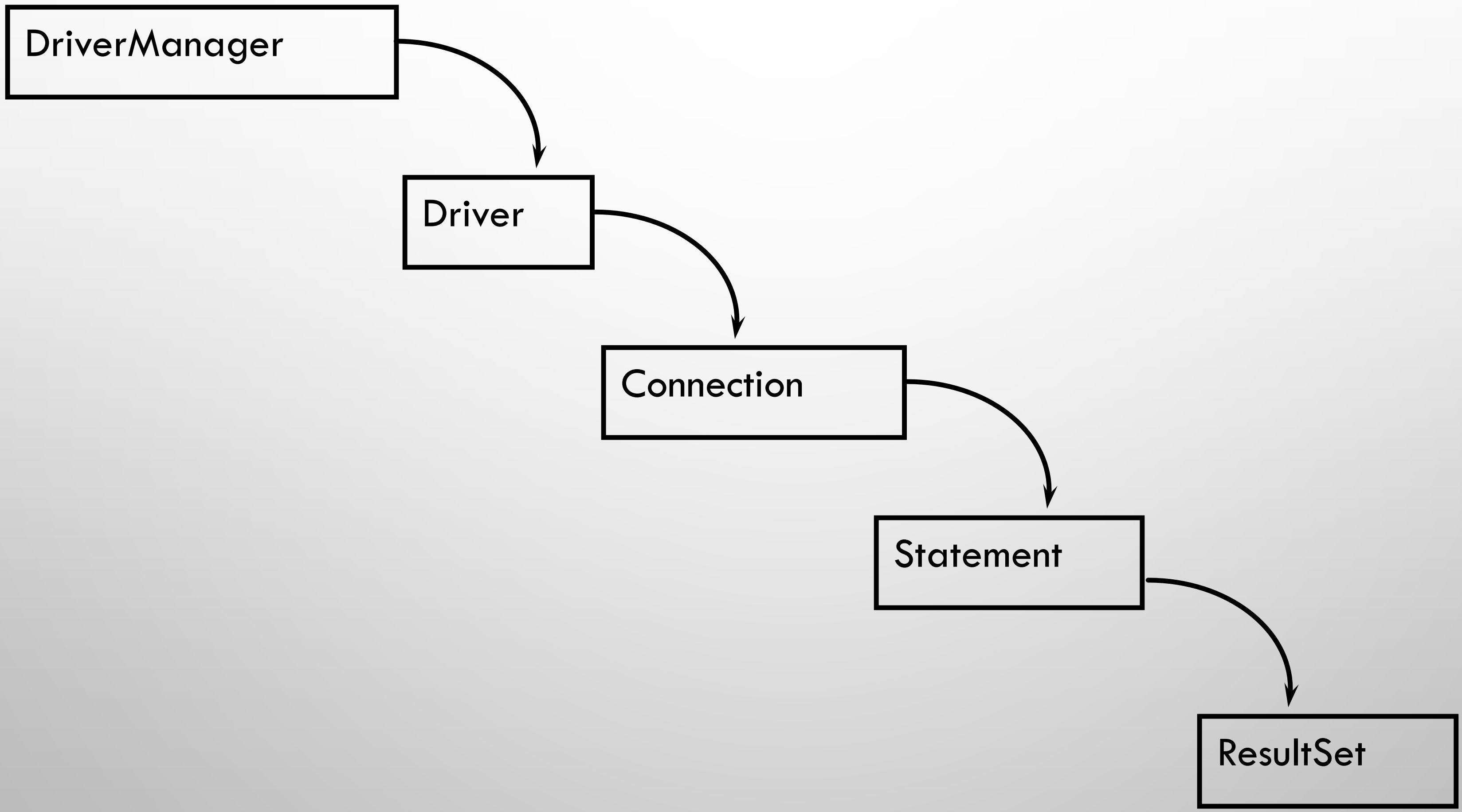
- ✓Driver interface
- ✓Connection Interface
- ✓Statement Interface
- ✓PreparedStatement Interface
- ✓CallableStatement Interface
- ✓ResultSet Interface
- ✓ResultSetMetadata Interface
- ✓Databasemetadata Interface
- ✓RowSet Interface

JDBC API CONTI….

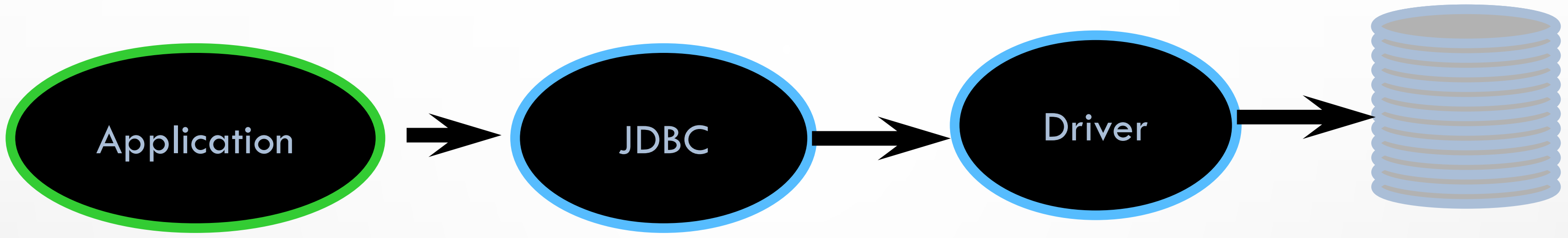
❖A list of popular classes of JDBC API are given below:

- ✓DriverManager class
- ✓Blob class
- ✓Clob class
- ✓Types class

JDBC API USAGE



JDBC ARCHITECTURE

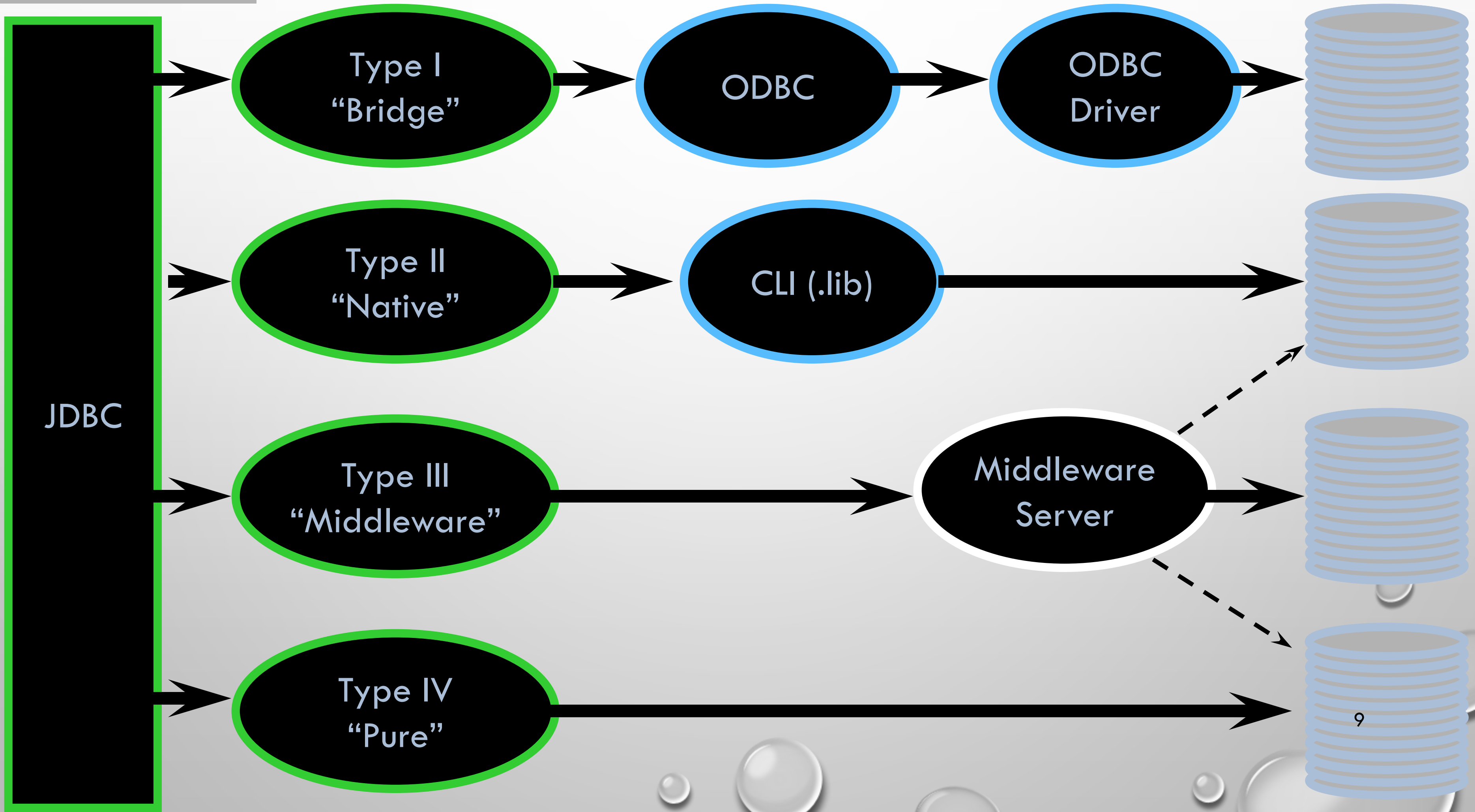


- Java code calls JDBC library
- JDBC loads a *driver*
- Driver talks to a particular database
- Can have more than one driver -> more than one database
- Ideal: can change database engines <u>without changing any application code</u>

JDBC DRIVERS

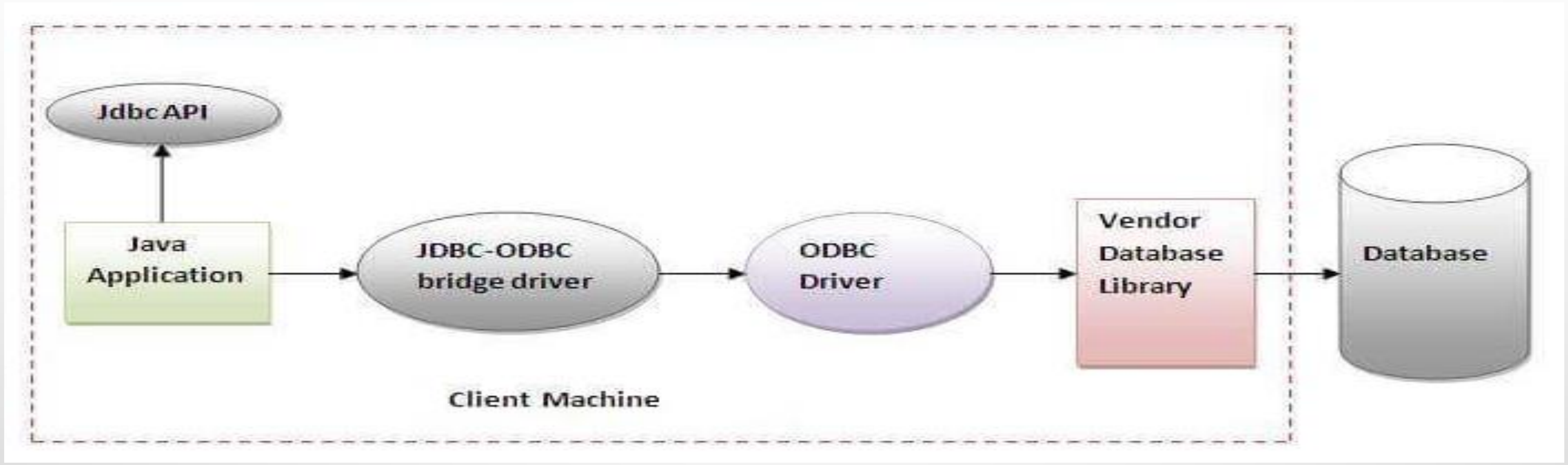
- Type I: JDBC-ODBC bridge driver
- Type II: “Native Driver” (partially java driver)
- Type III: “Middleware” (network protocol driver) (fully java driver)
- Type IV: “Pure” (thin driver) (fully java driver)

JDBC DRIVERS



Type I: JDBC-ODBC BRIDGE DRIVER

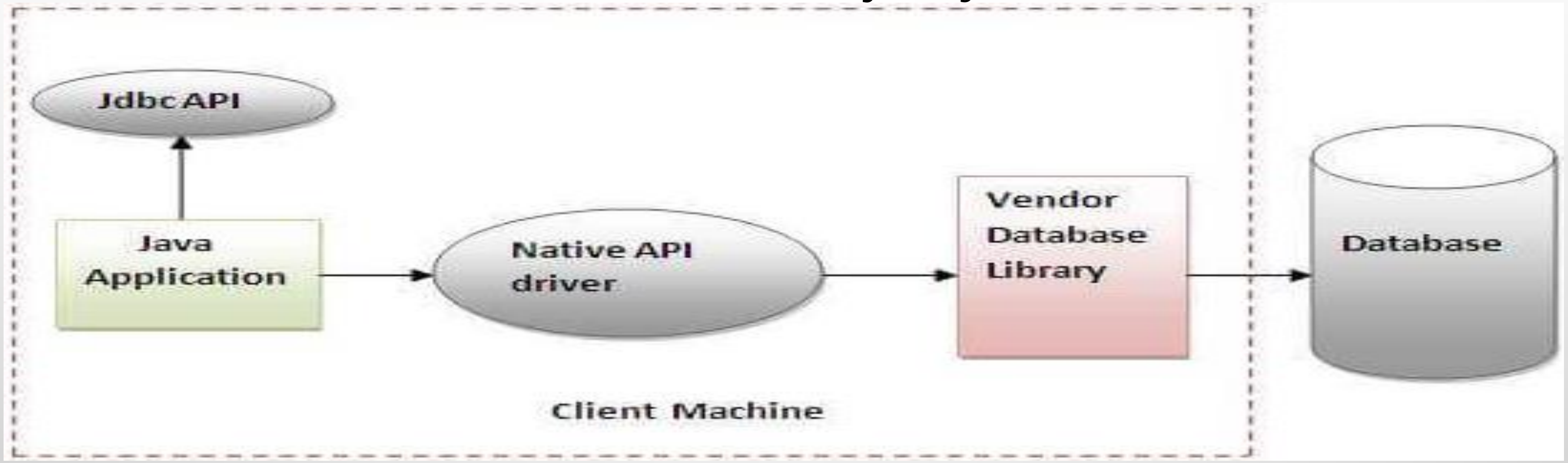
- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database.



- **Advantages:**
  - Easy to use.
  - Can be easily connected to any database.
- **Disadvantages:**
  - Performance degraded because JDBC method call is converted into the ODBC function calls.
  - The ODBC driver requires to be installed /Configured on the client machine.
  - Not good for web

Type II: Native-API driver

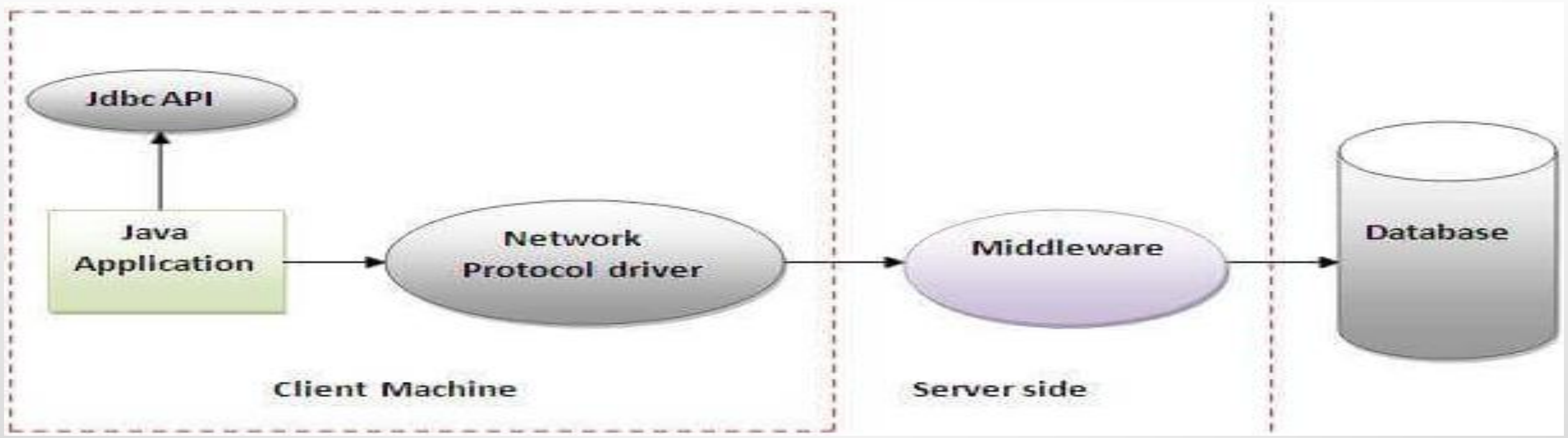
- The native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java..



- Advantage:
  - performance upgraded than JDBC-ODBC bridge driver.
- Disadvantage:
  - The Native driver needs to be installed on the each client machine.
  - The Vendor client library needs to be installed on client machine.

Type III: **Network Protocol Driver**

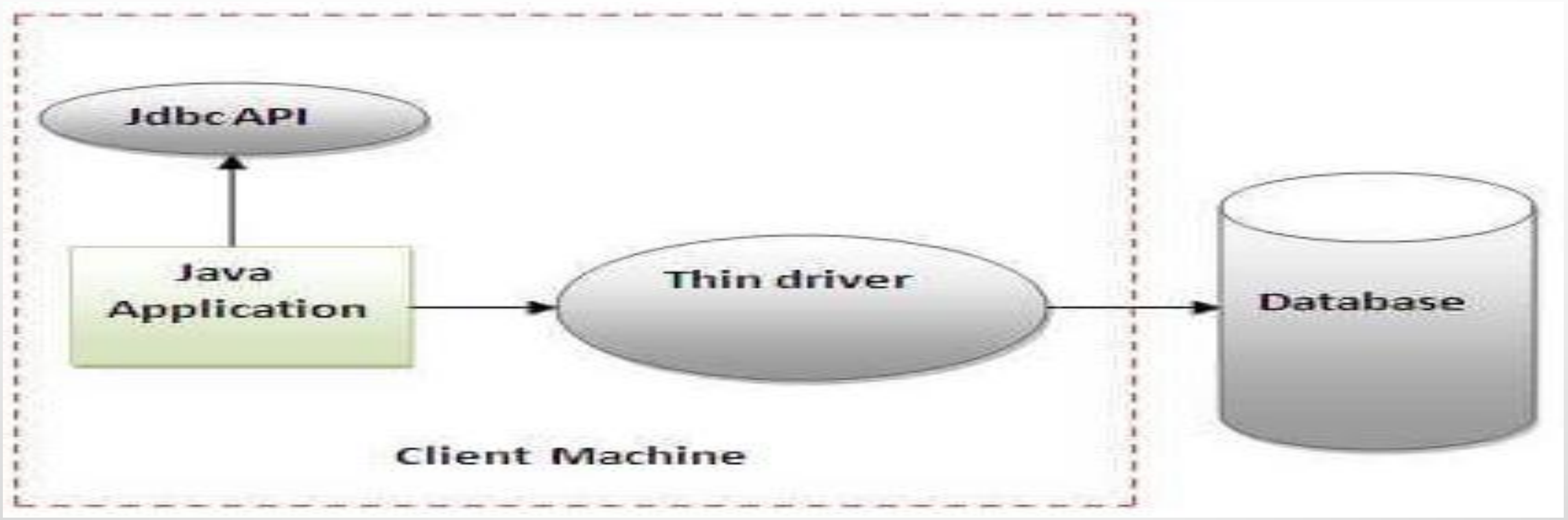
- The network protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.



- **Advantage:**
  - No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.
- **Disadvantages:**
  - Network support is required on client machine.
  - Requires database-specific coding to be done in the middle tier.
  - Maintenance of network protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

Type IV: **Thin Driver**

- The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in java language.



- **Advantage:**
  - Better performance than all other drivers.
  - No software is required at client side or server side.
- **Disadvantage:**
  - Drivers depend on the database.

Java Database Connectivity

- There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:
  - Register the driver class
  - Create connection
  - Create statement
  - Execute queries
  - Close connection

Java Database Connectivity Steps

1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax:

public static void forName(String classname) throws ClassNotFoundException

Register ODBC Drive

```
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver")
```

Register MySql Drive

```
Class.forName(“com.mysql.jdbc.Driver")
```

.Register Oracle Drive

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

Conti….

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax:

```
public static Connection getConnection(String url) throws SQLException

public static Connection getConnection(String url, String name,
                                       String password)  throws SQLException
```

- **Connection with the Access database**

```
Connection c=DriverManager.getConnection("jdbc:odbc:mydsn");
```

- **Connection with the MySql database**

```
Connection con=DriverManager.getConnection(
               "jdbc:mysql://localhost:3306/sonoo","root","root");
```

- **Connection with the Oracle database**

```
Connection con=DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:xe","system","password");
```

Conti….

3) Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax:

```
public Statement createStatement()throws SQLException
```

Example to create the statement object

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

Conti….

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax:

```
public ResultSet executeQuery(String sql) throws SQLException
```

Example:

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

Conti…..

5) Close the connection object

Syntax:

```
public void close()throws SQLException
```

Example:

```
con.close();
```

- **Create Database & Table**

```
create database dev;

use dev;

create table emp(id int(10),name varchar(40),age int(3));
```

Database Connectivity with Access to Select

```
import java.sql.*;
   class Test{
   public static void main(String ar[ ]){
    try{
      String url="jdbc:odbc:mydatasource";
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection c=DriverManager.getConnection(url);
      Statement st=c.createStatement();
      ResultSet rs=st.executeQuery("select * from student");
while(rs.next()){
        System.out.println(rs.getString(1));
       }
    }catch(Exception e){System.out.println(e);}
       }}
```

Database Connectivity with MySql to Select

```
import java.sql.*;
class MysqlCon{
public static void main(String args[ ]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/dev","root","root");
 Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
    System.out.println(rs.getInt(1)+"
"+rs.getString(2)+"  "+rs.getString(3));
    con.close();
    }catch(Exception e){ System.out.println(e);}
    } }
```

PreparedStatement interface

- The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.
- The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

Syntax:

```
public PreparedStatement prepareStatement(String query)throws SQLException{}
```

Insert Record into Database with MySql

```
import java.sql.*;

class InsertPrepared{

public static void main(String args[]){

try{

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

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

PreparedStatement stmt=con.prepareStatement("insert into
emp values(?,?,?)");
stmt.setInt(1,101);//1 specifies the first parameter in
the query

stmt.setString(2,"Dev");

stmt.setInt(3,11);

int i=stmt.executeUpdate();

System.out.println(i+" records inserted");

con.close();

}catch(Exception e){ System.out.println(e);}

}}
```

Retrieve Record from Database with MySql

```
import java.sql.*;
class RetrievePrepared{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:
3306/dev","root","root");
PreparedStatement stmt=con.prepareStatement("select *
from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
con.close();
}catch(Exception e){ System.out.println(e);}
}}
```

Update Record with MySql

```
import java.sql.*;
class UpdatePrepared{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:
3306/dev","root","root");
PreparedStatement stmt=con.prepareStatement("update emp
set name=?,age=? where id=?");
stmt.setString(1,"Devendra Gahlot");//1 specifies the first parameter in the query i.e. name

stmt.setInt(2,111);

stmt.setInt(3,101);

int i=stmt.executeUpdate();

System.out.println(i+" records updated");

con.close();

}catch(Exception e){ System.out.println(e);}

}}
```

Delete Record from database with MySql

```
import java.sql.*;
class DeletePrepared{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:
3306/dev","root","root");
PreparedStatement stmt=con.prepareStatement("delete from
emp where id=?");
stmt.setInt(1,101);
int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
con.close();
}catch(Exception e){ System.out.println(e);}
}}
```

THANKS