Prepared By Abhiyan Dev Rimal
https://github.com/abhirimal/be-software-notes

PAGE :
DATE : / /

Chapter-8
Java Database Connectivity (JDBC)

1. Why JDBC is required in java. Explain the role of JDBC driver.

Ans: JDBC stands for Java Database Connectivity. It is a Java API to connect and execute query with a database. It is an advancement for ODBC (Open Database Connectivity). JDBC 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 API

= JDBC is an application programming interface for the programming language Java, which defines how a client may access a database. It is a Java-based data access technology used for Java database connectivity.
JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database. JDBC works with Java on a variety of platforms, such as Windows, Mac OS, and the various version of UNIX.

By the help of JDBC API, we can save, update, delete and fetch data from the database.

Scanned with CamScanner

PAGE :
DATE : / /

~~Chapter-8~~

JDBC enables us to

1) make connection to database.
2) create SQL and MySQL statement
3) Execute queries in database
4) View and modify results

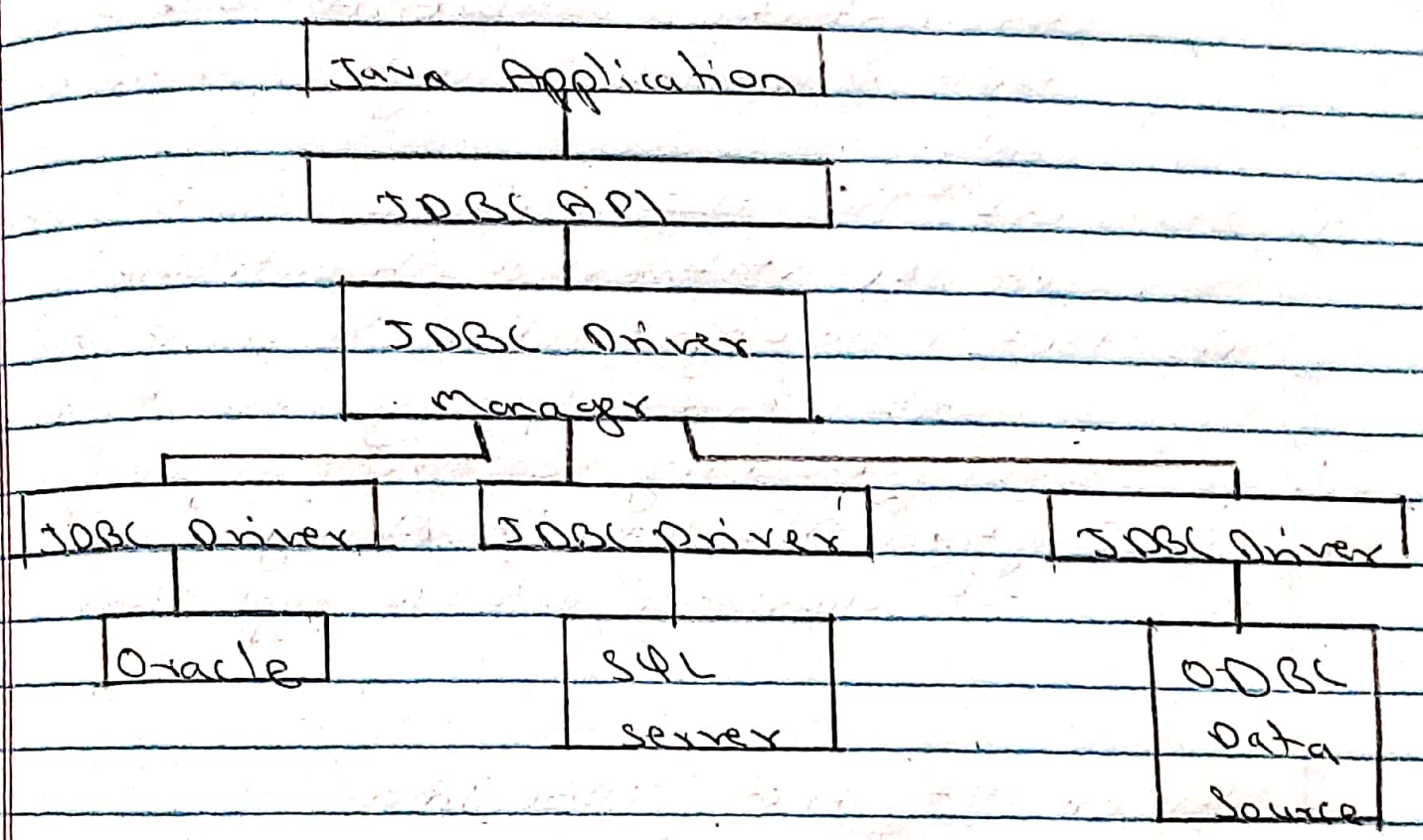

fig:- JDBC Architecture

~~JDB~~

P.T.O ⇒

Scanned with CamScanner

PAGE :
DATE : / /

The JDBC API provides following interfaces and classes

1) DriverManager: This class manages a list of database driver.
2) Driver: This interface handles the communications with the database server.
3) Connection: This interface is the only methods for contacting a database. All communication with database is through connection object only.
4) Statement: This interface is used to submit the SQL statements to the database.
5) ResultSet: These objects hold data received from a database after you execute an SQL query using Statement objects.
6) SQLException: This class handles any errors that occur in a database application.

There are 4 types of JDBC Drivers

1) JDBC-ODBC Bridge Driver
2) Native Driver
3) Network Protocol Driver
4) Thin Driver

Scanned with CamScanner

PAGE:
DATE: / /

2) what are the different types of JDBC statements available? Explain with example.

→ There are 3 types of statements. They are:-

i) Statement → It can be used for general-purpose access to the database. It is useful when you are using static SQL.

ii) Prepared Statement → It can be used when you plan to use the same SQL statement many times. The PreparedStatement interface accepts input parameters at run time.

iii) CallableStatement → It can be used when you want to access database stored procedures.

ODBC (Open Database Connectivity)

→ It is a procedural oriented Database Connectivity which is used to provide connection between front-end application and back-end application. It is slower than JDBC

Scanned with CamScanner

PAGE :
DATE : / /

v.v.Imp

3) Explain different types of JDBC Drivers.
Or
Explain different types of Database Drivers

→ JDBC API cannot directly communicate with the database. It uses JDBC driver of the database to interact with the database.

→ ~~JDBC~~ JDBC driver is a software component provided along with the database which is required by the JDBC API to interact with the database.

There are 4 types of JDBC Drivers
1) Type 1 JDBC Driver / JDBC-ODBC Bridge Driver
2) Type 2 JDBC Driver / Native API Driver
3) Type 3 JDBC Driver / Network Protocol Driver
4) Type 4 JDBC Driver / Native Protocol Driver

1) JDBC-ODBC Driver

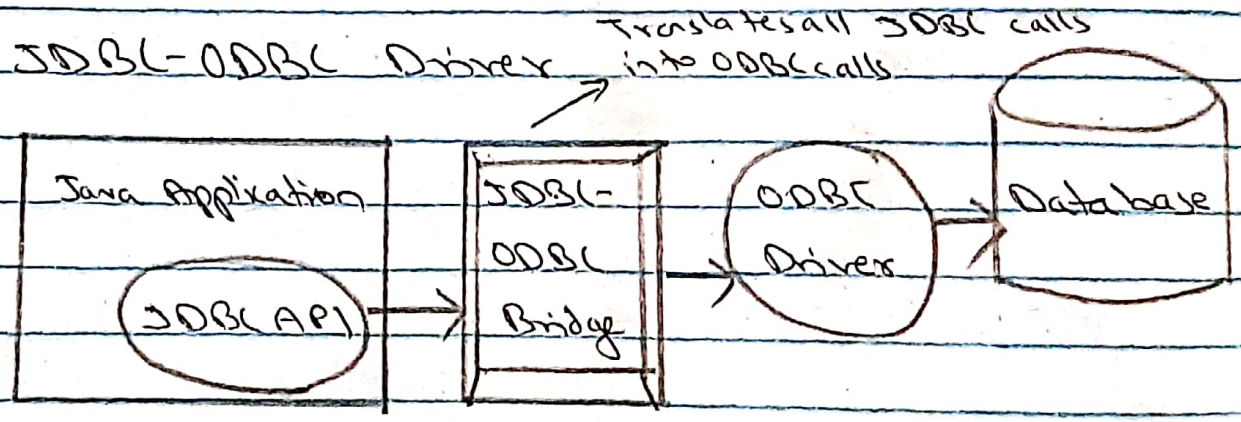

fig: JDBC-ODBC Bridge Driver

→ ~~It uses~~ This driver provides the bridge between JDBC and ODBC API. It uses ODBC driver to connect to the database.

Scanned with CamScanner

PAGE :
DATE : / /

→ This type of drivers translate all JDBC calls into ODBC calls and sends them to ODBC driver which interacts with the database.
→ These types of drivers are slowest of all types. Because, all JDBC calls will go to the ODBC driver through the bridge and then to database.
→ This driver is not entirely written in java language. It causes the portability issues.
→ It has been removed in Java 8.

Advantages
1. Easy to use
2. Can be easily connected to any database.

Disadvantages
1. It is slow
2. The ODBC driver needs to be installed on the client machine.

Scanned with CamScanner

2) Native-API Driver

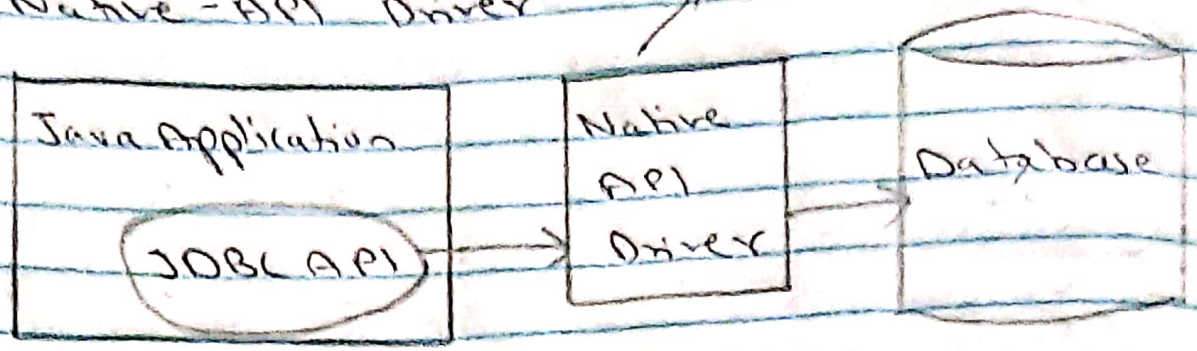

fig:- Native-API Driver

→ Native-API Driver translates all JDBC methods calls into database specific calls using native API of the database.
→ It uses client side libraries of the database.
→ This driver is database specific. So once we switch from one database to another, we have to change the driver.
→ Its performance is slightly better than JDBC-ODBC bridge driver as communication layer is reduced in this driver.
→ It is partially written in Java, so this also causes the portability issues.

Advantages
1. Performance upgraded than JDBC-ODBC.

Disadvantages
1. It needs to be installed into each client machine.

PAGE :
DATE : / /

3) Network Protocol Driver

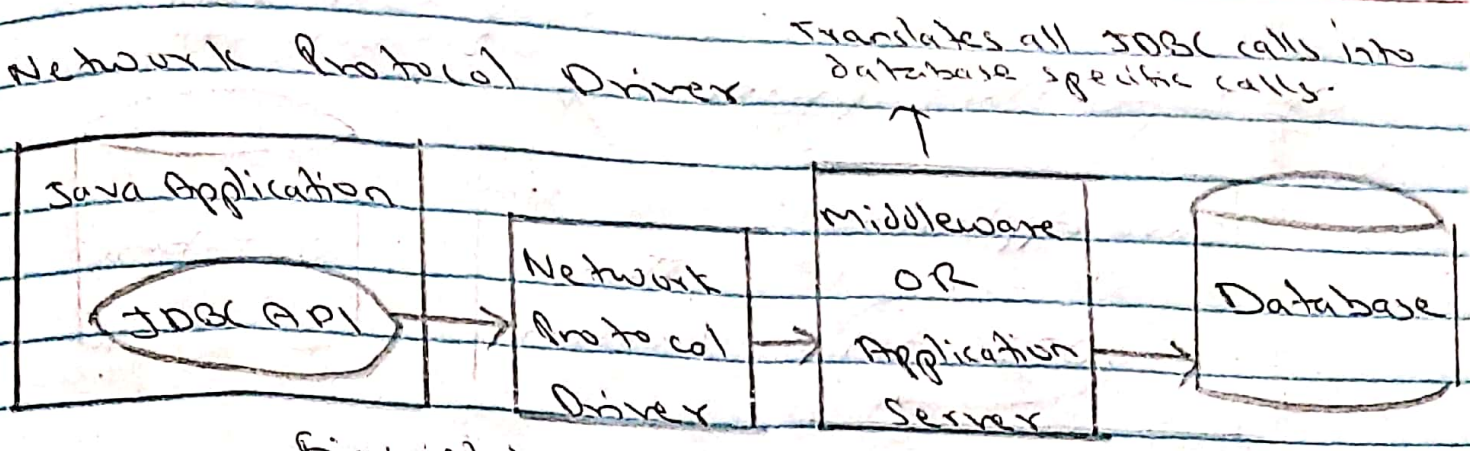

fig:- Network Protocol Driver

→ Network Protocol Driver uses middle ware or application server that converts/translates all JDBC calls into database specific calls.
→ One of the main advantage of this driver is that is entirely written in java language. So no portability issues.
→ It is costly as extra application server or middle ware component has to be maintained.

Advantages
1. No client side library is required

Disadvantages
1. Maintenance becomes costly
2. Requires database-specific coding to be done in the middle tier.

Scanned with CamScanner

3) Network Protocol Driver

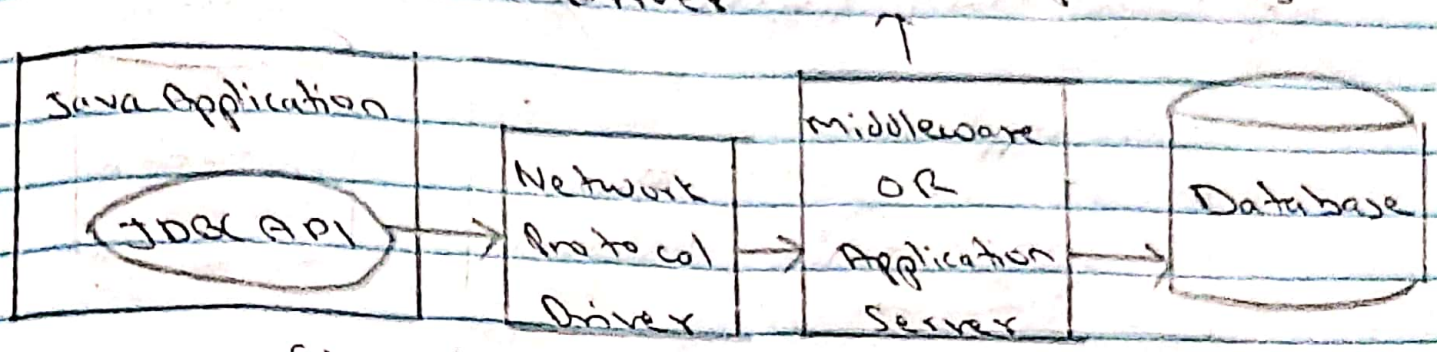

fig:- Network Protocol Driver

→ Network Protocol Driver uses middle ware or application server that converts/translates all JDBC calls into database specific calls.

→ One of the main advantage of this driver is that is entirely written in java language. So no protability issues.

→ It is costly as extra application server or middle ware component has to be maintied.

Advantages

1. No client side library is required

Disadvantages

1. Maintenance becomes costly
2. Requires database-specific coding to be done in the middle tier.

Scanned with CamScanner

4) Type Thin Driver

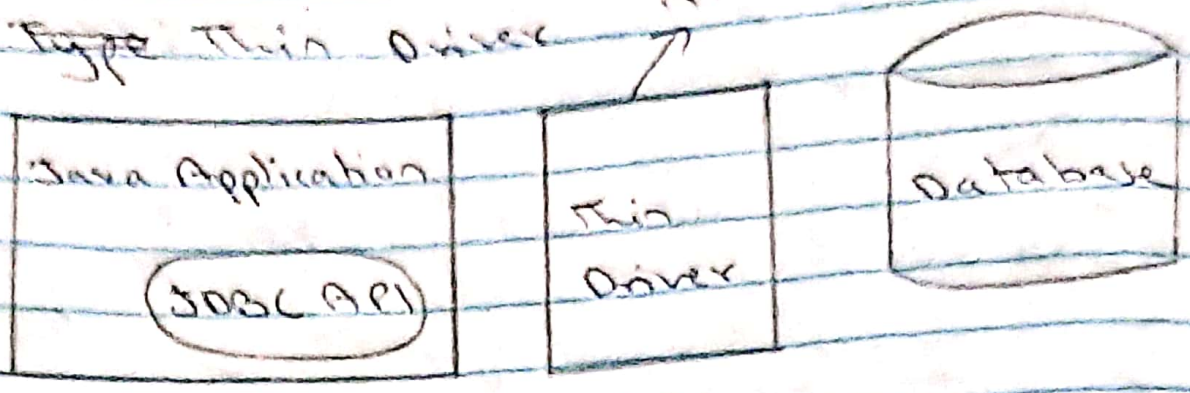

fig:- Thin Driver

→ Thin Driver converts JDBC calls into database specific calls.
→ It is most popular among all 4 types of JDBC drivers
→ It is completely written in Java & portable
→ It is preffered over Network Protocol Driver as it removes extra layer of communication and this makes it faster than the Type 3 driver.

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

Disadvantages
→ Driver depends on the database.

Scanned with CamScanner

PAGE :
DATE : / /

Steps to connect to a database

1. Register Driver Class
2. Create Connection
3. Create Statement
4. ~~Close Connection~~ Execute Queries
5. Close Connection

Steps	Code Snippets
1. Register Driver Class	Class.forName("com.mysql.jdbc.Driver")
2. Create Connection	Connection con = DriverManager.getConnection("jdbc:mysql://localhost/dbaseName", "usersname", "pwd")
3. Create Statement	Statement stmt = con.Create Statement();
4. Execute Queries	ResultSet rs = stmt.executeQuery("Select * from table")
5. Close Connection	con.close()

(SELECT * FROM)

Scanned with CamScanner

PAGE :
DATE : / /

3) WAP to display all records from the database table.

DB name = testdb

```
import java.sql.*;

class SqlDemo {
    public static void main(String args[]) {
        String username = "root";
        String pwd = "password";
    String dbaseURL = "jdbc:mysql://localhost:
        3306/testdb";
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    Connection c = DriverManager.getConnection
        (dbaseURL, username, pwd);
    Statement stmt = c.createStatement();
    ResultSet r = stmt.executeQuery("SELECT
        * from employee");

    while(r.next()) {
    System.out.println(r.getString(1) + " " + rs.getString(2));
        }
    c.close();
    }   catch (SQLException e) {
        System.out.println(e.getMessage());
    }
        catch(ClassNotFoundException e) {
    }   System.out.println(e.getMessage());
}   }
```

Scanned with CamScanner

PAGE :
DATE : / /

6) A database 'testdb' contains a table 'employee' with some records having id, name, post, salary. WAP to update the salary to 5000 whose post is 'Manager'.

```
import java.sql.*;

class Query {
  public static void main(String[] args) {
  try {
    String url = "jdbc:mysql://200.210.220.1:1114/
                testdb";
    Connection conn = DriverManager.getConnection
          (url," "," ");
    Statement stmt = conn.createStatement();
    ResultSet rs;
    rs = stmt.executeQuery("UPDATE employee
        SET salary=50000 WHERE post='Manager'");
    conn.close();
  }
    catch (Exception e) {
      System.out.exr.println("not an Exception");
      System.exr.println(e.getMessage());
    }
  }
}
```

Scanned with CamScanner