

Unit 3: Contents

UNIT-III: WORKING WITH DATABASE:

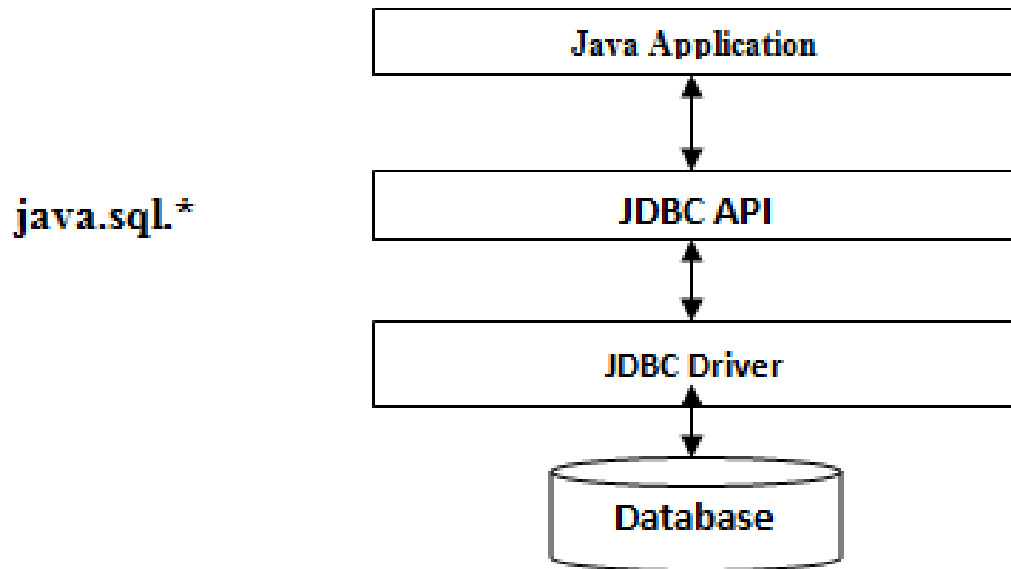
Getting started with JDBC , Defining ODBC, Introduction to JDBC, Components of JDBC, JDBC Architecture, Types of Drivers, Working with JDBC APIs, Creating a Simple Application, Working with Prepared Statement.

Getting Started with JDBC

1) What is JDBC?

Java Database Connectivity in short called as JDBC. It is a java API which enables the java programs to execute SQL statements. It is an application programming interface that defines how a java programmer can access the database in tabular format from Java code using a set of standard interfaces and classes written in the Java programming language.

The **JDBC DriverManager** class defines objects which can connect Java applications to a JDBC driver. DriverManager has traditionally been the backbone of the JDBC architecture. It is quite small and simple.

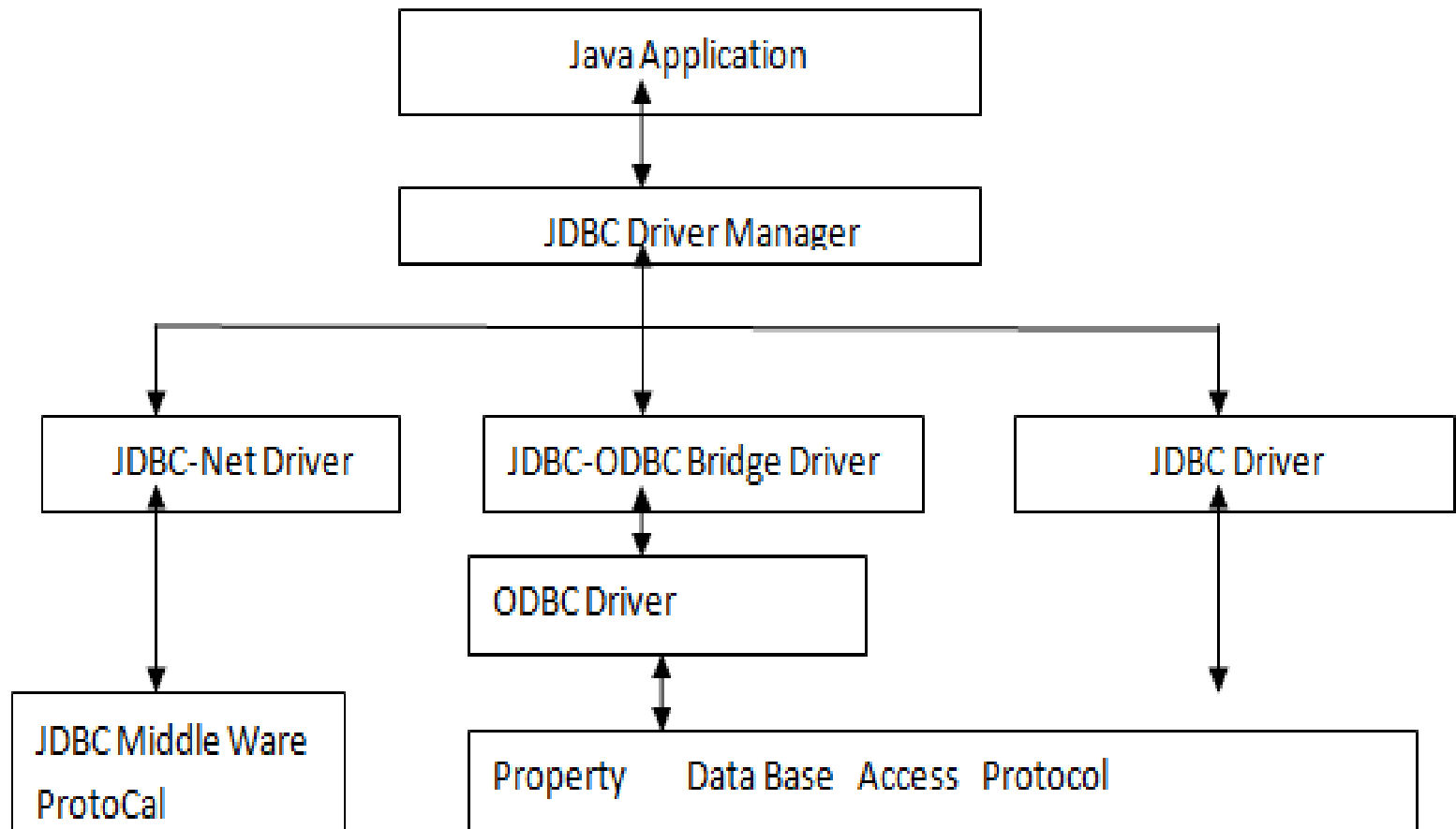


JDBC Components

JDBC provides the following components as part of the JDK.

JDBC Driver Manager	The JDBC driver manager is the backbone of the JDBC Architecture. It actually is quite small and simple; its primary function is to connect Java applications to the correct JDBC Driver and then get out of the way.
JDBC-ODBC Bridge	The JDBC-ODBC bridge allows ODBC drivers to be used as JDBC drivers. It provides a way to access less popular DBMS.If JDBC driver is not implemented for it.

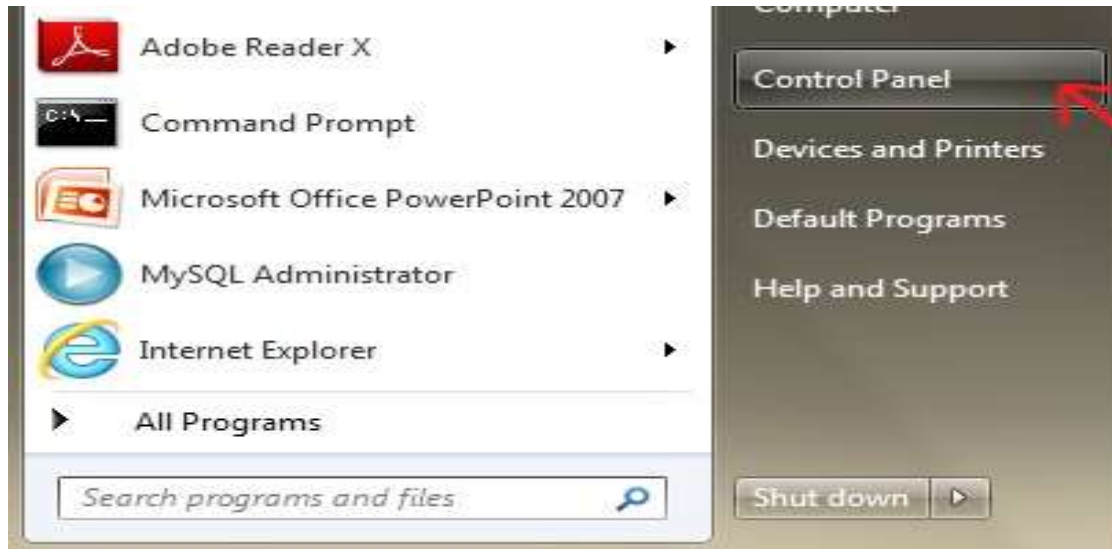
JDBC Driver Component Architecture



Defining ODBC

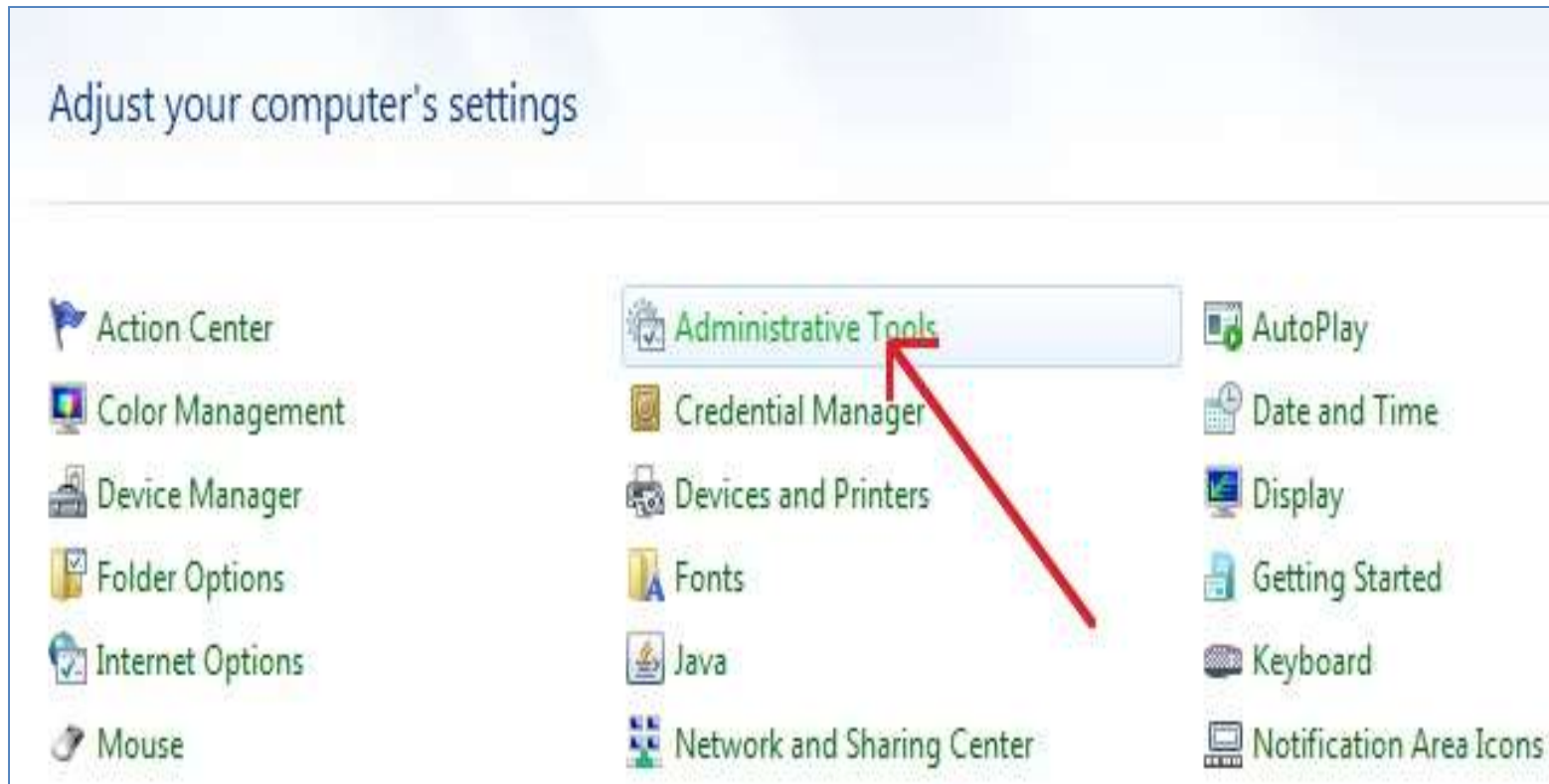
ODBC is known as Open Database Connectivity, this is present in built in every operating system. The following are the steps to check where the ODBC is present in our operating system.

Go to **Start menu..Choose Control Panel**









Contd...


Next choose Administrative tools from that control panel.



Contd...

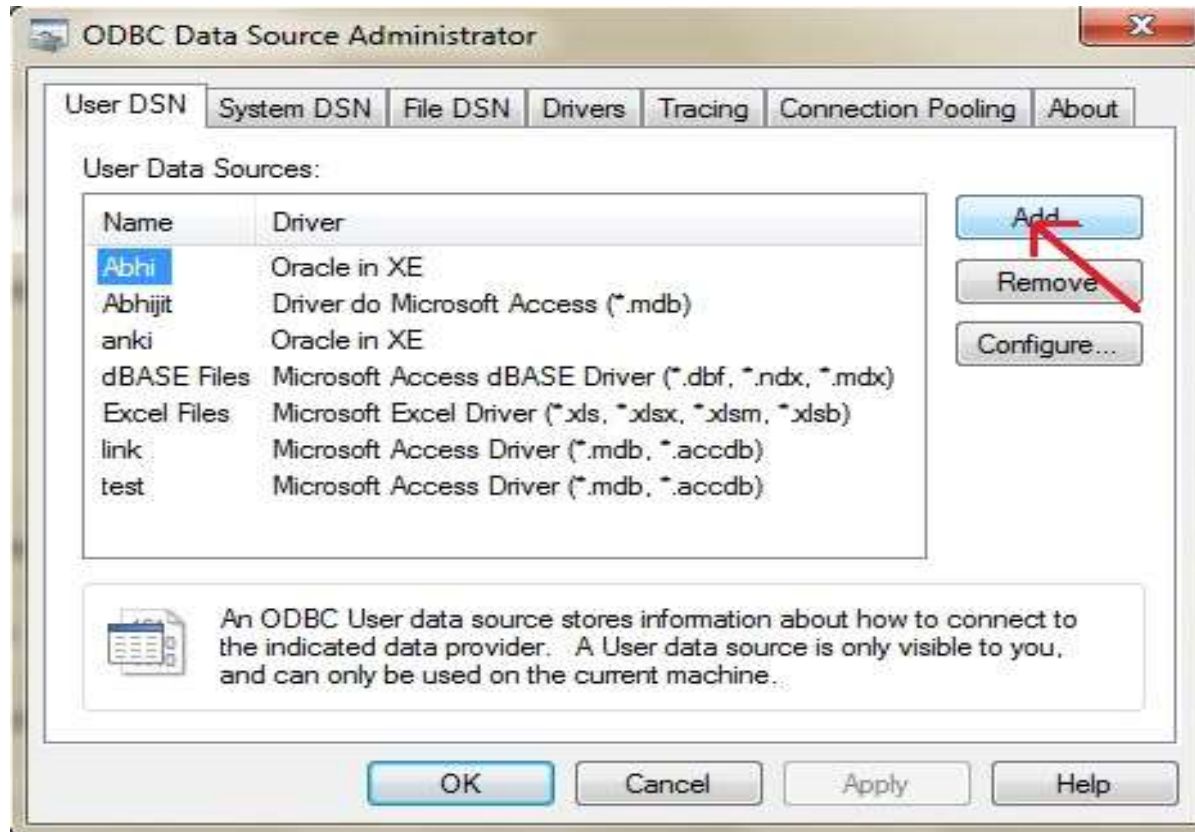
After that choose Data Sources(ODBC option)

Name	Date modified	Type	Size
 Component Services	14-07-2009 10:16	Shortcut	2 KB
 Computer Management	14-07-2009 10:11	Shortcut	2 KB
 Data Sources (ODBC)	14-07-2009 10:11	Shortcut	2 KB
 Event Viewer	14-07-2009 10:12	Shortcut	2 KB
 Internet Information Services (IIS) Manager	06-01-2013 10:38	Shortcut	2 KB
 iSCSI Initiator	14-07-2009 10:11	Shortcut	2 KB

 Maintains ODBC data sources and drivers.

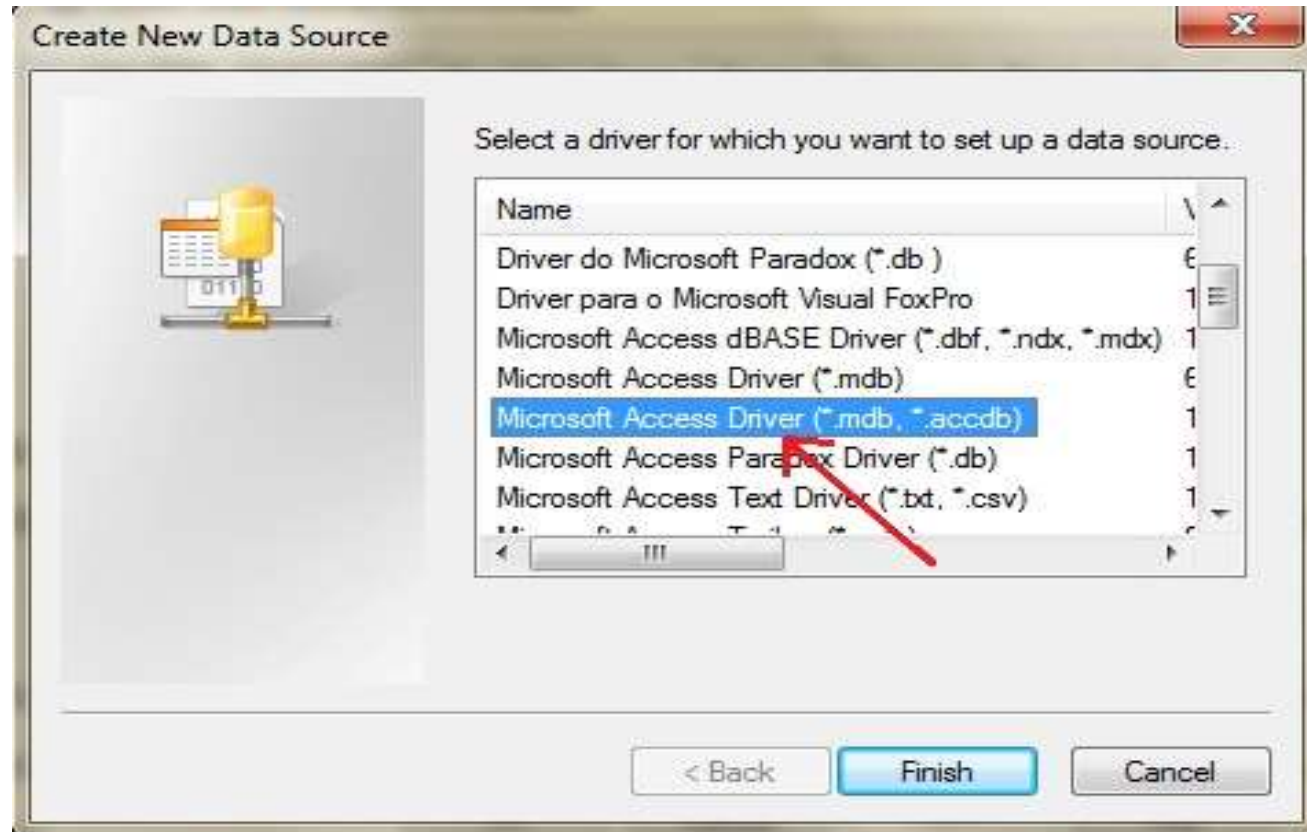
Contd...

Next we can add the Data Source Name for the type of database which we want to connect



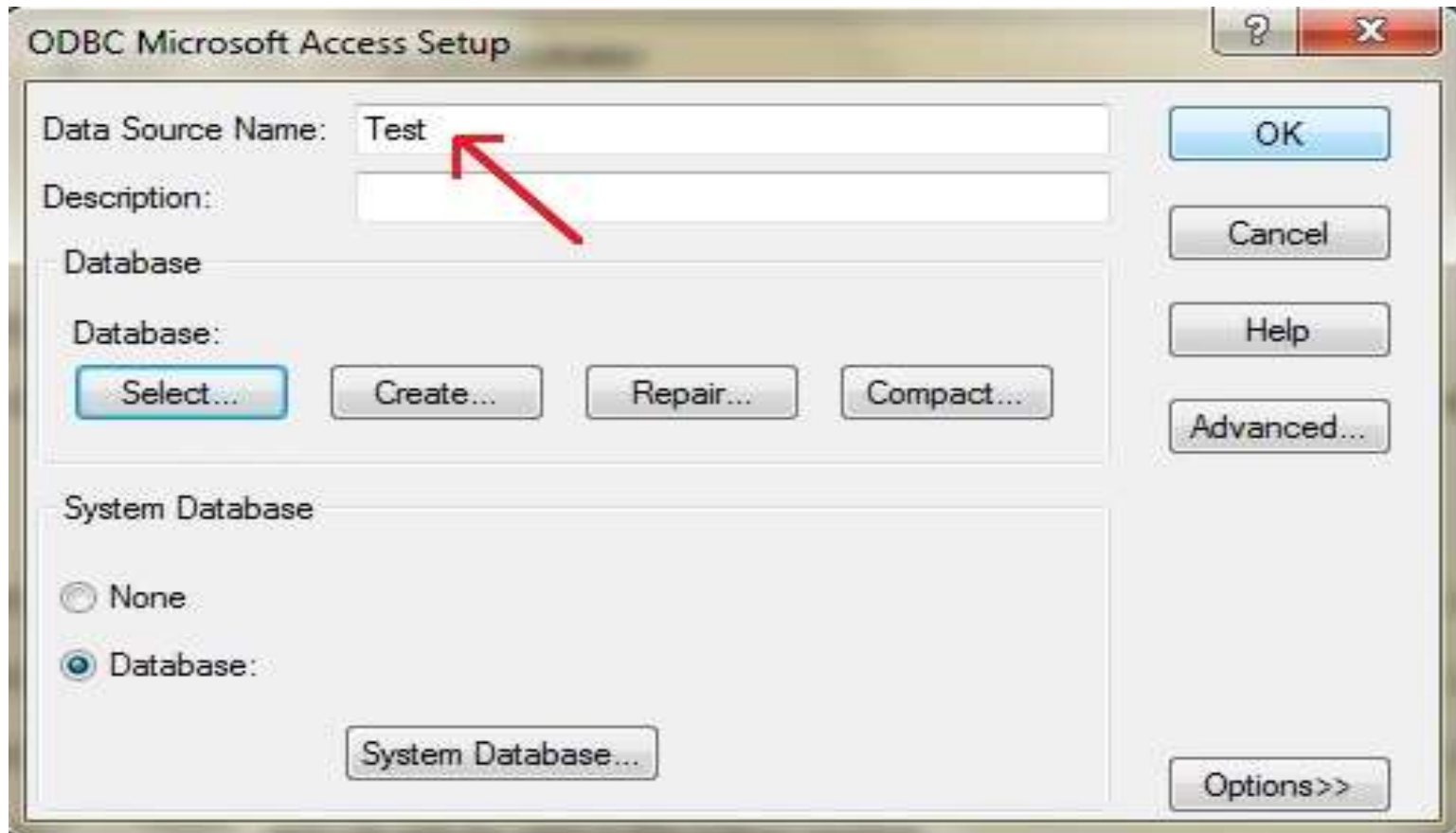
If we use 32-bit Operating system

If we use 32 bit operating system in our system, then we can see the list of options in the ODBC:

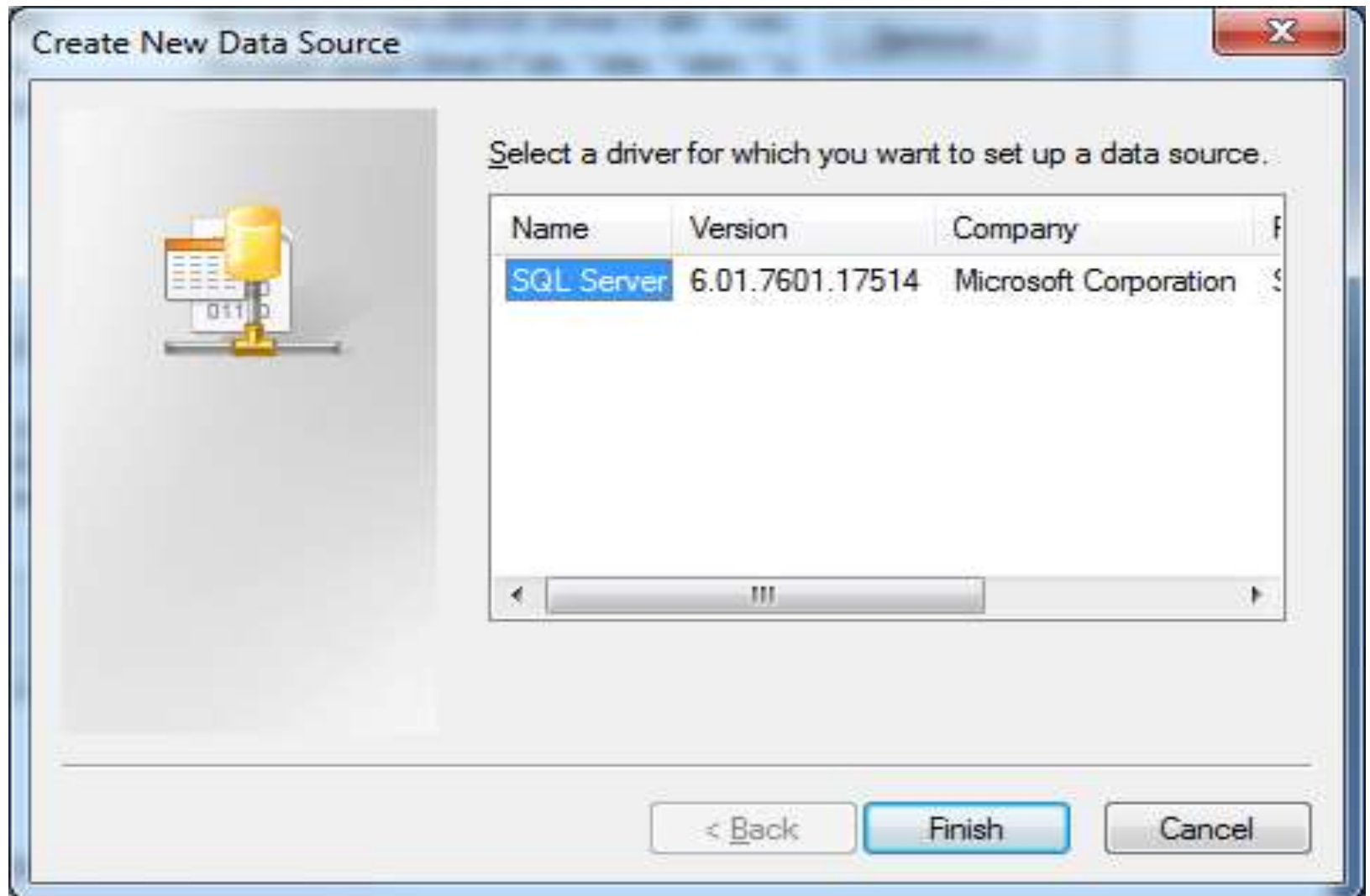


We can choose DSN Name for that database

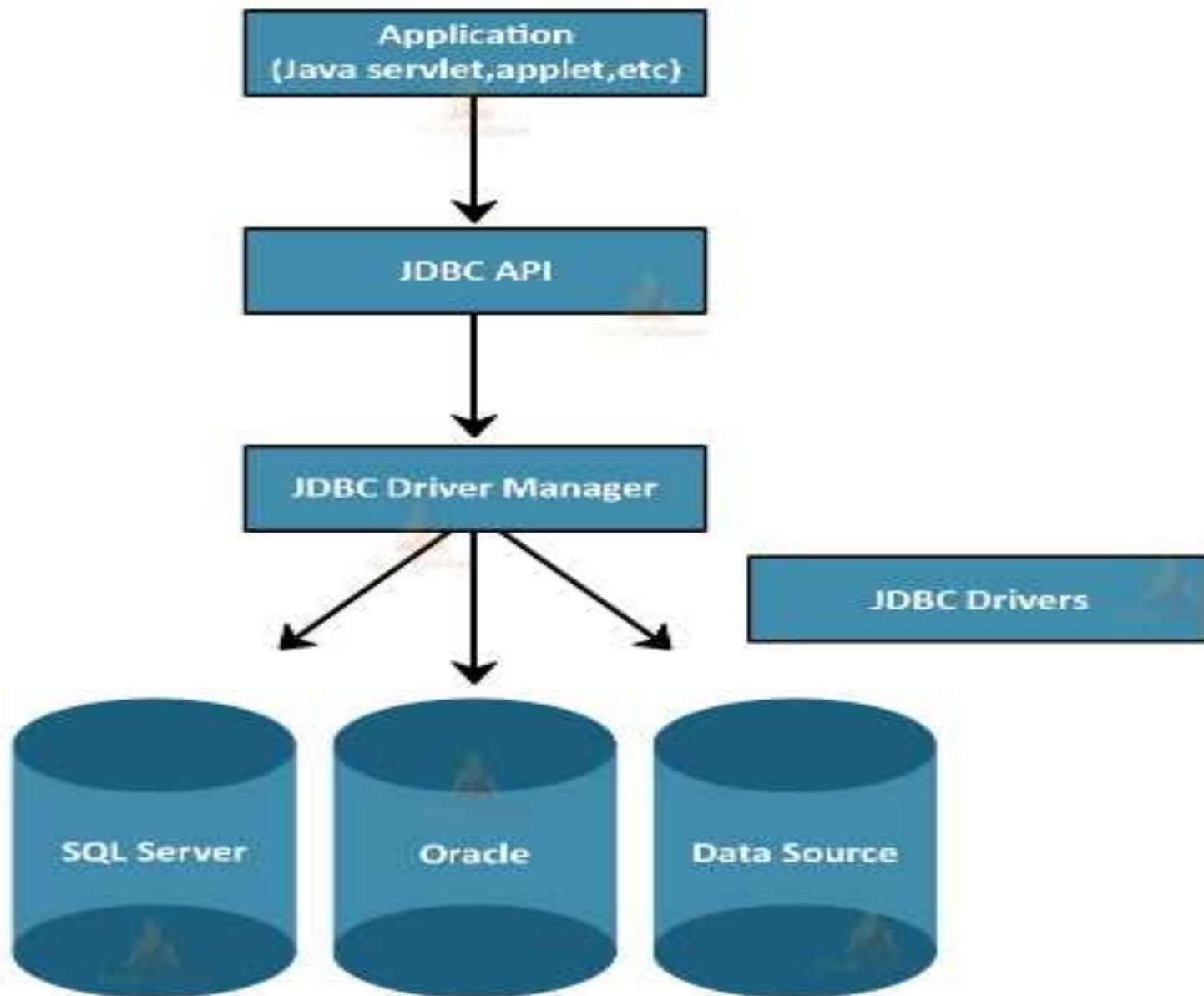
Here we choose **Test** as data source name.



If we use 64 Bit OS, then we cannot see multiple databases, only SQL server can be seen



Architecture of JDBC



JDBC has four major components that are used for the interaction with the database.

1. JDBC API
2. JDBC Test Suite
3. JDBC Driver Manager
4. JDBC ODBC Bridge Driver

1) JDBC API: JDBC API provides various interfaces and methods to establish easy connection with different databases.

`javax.sql.*;`

`java.sql.*;`

2) JDBC Test suite: JDBC Test suite facilitates the programmer to test the various operations such as deletion, updation, insertion that are being executed by the JDBC Drivers.

3) JDBC Driver manager: JDBC Driver manager loads the database-specific driver into an application in order to establish the connection with the database. The JDBC Driver manager is also used to make the database-specific call to the database in order to do the processing of a user request.

4) JDBC-ODBC Bridge Drivers: JDBC-ODBC Bridge Drivers are used to connect the database drivers to the database. The bridge does the translation of the JDBC method calls into the ODBC method call. It makes the usage of the **sun.jdbc.odbc** package that encompasses the native library in order to access the ODBC (Open Database Connectivity) characteristics.

Working with JDBC API

Java Database Connectivity

Register driver

Get connection

Create statement

Execute query

Close connection



Components of JDBC API

In JDBC API, we can have several pre-defined classes, interfaces and corresponding methods to execute the task. The following are some of the classes, interfaces and methods which are present in the JDBC API.

In java, we can use JDBC with the help of following package

```
import java.sql.*;
```

Class/interface	Usage	Methods
DriverManager	This class manages the JDBC drivers. You need to register your drivers to this.	registerDriver(), getConnection() are the two methods used from this <u>class</u> .
Driver	This interface is the Base interface for every driver class i.e. If you want to create a JDBC Driver of your own you need to implement this interface.	createInstance()
Class	This is one predefined class in JDBC API which is used to connect the type of driver what we are using for the program.	The following is the method used to link driver: forName("driver path"); Syntax: Class.forName("driver path");

Connection	This interface represents the connection with a specific database. SQL statements are executed in the context of a connection.	This interface provides methods such as close(), commit(), rollback(), createStatement(), prepareCall(), prepareStatement(), setAutoCommit() setSavepoint()
Statement	This interface represents a static SQL statement . Using the Statement object and its methods, you can execute an SQL statement and get the results of it.	It provides methods such as execute(), executeBatch(), executeUpdate()

PreparedStatement	<p>precompiled SQL statement. An SQL statement is compiled and stored in a prepared statement and you can later execute this multiple times.</p>	<p><code>executeQuery()</code>, <code>executeUpdate()</code>, and <code>execute()</code> to execute the prepared statements and <code>getXXX()</code>, <code>setXXX()</code> (where XXX is the datatypes such as long int float etc..) methods to set and get the values of the bind variables of the prepared statement.</p>
ResultSet	<p>This interface represents the database result set, a table which is generated by executing statements</p>	<p>There are several methods used in this interface to retrieve the data. They are as follows:</p> <p><code>getInt(column index)</code></p> <p><code>getInt(column name)</code></p> <p><code>getString(column name)</code></p> <p><code>getString(column index)</code></p>

Types of Drivers

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4 :

- 1) Type 1 – JDBC-ODBC Bridge Driver
- 2) Type 2 – JDBC-Native API
- 3) Type 3 – JDBC-Net pure Java
- 4) Type 4 – 100% Pure Java

Type 1-JDBC-ODBC Bridge Driver

- 1) In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine.
- 2) Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.
- 3) When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

Type 1 Architecture

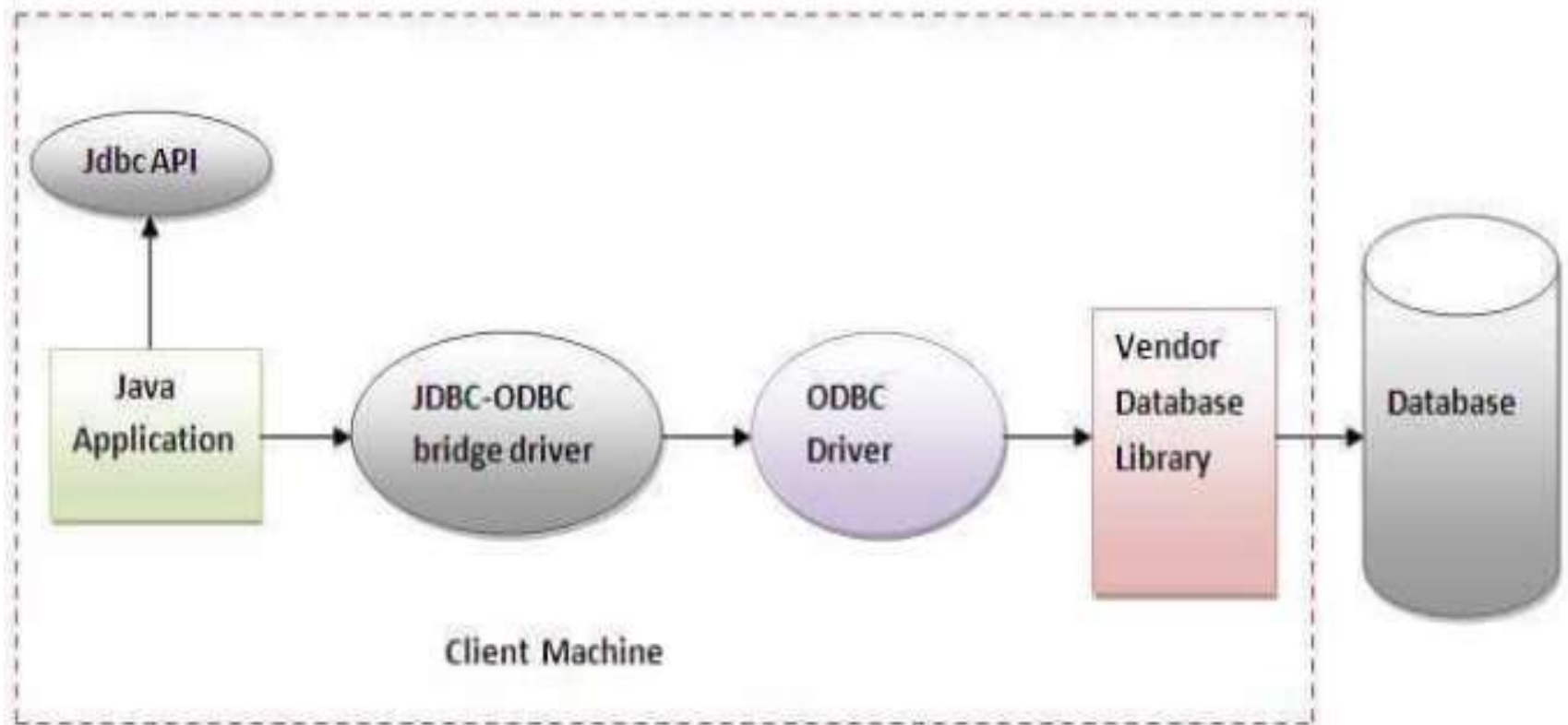


Figure- JDBC-ODBC Bridge Driver

Pros and Cons for Type 1 Driver

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 needs to be installed on the client machine.

Type-2 : Native-API driver

- The Native API driver is also known as **Type 2 driver**.
- Type 2 driver uses the native code part instead of ODBC parts. It uses the client-side libraries of the database.
- It is vendor specific driver, so must be installed on each client system. This driver converts the JDBC method call into native call of database.
- The **Oracle Call Interface (OCI)** driver is an example of a Native API Driver.

Type -2 Architecture

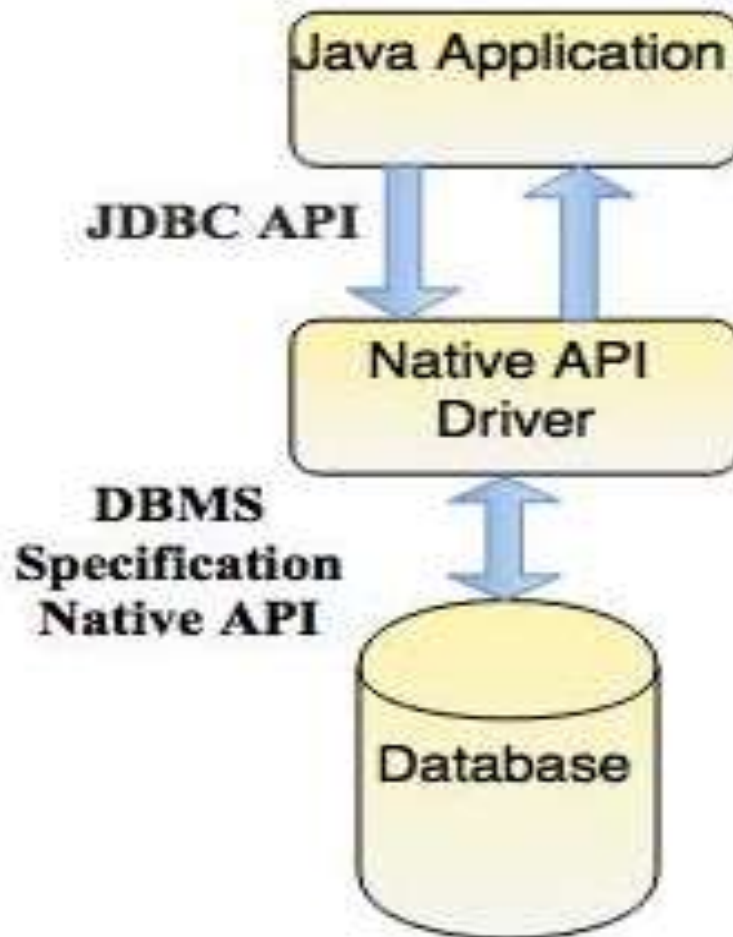


Figure: Native API Driver

Pros and Cons of Type -2

Advantages

- It is faster than type 1 driver.

Disadvantages

- Type 2 driver is platform dependent.
- It uses the vendor class libraries, so needs extra installation on client machine.
- It does not support applet programming.

Type-3: 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.

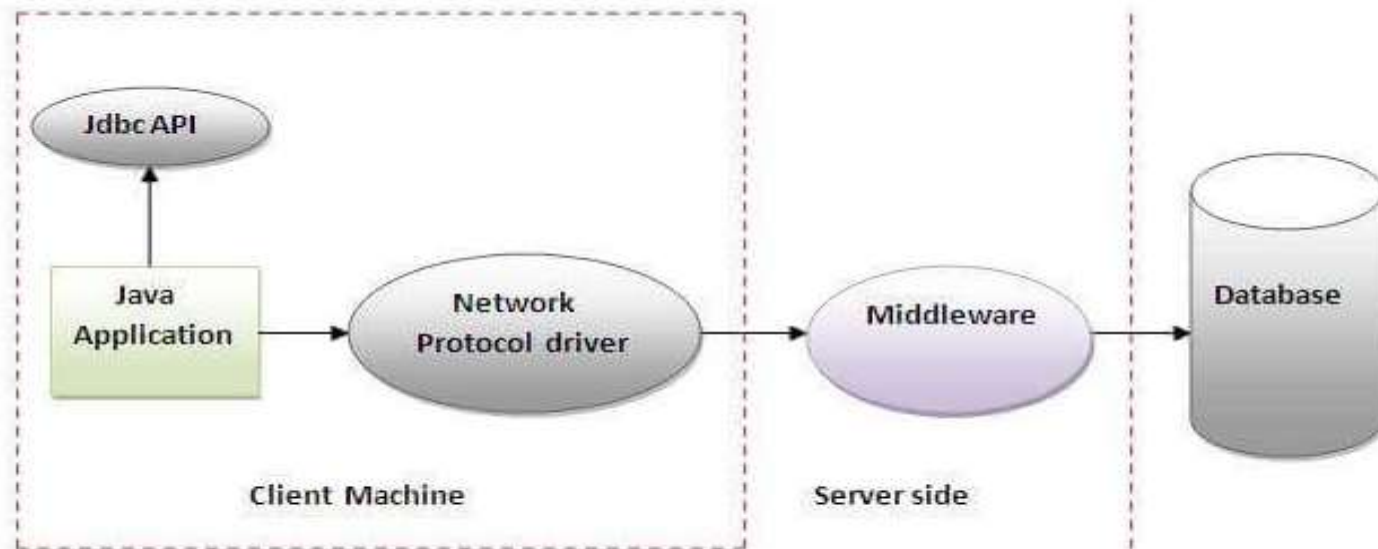


Figure- Network Protocol Driver

Pros and Cons

Advantage:

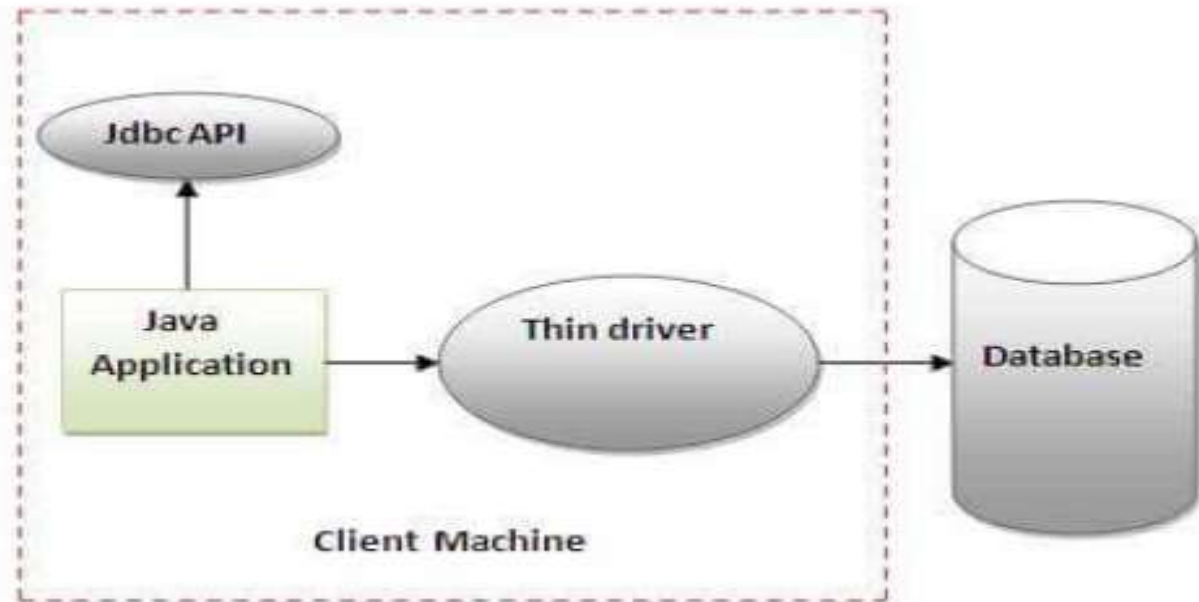
- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

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 4 Driver: 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.



Pros and Cons

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depend on the Database.

MYSQL connectivity using Type 4 Driver for Data Retrieval

```
import java.sql.*;
class MysqlCon
{
public static void main(String args[])
{
    try
    {
        Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax
        Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");

        //here vasavi is the database name, root is the username and root is the password of mysql database
        Statement stmt=con.createStatement();

        ResultSet rs=stmt.executeQuery("select * from emp order by name"); //select * from emp order by name

        while(rs.next())
        {
            System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
        }
        con.close();

    }catch(Exception e)

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

Before executing the program,just try to create database with following tables

```
create database vasavi;
```

```
use vasavi;
```

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

```
insert into emp values(1,'praveen',31);
```

```
insert into emp values(2,'Kumar',30);
```

```
insert into emp values(3,'Sowmya',29);
```

MYSQL connectivity using Type 4 Driver for Data Update

```
import java.sql.*;

class MysqlCon1
{
    public static void main(String args[])
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

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

            Statement stmt=con.createStatement();

            String query1 = "update emp set name='kishore' " + " where id in(1,4)";// here we use Update Query

            stmt.executeUpdate(query1);

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

MYSQL connectivity using Type 4 Driver for Data Deletion

```
import java.sql.*;

class MysqlCon2
{
    public static void main(String args[])
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

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

            //here vasavi is the database name, root is the username and root is the password of mysql database

            Statement stmt=con.createStatement();
            String query1 = "delete from student "+" where stdname='a'"; // delete operation in mysql using type 4
            stmt.executeUpdate(query1);

            con.close();

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

Database Tables for Delete Operation Program

- `CREATE DATABASE "vasavi";`
- `USE "vasavi";`
- `CREATE TABLE /*!32312 IF NOT EXISTS*/ "student" ("stdno" int(50) , "stdname" varchar(50) , "stdmarks" int(10));`
- Try to insert some student details in the above table.s

PreparedStatement

A **PreparedStatement** is a pre-compiled SQL statement. It is a **subinterface** of **Statement**. Prepared Statement objects have some useful additional features than Statement objects. Instead of hard coding queries, PreparedStatement object provides a feature to execute a parameterized query.

When PreparedStatement is created, the SQL query is passed as a parameter. This Prepared Statement contains a pre-compiled SQL query, so when the PreparedStatement is executed, DBMS can just run the query instead of first compiling it.

Method : `prepareStatement()`

Prepared Statement for Retrieving Query in Dynamic Way

```
import java.sql.*;

class MysqlCon3
{
    public static void main(String args[])
    {
        try
        {
            Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax

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

            PreparedStatement ps = con.prepareStatement("SELECT * from emp WHERE name =?"); //Here we use select query under dynamic way
            ps.setString(1,"kishore");

            ResultSet rs=ps.executeQuery();
            while(rs.next())
            {
                System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
            }
            con.close();

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

Prepared Statement for insert the records in Dynamic Way

```
1 //Example on Type 4 driver using Prepared Statement to insert the data into the database
2 import java.sql.*;
3 class MysqlCon4
4 {
5     public static void main(String args[])
6     {
7         try
8         {
9
10            Class.forName("com.mysql.jdbc.Driver");//Type4 Driver Syntax
11
12            Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/vasavi","root","root");
13            PreparedStatement ps = con.prepareStatement("insert into emp values(?,?,?)");
14            ps.setInt(1,7);
15            ps.setString(2,"Ram");
16            ps.setString(3,"35");
17            int res = ps.executeUpdate();
18
19            // Display the records inserted
20            System.out.println(res + " records inserted");
21            con.close();
22        }catch(Exception e)
23        {
24            System.out.println(e);
25        }
26    }
27 }
28 }
```


THANK YOU