# UNIT 4: JDBC

## Introduction to JDBC (Java Database Connectivity)

JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction(API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets. The enterprise data stored in a relational database(RDB) can be accessed with the help of JDBC APIs.

**Definition of JDBC(Java Database Connectivity)**
JDBC is an API(Application programming interface) used in java programming to interact with databases. *The classes and interfaces of JDBC* allow the *application to send* requests *made by users to the specified database.*
**Purpose of JDBC**
Enterprise applications created using the JAVA EE technology need to interact with databases to store application-specific information. So, interacting with a database requires efficient database connectivity, which can be achieved by using the ODBC(Open database connectivity) driver. This driver is used with JDBC to interact or communicate with various kinds of databases such as Oracle, MS Access, Mysql, and SQL server database.

### Components of JDBC

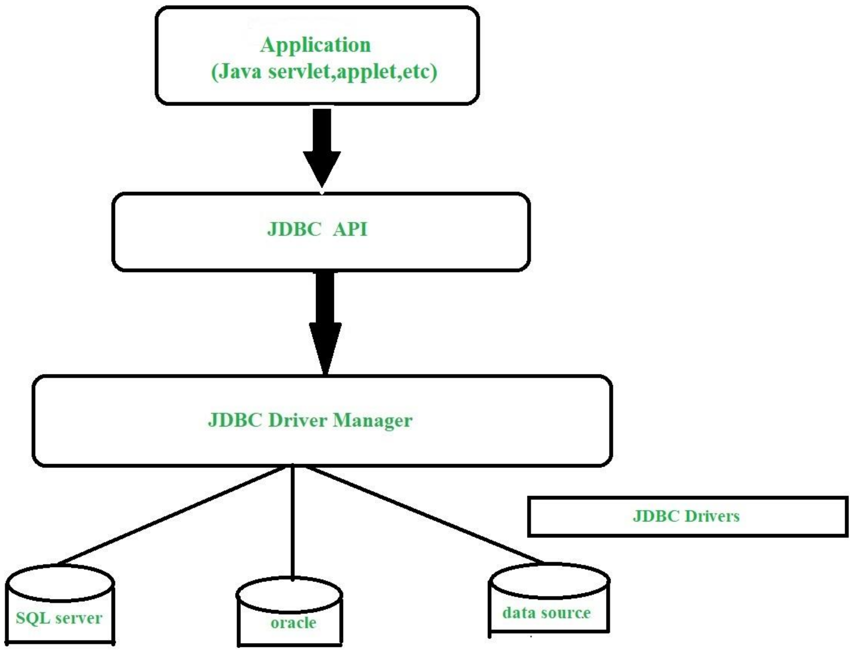
There are generally four main components of JDBC through which it can interact with a database. They are as mentioned below:

**1. JDBC API:** It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE and Java EE platforms to exhibit WORA(write once run anywhere) capabilities.
java.sql.*;

It also provides a standard to connect a database to a client application.

**2. JDBC Driver manager:** It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.
**3. JDBC Test suite:** It is used to test the operation(such as insertion, deletion, updation) being performed by JDBC Drivers.
**4. JDBC-ODBC Bridge Drivers**: It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the **sun.jdbc.odbc** package which includes a native library to access ODBC characteristics.

## Architecture of JDBC



*Architecture of JDBC*

**Description:**

1. **Application:** It is a java applet or a servlet that communicates with a data source.
2. **The JDBC API:** The JDBC API allows Java programs to execute SQL statements and retrieve results. Some of the important classes and interfaces defined in JDBC API are as follows:

3. **DriverManager:** It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.
4. **JDBC drivers:** To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

## JDBC Drivers

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver
3. Type-3 driver or Network Protocol driver
4. Type-4 driver or Thin driver

## Types of JDBC Architecture(2-tier and 3-tier)

The JDBC architecture consists of two-tier and three-tier processing models to access a database. They are as described below:

1. **Two-tier model:** A java application communicates directly to the data source. The JDBC driver enables the communication between the application and the data source. When a user sends a query to the data source, the answers for those queries are sent back to the user in the form of results.
   The data source can be located on a different machine on a network to which a user is connected. This is known as a **client/server configuration**, where the user's machine acts as a client, and the machine has the data source running acts as the server.

2. **Three-tier model:** In this, the user's queries are sent to middle-tier services, from which the commands are again sent to the data source. The results are sent back to the middle tier, and from there to the user.
   This type of model is found very useful by management information system directors.

**Interfaces of JDBC API**

A list of popular *interfaces* of JDBC API is given below:

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface

- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

## Classes of JDBC API

A list of popular *classes* of JDBC API is given below:

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

## Working of JDBC

Java application that needs to communicate with the database has to be programmed using JDBC API. JDBC Driver supporting data sources such as Oracle and SQL server has to be added in java application for JDBC support which can be done dynamically at run time. This JDBC driver intelligently communicates the respective data source.

**Creating a simple JDBC application**

- Java

```
package com.vinayak.jdbc;



import java.sql.*;



public class JDBCDemo {



    public static void main(String args[])

        throws SQLException, ClassNotFoundException

    {

        String driverClassName
    = "sun.jdbc.odbc.JdbcOdbcDriver";

String url = "jdbc:odbc:XE";

String username = "scott";

String password = "tiger";

String query

    = "insert into students values(109, 'bhatt')";



// Load driver class

Class.forName(driverClassName);



// Obtain a connection

Connection con = DriverManager.getConnection(

    url, username, password);



// Obtain a statement

Statement st = con.createStatement();



// Execute the query

int count = st.executeUpdate(query);
```

```
        System.out.println(
            "number of rows affected by this query= "
            + count);



        // Closing the connection as per the
        // requirement with connection is completed
        con.close();
    }
} // class
```

The above example demonstrates the basic steps to access a database using JDBC. The application uses the JDBC-ODBC bridge driver to connect to the database. You must import **java.sql** package to provide basic SQL functionality and use the classes of the package.

## JDBC Driver

1. JDBC Drivers
1. JDBC-ODBC bridge driver
2. Native-API driver
3. Network Protocol driver
4. Thin driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 type

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)

4. Thin driver (fully java driver)

## 1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver conv
calls into the ODBC function calls. This is now discouraged because of thin driver.

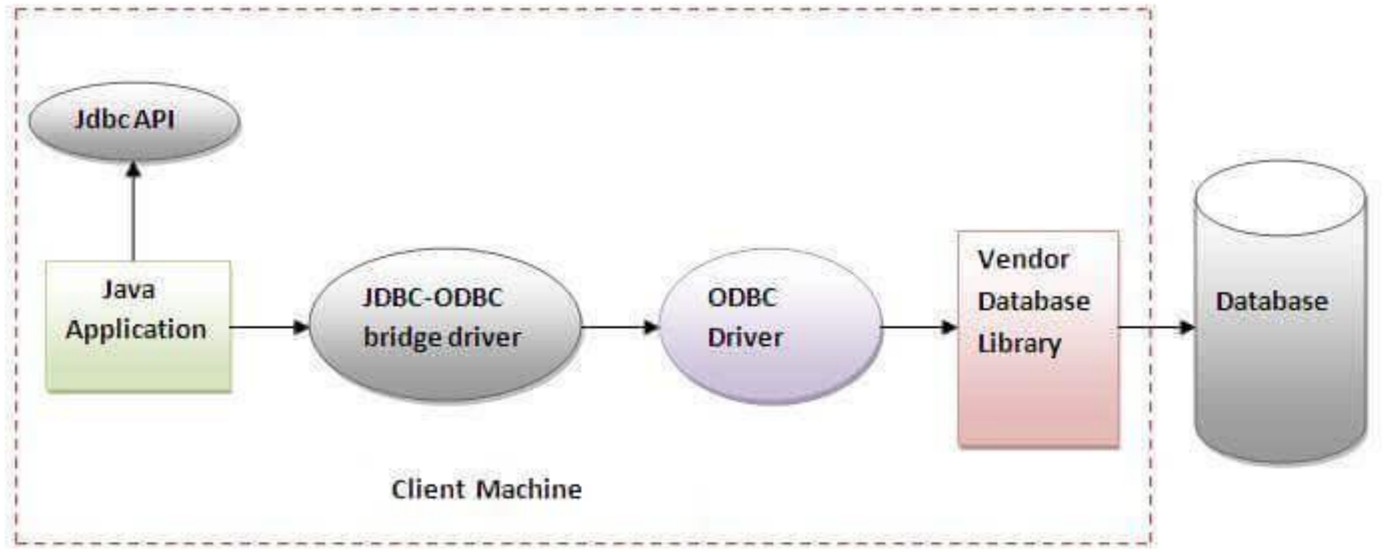


Figure- JDBC-ODBC Bridge Driver

***In Java 8, the JDBC-ODBC Bridge has been removed.***

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

### 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.

## 2) Native-API driver

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

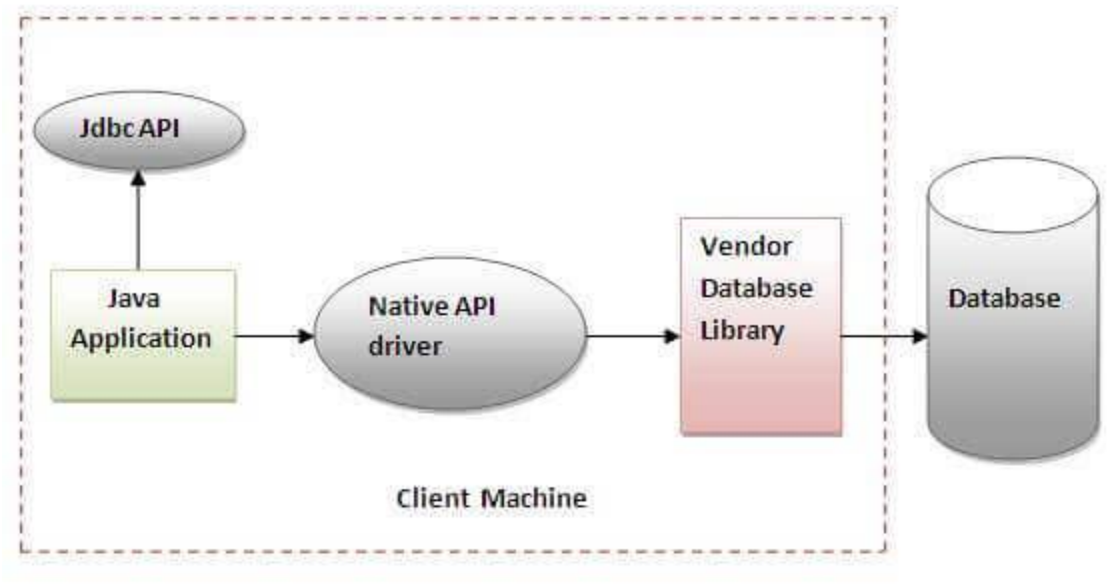


Figure- Native API Driver

### 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.

## 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.

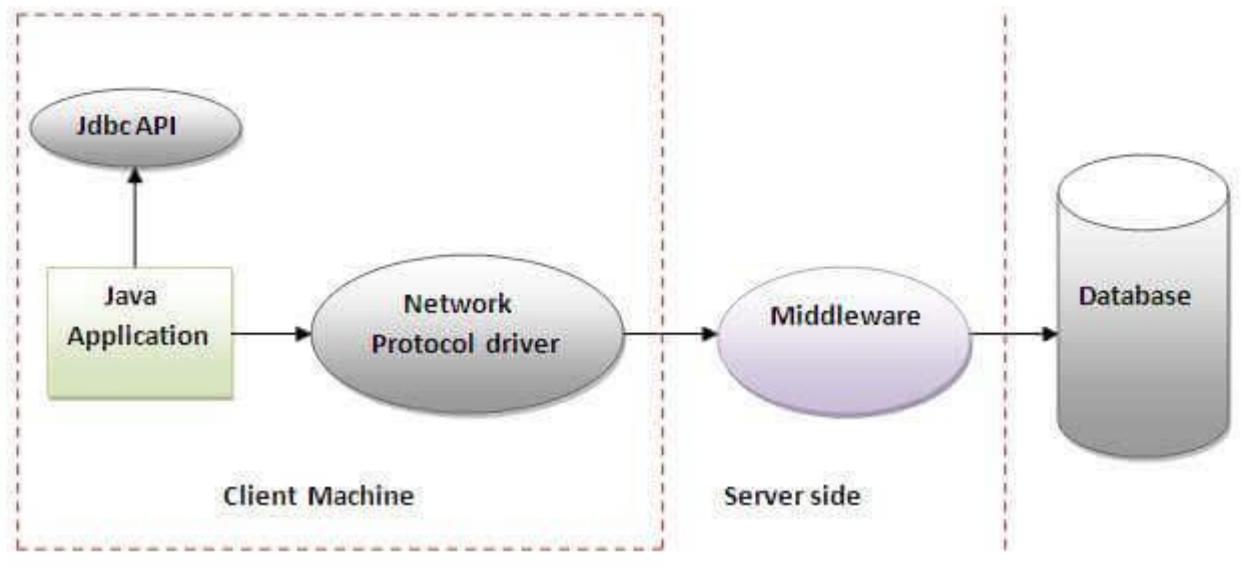


Figure- Network Protocol Driver

### 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.

---

## 4) Thin driver

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

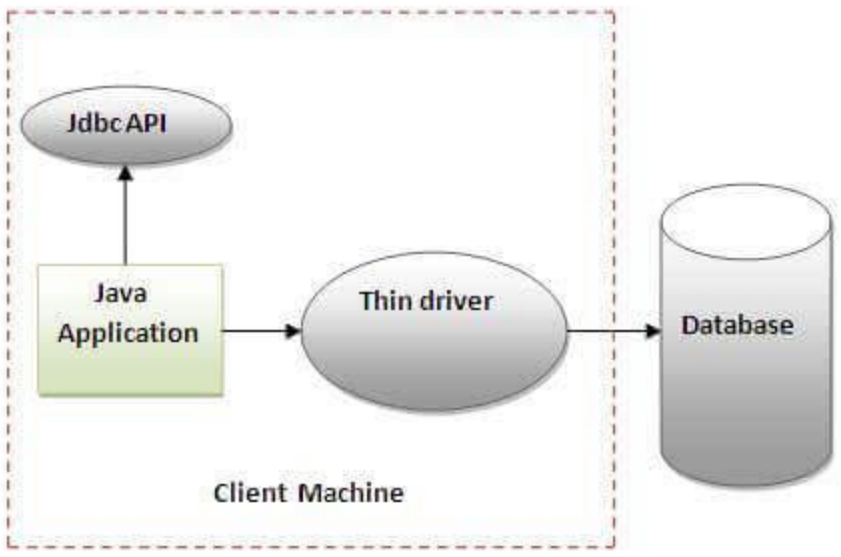


Figure- Thin Driver

## Advantage:

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

## Disadvantage:

- Drivers depend on the Database.

## java.sql

We have relational databases, from which at many times we need to access the data. For various data processing related matters from RDDBMS we have java.sql package. The various classes in the package are shown below:

| **Class** | **Description** |
|---|---|
| Date | It gives time in milliseconds. It is a wrapper type. It provides sql with dates. The class is declared as: public class Date extends Date The class methods are inherited from date class. |
| DriverManager | The class is designed for managing the various JDBC drivers. The class is declared as follows: public class DriverManager extends Object The class methods are inherited from Object class. |
| DriverPropertyInfo | The class keeps an account for managing the various properties of JDBC drivers which are required for making a secure connection. The class is declared as follows: public class DriverPropertyInfo extends Object The class methods are inherited from Object class. |
| SQLPermission | The class manages the various SQL related permissions which are provided to the accessing objects. The class is declared as follows: public final class SQLPermission extends BasicPermission The class methods are inherited from BasicPermission class. |
| Time | It is wrapper class around java.util. The class provides time related information. The class is declared as follows: public class Time extends Date The class methods are inherited from Date class. |

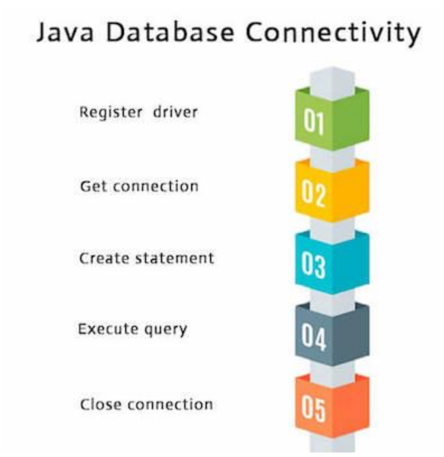
| | |
|---|---|
| Timestamp | It is wrapper class around java.util. The class allows JDBC API to identify as TIMESTAMP value. The class is declared as follows: public class Timestamp extends Date The class methods are inherited from Date class. |
| Types | The class defines various SQL constants. The class is declared as follows: public class Types extends Object The class methods are inherited from Object class. |

## Java Database Connectivity with 5 Steps

1. 5 Steps to connect to the database in java
1. Register the driver class
2. Create the connection object
3. Create the Statement object
4. Execute the query
5. Close the connection object

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



## 1) Register the driver class

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

### Syntax of forName() method

```
public static void forName(String className)throws ClassNotFoundException
```

***Note: Since JDBC 4.0, explicitly registering the driver is optional. We just need to put vender's Jar in the classpath, and then JDBC driver manager can detect and load the driver automatically.***

## Example to register the OracleDriver class

Here, Java program is loading oracle driver to esteblish database connection.

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

## 2) Create the connection object

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

### Syntax of getConnection() method

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

## Example to establish connection with the Oracle database

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

## 3) Create the Statement object

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

### Syntax of createStatement() method

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

## Example to create the statement object

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

## 4) Execute the query

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

### Syntax of executeQuery() method

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

## Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");

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

---

## 5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Conn
used to close the connection.

### Syntax of close() method

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

## Example to close connection

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

***Note: Since Java 7, JDBC has ability to use try-with-resources statement to automatically close resources of type Connection, ResultSet, and Statement.***

It avoids explicit connection closing step.