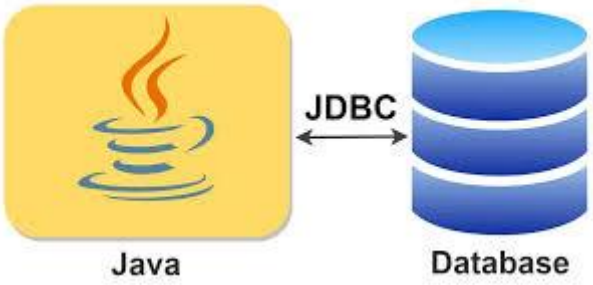
JDBC(Java Database Connectivity)

Created By Foram Chudasama

Java JDBC

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database.

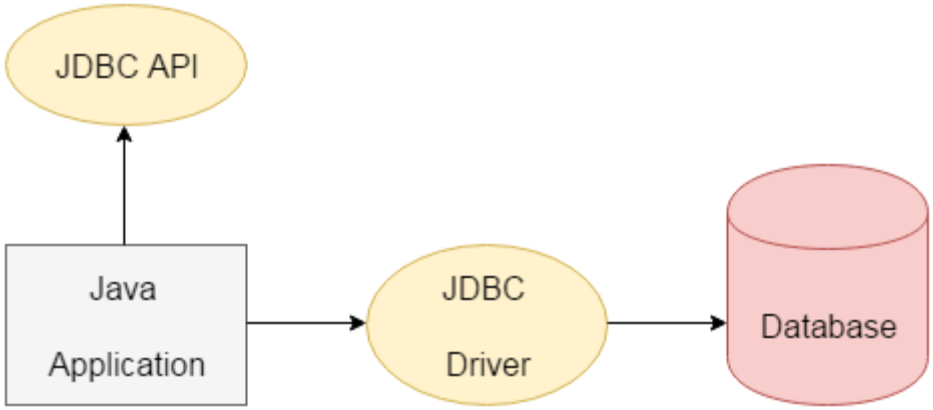


It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

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.

Java Database Connectivity



The **java.sql** package contains classes and interfaces for JDBC API. 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

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

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

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

JDBC Driver

JDBC Driver is a software component that enables java application

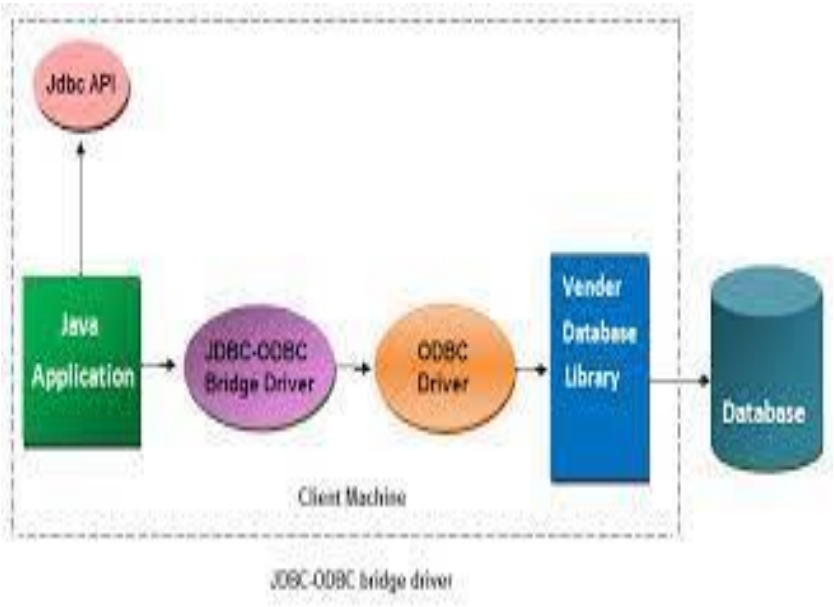
To interact with the database. There are 4 types of JDBC drivers:

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 converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.



JDBC-ODBC bridge driver

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 native calls of the database API. It is not written entirely in java.

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

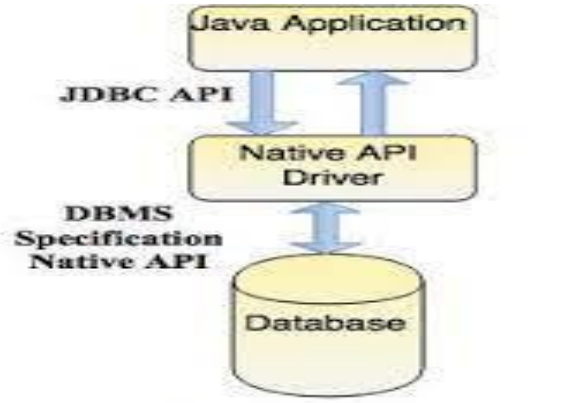


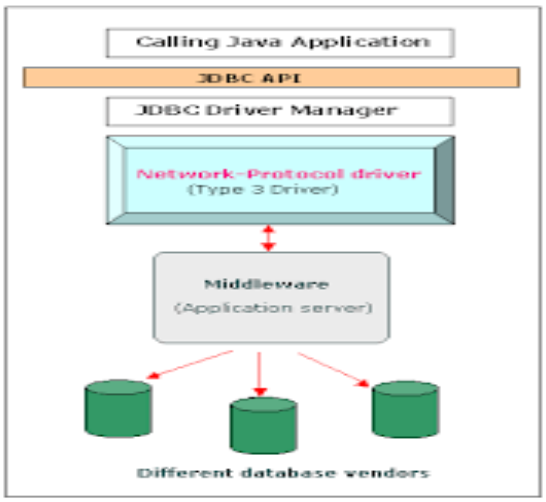
Figure: Native API Driver

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.



Advantage:

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

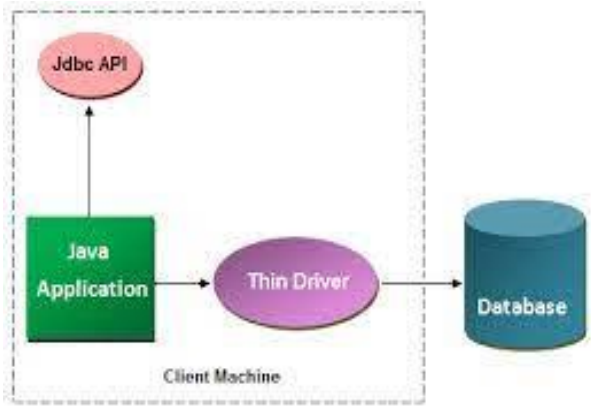
Disadvantages:

- 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 as thin driver.

It is fully written in Java language.



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 Database Connectivity with 5 Steps

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

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 of forName() method

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

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

```
public static Connection getConnection(String url)throws SQLException
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:8001:ex","root","password"
);
```

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 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 returns the object of 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 Connection interface is used to close the connection.

Syntax of close() method

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

Example to close connection

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

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database
   is **jdbc:mysql://localhost:3306/employee** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and **employee** is the database name. We may use any database, in such case, we need to replace the **employee** with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

first create a table in the mysql database, but before creating table, we need to create database first.

1. create database employee;
2. use employee;
3. create table emp(id **int**(10),name varchar(40),age **int**(3));