JDBC

Connect to Relational DBs via Java

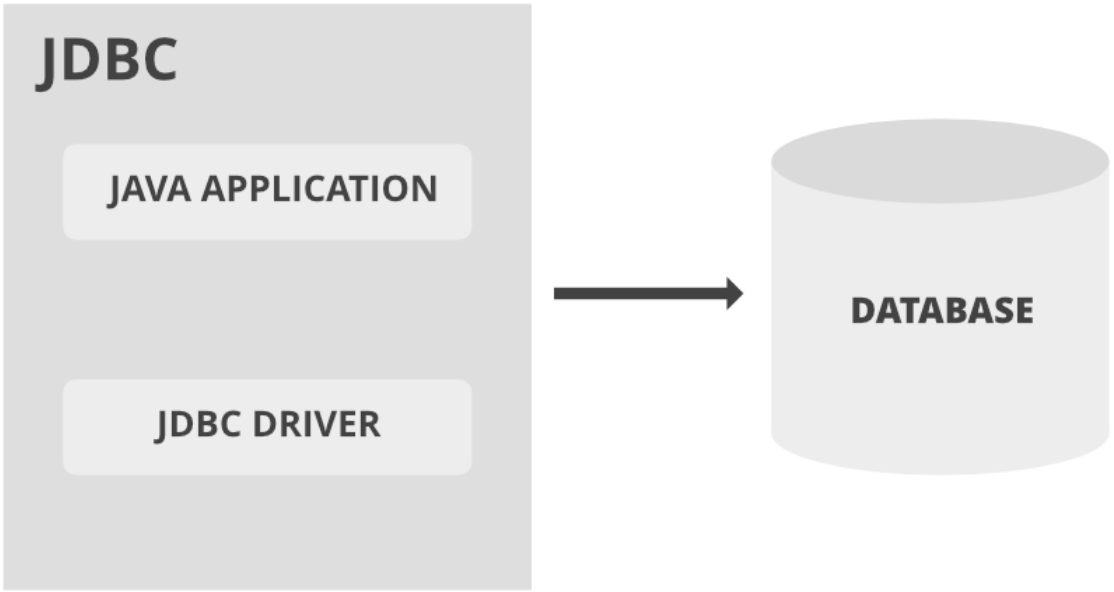
IBM Client Innovation Center - Bari

Bari, */06/2023

IBM Client Innovation Center
Italy

What is JDBC?

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute statements against a database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. In the following, a pictorial representation of the JDBC architecture.



Why Should We Use JDBC

Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

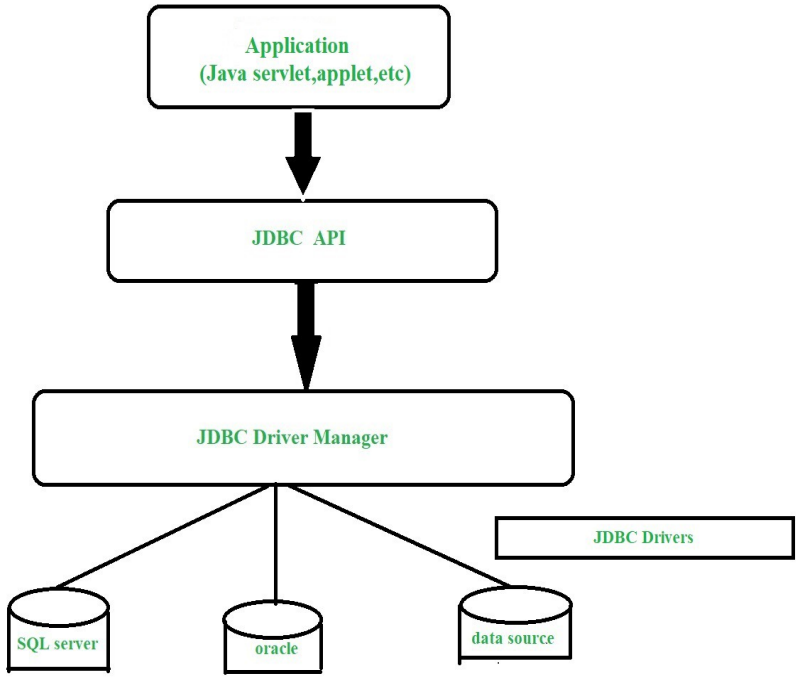
We can use JDBC API to handle database using Java program and can perform the following activities:

- Connect to the database
- Execute queries and update statements to the database
- Retrieve the result received from the database.

What is API?

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

JDBC Architecture



So from the close image, we will be discussing out major 5 terminologies encountered which are as follows:

1. Driver Software
2. Statement object
3. ResultSet
4. Connection object
5. SQL query

JDBC Environment Setup

Now let us do discuss out various types of drivers in JDBC.

So basically there are 4 types of standard drivers listed below:

- Type-1 driver or JDBC-ODBC bridge driver (*Bridge Driver*)
- Type-2 driver or Native-API driver (*Native API*, partially java driver)
- Type-3 driver or Network Protocol driver (*Network Protocol*, fully java driver)
- Type-4 driver or Thin driver (*Native protocol*, fully java driver)

Which driver we will use? The solution to this is as simple as we need to use the driver as per accordance with the database with which we are interacting.

Where available, we will prefer Type-4 driver.

1. JDBC-ODBC bridge driver

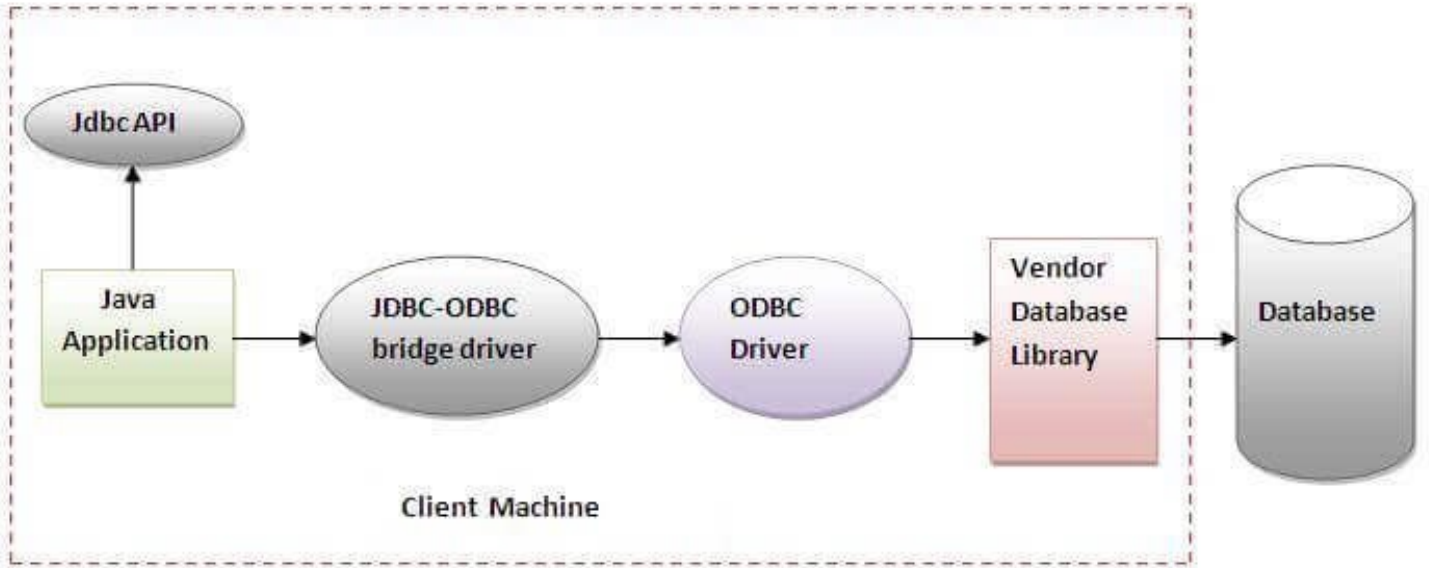


Figure- JDBC-ODBC Bridge Driver

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

2. Native-API driver

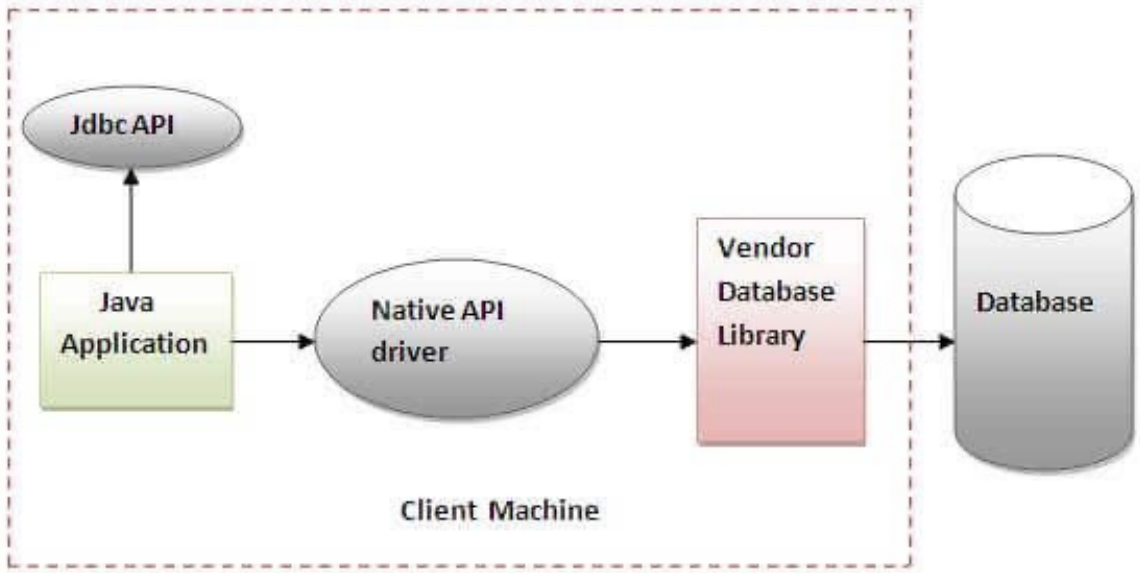


Figure- Native API Driver

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

3. Network Protocol driver

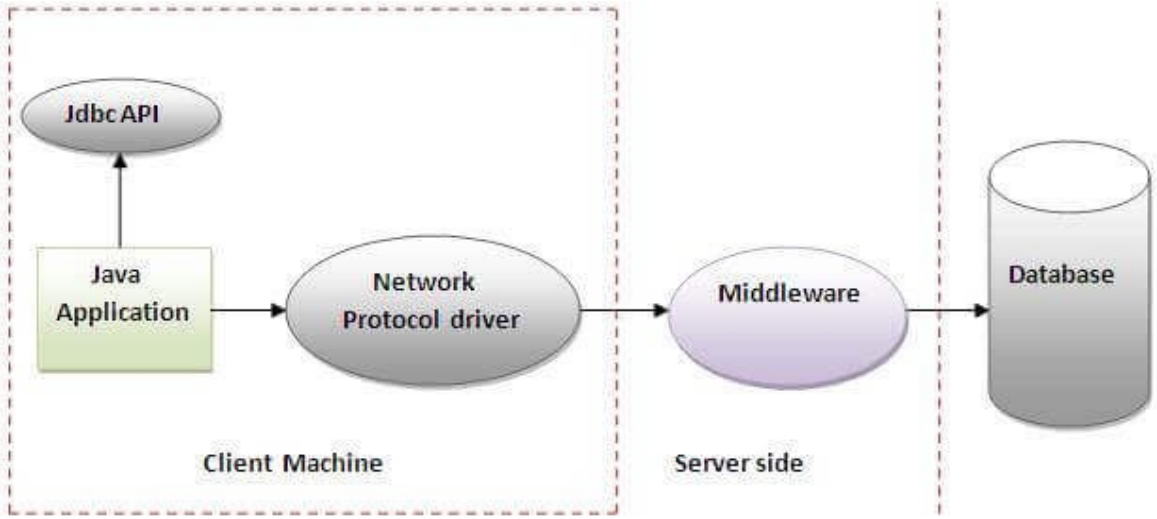


Figure- Network Protocol Driver

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.

4. Thin driver

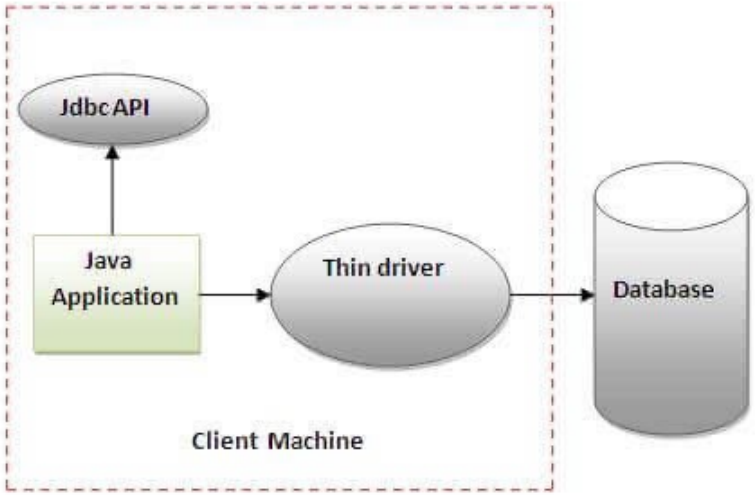


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.

- No software is required at client side or server side.

Disadvantage:

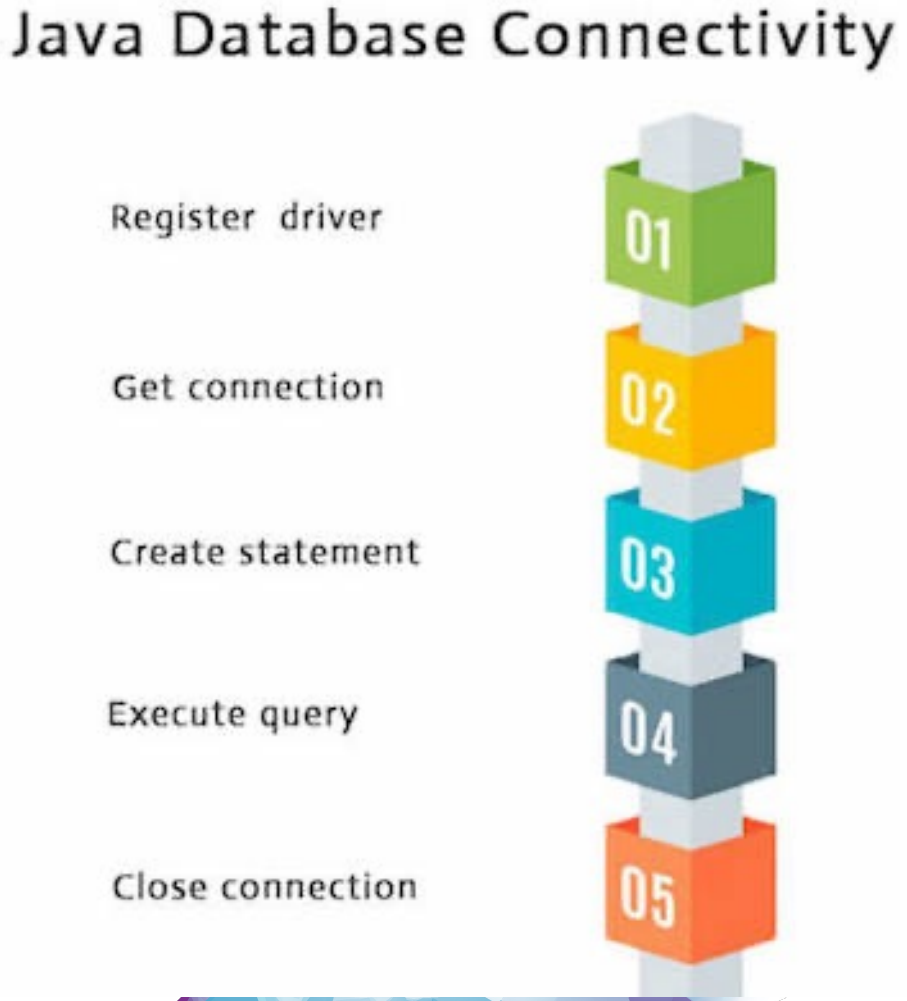
Drivers depend on the Database.

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.

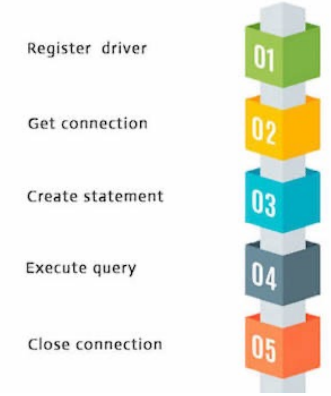
Java Database Connectivity with 5 Steps

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



JDBC 1) Register the driver Class



Import the required package for the corresponding database (add the jar to Java ClassPath).

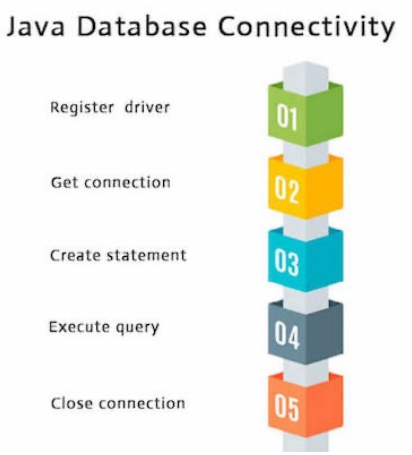
The **forName()** method of *Class* class is used to register the driver class. This method is used to dynamically load the driver class.

Example:

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

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

JDBC 2) Create the connection object



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

1) **public static** Connection getConnection(String url) **throws** SQLException

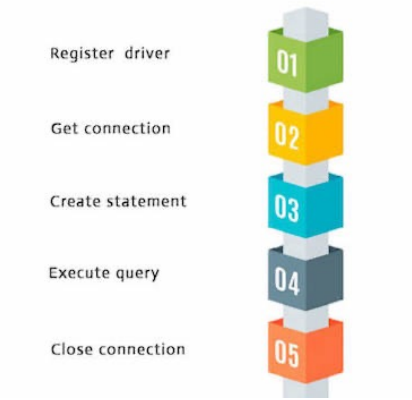
2) **public static** Connection getConnection(String url, String name, String password) **throws** SQLException

Example:

```
Connection con=DriverManager.getConnection(

        "jdbc:mysql://<server-name>:<server-port>/<database name>",”<name>",”<password>");
```

JDBC 3) Create the Statement object



The **createStatement()** method of *Connection* interface is used to create statement.

The object of statement is responsible to execute statements with the database.

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

Example:

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

JDBC 4) Execute the SQL Statement

There are 3 different methods on the *Statement* interface.

- **executeQuery(**String sql**)** 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.
- **executeUpdate(**String sql**)** method of *Statement* interface is used to execute SQL statements which update or modify database.
- **execute(**String sql**)** method of *Statement* interface is used for any kind of SQL statements.

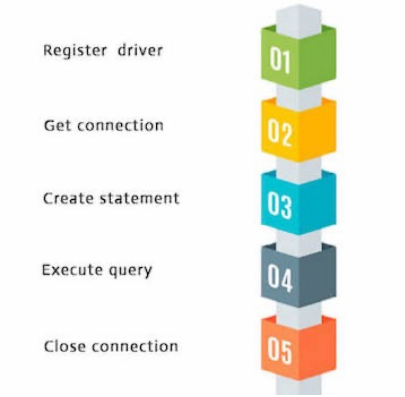
```
public ResultSet executeQuery(String sql) throws SQLException
```

Example:

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

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

JDBC 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.

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

Example:

```
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.

ResultSet Interface

The object of *ResultSet* maintains a cursor pointing to a row of a table.

Initially, cursor points to before the first row.

Note: By default, ResultSet object can be moved forward only and it is not updatable.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

Example:

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
```

ResultSet Interface (cont.)

Commonly used methods of *ResultSet* interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

ResultSet Interface: example

Let's see the simple example of ResultSet interface to retrieve the data of 3rd row.

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");

//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));

con.close();
}}
```