# JDBC:

1. JDBC stands for Java Database Connectivity.
2. It is used for connecting a Java program to some database.
3. JDBC is a Java API to connect and execute the query with the database.
4. It is a part of Java SE (Java Standard Edition).
5. There are four types of JDBC drivers:
6. JDBC-ODBC Bridge Driver,
7. Native Driver,
8. Network Protocol Driver,
9. Thin Driver
10. We can use JDBC API to access tabular data stored in any relational database.
11. By the help of JDBC API, we can save, update, delete and fetch data from the database.
12. It is like Open Database Connectivity (ODBC) provided by Microsoft.



1. The current version of JDBC is 4.3.
2. It is the stable release since 21st September, 2017.
3. It is based on the X/Open SQL Call Level Interface.
4. The **java.sql** package contains classes and interfaces for JDBC API.
5. 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

1. Before JDBC, ODBC API was the database API to connect and execute the query with the database.
2. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured).
3. 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.

## What is API (Application programming interface):

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

# JDBC Driver

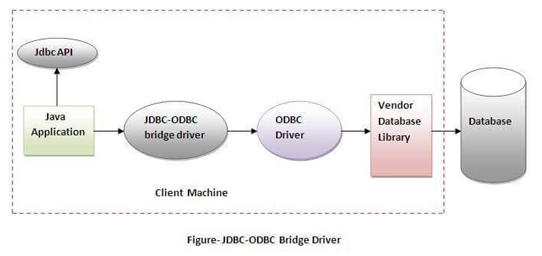
JDBC Driver is a software component that enables java application to interact with the database.

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)

### 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. |



#### 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.

### 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.

### Disadvantage:

* The Native driver needs to be installed on the each client machine.
* The Vendor client library needs to be installed on client machine.

### 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.

### 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. |



### 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:

1. Register the Driver class
2. Create connection
3. Create statement
4. Execute queries
5. Close connection

Connectivity with Access without DSN

There are two ways to connect java application with the access database.

1. Without DSN (Data Source Name)
2. With DSN

Java is mostly used with Oracle, mysql, or DB2 database.

### Example to Connect Java Application with access without DSN

In this example, we are going to connect the java program with the access database. In such case, we have created the login table in the access database. There is only one column in the table named name. Let's get all the name of the login table.

**import** java.sql.\*;

**class** Test

{

**public** **static** **void** main(String ar[])

{

**try**{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

//Here database exists in the current directory

String database="student.mdb";

String url="jdbc:odbc:Driver= {Microsoft Access Driver (\*.mdb)};

                     DBQ=" + database + ";DriverID=22;READONLY=**true**";

   Connection c=DriverManager.getConnection(url);

   Statement st=c.createStatement();

   ResultSet rs=st.executeQuery("select \* from login");

**while**(rs.next()){

    System.out.println(rs.getString(1));

   }

}**catch**(Exception ee){System.out.println(ee);}

}

}

### Example to Connect Java Application with access with DSN

Connectivity with type1 driver is not considered good. To connect java application with type1 driver, create DSN first, here we are assuming your dsn name is mydsn.

**import** java.sql.\*;

**class** Test

{

**public** **static** **void** main(String ar[])

{

 t**ry**

{

   String url="jdbc:odbc:mydsn";

   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

   Connection c=DriverManager.getConnection(url);

   Statement st=c.createStatement();

   ResultSet rs=st.executeQuery("select \* from login");

**while**(rs.next())

{

     System.out.println(rs.getString(1));

    }

}

**catch**(Exception ee)

{

System.out.println(ee);

}

}

}