**JDBC ( UNIT -5 )**

Q) What is JDBC?

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

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.



**Why Should We Use JDBC? Need of 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).

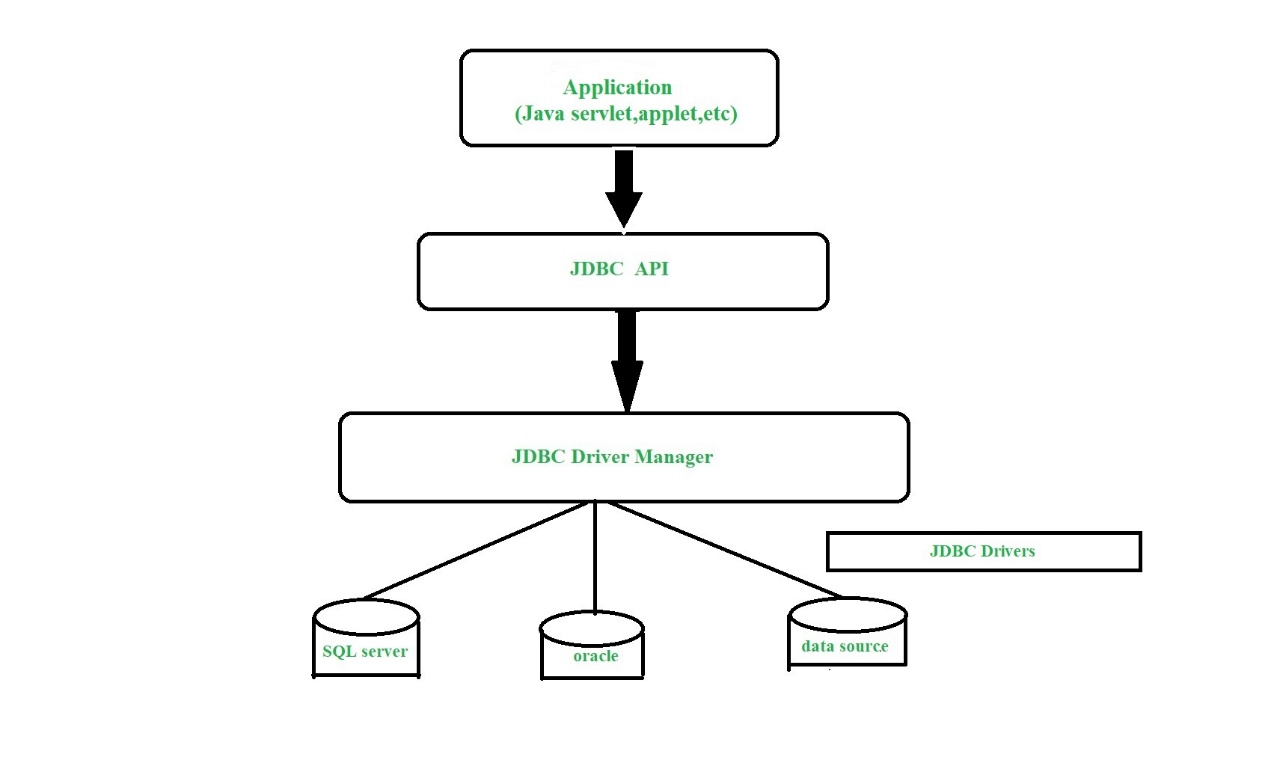
We can use JDBC API to handle database using Java program and can perform the following activities:

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

JDBC architecture?

**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.



Q. What is JDBC driver? Types of drivers \*\*\*\*?

# DBC 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.  **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.

**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. |



**Advantage:**

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

**Disadvantage:**

* Drivers depend on the Database.

Q. What are the steps to connect to a jdbc database. Show demo with a prpgram?

Ans. Here 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 load the driver class. |

### 2) Create the connection object

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

### 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. |

### 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. |

### 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. |

import java.sql.\*;

class OracleCon{

public static void main(String args[]){

try{

//step1 load the driver class

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

//step2 create the connection object

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object

Statement stmt=con.createStatement();

//step4 execute query

ResultSet rs=stmt.executeQuery("select \* from emp"); **//THIS IS A SQL QUERY TO RETRIEVE ALL THE RECORDS FROM THE DATABASE**

while(rs.next())

System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));

//step5 close the connection object

con.close();

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

}

}