SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF COMPUTER APPLICATIONS

23CAT606 – JAVA PROGRAMMING

I YEAR II SEM

UNIT III – Networking & IO Packages

TOPIC – Java Database Connectivity:

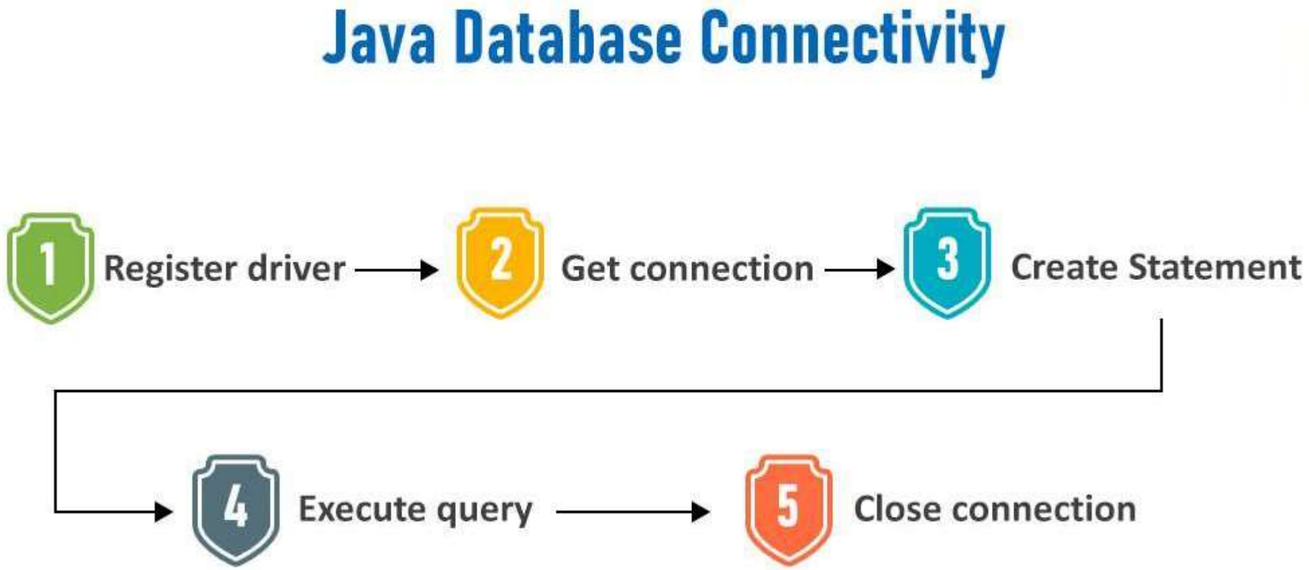
Drivers

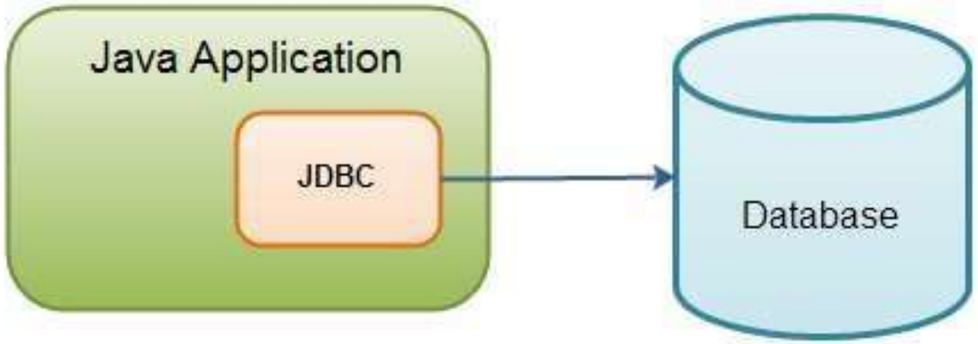
Connection

Query Execution

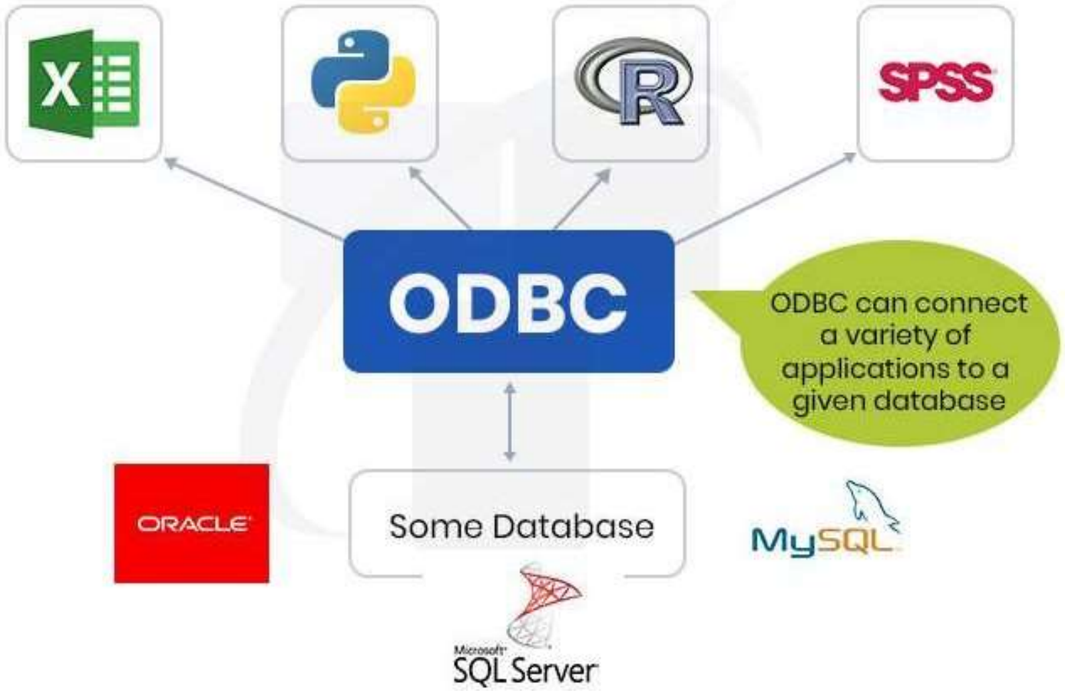
Java Database Connectivity

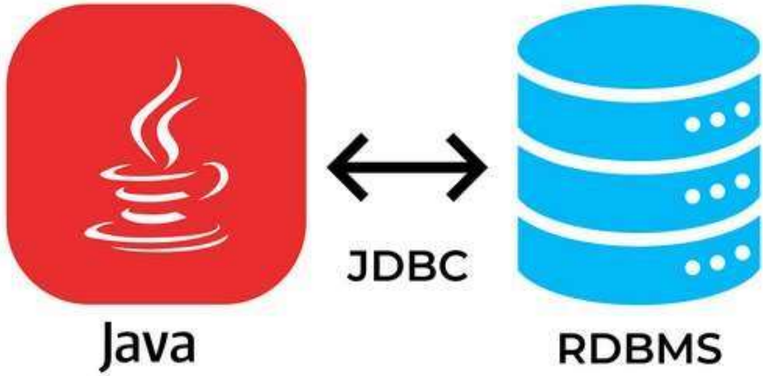
Definition: The term JDBC is an abbreviation for Java Database Connectivity. It is a built-in API available in Java that is used to connect with databases and execute queries. JDBC is specifically available in JavaSE.





Why should learn JDBC?





Drawback:

1.Written in C Programming

2. No Security

3. Platform Dependent

Highlights:

1. Java API

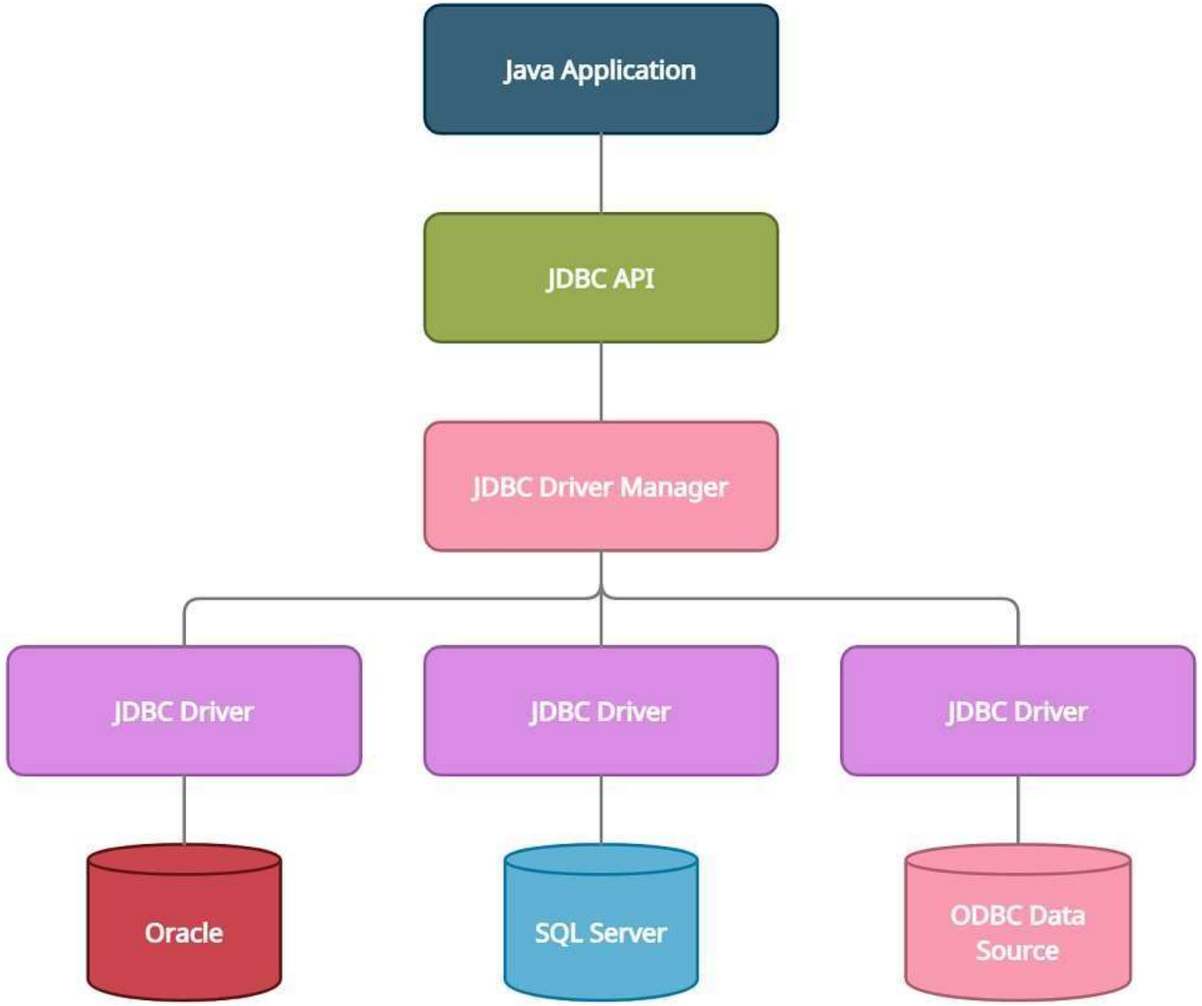
2. Secured

3. Platform Independent

JDBC Architecture

JDBC API can be used to do the following:

1. Directly connect to a database
2. Creating SQL statements and queries
3. Execute queries (with checks) to the database
4. Retrieve database objects such as Tables, Indices, etc. from the database.



The JDBC Architecture has two layers, they are

1. JDBC API − Provides connection from Application to JDBC Manager.
2. JDBC Driver API – Supports the connection between JDBC Manager and the Driver.

JDBC Driver

JDBC in Java uses a driver manager and a database-specific driver to connect to a database. The JDBC driver manager makes sure that the correct driver is being used to access the databases. It is also capable of handling multiple drivers connected to multiple databases simultaneously.

Types of JDBC Drivers

1. JDBC – ODBC Bridge Driver
2. Native Driver
3. Network Protocol Driver
4. Thin Driver

JDBC – ODBC Bridge Driver

1. It uses ODBC driver to connect to the database.
2. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.
3. This is now discouraged because of thin driver.

Advantages:

1. Easy to use.
2. Can be easily connected to any database.

Disadvantages:

1. Performance degraded because JDBC method call is converted into the ODBC function calls.
2. The ODBC driver needs to be installed on the client machine.

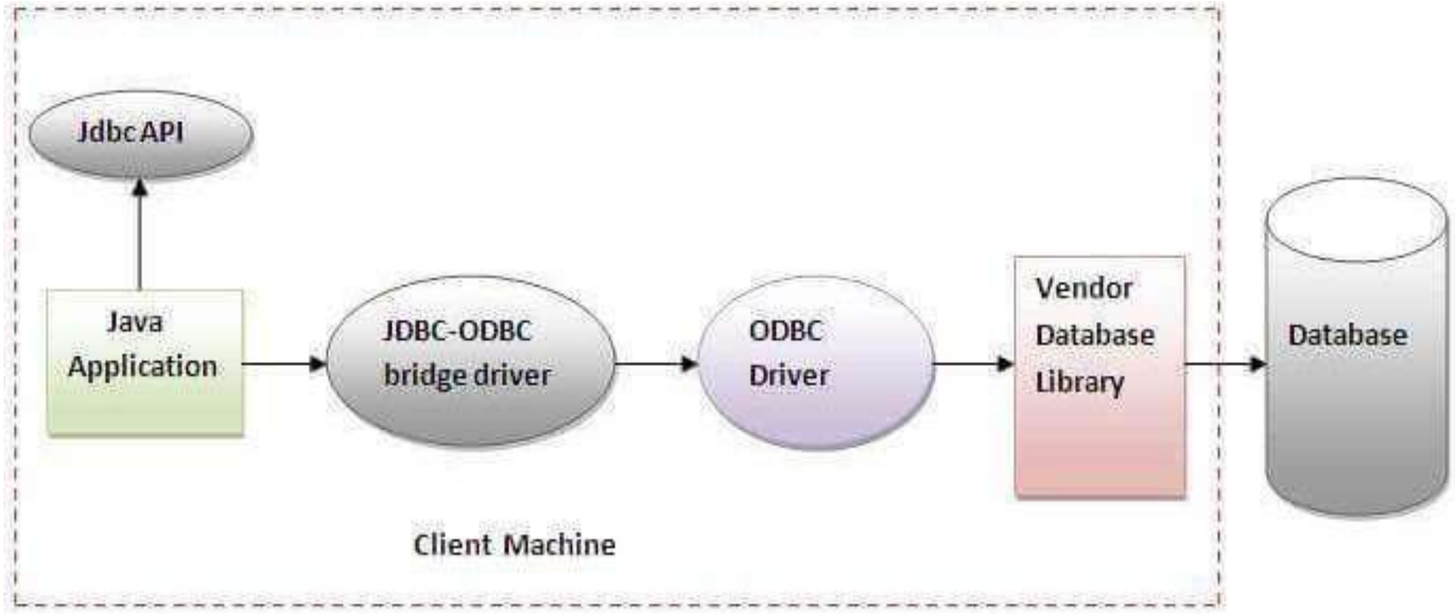


Figure- JDBC-ODBC Bridge Driver

Native-API driver

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

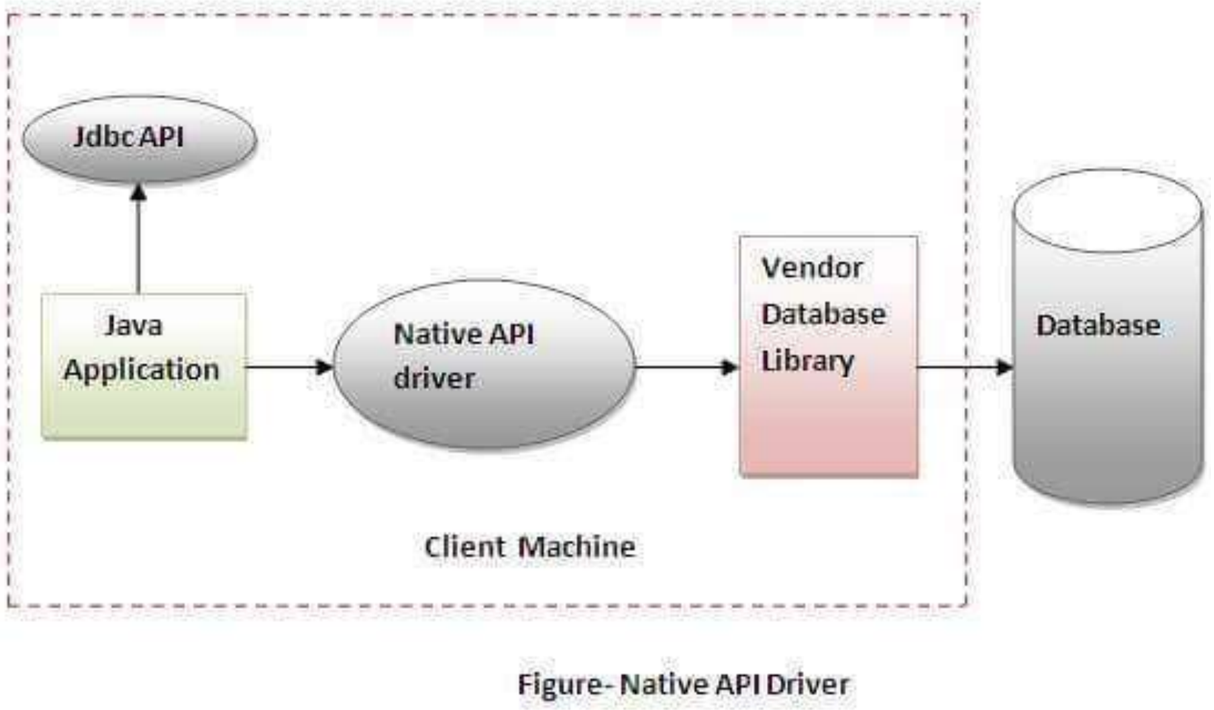


Figure- Native API Driver

Advantage:

1. Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

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

Network Protocol driver

1. 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:

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

Disadvantages:

1. Network support is required on client machine.
2. Requires database-specific coding to be done in the middle tier.
3. Maintenance of Network Protocol driver becomes costly

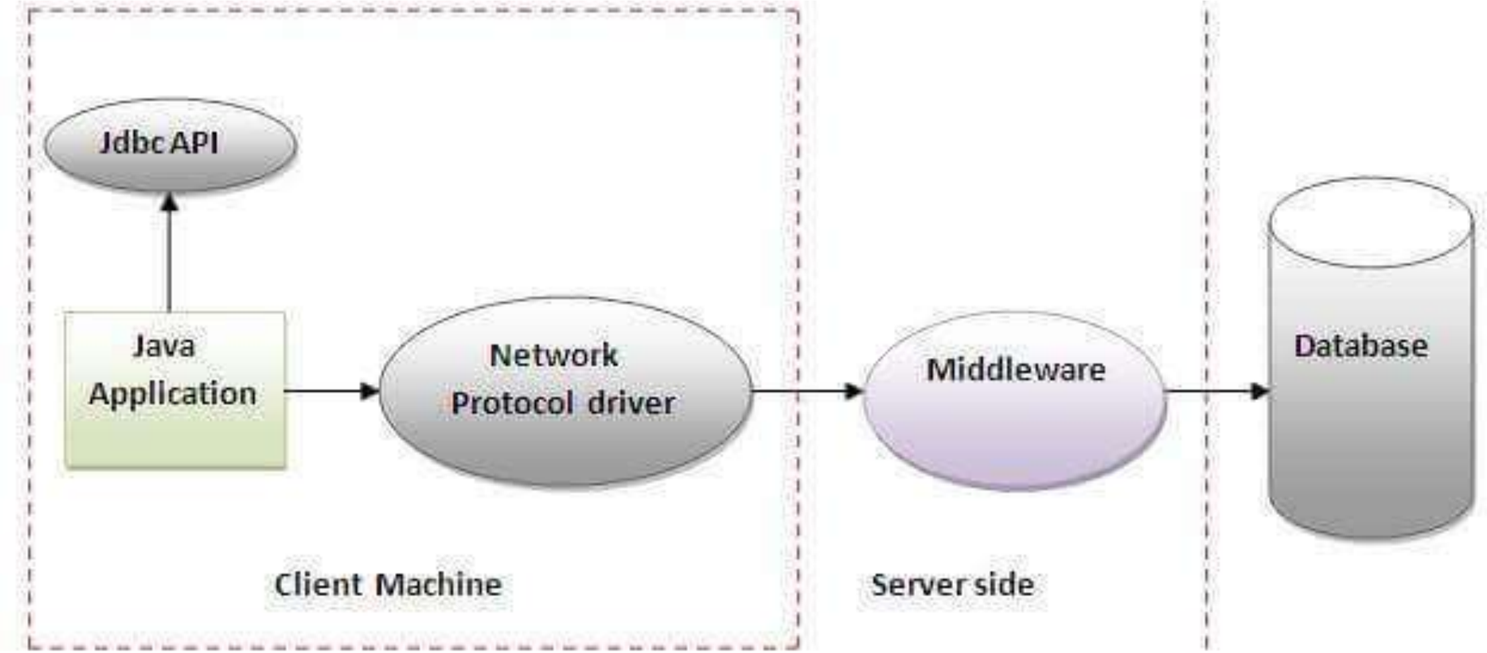


Figure- Network Protocol Driver

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.

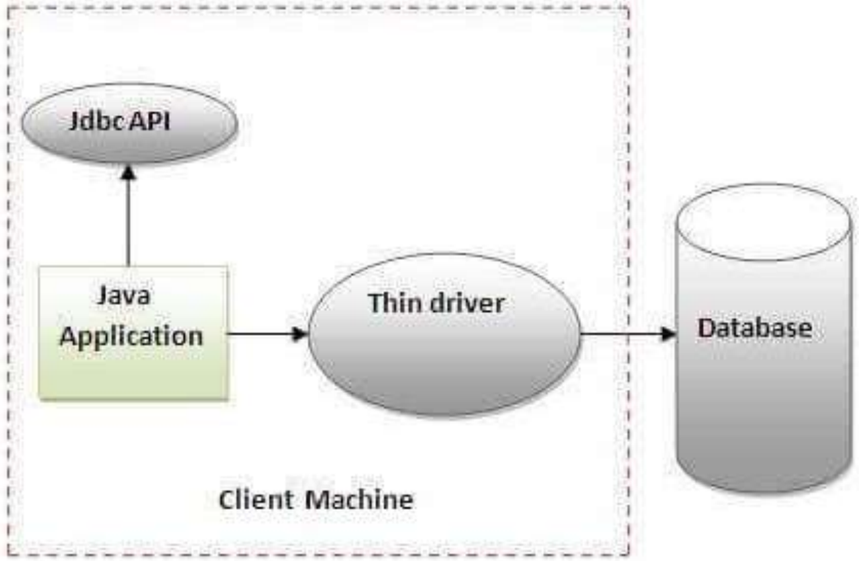


Figure- Thin Driver

Java Database Connectivity with 5 Steps

1 :import Packages

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

2: Register Database Driver to JDBC

```
Class.forName("Database Driver");
```

3: Open Connection

```
Connection connection=DriverManager.getConnection("jdbc:Database url", "username", "password");
```

4: Create Statement

```
Statement smt = conn.createStatement();
```

5: Execute Query

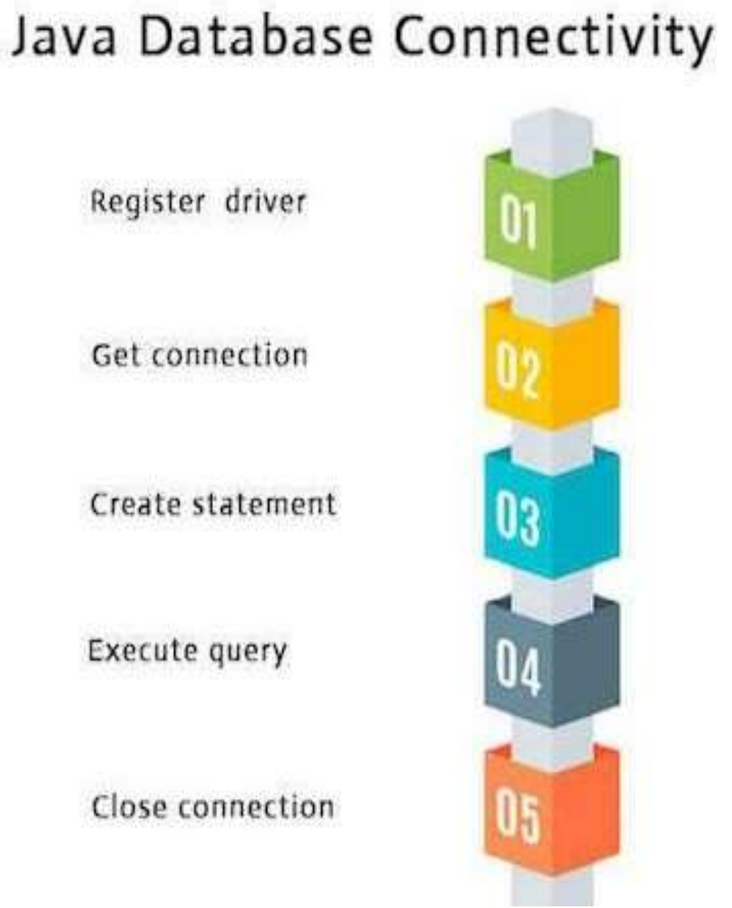
```
ResultSet rs=smt.executeQuery("SQL statement");
```

6: Extract Database

```
rs.next()
int i=rs.getInt(index);
int i=rs.getInt(columnName);
```

7: Close connection

```
connection.close()
```



Steps by step: Load the drive

Two ways:

1. **Class.forName()**
2. **DriverManager.registerDriver()**

Public static void registerDriver(driver)

Class.forName()

DB Name	JDBC Driver Name
MySQL	com.mysql.jdbc.Driver
Oracle	oracle.jdbc.driver.OracleDriver
Microsoft SQL Server	com.microsoft.sqlserver.jdbc.SQLServerDriver
MS Access	net.ucanaccess.jdbc.UcanaccessDriver

1.DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())

2.DriverManager.registerDriver(new com.microsoft.sqlserver.jdbc.SQLServerDriver())

Steps by step: Establish Connection, Create the Statement object

Two ways:

1. **getConnection(URL,username,password);**
2. **getConnection(URL);**

String database = "D:\\Users\\TCP\\Documents\\Database1.accdb";

String url = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=" + database;

Connection con = DriverManager.getConnection(url);

Create statement

1.Statement stmt=con.createStatement();

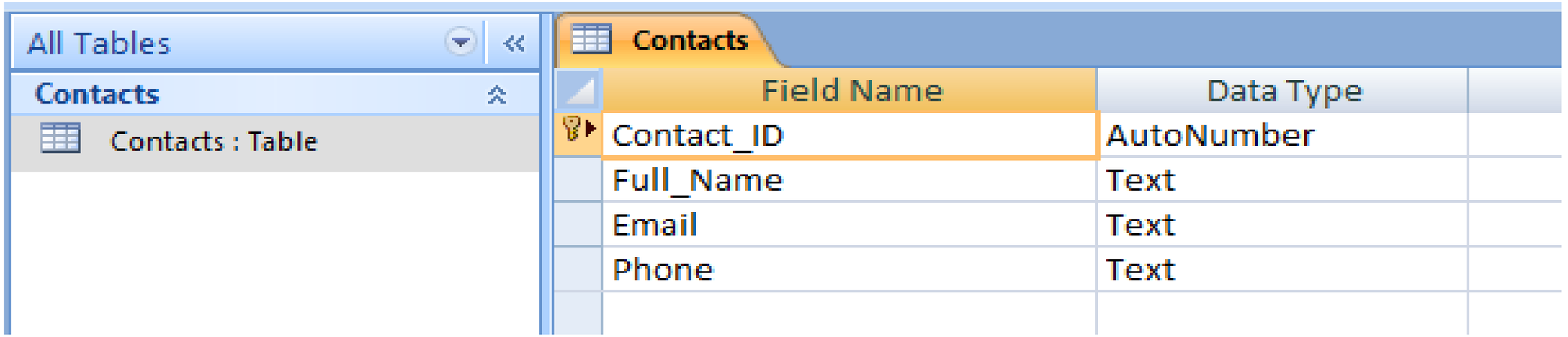
Execute the query

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

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

Close the connection con.close();

Example


All Tables
Contacts
Contacts : Table
Contacts
Field Name
Data Type
Contact_ID
AutoNumber
Full_Name
Text
Email
Text
Phone
Text

Example

```
public class JdbcAccessTest {

    public static void main(String[] args) {

        String databaseURL = "jdbc:ucanaccess://e://Java//JavaSE//MsAccess//Contacts.accdb";

        Connection connection = DriverManager.getConnection(databaseURL)

        String sql = "INSERT INTO Contacts (Full_Name, Email, Phone) VALUES (?, ?, ?)";

      PreparedStatement preparedStatement = connection.prepareStatement(sql);

      preparedStatement.setString(1, "Raj");

      preparedStatement.setString(2, "raj@gmail.com");

      preparedStatement.setString(3, "0919989998");

     int row = preparedStatement.executeUpdate();

     if (row > 0) {

         System.out.println("A row has been inserted successfully.");

      }
```

Example

```
sql = "SELECT * FROM Contacts";
Statement statement = connection.createStatement();
ResultSet result = statement.executeQuery(sql);
        while (result.next()) {
            int id = result.getInt("Contact_ID");
            String fullname = result.getString("Full_Name");
            String email = result.getString("Email");
            String phone = result.getString("Phone");
            System.out.println(id + ", " + fullname + ", " + email + ", " + phone);
        }
    }   }
```


Thank
you