



Java Programming

Unit 5 Interacting with Database





Course Objective(CO)

- Develop java programs using Database



What is JDBC?

- JDBC stands for **J**ava **D**atab**a**se **C**onnectivity
- 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.
- Functions of JDBC are
 - Making a connection to a database.
 - Creating SQL or MySQL statements.
 - Executing SQL or MySQL queries in the database.
 - Viewing & Modifying the resulting records.



Applications of JDBC

- JDBC is used for design and development of
 - Java Applications
 - Java Applets
 - Java Servlets
 - Java ServerPages (JSPs)
 - Enterprise JavaBeans (EJBs).



The JDBC architecture

- consists of two-tier and three-tier processing models to access a database.

1. Two-tier model:

1. A java application communicates directly to the data source.
2. The JDBC driver enables the communication between the application and the data source.
3. When a user sends a query to the data source, the answers for those queries are sent back to the user in the form of results.
4. The data source can be located on a different machine on a network to which a user is connected.
5. This is known as a **client/server configuration**, where the user's machine acts as a client and the machine having the data source running acts as the server.

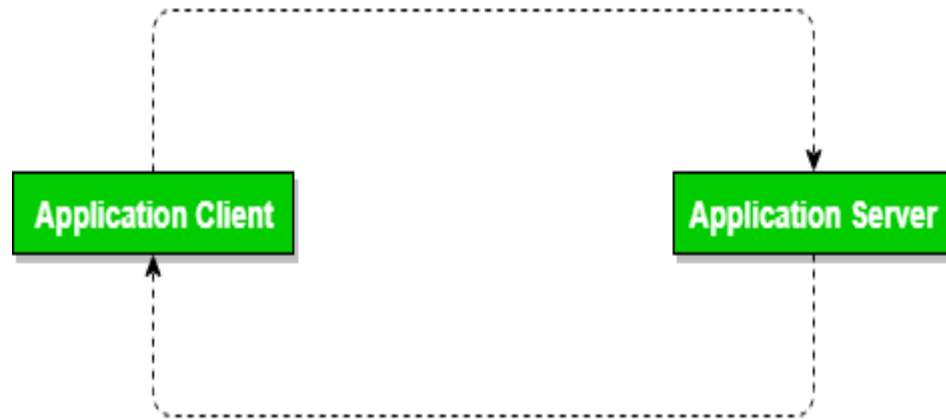
2-tier Architecture

Advantages:

Easy to understand, implement and maintain

Disadvantage:

Not suitable for large amount of data





Three Tier architecture:

- another layer between the client and the server
- The client does not directly communicate with the server.
- This intermediate layer acts as a medium for exchange of data between server and client.
- This type of architecture is used in case of large web applications.

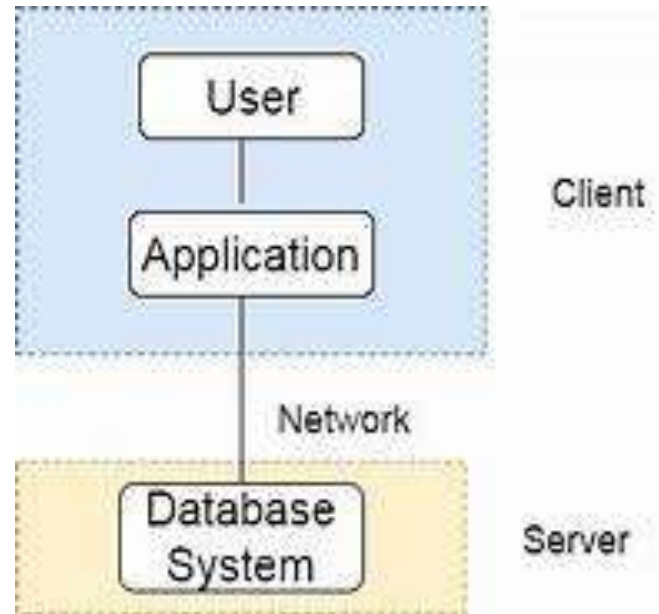
Advantages:

- Enhanced scalability, Data Integrity, Security

■ **Disadvantages:**

complexity of implementation and communication

Figure of three tier architecture

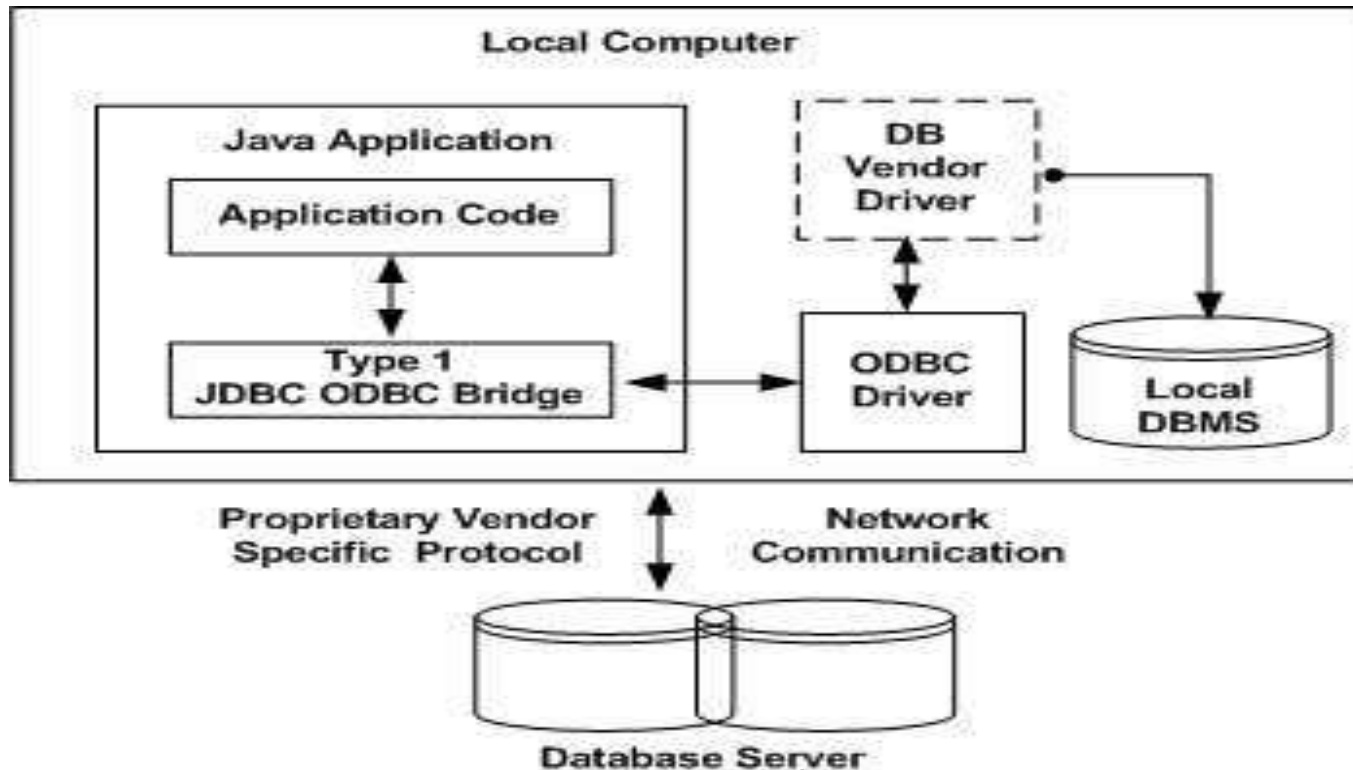




JDBC Driver

- JDBC Driver is a software component that enables java application to interact with the database.
- There are 4 types of JDBC drivers:
 - JDBC-ODBC bridge driver
 - Native-API driver (partially java driver)
 - 100% pure java, JDBC-network
 - 100% pure java(Thin driver) (fully java driver)

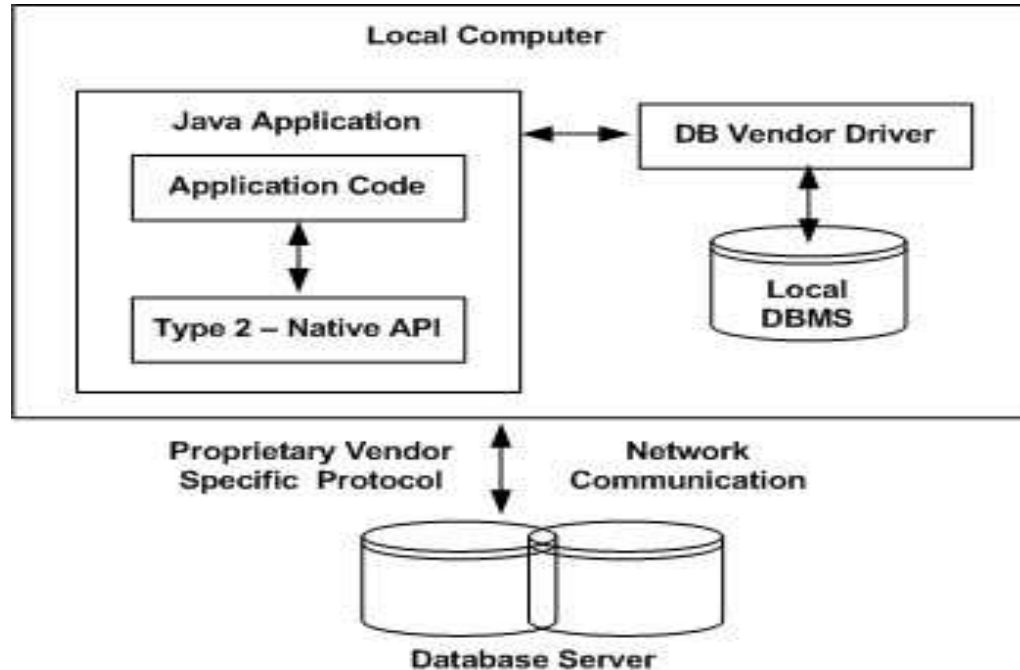
Type1 JDBC/ODBC Bridge



Advantages: Easy to use & Free of cost

Disadvantages: Slow, Platform Dependent, More time, ODBC drivers must be loaded on client

JDBC Type2

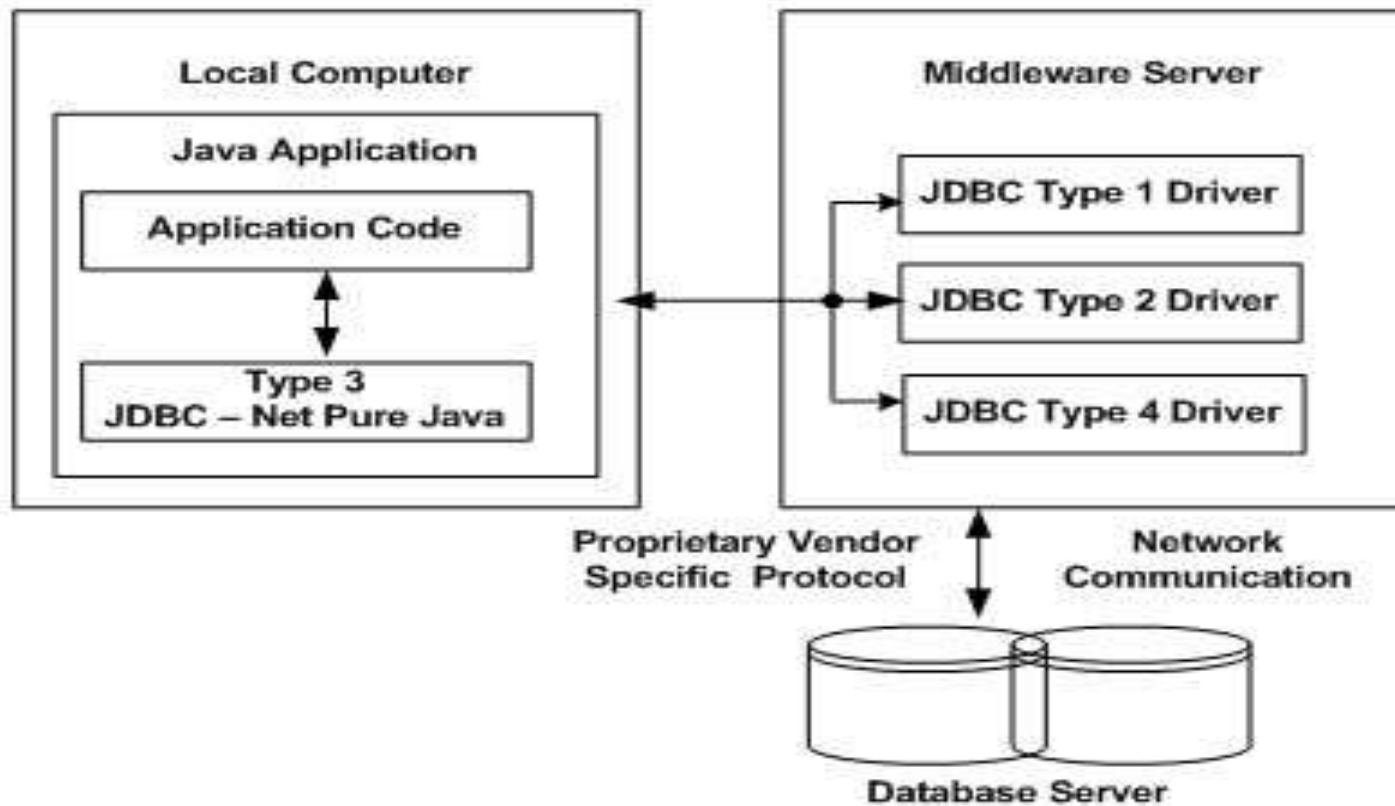


OS specific, same as Type1, JDBC converted to native API(C,C++)

Adv: speed is increased, performance improved

Disadv: less portable, native APIs must be loaded to target machine

JDBC Type3: 100% pure java, JDBC-network





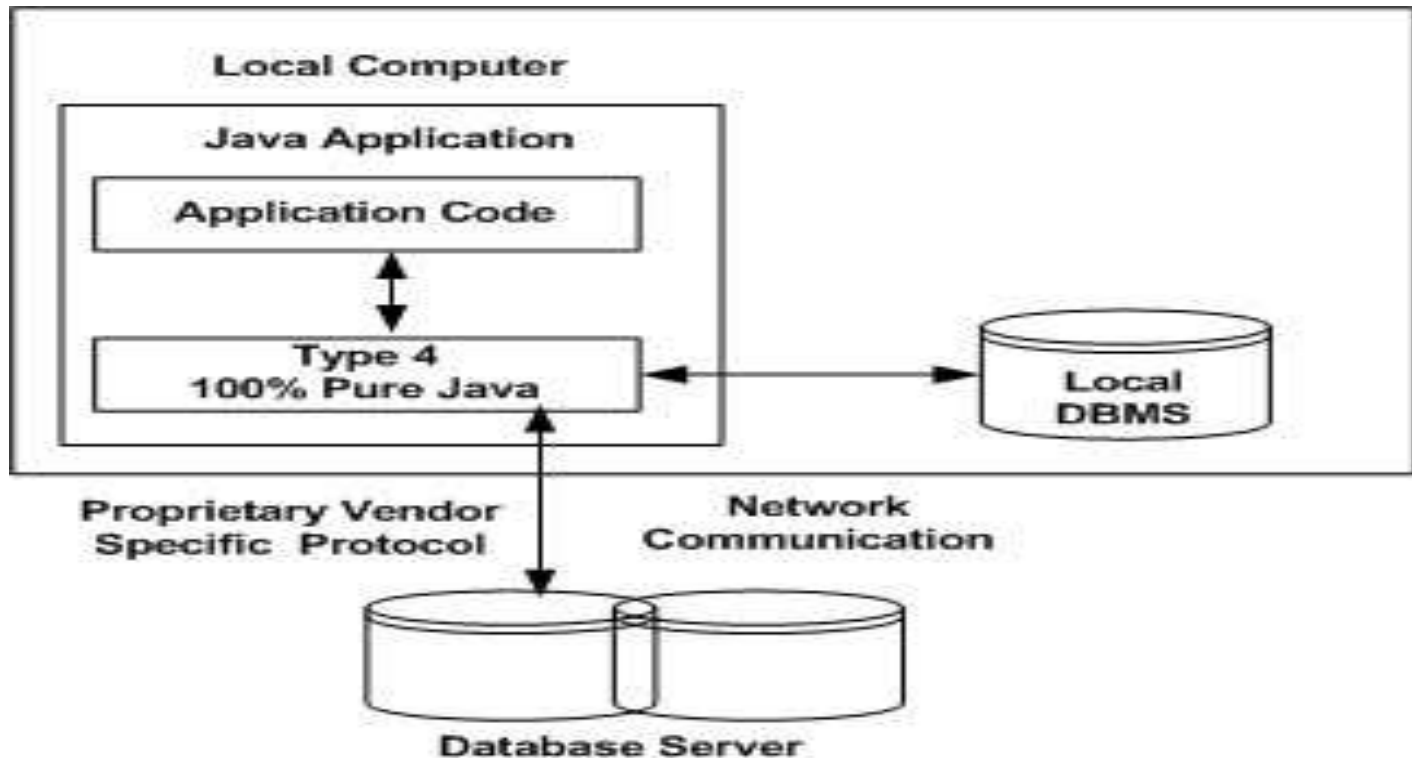
Type3

- 3-tire architecture
- Network application
- Socket programming
- Advantages: speed is increased
flexible & versatile
deployment is easy
support for networking
no need of client machine installation

Disadvantages:

establishment of network connection is tedious.

JDBC Type 4: 100% Java





Type 4

- Implemented entirely in Java
- It has database-specific network protocols to communicate directly with database
- Database vendors usually supply type4 driver.



JDBC INTERFACES

INTERFACES	Description
Connection	Connects java program to database
Statement	Execute normal SQL statement
PreparedStatement	Execute dynamic SQL statement
ResultSet	Accepts results from SQL statement
DatabaseMetaData	Information about database driver & interfaces for each Vendor database
ResultSetMetaData	Information about result set returned from database



Java Objects

Objects	Description
Date	Accept database date values
<i>DriverManager</i>	Another way to connect database
DriverPropertyInfo	Manage Driver object
Time	Accept database time value
TimeStamp	
Types	Provides list of predefined integer values that can be used in JDBC



JDBC Exceptions

Name	Description
DataTruncation Exception	If Unexpectedly truncates values
SQLException Exception	Thrown by all methods of JDBC API
SQLWarning Exception	When database issues any warning



DriverManager class

1. getConnection() method

- creates connection to database
- returns connection object
- 3 forms are available

1. getConnection(String url)

2. getConnection(String url, Property arg)

3. getConnection(String url, String userid, String password)

String url="jdbc:odbc:xyz"



Connection Interface

- Provides methods to handle transaction processing
- Provides some methods for error handling
- `Connection con = DriverManager.getConnection(url)`
- Above line will create connection to database.



Statement Interface

- `createStatement()`: creates statement object and returns it.

Syntax:

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

This line will create Statement object & this object will execute SQL commands.

- Here con is object of Connection interface
- SQL queries will *be static*



Statement Interface Methods

- `execute()`-boolean method
- Eg: create table
- `executeQuery()`-return type is integer
- Usefull for delete,insert &update commands
- Returns how many records are inserted,deleted or updated
- `executeUpdate()`-return tpye is object of ResultSet class
- Eg: select query



ResultSet Class

- Methods & Description
- **public void beforeFirst() throws SQLException**
Moves the cursor to just before the first row
- **public void afterLast() throws SQLException**
Moves the cursor to just after the last row
- **public boolean first() throws SQLException**
Moves the cursor to the first row
- **public void last() throws SQLException**
Moves the cursor to the last row.



ResultSet class

- **public boolean absolute(int row) throws SQLException**

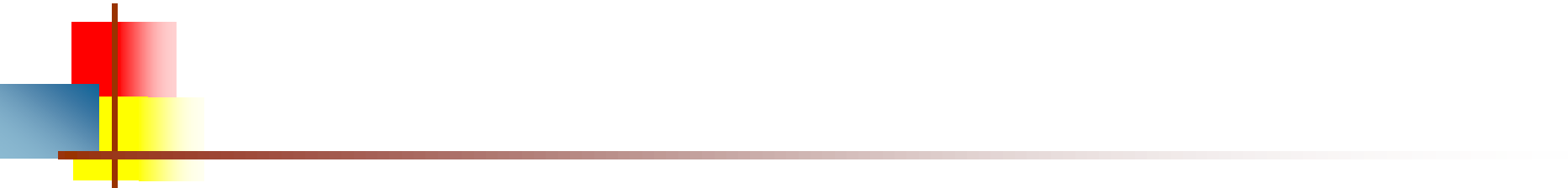
Moves the cursor to the specified row

- **public boolean relative(int row) throws SQLException**

Moves the cursor the given number of rows forward or backwards from where it currently is pointing.

- **public boolean previous() throws SQLException**

Moves the cursor to the previous row. This method returns false if the previous row is off the result set

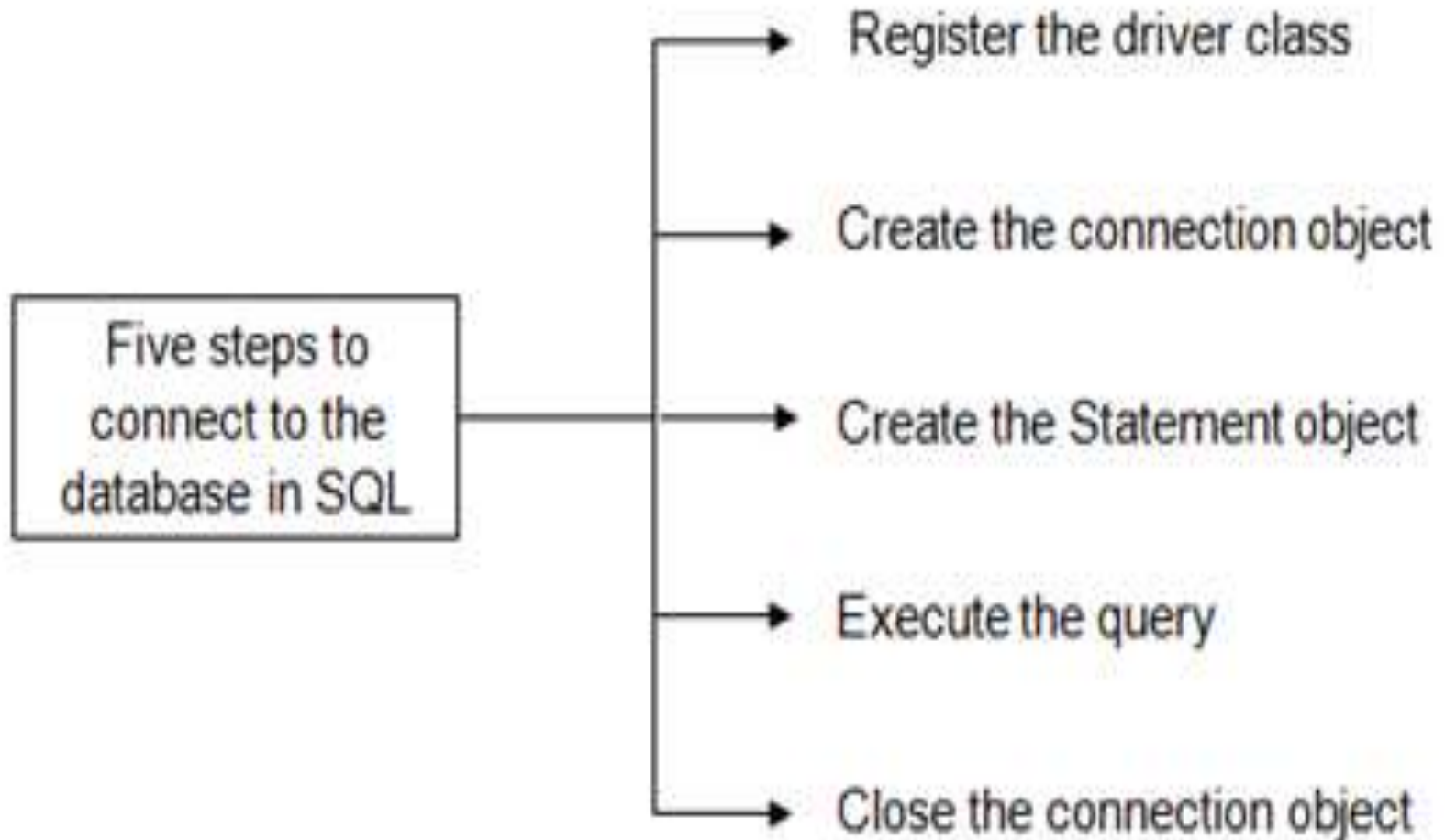
- 
- **public boolean next() throws SQLException**
Moves the cursor to the next row.
 - **public int getRow() throws SQLException**
Returns the row number that the cursor is pointing to.
 - **public void moveToInsertRow() throws SQLException**
Moves the cursor to a special row in the result set that can be used to insert a new row into the database.
 - **public void moveToCurrentRow() throws SQLException**
Moves the cursor back to the current row if the cursor is currently at the insert row;

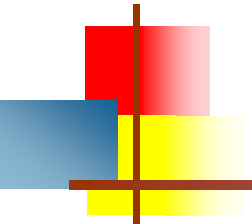


Steps for JDBC Program

1. Create datasource
2. Load Driver using `class.forName()`(optional for higher versions of Java)
3. Define URL
4. Establish a connection to database using `getConnection()` & `DriverManager` object
5. Create Statement object & use `createStatement()`
6. Use Statement object & execute SQL query
7. Process Result
8. Close connection

Steps to connect database





Register the driver class

(optional for higher versions of Java)

- The `forName()` method of `Class` class is used to register the driver class.

This method is used to dynamically load the driver class.

- Syntax: `public static void forName(String className)`
`throws ClassNotFoundException`

- Example to register the `OracleDriver` class, Here Type1 driver software
- `Class.forName("sun.jdbc.odbc.JdbcodbcDriver");`
- Here we can load driver
- To register `MySQL` database drivers we will use Type 4 driver software
- `Class.forName("com.mysql.jdbc.Driver");`



Create the connection object

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

- Example to establish connection with the Oracle database

- Connection con=

```
DriverManager.getConnection("Jdbc.odbc : DSN Name");
```

- For MSAccess

```
con=DriverManager.getConnection("Jdbc.odbc.mydsn");
```

- mydsn is DSN given to Database

- For MySQL

- Connection con=

- DriverManager.getConnection

```
(jdbc:mysql://localhost/database1?user=root & password="pw");
```



Program to check drivers and connection

```
import java.sql.Connection;
import java.sql.DriverManager;
public class JavaDataBase1 {
    public static void main(String args[]) throws ClassNotFoundException
    {
        String url;
        Connection con = null;
        try {
            Class.forName("com.mysql.jdbc.Driver");
            System.out.println("Drivers are loaded");
            url="jdbc:mysql://localhost:3306/spring";
            con = DriverManager.getConnection(url);
            System.out.println("Connection created");
            con.close();
            System.out.println("Connection closed");
        }
        catch (Exception e) {
            System.out.println(e.toString());
        }
    }
}
```



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.

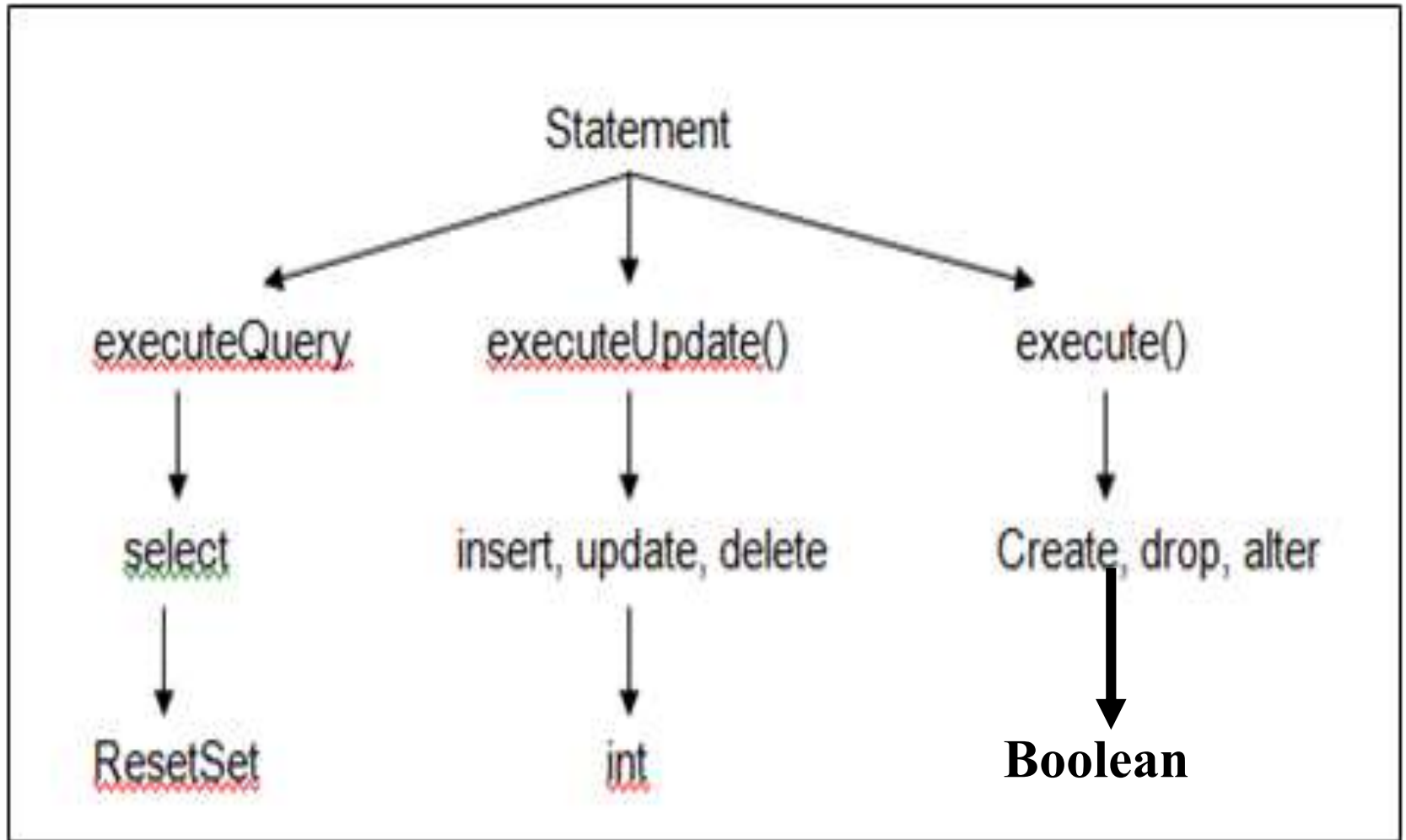
- Syntax:

`public Statement createStatement()throws SQLException`

- Example to create the statement object

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

Methods of statement interface





PreparedStatement Interface

- It is useful when user wants to execute SQL statements many times.
- The PreparedStatement interface accepts input parameters at runtime.
- The PreparedStatement interface extends the Statement interface.
- All parameters in JDBC are represented by the ‘?’ symbol, which is known as the “**parameter marker or place holder**”. We must supply values for every parameter before executing the SQL statement.
- The **setXXX()** methods bind values to the parameters, where XXX represents the Java data type of the value you wish to bind to the input parameter. If we forget to supply the values, you will receive an SQLException.



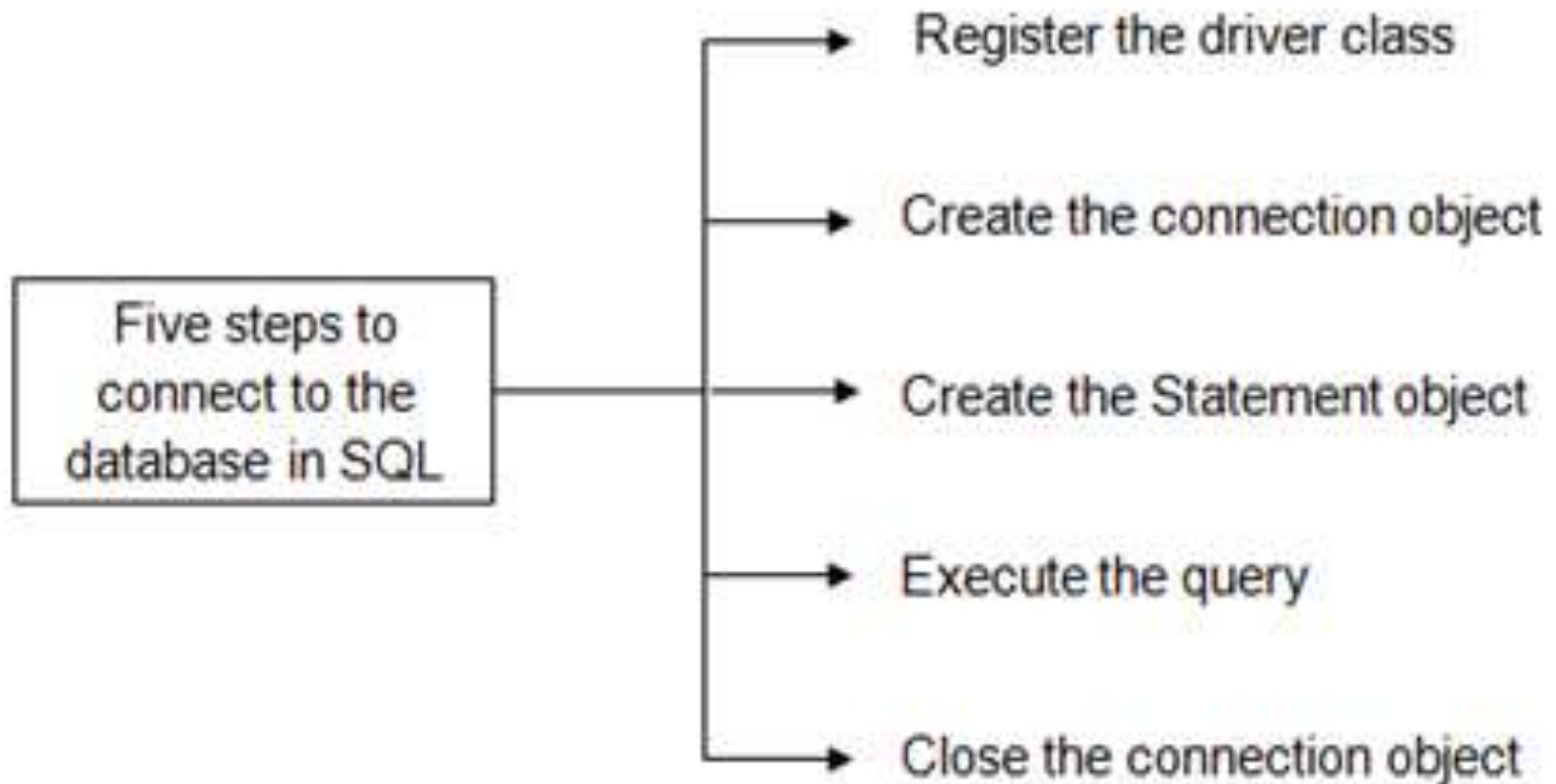
Example of PreparedStatement

```
String sql="insert into table values(?,?)";
```

```
PreparedStatement ps=con.prepareStatement(sql);
```

- `Int r2=5;`
- `String s2="MIT";`
- `ps.setInt(1,r2);`
- `ps.setString(2,s2);`
- `int i2=ps1.executeUpdate();`
- `System.out.println("data inserted"+i2);`

Steps to connect database





Program to create a table

```
import java.sql.*;
public class create{
    public static void main(String[] args) {
        System.out.println("Creating table in Mysql database table!");
        try {
            //Class.forName("com.mysql.cj.jdbc.Driver");Not required in latest versions of Java
            //System.out.println("Drivers loaded");//optional
            Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/ jdbc_example", "root", "");
            System.out.println("Connection Established");
            try {
                String sql="create table employee1(emp_id int, emp_name char)";
                PreparedStatement ps1=con.prepareStatement(sql);
                ps1.executeUpdate();
                System.out.println("table created");
            } catch (SQLException s)
            {System.out.println("SQL statement is not executed! Error is: " + s.getMessage());}
            } catch (Exception e) {
                e.printStackTrace();} } }
```



Program to insert dynamic values

```
import java.sql.*;
import java.io.*;
public class InsertCValues {
    public static void main(String[] args) {
        System.out.println("Inserting values in Mysql database table!");
        String url = "jdbc:mysql://localhost:3306/" + "jdbc_example";
        try {
            //Class.forName("com.mysql.cj.jdbc.Driver");
            Connection con = DriverManager.getConnection(url, "root", "***");
            try {
                String sql = "INSERT employee VALUES(?,?,?)";
                PreparedStatement ps = con.prepareStatement(sql);
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            }
        }
    }
}
```



Program to insert dynamic values

```
do{
    System.out.println("enter id:");
    int id=Integer.parseInt(br.readLine());
    System.out.println("enter name:");
    String name=br.readLine();
    System.out.println("enter salary:");
    float salary=Float.parseFloat(br.readLine());
    ps.setInt(1,id);
    ps.setString(2,name);
    ps.setFloat(3,salary);
    int i=ps.executeUpdate();
    System.out.println(i+" records affected");
    System.out.println("Do you want to continue: y/n");
    String s=br.readLine();
    if(s.startsWith("n")){
        break; }
    }while(true);
} catch (SQLException s) {
    System.out.println("SQL statement is not executed! Error is: " + s.getMessage());}
} catch (Exception e) {
    e.printStackTrace();}}}
```

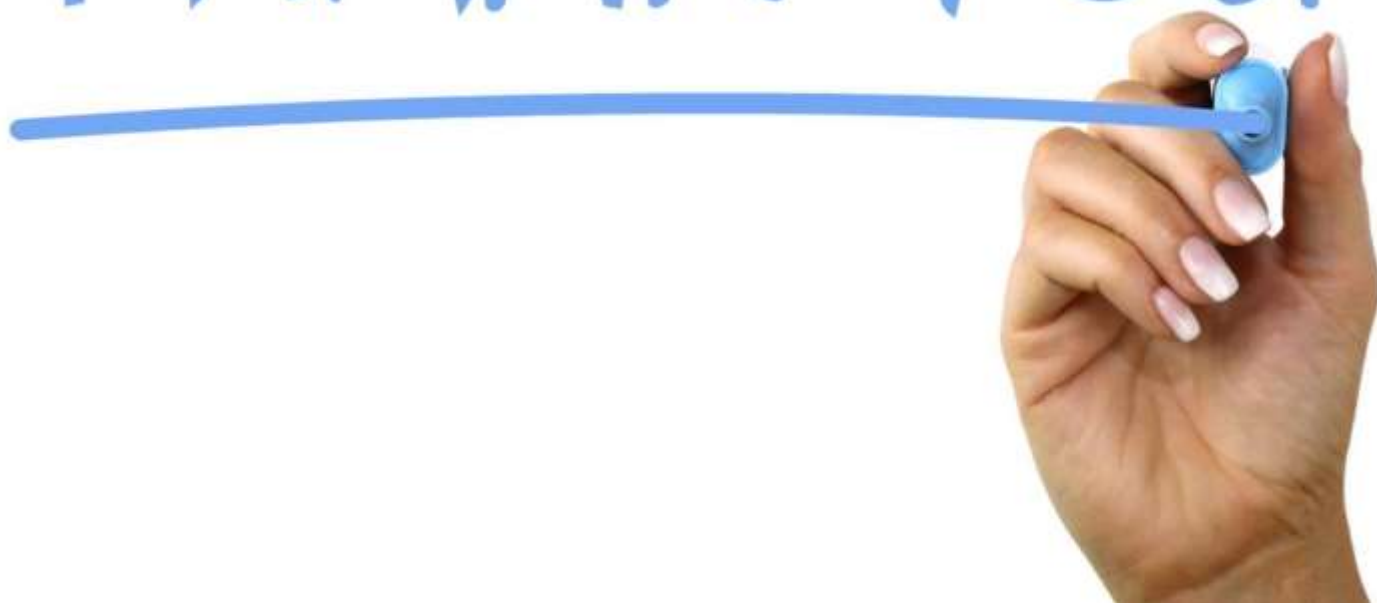


Points to remember

- Prepare similar JDBC programs for static queries
- Here contents are given pointwise but while writing answers write it as per the mark's allocation.
- Execute the JDBC programs for better understanding



THANK YOU



[This Photo](#) by Unknown Author is licensed under [CC BY-SA](#)