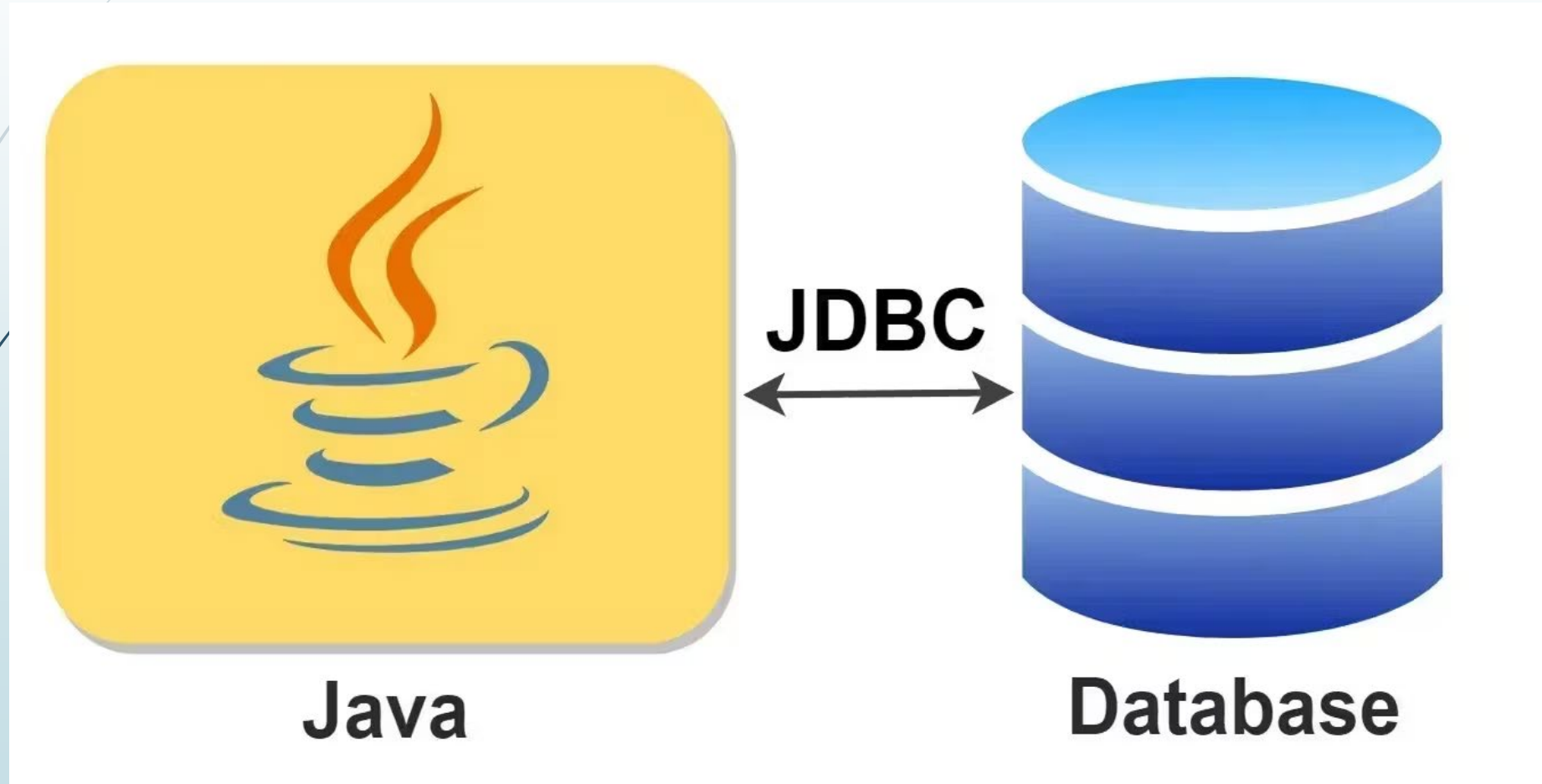
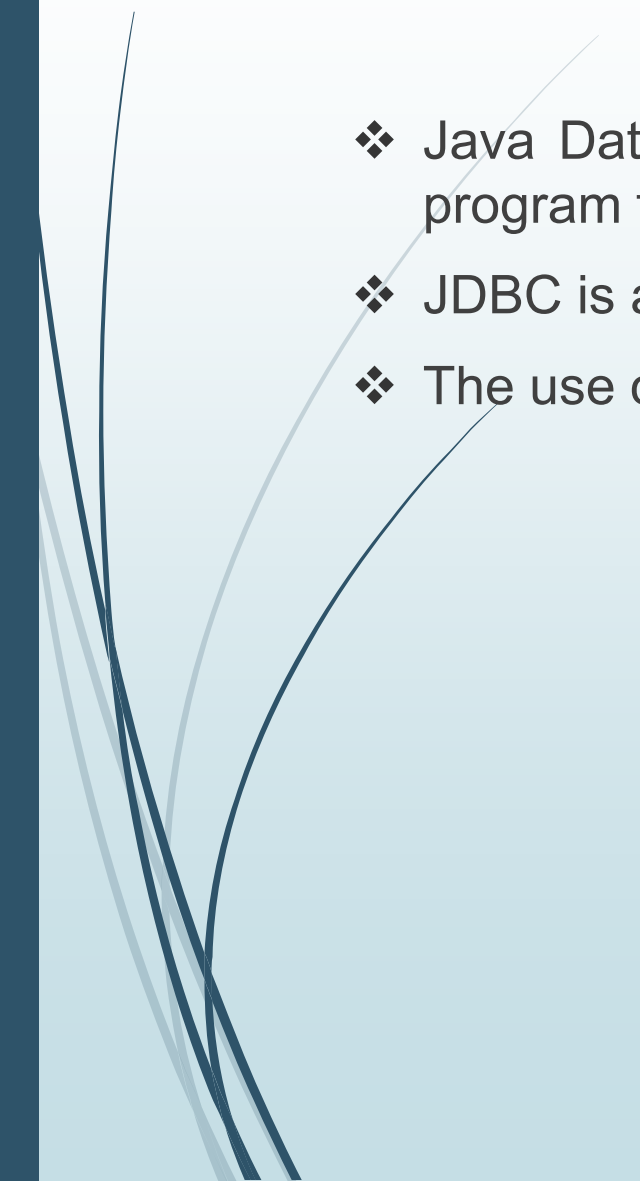


# JDBC in Java

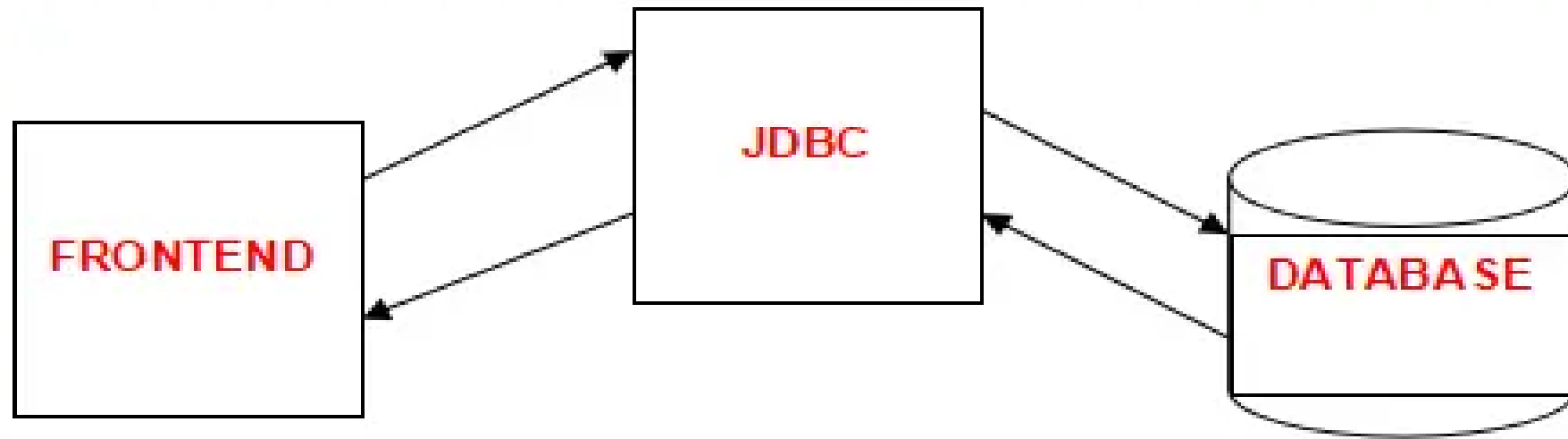




# JDBC in Java

- ❖ Java Database Connectivity(JDBC) is a technology that enables the java program to manipulate data stored in the database.
  - ❖ JDBC is a database-related technology given by SUN Microsystems.
  - ❖ The use of JDBC technology is to communicate with the database.
- 

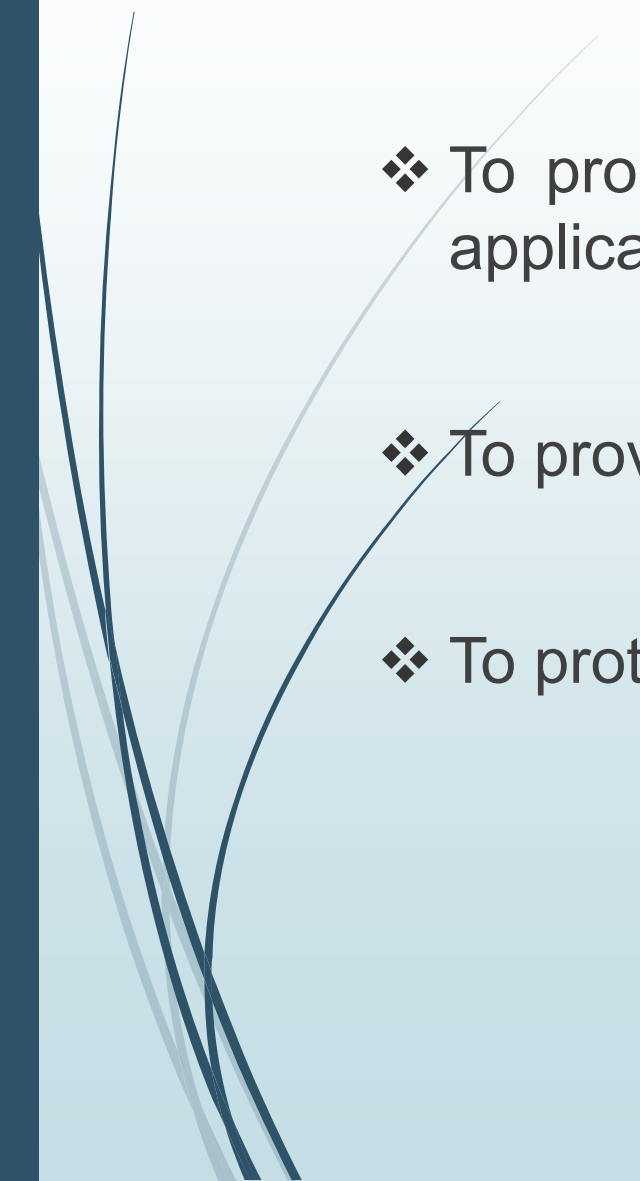
# Java Database Communication



- ❖ JAVA program only knows method calls and DBMS only knows SQL statements.
- ❖ Both have a heterogeneous environment, they cannot communicate directly.
- ❖ So for communication between the JAVA program and database, we have to use JDBC.



## Need of Database communication

- ❖ To promote a dynamic interaction between user and application.
  - ❖ To provide security for user confidential data.
  - ❖ To protect data from multiple users.
- 

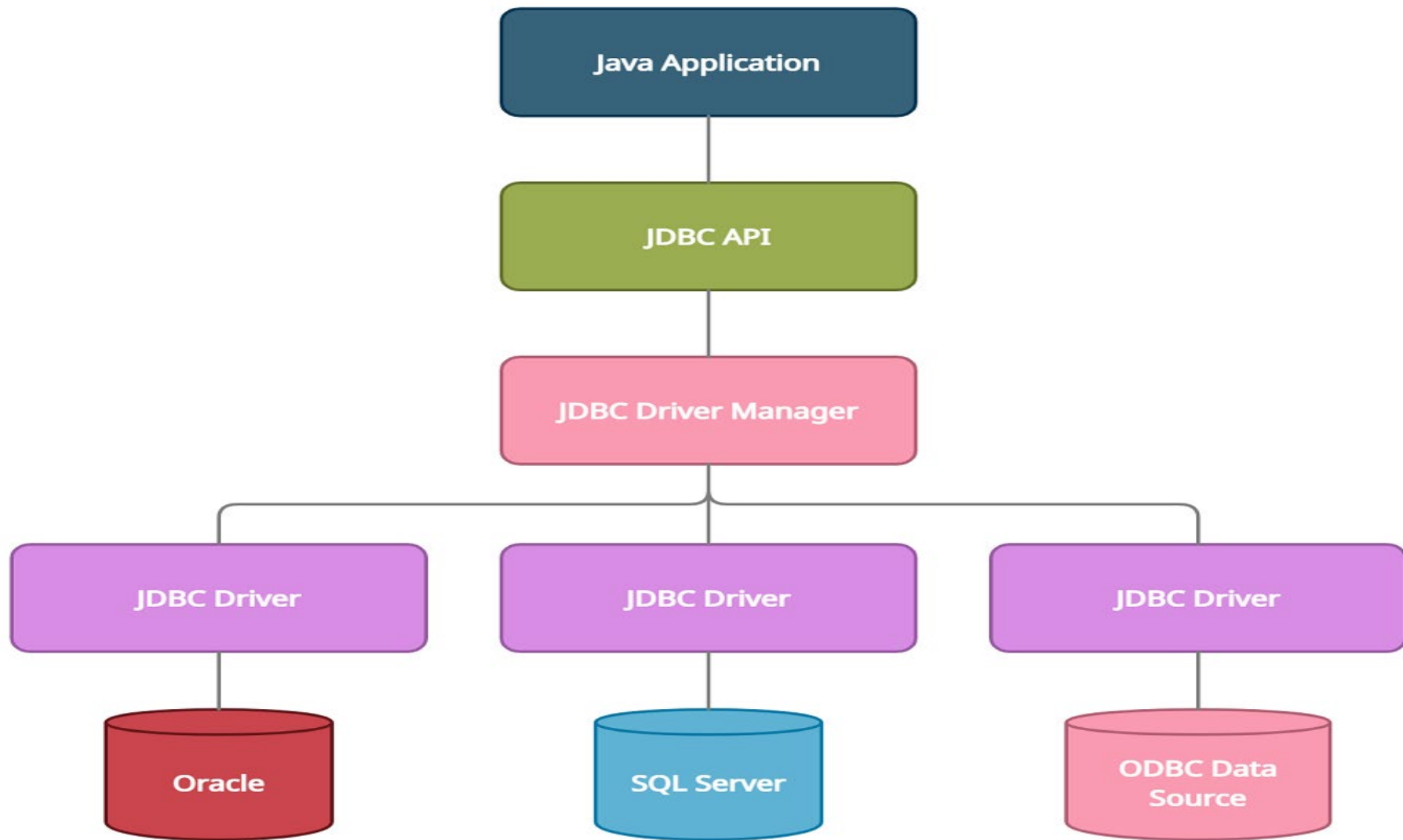
# What is JDBC?

- ❖ **JDBC is a java based data access technology from Sun Microsystems.**
- ❖ **JDBC is an open specification that contains rules and guidelines that have to be followed by the vendor for developing JDBC drivers.**
- ❖ **The process of interacting with the database from Java Applications is called JDBC.**
- ❖ **JDBC is a Java API which enables java programs to execute SQL statements.**
- ❖ **JDBC is an API, which will provide a very good predefined library to connect with databases from JAVA applications in order to perform the basic database operations.**

## What is JDBC? (Conti..)

- ❖ To execute JDBC applications we should require a conversion mechanism to convert the database logic from Java representations to Query Language representations and from Query Language representations to Java representations.
- ❖ For this, the required conversion mechanisms are available in the form of software called “Driver”

## JDBC Architecture:



## JDBC Architecture layers

➡ The JDBC Architecture has two layers:

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

**Note:** An API (Application Programming Interface) is a set of rules and protocols that allow software applications to communicate with each other. API is act as an interface, or contract, between two applications, defining how they communicate with requests and responses.

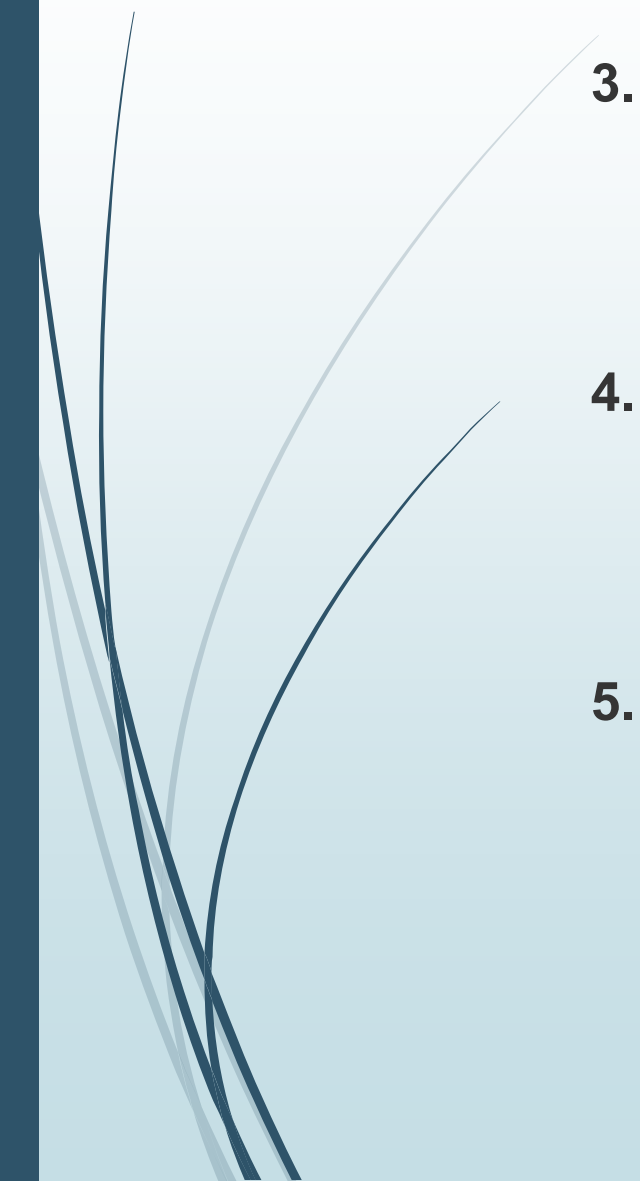


## Components of JDBC

- ❖ **JDBC architecture is a client-server architecture.**
- ❖ **JDBC architecture has 5 elements.**
  1. **JDBC client:** Here JDBC Client is nothing but a java application that wants to communicate with the database
  2. **JDBC API:** To write a java program that communicates with any database without changing the code Sun Microsystems has released JDBC API. By JDBC API implementation we can call JDBC Driver.




## Components of JDBC (Cont..)

- 
3. **JDBC Driver:** It is a java application that acts as an interface between java applications and databases. It understands the given java calls and converts them into database calls and vice-versa.
  4. **JDBC Driver Manager:** It acts as a data structure that collects a group of JDBC Drivers and allows the programmer to select the driver dynamically for the database connection.
  5. **Database Server:** It is nothing but the Database server like Oracle, MySQL, SQL Server, etc. with which the JDBC client wants to communicate.



## JDBC API

- ❖ API (Application Programming Interface) is nothing but a collection of library methods.
  - ❖ Library classes and interfaces contain library methods and these library classes and interfaces are present inside some predefined packages.
  - ❖ Packages are just like a folder that contains files.
- 



## JDBC Driver:

- JDBC driver is a translation software written in java language according to JDBC specifications.
- JDBC drivers do the following operations:
  1. Establishing the database connection for the JDBC client with the database server.
  2. Receiving the JDBC method calls from the JDBC client, translating it into DBMS understandable format, and passing on the same to the database server.
  3. Receiving the response from the DBMS, translating it into java format, and passing on the same to the java client.



## Types of drivers:

All the drivers classified into 4 types:


1. JDBC-ODBC bridge driver (**TYPE-1 Driver**):
2. Native API, partly java driver (**TYPE-2 Driver**)
3. Net protocol, All java driver(**TYPE-3 Driver**)
4. Native protocol, All java driver (**TYPE-4 Driver**)

## JDBC-ODBC bridge driver (TYPE-1 Driver)

- ✓ This is the first driver ever developed. The driver class of type1 driver software is “**sun.jdbc.odbc.JdbcOdbcDriver**”.
- ✓ This driver receives any JDBC method calls and sends them to ODBC (Open Database Connectivity) driver.
- ✓ ODBC driver understands these calls and communicates with the database library provided by the vendor.
- ✓ The ODBC driver and the vendor database library must be present in the client machine.
- ✓ As the type1 driver acts as a bridge between sun Microsystems JDBC technology and Microsoft's ODBC technology it is known as the JDBC-ODBC bridge driver.

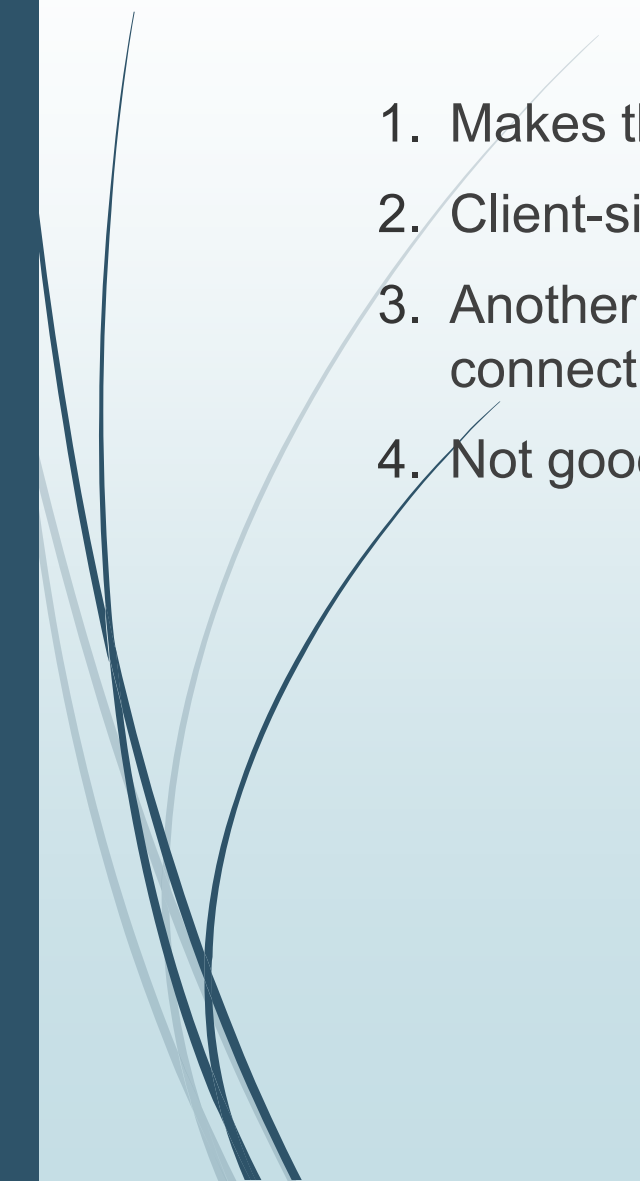


## **Advantages of JDBC – ODBC Bridge Driver**

1. It is easy to use the driver.
  2. It can be used to connect to any database.
- 

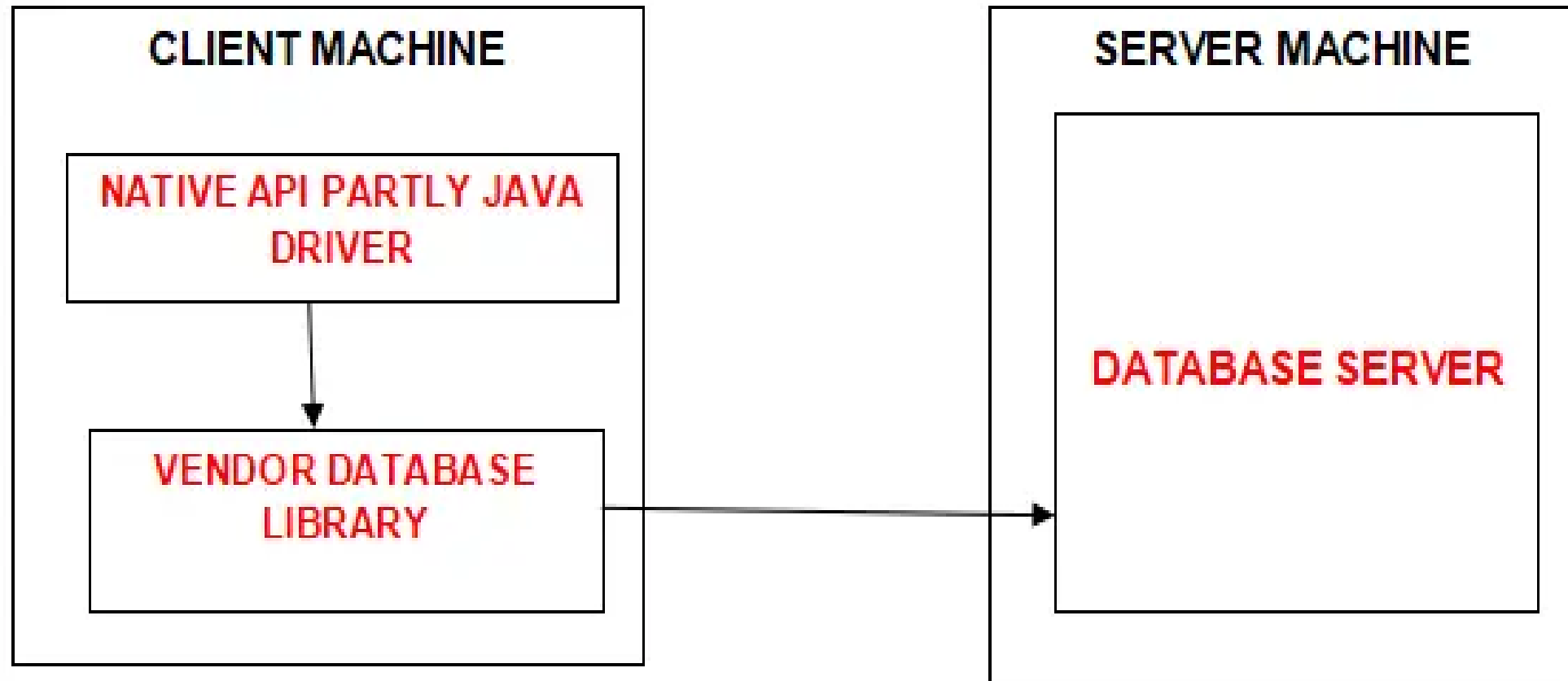


## Disadvantages of JDBC – ODBC Bridge Driver

1. Makes the java application OS-dependent (Windows).
  2. Client-side maintenance is very high.
  3. Another requirement is that the ODBC driver and the native database connectivity library must already be installed on the client machine.
  4. Not good for the web applications.
- 



## Type 2 JDBC Driver (Native API Partly Java Driver)



Type 2 Driver

## Type 2 JDBC Driver (Native API Partly Java Driver) (Cont..)

- ❖ Type 2 JDBC driver is known as Native API Partly Java Driver.
- ❖ DBMS vendor only develops type 2 drivers.
- ❖ The Type 2 driver translates JDBC method calls into DBMS vendor-provided native library calls.
- ❖ These native libraries are developed using C or C++ which means java software known as Type 2 driver makes use of JNI (Java Native Interfaces) to invoke the native library.
- ❖ Therefore it is called Partly Java Driver.
- ❖ For example, oracle's OCI(Oracle Call Interface) driver
  - Class name = `oracle.jdbc.driver.OracleDriver`
  - Connection string = `jdbc:oracle:OCI8:@server`



## Advantages of Type 2 Driver

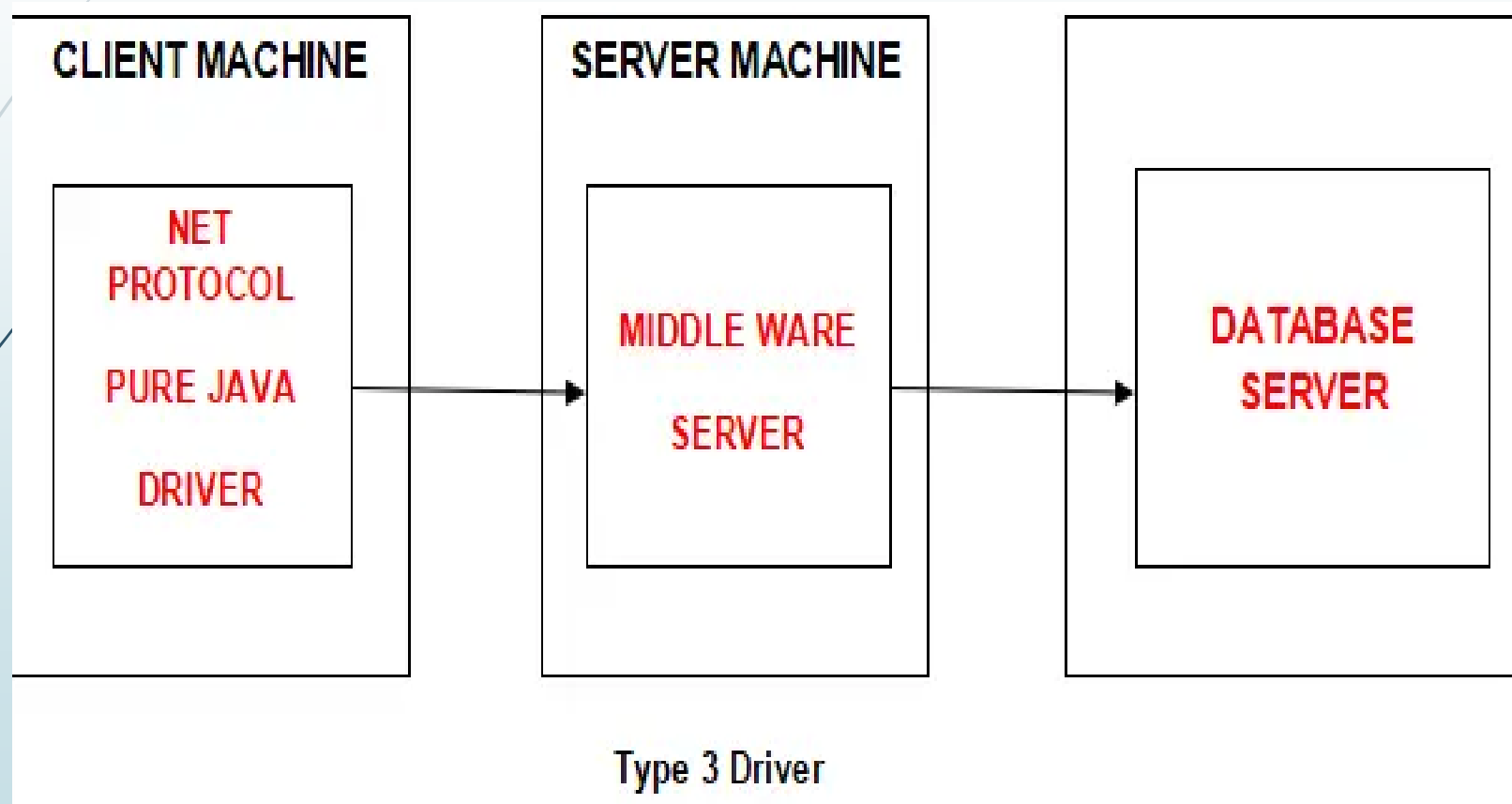
- ❖ Better performance than Type 1 driver as ODBC layer is completely eliminated and it uses native API which is database specific.
- ❖ OS in-dependency is achieved i.e. Type 2 drivers are available for all platforms of java.
- ❖ Client-side maintenance is slightly reduced.



## Disadvantages of Type 2 Driver:

- It doesn't provide optimum performance as it doesn't directly communicate with the database.
- Because of native libraries, there is a chance of a system crash.
- Client-side maintenance is not completely eliminated. If DBMS product changes, in all client machines that vendor-provided native library has to be copied.
- The vendor database library needs to be loaded on each client machine and hence can not be used for the internet.

## Type 3 JDBC Driver (NET Protocol All Java Driver)





## **Type 3 JDBC Driver (NET Protocol All Java Driver)**

1. Type 3 driver is known as Net Protocol All Java (Pure Java) Driver.
2. The Type 3 driver translates JDBC method calls into a network server understandable protocol call.
3. That network server (middleware server) internally uses a Type1 or Type 2 driver to communicate with the DBMS.
4. Type 3 is also called Middleware Database server access Driver and Network Driver.
5. Type 3 Driver is purely designed for Enterprise applications it is not suggestible for standalone applications.
6. Type 3 Driver portability is very good when compared to Type 1 and Type 2 Drivers.
7. Type 3 drivers will provide a very good environment to interact with multiple databases.

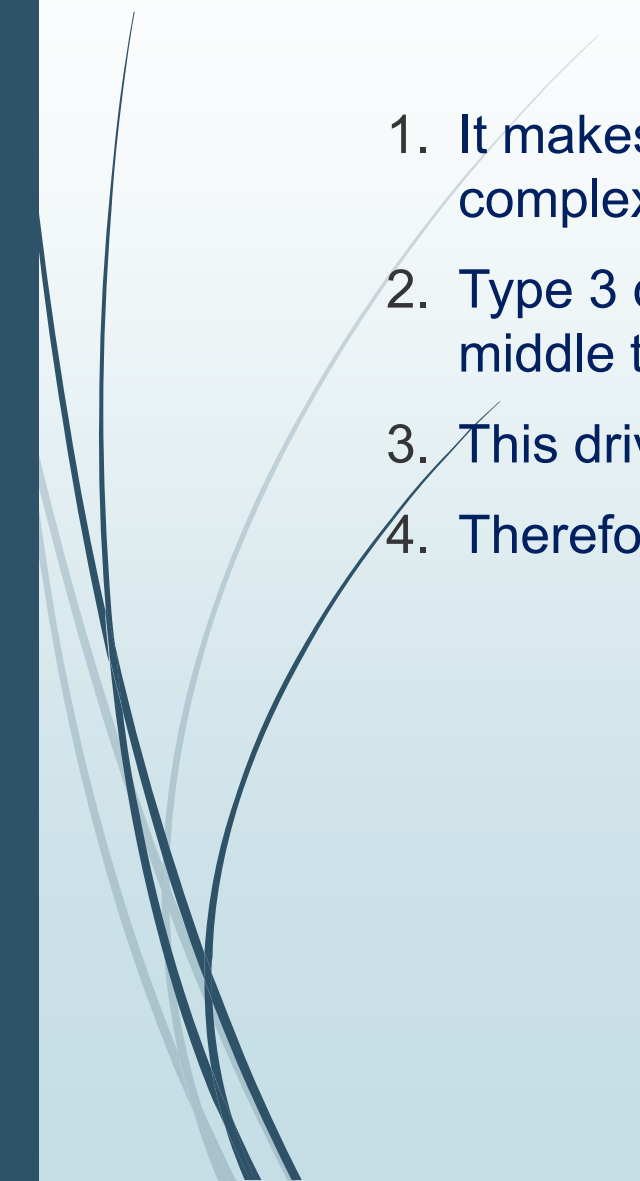


## Advantages of Type 3 Driver:

1. Client-side maintenance not required.
2. Better performance as connection optimization can be done in the network server and protocols are faster.
3. This driver is server-based so there is no need for any vendor database library to be present on the client machine.
4. This driver is fully written in java and hence portable.
5. It is suitable for the internet.

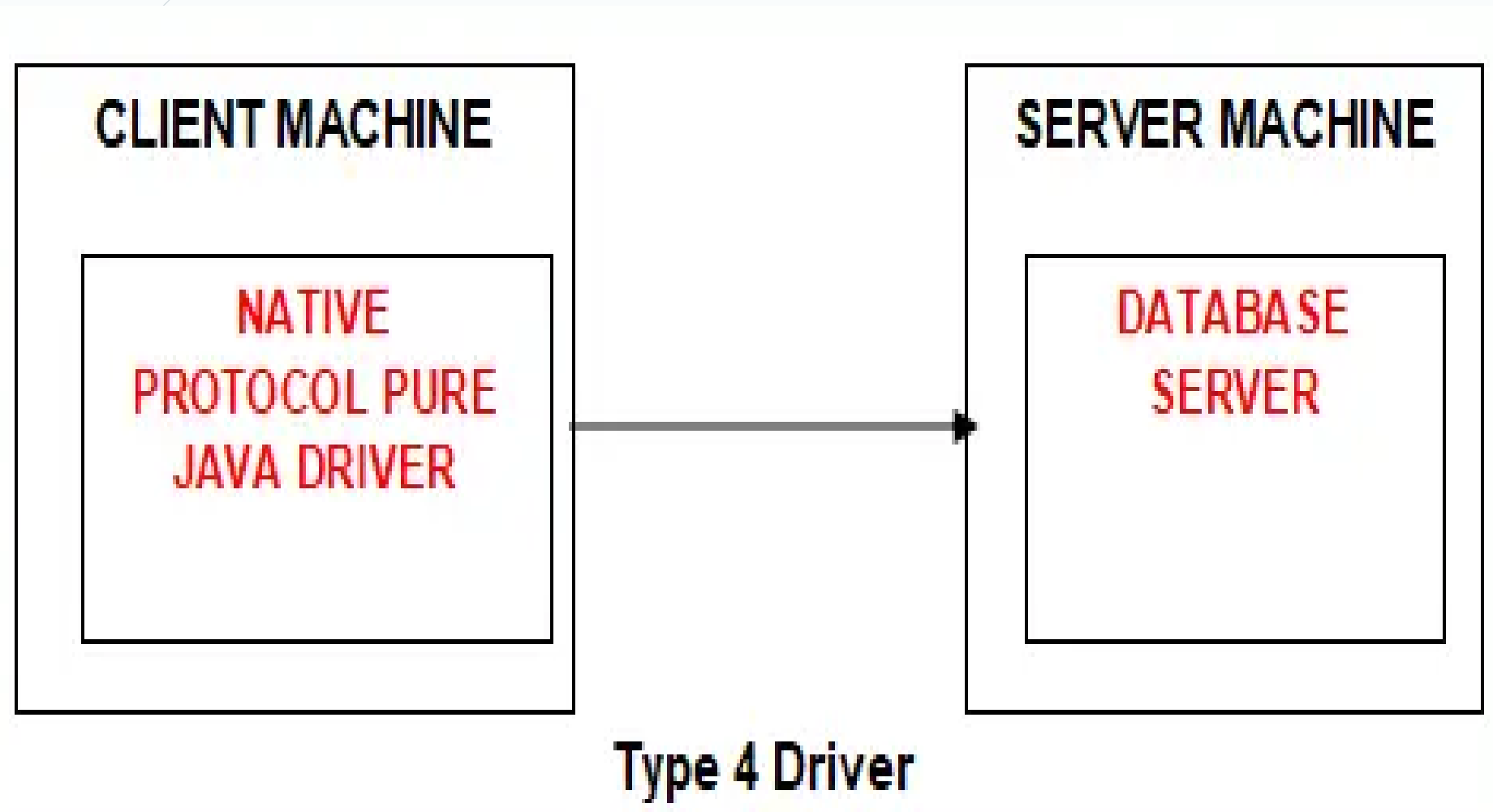


## Disadvantages of Type 3 Driver:

1. It makes java database communication a 3 tier architecture that is complex to maintain.
  2. Type 3 driver requires database-specific coding to be done in the middle tier. Maintenance of the middle tier becomes costly.
  3. This driver also doesn't directly communicate with the database.
  4. Therefore, still, there is a scope for improvement in performance.
- 



## Type 4 JDBC Driver (Native Protocol All Java Driver)

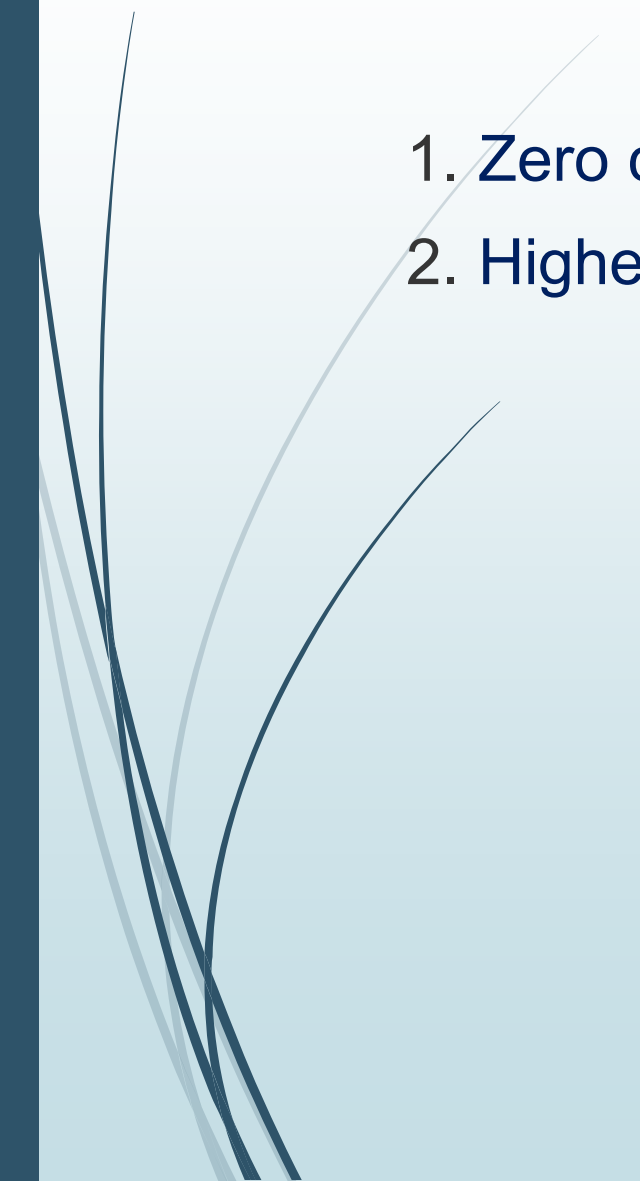


## Type 4 JDBC Driver (Native Protocol All Java Driver)

1. Type 4 Driver is also called pure Java Driver and Thin Driver because Type 4 driver was implemented completely by using java implementations.
2. Type 4 Driver is the frequently used Driver when compared to all the remaining Drivers.
3. Type 4 Driver is recommended for any type of application including standalone applications, Network applications, etc.
4. Type 4 Driver portability is very good when compared to all the remaining drivers.
5. Type 4 Driver should not require any native library dependencies and it should require one-time conversion to interact with the database from Java applications.
6. Type 4 is the cheapest Driver among all.

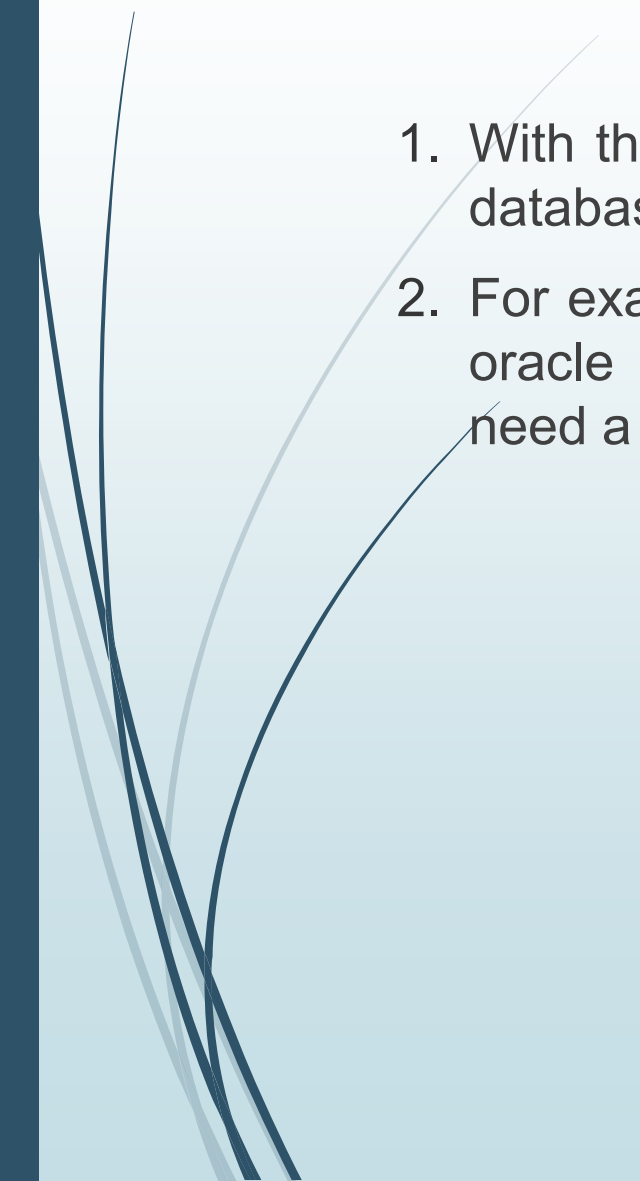


## Advantages of Type 4 Driver

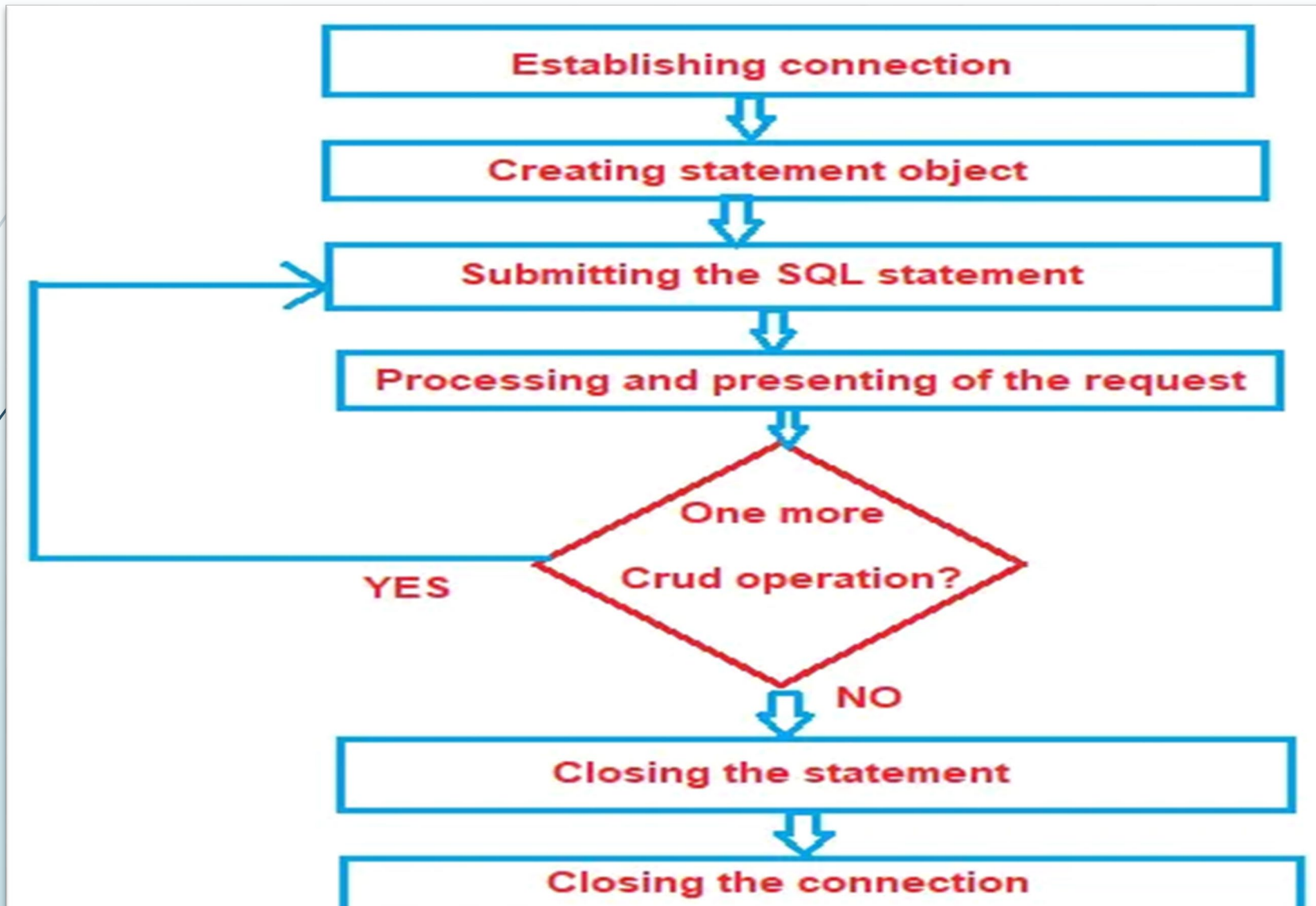
1. Zero client-side maintenance required.
  2. Highest performance.
- 



## Disadvantages of Type 4 Driver

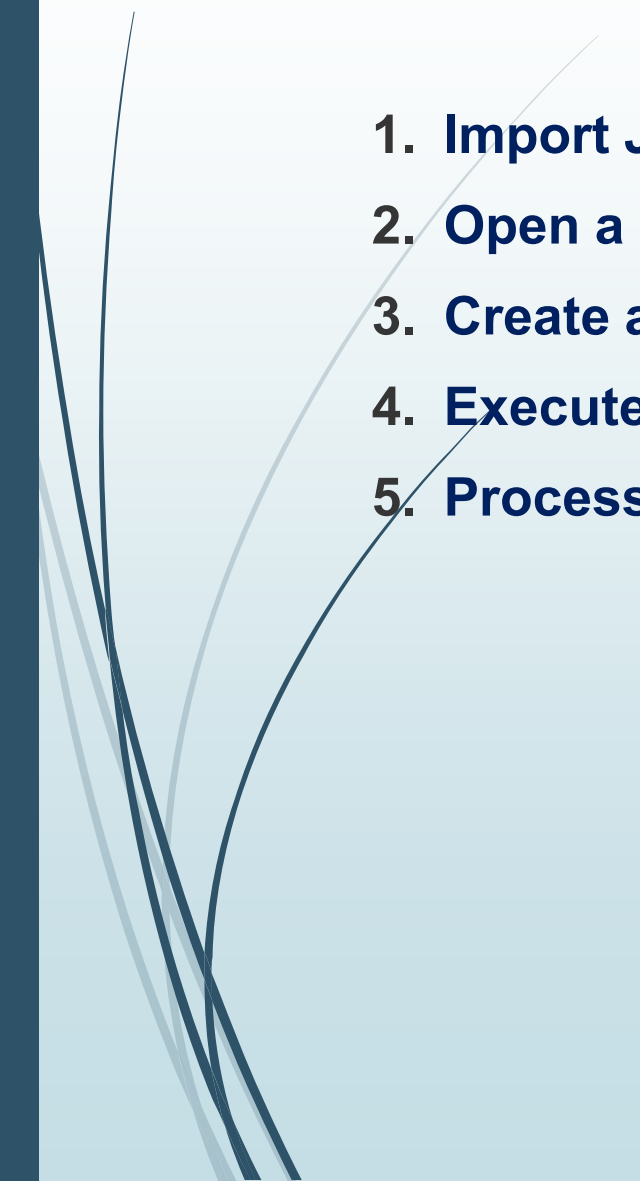
1. With the type 4 driver, the user needs a different driver for each database.
  2. For example, to communicate with the oracle server we need an oracle driver and to communicate with the MySQL server we need a MySQL driver.
- 

## Steps to Design JDBC Applications in Java

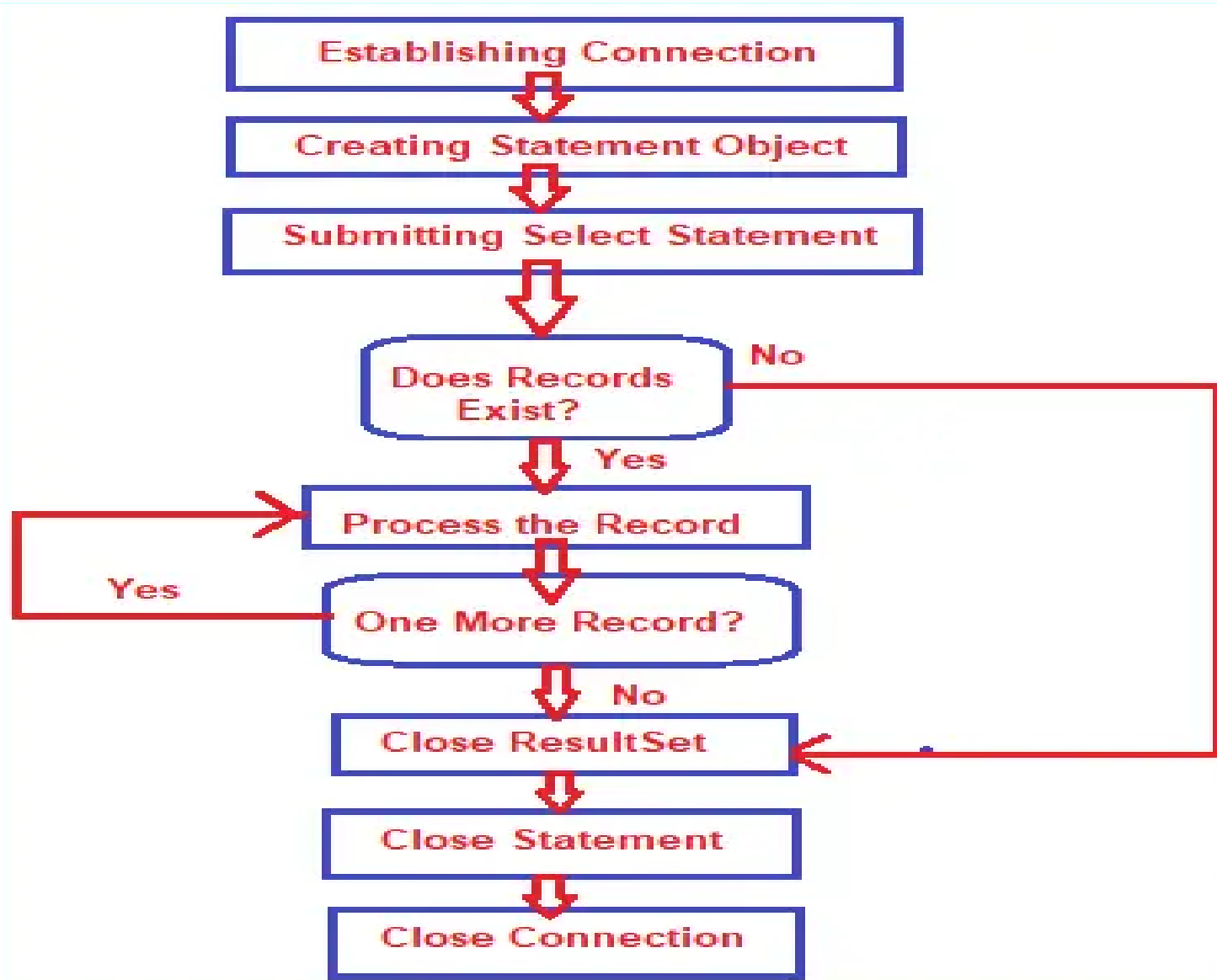




## **Steps to Design JDBC Applications in Java**

- 1. Import JDBC packages**
  - 2. Open a connection to the database.**
  - 3. Create a statement object to perform a query.**
  - 4. Execute the statement object and return a query resultset.**
  - 5. Process the resultset.**
- 

## How to retrieve the data from a Database in JDBC Application?



## Submitting the Select Statement:

In order to retrieve data from the database, the **JDBC** application has to submit **SELECT SQL** statements to the **DBMS**.

The **executeQuery()** method of statement object is used to submit a select statement to the **DBMS**.

**For example:**

```
ResultSet rs = st.executeQuery("select * from employee where empId = 101");
```

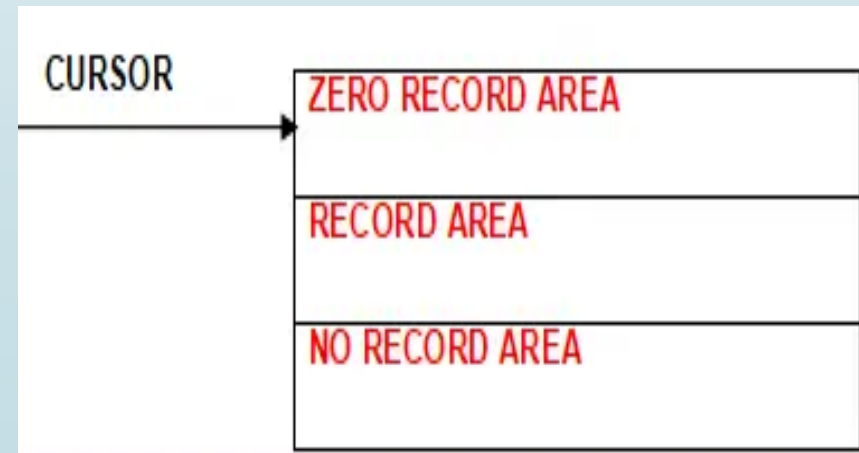


## Submitting the Select Statement (Cont..)

Object-oriented representation of the records received from the **DBMS** is nothing but the **ResultSet** object. **ResultSet** object is logically divided into three partitions.

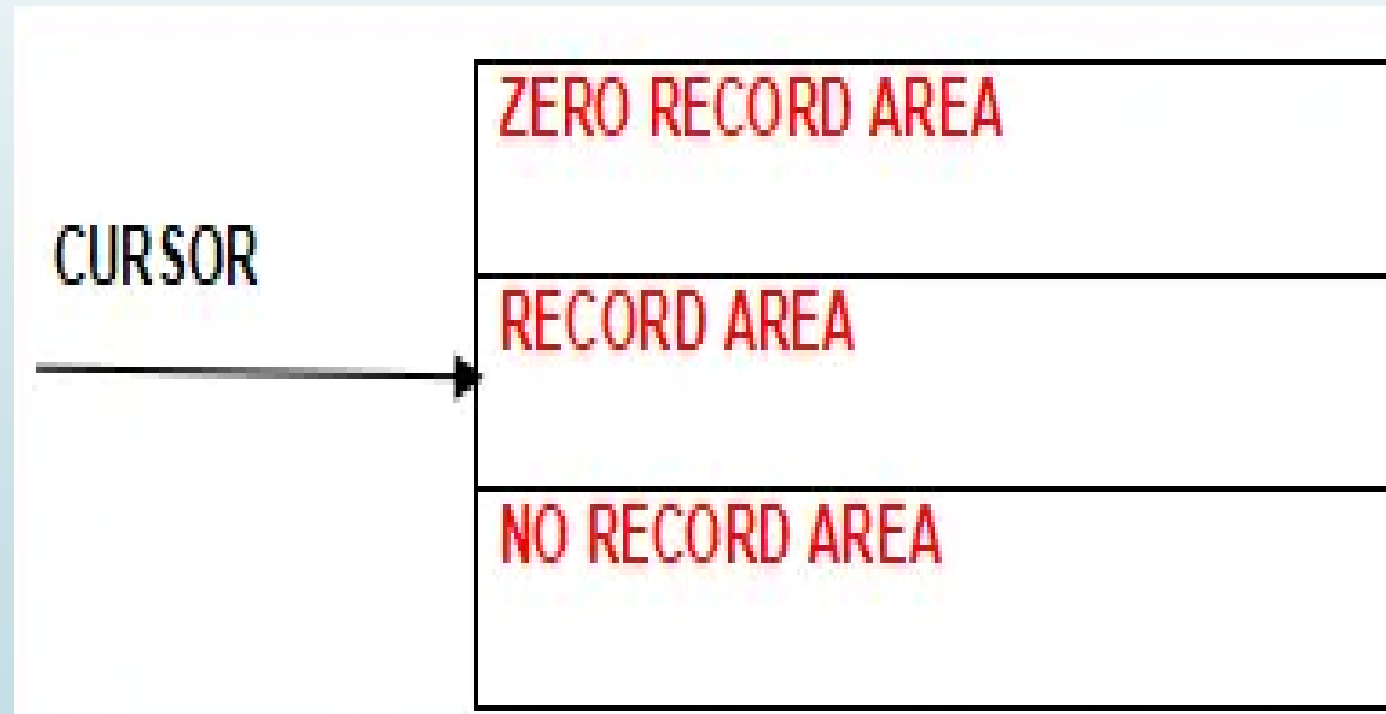
1. **ZERO RECORD AREA**
2. **RECORD AREA**
3. **NO RECORD AREA**

When the **ResultSet** object is created a logical pointer points to the **ZERO RECORD AREA** which is known as the **cursor**.



## Submitting the Select Statement (Cont..)

Processing a record of the **ResultSet** is nothing but receiving its column values. We should not process the record when the cursor points to **ZERO RECORD AREA** or **NO RECORD AREA** of the **ResultSet**, otherwise, **SQL Exception** is raised i.e. we should read column values only when the cursor points to the **RECORD AREA** of the **ResultSet**.



## Submitting the Select Statement (Cont..)

**ResultSet** object has 15 methods to deal with the cursor.

The most frequently used method is “**next()**”.

The **next()** method does two things:

1. Moves the cursor by one record (area) in the forward direction.
2. Returns **boolean** value after moving the cursor. It returns **true** if the record exists there, otherwise, it returns **false**.

**ResultSet** object has **getXXX** methods to read column values. This method takes the column number of the record of the **ResultSet** as arguments and returns the column values.

## Retrieve Operation in Java using JDBC

```
try (
    Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/javadb",
    "root", "cdac"); Statement stmt = conn.createStatement();) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the product id: ");
    int prodId = sc.nextInt();
    ResultSet rs = stmt.executeQuery("select * from product where productId =" +
    prodId);
    if (rs.next()) {
        System.out.println("Product Id: " + rs.getInt(1));
        System.out.println("Product Name: " + rs.getString(2));
        System.out.println("Product Price: " + rs.getFloat(3));
        System.out.println("Product Quantity: " + rs.getString(4));
    } else {
        System.out.println("account doesnot exist");
    }
    sc.close();
} catch (SQLException ex) {
    ex.printStackTrace();
}
```

## JDBC ResultSet Interface in Java Application

- ❖ In JDBC applications if we use Select **SQL Query** as a **parameter** to **executeQuery()** method then **JVM** will send that selection group **SQL Query** to the **database engine**, where **Database Engine** will execute that **SQL Query**, fetch the data from Database and send back to **Java application**.
- ❖ At Java application, the **fetches data** will be stored in the form of an **object** at the **heap memory** called **ResultSet**. As per the predefined implementation of the **executeQuery** method, JVM will return the generated **ResultSet object reference** as the return value.

```
ResultSet rs = st.executeQuery("select * from Employee");
```

## JDBC ResultSet Interface in Java Application (Conti..)

When **ResultSet** object is created automatically a cursor will be created positioned before the first record.

If we want to read the records data from **ResultSet** object, for each and every record we have to check whether the next record is available or not from **ResultSet** cursor position, if it is available then we have to move **ResultSet** cursor to the **next** record position.

To perform the above work we have to use the following method from **ResultSet**.

```
public boolean next()
```

## JDBC ResultSet Interface in Java Application (Conti..)

After getting **ResultSet** cursor to a particular Record position we have to retrieve the data from **respective columns**, for this, we have to use the following **overloaded method**:

1. **public xxx getxxx(int field\_No)**
2. **public xxx getxxx(String field\_Name)**

Where, xxx may be a byte, short, int, etc.

**Example :**

```
while(rs.next())  
{  
    System.out.println(rs.getInt(1));  
    System.out.println(rs.getString(2));  
    System.out.println(rs.getFloat(3));  
}
```

## Prepared Statement in JDBC

- ❖ **PreparedStatement** is another statement object which is also used to execute all SQL select and non-select statements.
- ❖ The main advantage of **Prepared Statement** is, that it will improve the performance when we want to execute the same query multiple times compared to **Statement** object
- ❖ But if we send different queries using **PreparedStatement** then there is no difference between **PreparedStatement** and **Statement** but in this scenario **Statement** object is **preferable** one.



## How to use PreparedStatement in Java?

Creating a prepared statement object: By calling the **prepareStatement()** method on the connection object, **java.sql.PreparedStatement** object is created.

**For example:**

```
PreparedStatement ps = con.prepareStatement("INSERT INTO VALUES(?,?,?)");
```

The Object-oriented representation of a pre-compiled or already prepared SQL statement is nothing but a **PreparedStatement** object.

## Architecture of PreparedStatement

