1. Write the difference between Core Java and Advanced Java.

Core java	Advanced Java
Includes Java Standard Edition (J2SE)	Includes Java Enterprise Edition (J2EE)
Category of java that covers the fundamental concepts of Java programming language to develop general applications	Category of java that covers the advanced concepts to build enterprise applications using Java programming language
OOP, data types, operators, exception handling, threading, swing and collection are some topics	Database connectivity, Web services, Servlets, JSP, EJB and some topics
Uses single tier architecture. single-tier applications are desktop applications like MS Office, PC Games, image editing software like Gimp, Photoshop	Uses two tier architecture client and server architecture. presentation layer runs on a client and data is stored on a server. PC, Mobile, Tablet,
Helps to build general applications	Helps to build enterprise level application .web applications

Advanced Java is the level ahead of Core Java, and covers more advanced concepts such as web technologies, and database accessing. Java Enterprise Edition (J2EE) is categorized as Advanced Java.

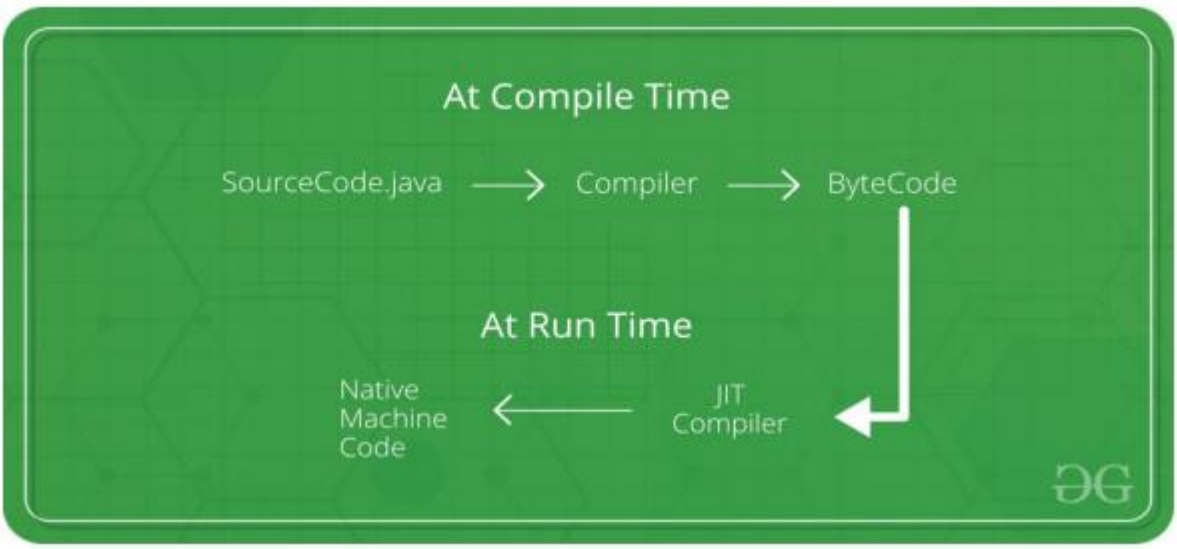
Advanced Java covers a number of topics. JDBC stands for Java Database Connectivity. It is a standard Java API to build independent connectivity between the Java language-based application and databases such as MySQL (My structured query language), MsSQL (Microsoft SQL server) and Oracle. Additionally, Servlets and JSP allow developing dynamic web applications.

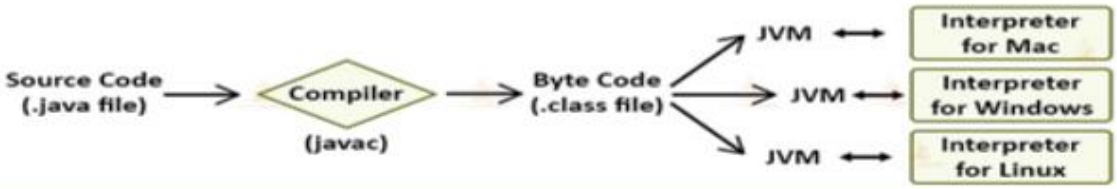
EJB provide distributed and highly transactional features to build enterprise applications. Furthermore, Java web services help to build SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) Architecture style expose the URL/users full web services. They provide a common platform for the applications to communicate with each other.

2. Write a short note on the Java Virtual Machine.

The Just-In-Time (JIT) compiler is a component of the runtime environment that improves the performance of Java™ applications by compiling bytecodes to native machine code at run time.

JVM compiles complete byte code to machine code. JIT compiles only the reusable byte code to machine code. JVM provides platform independence. JIT improves the performance of JVM.



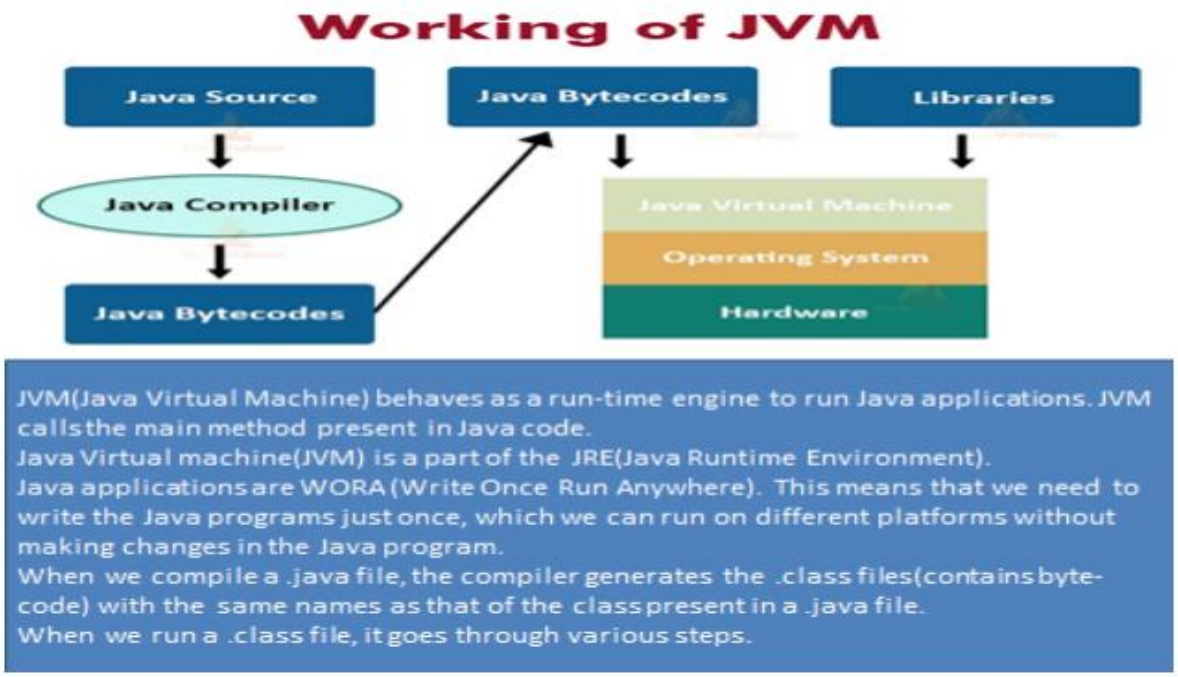


Java programs are first compiled into Java Byte Code(Binary form) and then a special Java interpreter interprets them for a specific platform. Java ByteCode is the machine language for Java Virtual machine(JVM). The JVM converts the compiled binary byte code into a specific machine language.

Compile time vs Run time

Runtime and compile time, these are two programming terms that are more frequently used in java programming language. The programmers specially beginners find it little difficult to understand what exactly they are. So let's understand what these terms means in java with example.
In java running a program happens in two steps, compilation and then execution. The image below shows where does compile time and runtime takes place in execution of a program.

As soon as the programmer starts executing the program using java command, runtime gets started and it ends when execution of program ended either successfully or unsuccessfully. In other way the process of running a program is known as runtime.
Difference between runtime and compile time
Compile time is a process in which java compiler compiles the java program and generates a .class file. In other way, in compile time java source code(.java file) is converted into .class file using java compiler. While in runtime, the java virtual machine loads the .class file in memory and executes that class to generate the output of program.



3. Explain the JDBC driver types with an example.

JDBC Driver is a software component that enables java application to interact with the database. The classes and interfaces of JDBC allow the application to send requests made by users to the specified database. There are 4 types of JDBC drivers:

1) **<u>JDBC-ODBC bridge driver</u>**
   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.

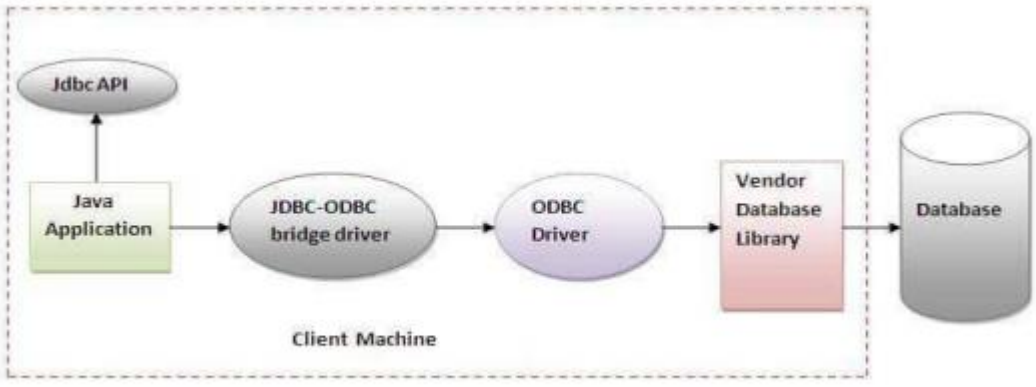


Figure- JDBC-ODBC Bridge Driver

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.

2) <u>Native-API driver (partially java driver)</u>

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.

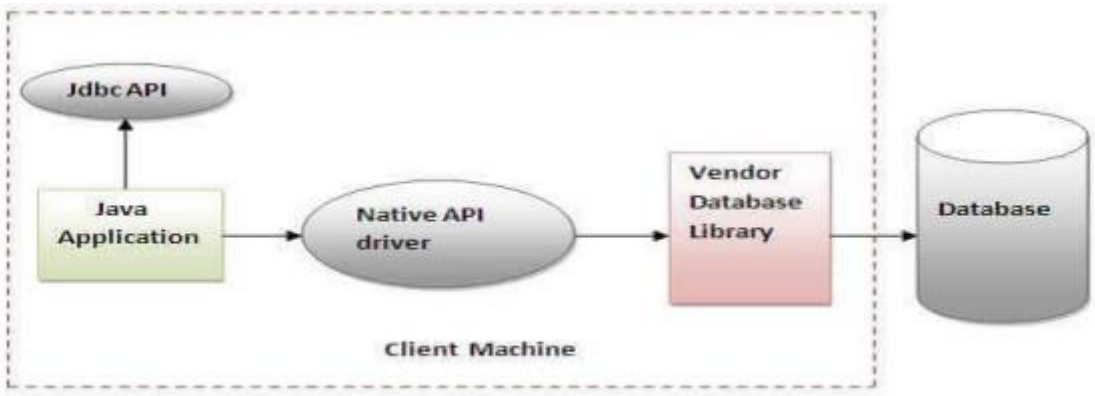


Figure- Native API Driver

Advantage:

- Upgraded performance than JDBC-ODBC bridge driver.

Disadvantage:

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

3) <u>Network Protocol driver (fully java driver)</u>

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.

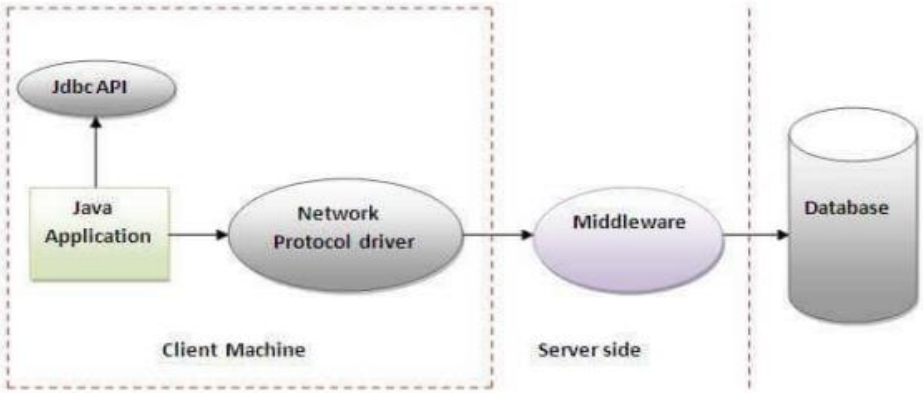


Figure- Network Protocol Driver

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 (fully java 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.

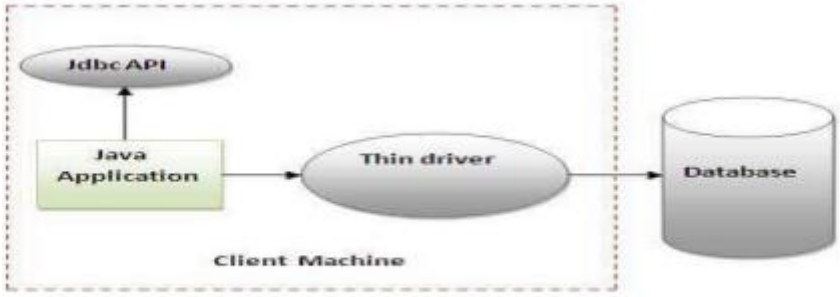


Figure- Thin Driver

Advantage:

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

Disadvantage:

- Drivers depend on the Database

4. Write a Java program to access the MySQL database.

MySQL database:

MySQL> use college

MySQL> desc mca;

MySQL> create table mca(S_id int(5) primary key, Sname varchar(20), DOB date, Address varchar(20), Email_id varchar(20));

MySQL> insert into mca values (101, ‘Raja’, ’2023-07-09’, ‘Chennai’, ‘ss@gmail.com’);

MySQL> insert into mca values (102, ‘John’, ’2023-07-10’, ‘Mangalore’, ‘vv@gmail.com’);

MySQL> select * from mca;

MySQL JDBC program

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

public class demo {
    public static void main(String args[]) {
        try {
            Connection con = (Connection)
            DriverManager.getConnection("jdbc:mysql://localhost:3306/college", "root",
        "root");
            Statement stnt = con.createStatement();
            String query = "select*from mca";
            ResultSet rs = stnt.executeQuery(query);
            while (rs.next()) {
                for (int i=1; i<=5; i++) {
                    System.out.print(rs.getString(i));
                    System.out.println("|");
                }
                System.out.println();
            }
        }
        catch (SQLException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

5. Explain the JDBC Statements, Prepared Statements and Callable Statements.

The JDBC Statement, Callable Statement, and Prepared Statement interfaces define the methods and properties that enable to send SQL or PL/SQL commands and receive data from your database. They also define methods that help bridge data type differences between Java and SQL data types used in a database.

JDBC Statements

Used for general-purpose access to your database. Useful when using static SQL statements at runtime. The statement interface cannot accept parameters.

Syntax:

```
Statement stmt = null;
try {
```

```
        stmt = conn.createStatement( );
        . . .
}
catch (SQLException e) {
        . . .
}
finally {
        stmt.close();
}
```

JDBC Prepared Statement

The Prepared Statement interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object. Used for precompiling SQL statements that might contain input parameters.

Syntax:

```
PreparedStatement pstmt = null;
try {
        String SQL = "Update Employees SET age =? WHERE id =?";
        pstmt = conn.prepareStatement(SQL);
        . . .
}
        catch (SQLException e) {
        . . .
}
finally {
        pstmt.close();
}
```

Callable Statement

Callable Statement object is used to execute a call to a database stored procedure. Using the Callable Statement objects is much like using the Prepared Statement objects. You must bind values to all the parameters before executing the statement, or you will receive an SQL Exception.

Syntax:

```
CallableStatement cstmt = null;
try {
        String SQL = "{call getEmpName (?, ?)}";
        cstmt = conn.prepareCall (SQL);
        . . .
}
catch (SQLException e) {
        . . .
```

```
}
finally {
        cstmt.close();
}
```

6. Write a short note on JDBC data types.

The JDBC driver converts the Java data type to the appropriate JDBC type, before sending it to the database. It uses a default mapping for most data types. For example, a Java int is converted to an SQL INTEGER. Default mappings were created to provide consistency between drivers.

- The setXXX() and updateXXX() methods enable you to convert specific Java types to specific JDBC data types.
- The methods, setObject() and updateObject(), enable you to map almost any Java type to a JDBC data type.
- ResultSet object provides corresponding getXXX() method for each data type to retrieve column value. Each method can be used with column name or by its ordinal position.

S.N0	SQL	JDBC/Java	setXXX	updateXXX
1	VARCHAR	java.lang.String	setString	updateString
2	CHAR	java.lang.String	setString	updateString
3	LONGVARCHAR	java.lang.String	setString	updateString
4	BIT	boolean	setBoolean	updateBoolean
5	NUMERIC	java.math.BigDecimal	setBigDecimal	updateBigDecimal
6	TINYINT	byte	setByte	updateByte
7	SMALLINT	short	setShort	updateShort
8	INTEGER	int	setInt	updateInt
9	BIGINT	long	setLong	updateLong
10	REAL	float	setFloat	updateFloat
11	FLOAT	float	setFloat	updateFloat

12	DOUBLE	double	setDouble	updateDouble
13	VARBINARY	byte[ ]	setBytes	updateBytes
14	BINARY	byte[ ]	setBytes	updateBytes
15	DATE	java.sql.Date	setDate	updateDate
15	TIME	java.sql.Time	setTime	updateTime
17	TIMESTAMP	java.sql.Timestamp	setTimestamp	updateTimestamp
18	CLOB	java.sql.Clob	setClob	updateClob
19	BLOB	java.sql.Blob	setBlob	updateBlob
20	ARRAY	java.sql.Array	setARRAY	updateARRAY
21	REF	java.sql.Ref	SetRef	updateRef
22	STRUCT	java.sql.Struct	SetStruct	updateStruct

7. Describe the Advanced Java applications.

There are a wide range of applications for advanced Java. Typically, programmers use it for web and network-focused applications and databases. Some of its applications include:

Mobile: Java is popular with mobile app developers because of its compatibility range.

Graphical user interfaces (GUIs): When developing GUIs within corporate networks, programmers often use advanced Java.

Web: Advanced Java is a popular choice for web applications, as it's easy to use and has a high level of security.

Enterprise: Developers of enterprise applications, such as banking applications, often use advanced Java because of its advantageous runtime environment and compatibility with web services.

Scientific: Advanced Java is a popular choice for developers for coding mathematical and scientific calculations.

Gaming: Game developers often use advanced Java for designing 3D games.

Big data: Databases commonly use advanced Java to help organize large volumes of information.

Distributed applications: Developers frequently use Java for distributed applications because of its persistent and dynamic nature.

Cloud-based applications: Java is a popular choice for cloud-based applications, as it's compatible with software as a service (SaaS) and similar applications for platforms (PaaS) and infrastructure (IaaS).

Example: Java Platform, Micro Edition (Java ME) provides a robust, flexible environment for applications running on embedded and mobile devices in the Internet of Things: micro-controllers, sensors, gateways, mobile phones, personal digital assistants (PDAs), TV set-top boxes, printers and more.

8. Explain the Transaction in JDBC with an example.

A SQL transaction is a grouping of one or more SQL statements that interact with a database. A transaction in its entirety can commit to a database as a single logical unit or rollback (become undone) as a single logical unit. In SQL, transactions are essential for maintaining database integrity.

Things required for transaction in JDBC:

To do transaction management in JDBC, we need to follow the below steps.

1) Disable auto commit mode of JDBC.
2) Put all operation of a transaction in try block.
3) If all operation is done successfully then commit in try block, otherwise rollback in catch block

A transaction is a group of operation used to performed one task if all operations in the group are success, then the task is finished and the transaction is successfully completed. If any one operation in the group is failed then the task is failed and the transaction is failed.

Example:

Suppose a Movie or Travel ticket booking at online is a transaction. This task contains four operations.

1) Verify the seats.
2) Reserve the seats
3) Payment
4) Issue tickets

If all the above four operations are done successfully then a transaction is finished successfully. In the middle, if any one operation is failed then all operation is canceled and finally a transaction is failed.

Types of Transactions:

Local Transaction: A local transaction means, all operation in a transaction is executed against one database.

Global Transaction: A global transaction means, all operations in a transaction are executed against multiple databases.

Example:

```
import java.sql. *;

class FetchRecords {
```

```
    public static void main(String args[])throws Exception {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:xe", "system", "oracle");
            con.setAutoCommit(false);
            Statement stmt=con.createStatement();
            stmt.executeUpdate("insert into user420 values(190,'abhi',40000)");
            stmt.executeUpdate("insert into user420 values(191,'umesh',50000)");
            con.commit();
            con.close();
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

If you see the table emp400, you will see that 2 records has been added.

9. Write a short answer about Batch Processing in JDBC.

Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast. It is because when one sends multiple statements of SQL at once to the database, the communication overhead is reduced significantly, as one is not communicating with the database frequently, which in turn results to fast performance.

The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

Steps in Batch Processing

1) **Load the driver class:**
   Class.forName() An efficient way to load the JDBC driver is to invoke the Class. forName(). newInstance() method, specifying the name of the driver class, as in the following example: Class. The class loading process triggers a static initialization routine that registers the driver instance with the DriverManager and associates this class with the database engine identifier, such as oracle or Postgres. After the registration is complete, we can use this identifier inside the JDBC URL as jdbc:oracle.

2) **Create Connection:**
   Create the connection object
   The getConnection() method of DriverManager class is used to establish connection with the database. JDBC makes it possible to establish a connection with a data source, send queries and update statements, and process the results. Simply, JDBC makes it possible to do the following things within a Java application: Establish a connection with a data source. Send queries and update statements to the data source.

3) Create Statement

From the connection interface, you can create the object for this interface. It is generally used for general-purpose access to databases and is useful while using static SQL statements at runtime.
Syntax: Statement statement = connection.
The JDBC Statement, CallableStatement, and PreparedStatement interfaces define the methods and properties that enable you to send SQL or PL/SQL commands and receive data from your database. They also define methods that help bridge data type differences between Java and SQL data types used in a database.

4) Add query in the batch

The addBatch() method of Statement, PreparedStatement, and CallableStatement is used to add individual statements to the batch . The executeBatch() is used to start the execution of all the statements grouped together.
Batch is a group of SQL statements that are executed at one time by SQL Server. These statements are sent to SQL Server by a program, such as the Query Analyzer. The opposite of a batch query is a single query, containing only one SQL statement.

5) Execute Batch

The executeBatch() method begins the execution of all the statements grouped together. The method returns an integer array, and each element of the array represents the updated count for the respective update statement.

6) Close Connection

con.close() to close a JDBC Connection once you are done with it.
A database connection takes up number of resources, both inside your own application, but especially on the database server. Therefore, keeping database connections open that are unused will require the database to keep unnecessary resources allocated for the connection.