Part - A

1. **What is Core Java?**
   Core Java covers the fundamental concepts in the Java programming language to develop general applications. It covers topics like OOP, datatypes, operators, exception handling, swing and others.

2. **What is Advanced Java?**
   Advanced Java is used to build enterprise level applications. Advance Java covers advanced concepts and allows us to build Client-Server architecture for Web Applications.

3. **Define JVM.**
   JVM, or Java Virtual Machine, is a component of the Java runtime environment that executes Java bytecode, enabling platform-independent execution of Java applications. It provides features like memory management, garbage collection, and platform independence.

4. **What is full form of JDBC?**
   Java Database Connectivity.

5. **Thin JDBC driver is called as ________________**
   Fully java driver.

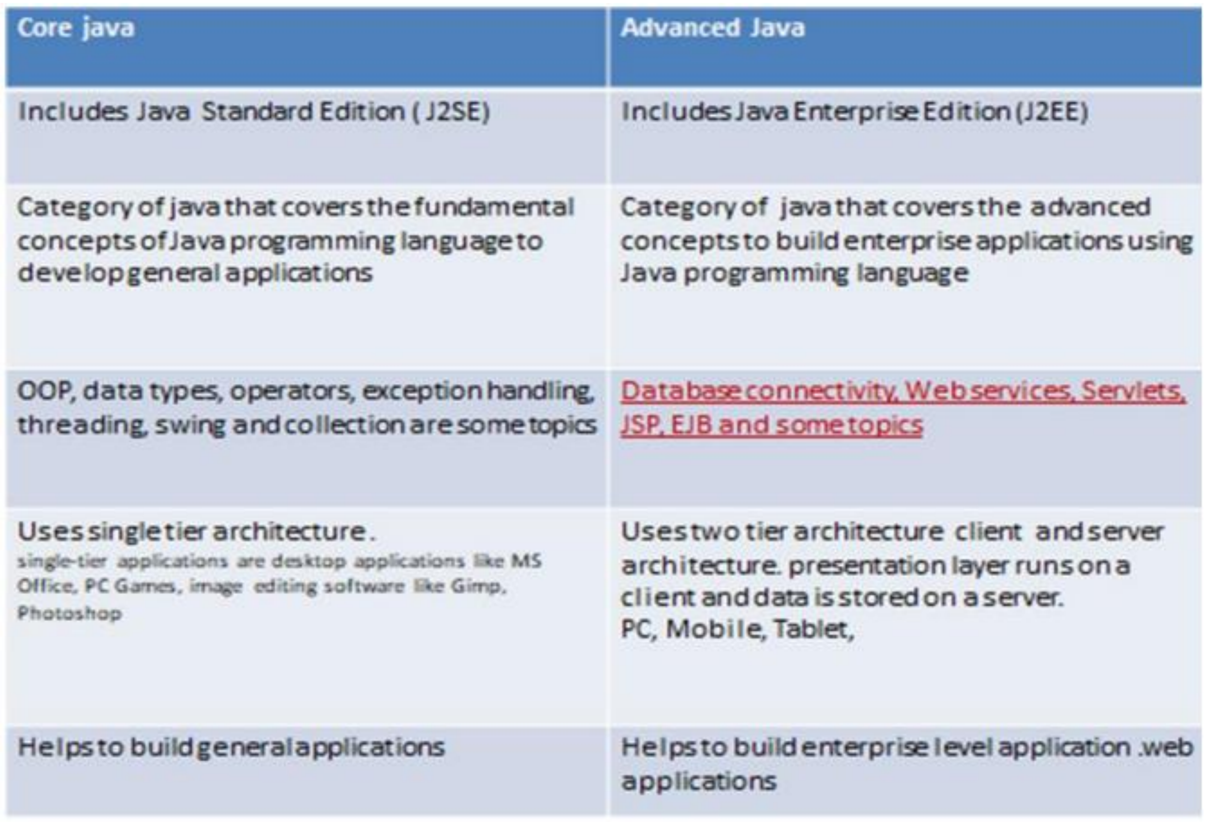
6. **What is JDBC Statements?**
   A JDBC statement is a Java interface that allows you to execute SQL queries and interact with a database. It's used to send SQL commands to the database and retrieve results, such as querying for data, inserting records, updating records, or deleting records.

7. **Define batch processing.**
   Batch processing refers to executing multiple SQL statements as a group or batch, rather than individually. This can be useful for processing multiple database operations together for efficiency and faster performance.

8. **What is Stored Procedure in SQL?**
   A stored procedure is a prepared SQL code that you can save and reuse repeatedly. A stored procedure can consist of multiple SQL statements like SELECT, INSERT, UPDATE, or DELETE. It can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Part - B

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. 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.

3. Define JDBC Concept with example.

JDBC (Java Database Connectivity) is a Java-based API (Application Programming Interface) that allows Java applications to interact with relational databases. It provides a standard interface for connecting to databases, sending SQL queries, and processing the results. JDBC makes it possible to access and manipulate data stored in a database from within a Java application.

Example: Connecting to a Database and Displaying Data

```
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());
        }
    }
}
```

In this example:

- We establish a database connection using **DriverManager.getConnection()**.
- We create a **Statement** to execute SQL queries on the database.
- We execute an SQL query to select data from the "**mca**" table.
- We process the query results using a **ResultSet**, extracting and displaying the data.

This example demonstrates a fundamental JDBC concept: connecting to a database, executing a query, and retrieving and processing data from the database. JDBC provides a powerful and flexible way to interact with databases in Java applications.

4. 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.

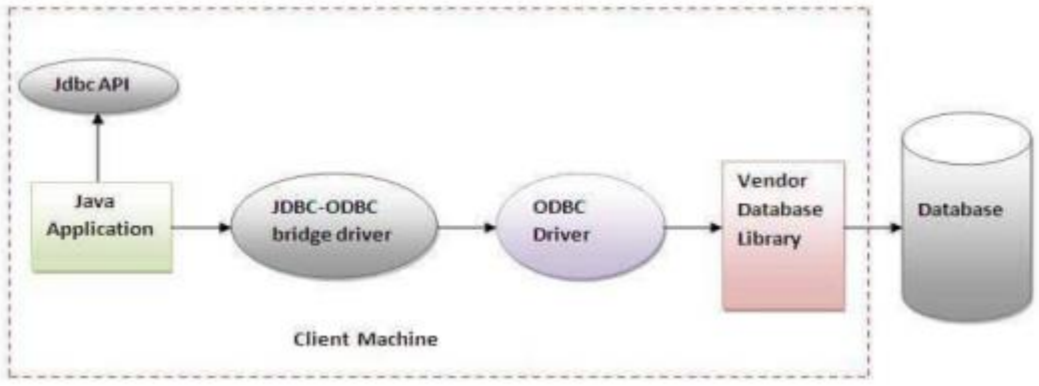


Figure- JDBC-ODBC Bridge Driver

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) **Native-API driver (partially java 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.

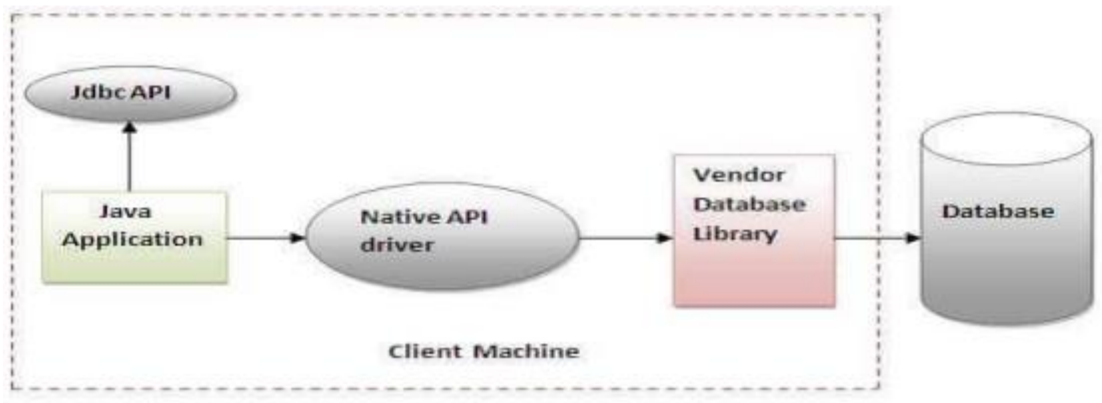


Figure- Native API Driver

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) **Network Protocol driver (fully java 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.

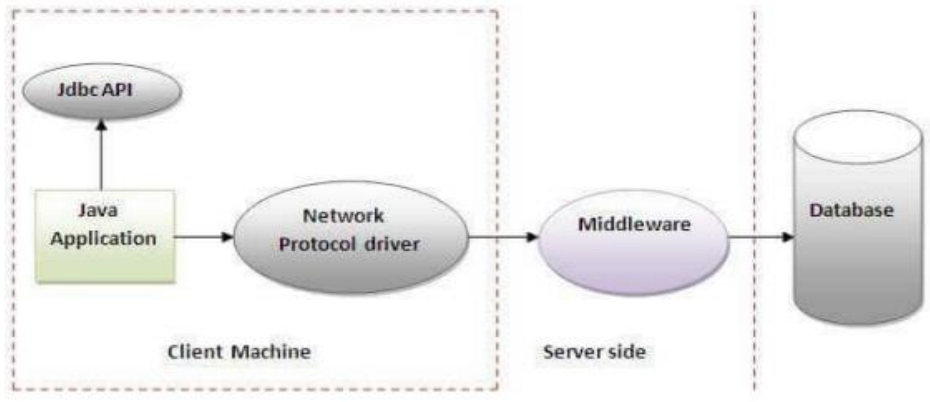


Figure- Network Protocol Driver

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) <u>Thin driver (fully java driver)</u>

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.

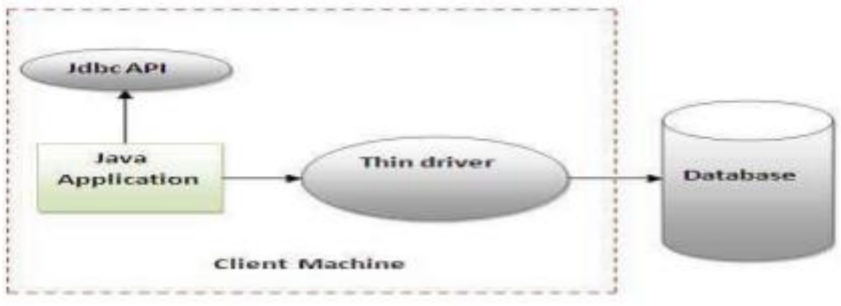


Figure- Thin Driver

Advantage:

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

Disadvantage:

- Drivers depend on the Database

5. Write a short note on JDBC data types.

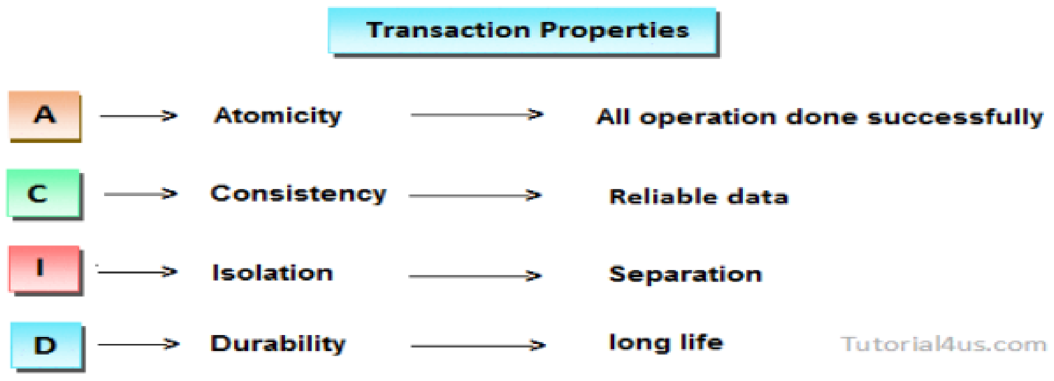
The JDBC driver converts the Java data type to the appropriate JDBC type, before sending it to the database. It uses a default mapping for most data types. For example, a Java int is converted to an SQL INTEGER. Default mappings were created to provide consistency between drivers.

- 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.NO	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

6. **Explain the properties of transaction managements.**

Every transaction follows some transaction properties these are called ACID properties.



Atomicity: Atomicity of a transaction is nothing but, in a transaction, either all operations can be done or all operation can be undone, but some operations are done and some operations are undone should not occur.

Consistency: Consistency means, after a transaction completed with successful, the data in the data store should be a reliable data this reliable data is also called as consistent data.

Isolation: Isolation means, if two transactions are going on same data then one transaction will not disturb another transaction.

Durability: Durability means, after a transaction is completed the data in the data store will be permanent until another transaction is going to be performed on that data.

7. **Explain the Advantage of Transaction Managements.**

Atomicity: Transactions ensure atomicity. If any part of the transaction fails, all changes made to the database are rolled back, maintaining the integrity of the data. This prevents the database from being left in an inconsistent state due to partial updates.

Consistency: Transactions maintain the consistency of the database. Before and after a transaction, the database adheres to predefined integrity constraints and business rules. If a transaction fails, the database returns to a consistent state without violating these constraints.

Isolation: Isolation levels prevent one transaction from seeing uncommitted changes made by other transactions, thus ensuring data reliability.

Durability: Once a transaction is committed, its changes are permanent and durable. Even in the event of a system crash or failure, the database system ensures that the committed changes are saved and not lost.

Error Handling: Transactions allow for proper error handling. In the case of an error or exception during a transaction, the application can choose to either commit the changes if everything is successful or roll back to the previous state if an error occurs. This helps maintain data integrity and recover from errors gracefully.

Concurrency Control: Transactions help manage concurrent access to the database by allowing multiple users to work with the data simultaneously. The database system automatically handles locking and unlocking of records to prevent conflicts and ensure data integrity.

Reusability and Maintainability: By encapsulating related database operations within a transaction, the code becomes more modular and maintainable. Transactions can be reused in various parts of the application, promoting code reusability.

Data Integrity: Transactions help maintain the integrity of data relationships in the database. When multiple related tables need to be updated together, a transaction ensures that these updates are consistent.

8. 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();
}
```

9. Explain the Batch Processing in JDBC with example.

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.

Example:

```
import java.sql.*;

class FetchRecords {
        public static void main(String args[])throws Exception{
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection con = DriverManager.getConnection ("jdbc:oracle:thin:@local
                host:1521:xe","system", oracle");
                con.setAutoCommit(false);
                Statement stmt=con.createStatement();
                stmt.addBatch("insert into user420 values(190,'abhi',40000)");
                stmt.addBatch("insert into user420 values(191,'umesh',50000)");
                stmt.executeBatch();//executing the batch
                con.commit();
                con.close();
        }
}
```

// If you see the table user420, two records have been added.

Steps in Batch Processing

1) **Load the driver class:**
Class.forName() An efficient way to load the JDBC driver is to invoke the Class, by specifying the name of the driver class. The class loading process triggers a static initialization routine that registers the driver instance with the DriverManager. After the registration is complete, we can use this identifier inside the JDBC URL as jdbc:oracle.

2) **Create Connection:**
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.

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.

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.

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. Therefore, keeping database connections open that are unused will require the database to keep unnecessary resources allocated for the connection.

PART - C

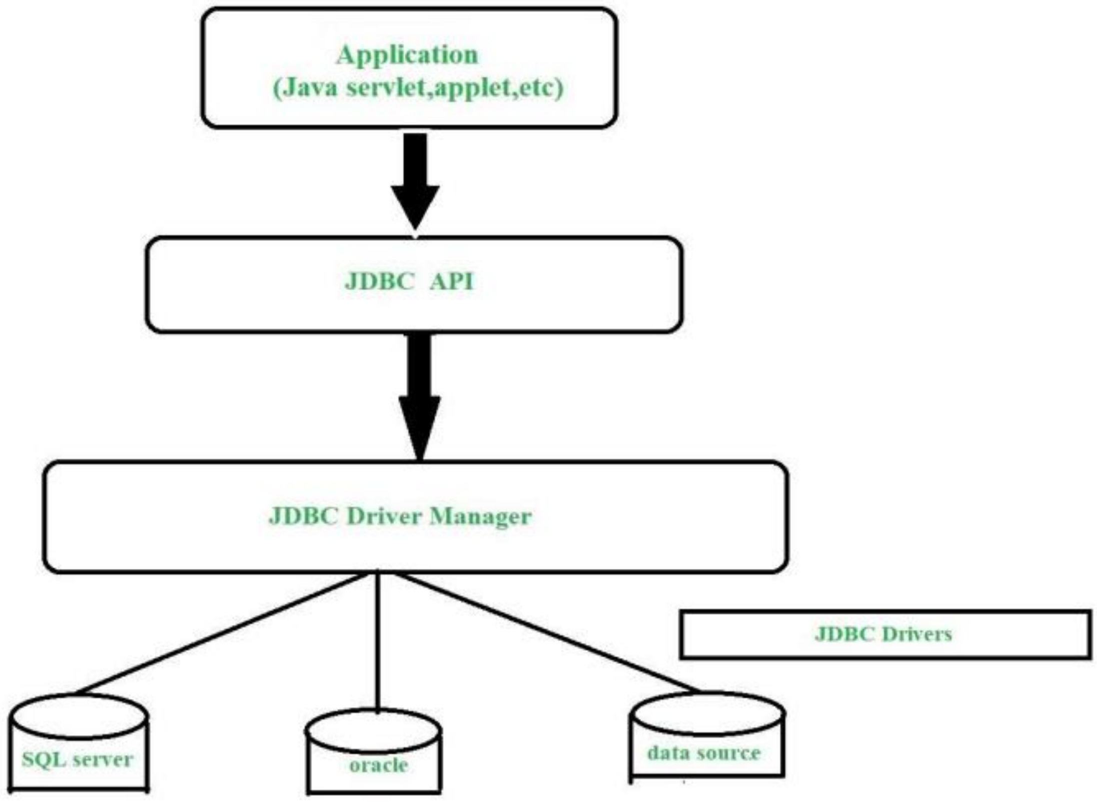
1. Explain the components of JDBC with neat diagram.

There are generally four main components of JDBC through which it can interact with a database. They are as mentioned below:

1) **JDBC API**: It provides various methods and interfaces for easy communication with the database. It provides two packages as follows, which contain the java SE and Java EE platforms to exhibit WORA (write once run anywhere) capabilities.
Syntax: java.sql.*;
It also provides a standard to connect a database to a client application.
2) **JDBC Driver manager:** It loads a database-specific driver in an application to establish a connection with a database. It is used to make a database-specific call to the database to process the user request.

3) **JDBC Test suite:** It is used to test the operation (such as insertion, deletion, updation) being performed by JDBC Drivers.

4) **JDBC-ODBC Bridge Drivers:** It connects database drivers to the database. This bridge translates the JDBC method call to the ODBC function call. It makes use of the **sun.jdbc.odbc** package which includes a native library to access ODBC characteristics.



Description:

- **Application:** It is a java applet or a servlet that communicates with a data source.
- **The JDBC API:** The JDBC API allows Java programs to execute SQL statements and retrieve results.
- **Driver Manager:** It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.
- **JDBC drivers:** To communicate with a data source through JDBC, you need a JDBC driver that intelligently communicates with the respective data source.

2. Write a java program to access the MySQL database.

<u>**MySQL database:**</u>

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());
 }
   }
}
```

3. Explain the steps to create a JDBC Database Connection.

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

1. Register the driver class
2. Create the connection object
3. Create the Statement object
4. Execute the query
5. Close the connection object

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.

Syntax of forName() method:

public static void forName(String className)throws ClassNotFoundException

Example to register the OracleDriver class
Here, Java program is loading oracle driver to esteblish database connection.
Class.forName("oracle.jdbc.driver.OracleDriver");

2) **<u>Create the connection object</u>**
The getConnection() method of DriverManager class is used to establish connection with the database.
Syntax of getConnection() method

- public static Connection getConnection(String url)throws SQLException

Example to establish connection with the Oracle database
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","pass
word");

3) **<u>Create the Statement object</u>**
The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.
Syntax of createStatement() method
public Statement createStatement()throws SQLException

Example to create the statement object
Statement stmt=con.createStatement();

4) **<u>Execute the query</u>**
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.
Syntax of executeQuery() method
public ResultSet executeQuery(String sql)throws SQLException
Example to execute query

```
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next()){
   System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

5) **<u>Close the connection object</u>**
By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.
Syntax of close() method
public void close()throws SQLException

Example to close connection
con.close();