Q1. What is JDBC? Explain the types of JDBC drivers?

Ans. **What is JDBC?**

- JDBC is an API, which is used in java programming for interacting with database.
- JDBC (Java DataBase Connection) is the standard method of accessing **databases** from **Java application**.
- JDBC is a specification from **Sun Microsystem** that provides a **standard API** for java application to communicate with different database.
- JDBC is a **platform independent** interface between relational database and java applications.

JDBC Drivers

1. **Type1 (JDBC-ODBC Driver)**
   - Depends on support for ODBC
   - Type1 is not portable driver
   - Translate JDBC calls into ODBC calls and use Windows ODBC built in drivers
   - ODBC must be set up on every client
   - For server side servlets ODBC must be set up on web server
   - Driver sun.jdbc.odbc.JdbcOdbc provided by JavaSoft with JDK
   - No support from JDK 1.8 (Java 8) onwards.
     E.g. MS Access

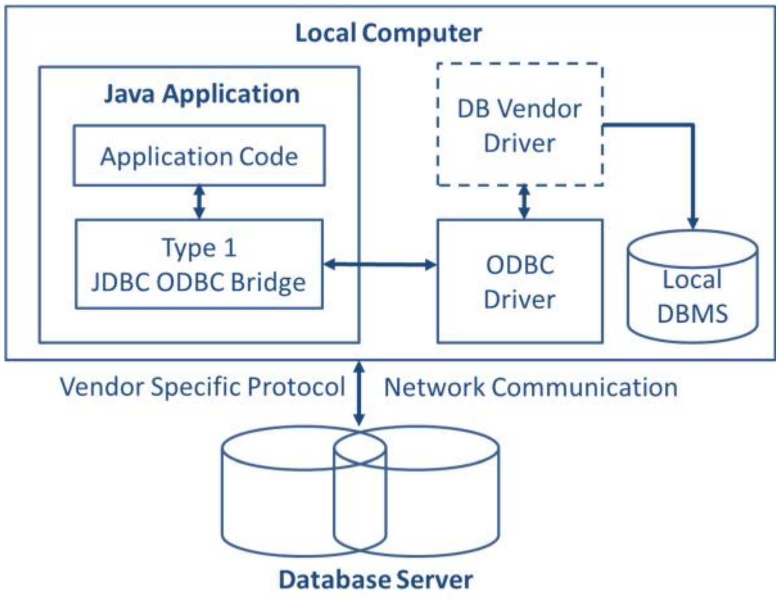

Figure: Type1 (JDBC-ODBC Driver)

Advantages :
- Allow to communicate with all database supported by ODBC driver
- It is vendor independent driver

Disadvantages:
- Due to large number of translations, execution speed is decreased
- Dependent on the ODBC driver
- ODBC binary code or ODBC client library to be installed in every client machine
- Uses java native interface to make ODBC call

- Because of listed disadvantage, type1 driver is not used in production environment. It can only be used, when database doesn't have any other JDBC driver implementation.

2. Type 2 (Native Code Driver)

- JDBC API calls are converted into native API calls, which are unique to the database.
- These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge.
- Native code Driver are usually written in C, C++.
- The vendor-specific driver must be installed on each client machine.
- Type 2 Driver is suitable to use with server side applications.
- E.g. Oracle OCI driver, Weblogic OCI driver, Type2 for Sybase

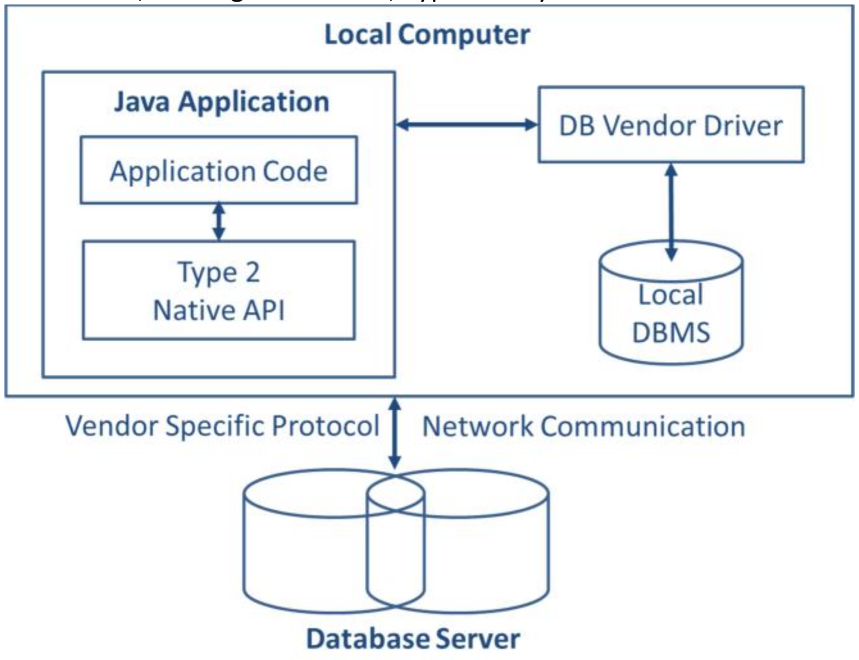

Figure: Type 2 (Native Code Driver)

Advantages

- As there is no implementation of JDBC-ODBC bridge, it may be considerably faster than a Type 1 driver.

Disadvantages

- The vendor client library needs to be installed on the client machine hence type 2 drivers cannot be used for the Internet.
- This driver is platform dependent.
- This driver supports all java applications except applets.
- It may increase cost of application, if it needs to run on different platform (since we may require buying the native libraries for all of the platform).
- Mostly obsolete now
- Usually not thread safe

3. **Type 3 (Java Protocol)**
   - This driver translate the jdbc calls into a database server independent and middleware server specific calls.
   - With the help of the middleware server, the translated jdbc calls further translated into database server specific calls.
   - This type of driver also known as net-protocol fully java technology-enabled driver.
   - Type-3 driver is recommended to be used with applets. its auto-downloadable.
   - Can interface to multiple databases – Not vendor specific.
   - Follows a three-tier communication approach.
   - The JDBC clients use standard network sockets to communicate with a middleware application server.
   - The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
   - This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

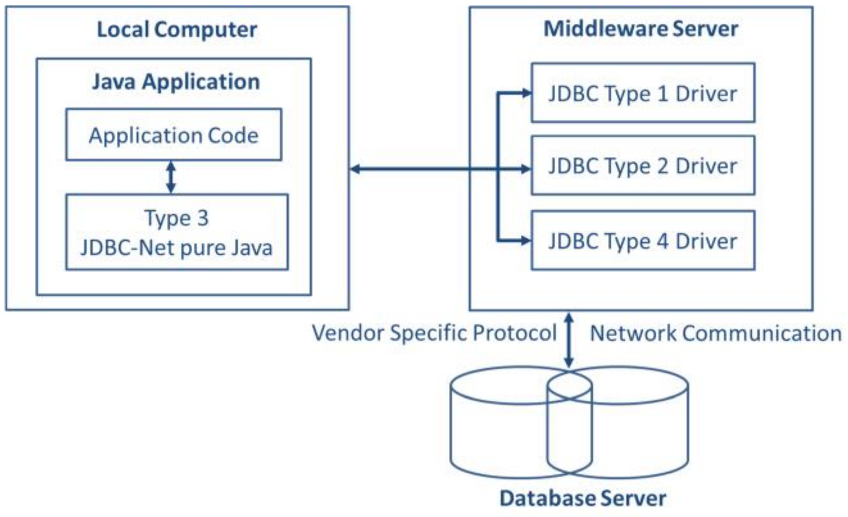

Figure: Type 3 (Java Protocol)

Advantages
- Since the communication between client and the middleware server is database independent, there is no need for the database vendor library on the client.
- A single driver can handle any database, provided the middleware supports it.
- We can switch from one database to other without changing the client-side driver class, by just changing configurations of middleware server.
  E.g.: IDS Driver, Weblogic RMI Driver

Disadvantages
- Compared to Type 2 drivers, Type 3 drivers are slow due to increased number of network calls.
- Requires database-specific coding to be done in the middle tier.
- The middleware layer added may result in additional latency, but is typically overcome by using better middleware services.

4. Type 4 (Database Protocol)

- It is known as the Direct to Database **Pure Java Driver**
- Need to download a new driver for each database engine
- Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection.
- This kind of driver is extremely flexible, you don't need to install special software on the client or server.
- This type of driver is lightweight and generally known as thin driver.
- You can use this driver when you want an auto downloadable option the client side application
- i.e. thin driver for oracle from oracle corporation, weblogic and ms sqlserver4 for ms sql server from BEA system

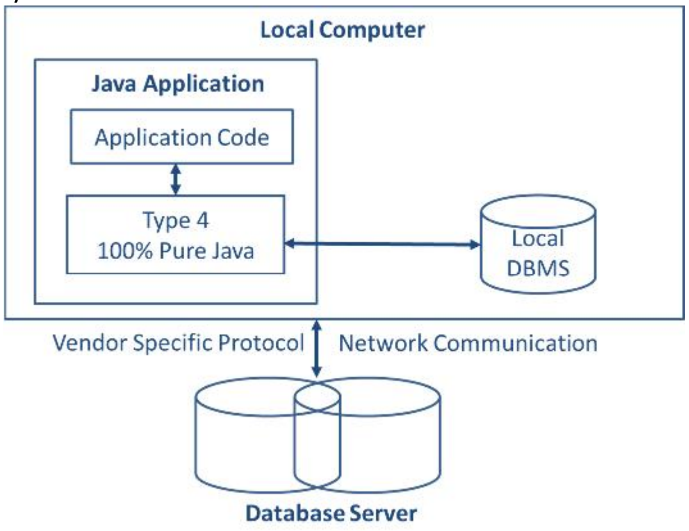

Figure: Type 4 (Database Protocol)

Advantages

- Completely implemented in Java to achieve platform independence.
- No native libraries are required to be installed in client machine.
- These drivers don't translate the requests into an intermediary format (such as ODBC).
- Secure to use since, it uses database server specific protocol.
- The client application connects directly to the database server.
- No translation or middleware layers are used, improving performance.
- The JVM manages all the aspects of the application-to-database connection.

Disadvantage

- This Driver uses database specific protocol and it is DBMS vendor dependent.

Comparison between JDBC Drivers

Type	**Type 1**	**Type 2**	**Type 3**	**Type 4**
Name	JDBC-ODBC Bridge	Native Code Driver/ JNI	Java Protocol/ Middleware	Database Protocol
Vendor Specific	No	Yes	No	Yes
Portable	No	No	Yes	Yes
Pure Java Driver	No	No	Yes	Yes
Working	JDBC-> ODBC call ODBC -> native call	JDBC call -> native specific call	JDBC call -> middleware specific. Middleware -> native call	JDBC call ->DB specific call
Multiple DB	Yes [only ODBC supported DB]	NO	Yes [DB Driver should be in middleware]	No
Example	MS Access	Oracle OCI driver	IDA Server	MySQL
Execution Speed	Slowest among all	Faster Compared to Type1	Slower Compared to Type2	Fastest among all
Driver	Thick Driver	Thick Driver	Thin Driver	Thin Driver

Q2. Explain Thick and Thin driver. Comment on selection of driver. Write code snippet for each type of JDBC connection.

Ans. **Thick driver**

- Thick client would need the client installation.
  E.g. Type 1 and Type 2.

Thin driver

- The thin client driver, which mean you can connect to a database without the client installed on your machine.
  E.g. Type 4

Comment on selection of driver

- If you are accessing one type of database such as MySQL, Oracle, Sybase or IBM etc., the preferred driver type is 4.
- If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.
- Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.
- The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

Write code snippet for each type of JDBC connection

1. **MySQL**

```
Class.forName("com.mysql.jdbc.Driver");

Connection conn=
DriverManager.getConnection("jdbc:mysql://localhost:PortNo/database
Name",“uid”, “pwd”);
```

2. **Oracle**

```
Class.forName("oracle.jdbc.driver.OracleDriver");

Connection conn=
DriverManager.getConnection("jdbc:oracle:thin:@hostname:port
Number:databaseName","root", "pwd");
```

3. **DB2**

```
Class.forName("com.ibm.db2.jdbc.net.DB2Driver");

Connection conn=
DriverManager.getConnection("jdbc:db2:hostname:port Number
/databaseName")
```

Q3. Explain Statement Interface with appropriate example.

Ans. **Java.sql.Statement**

- Used for general-purpose access to your database.
- Useful for **static** SQL statements, e.g. SELECT specific row from table etc.
- The Statement interface defines a standard abstraction to execute the SQL statements requested by a user and return the results by using the ResultSet object.
- The Statement interface is created after the connection to the specified database is made.
- The object is created using the createStatement() method of the Connection interface, as shown in following code snippet:

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

```
java.sql.*;
public class ConnDemo {
public static void main(String[] args) {
  try {
      // Load and register the driver
       Class.forName("com.mysql.jdbc.Driver");

     // Establish the connection to the database server
      Connection conn= DriverManager.getConnection
      ("jdbc:mysql://localhost:3306/database_name","root","pwd");
      // Create a statement
    Statement stmt = conn.createStatement();
// Execute the statement
       ResultSet rs = stmt.executeQuery("SELECT * from Table");

  // Retrieve the results
   while(rs.next()){
      System.out.print(rs.getInt(1)+"\t");
      System.out.print(rs.getString(“Name”)+"\t");
                 System.out.println(rs.getString(3));
   }//while

// Close the statement and connection
        stmt.close();
        conn.close();
 }catch(Exception e){System.out.println(e.toString());
}//PSVM
}//class
```

Q4. Explain Prepared Statement with example.

Ans.

- The PreparedStatement interface is subclass of the Statement interface, can be used to represent a precompiled query, which can be executed multiple times.
- Prepared Statement is used when you plan to execute same SQL statements many times.
- PreparedStatement interface accepts input parameters at runtime.
- A SQL statement is precompiled and stored in a PreparedStatement object.
- This object can then be used to efficiently execute this statement multiple times.
- The object is created using the prepareStatement() method of Connection interface, as shown in following snippet:

```
String query = “insert into emp values(? ,?)”;
PreparedStatement ps = con.prepareStatement(query);
ps.setInt(1,5);
ps.setString(2,”New Employee”);
int n = ps.executeUpdate();
```

Advantages:

- The performance of the application will be faster, if you use PreparedStatement interface because query is compiled only once.
- This is because creating a PreparedStatement object by explicitly giving the SQL statement causes the statement to be precompiled within the database immediately.
- Thus, when the PreparedStatement is later executed, the DBMS does not have to recompile the SQL statement.
- Late binding and compilation is done by DBMS.
- Provides the programmatic approach to set the values.

Disadvantage:

The main disadvantage of PreparedStatement is that it can represent only one SQL statement at a time.

Example of PreparedStatement

Write a program to insert student records to database using prepared statement

```
import java.sql.*;
public class PreparedInsert {
public static void main(String[] args) {
try {
Class.forName("com.mysql.jdbc.Driver");
Connection conn= DriverManager.getConnection
("jdbc:mysql://localhost:3306/DIET", "root","pwd");

String query="insert into dietstudent values(?,?,?,?)";
PreparedStatement ps=conn.prepareStatement(query);
ps.setString(1, "14092"); //Enr_no
```

```
        ps.setString(2, "abc_comp"); //Name
        ps.setString(3, "computer"); //Branch
        ps.setString(4, "cx"); //Division
        int i=ps.executeUpdate();

        System.out.println("no. of rows updated ="+i);
        ps.close();
        conn.close();
        }catch(Exception e){System.out.println(e.toString());} }//PSVM
}//class
```

Q5. Explain Callable Statement with example.

Ans.

- CallableStatement interface is used to call the stored procedures.
- Therefore, the stored procedure can be called by using an object of the CallableStatement interface.
- The object is created using the prepareCall() method of Connection interface.

```
CallableStatement cs=conn.prepareCall("{call Proc_Name(?,?)}");
cs.setInt(1,2222);
cs.registerOutParameter(2,Types.VARCHAR);
cs.execute();
```

- Three types of parameters exist: IN, OUT, and INOUT.
- PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.

Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. You bind values to IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. You retrieve values from the OUT parameters with the getXXX() methods.
INOUT	A parameter that provides both input and output values. You bind variables with the setXXX() methods and retrieve values with the getXXX() methods.

Example of CallableStatement

Writa a Callable Statement program to retrieve branch of the student using {getBranch() procedure} from given enrollment number. Also write code for Stored Procedure

Stored Procedure: getbranch()

```
DELIMITER @@
DROP PROCEDURE getbranch @@
CREATE PROCEDURE databaseName.getbranch
(IN enr_no INT, OUT my_branch VARCHAR(10))
BEGIN
SELECT branch INTO my_branch
FROM dietStudent
WHERE enr_no=enrno;
END @@
DELIMITER ;
```

Callable Statement program

```
import java.sql.*;
public class CallableDemo {
public static void main(String[] args) {
try {
  Class.forName("com.mysql.jdbc.Driver");
  Connection conn= DriverManager.getConnection
         ("jdbc:mysql://localhost:3306/Diet", "root","pwd");

  CallableStatement cs=conn.prepareCall("{call getbranch(?,?)}");
                 cs.setInt(1,2222);
                 cs.registerOutParameter(2,Types.VARCHAR);
                 cs.execute();
                 System.out.println("branch="+cs.getString(2));
                 cs.close();
                 conn.close();
             }catch(Exceptione){System.out.println(e.toString());}
            }//PSVM
}//class
```

Q6. Differentiate Statement, Prepared Statement and Callable Statement.

Ans.

Statement	**Prepared Statement**	**Callable Statement**
Super interface for Prepared and Callable Statement	extends Statement (sub-interface)	extends PreparedStatement (sub-interface)
Used for executing simple SQL statements like CRUD (create, retrieve, update and delete	Used for executing dynamic and pre-compiled SQL statements	Used for executing stored procedures
The Statement interface cannot accept parameters.	The PreparedStatement interface accepts input parameters at runtime.	The CallableStatement interface can also accept runtime input parameters.
stmt = conn.createStatement();	PreparedStatement ps=con.prepareStatement ("insert into studentDiet values(?,?,?)");	CallableStatement cs=conn.prepareCall("{call getbranch(?,?)}");
java.sql.Statement is slower as compared to Prepared Statement in java JDBC.	PreparedStatement is faster because it is used for executing precompiled SQL statement in java JDBC.	None
java.sql.Statement is suitable for executing DDL commands - CREATE, drop, alter and truncate in java JDBC.	java.sql.PreparedStatement is suitable for executing DML commands - SELECT, INSERT, UPDATE and DELETE in java JDBC.	java.sql.CallableStatement is suitable for executing stored procedure.

Q7. Explain JDBC Architecture.

Ans. **JDBC API**

- The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.
- JDBC API provides **classes** and **interfaces** to connect or communicate Java application with database.
- The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −
    1. JDBC API: This provides the application-to-JDBC Manager connection.
    2. JDBC Driver API: This supports the JDBC Manager-to-Driver Connection.

JDBC Driver Manager (*Class*)

- This class manages a list of database drivers.
- It ensures that the correct driver is used to access each data source.
- The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

- Matches connection requests from the java application with the proper database driver using communication sub protocol.
- The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.

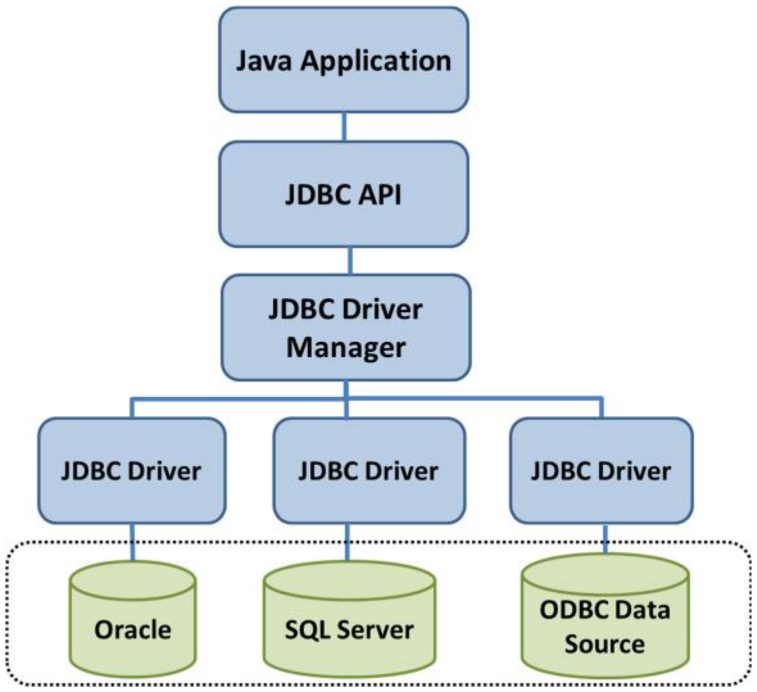

Figure: JDBC Architecture

Driver (*Interface*)

- This interface handles the communications with the database server.
- You will interact directly with Driver objects very rarely.
- Instead, you use DriverManager objects, which manages objects of this type.
- It also abstracts the details associated with working with Driver objects.

Connection (*Interface*)

- This interface with all methods for contacting a database.
- The connection object represents communication context, i.e., all communication with database is through connection object only.

Statement (*Interface*)

- You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

ResultSet (*Interface*)

- These objects hold data retrieved from a database after you execute an SQL query using Statement objects.
- It acts as an iterator to allow you to move through its data.

SQLException (*Class*)

This class handles any errors that occur in a database application.

Q8. Explain methods of ResultSet Interface.

Ans. **Categories**

1.	Navigational methods	Used to move the cursor around.
2.	Get methods	Used to view the data in the columns of the current row being pointed by the cursor.
3.	Update methods	Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

ResultSet: Navigational methods

boolean first() throws SQLException	Moves the cursor to the first row.
boolean last() throws SQLException	Moves the cursor to the last row.
boolean next() throws SQL Exception	Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
boolean previous() throws SQLException	Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
boolean absolute(int row) throws SQLException	Moves the cursor to the specified row.
boolean relative(int row) throws SQLException	Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
int getRow() throws SQLException	Returns the row number that the cursor is pointing to.

ResultSet: Get methods

int getInt (String columnName) throws SQLException	Returns the integer value to the current row in the column named columnName.
int getInt (int columnIndex) throws SQLException	Returns the integer value to the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

String getString (String columnLabel) throws SQLException	Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.
String getString (int columnIndex) throws SQLException	Retrieves the value of the designated column in the current row of this ResultSet object as a String in the Java programming language.

ResultSet: Update methods

void updateString (int col_Index, String s) throws SQLException	Changes the String in the specified column to the value of s.
void updateInt (int col_Index, int x) throws SQLException	Updates the designated column with an integer value.
void updateFloat (int col_Index, float x) throws SQLException	Updates the designated column with a float value.
void updateDouble (int col_Index,double x) throws SQLException	Updates the designated column with a double value.

Q9. Differentiate executeQuery(), executeUpdate() and execute() with appropriate example.

Ans.

executeQuery()	**executeUpdate()**	**execute()**
ResultSet executeQuery (String sql) throws SQLException	int executeUpdate(String sql) throws SQLException	Boolean execute(String sql) throws SQLException
This is used generally for reading the content of the database. The output will be in the form of ResultSet. Generally SELECT statement is used.	This is generally used for altering the databases. Generally DROP, INSERT, UPDATE, DELETE statements will be used in this. The output will be in the form of int. This int value denotes the number of rows affected by the query.	If you dont know which method to be used for executing SQL statements, this method can be used. This will return a boolean. TRUE indicates the result is a ResultSet and FALSE indicates it has the int value which denotes number of rows affected by the query.
E.g.:ResultSet rs= stmt.executeQuery(query);	E.g.: int i= stmt.executeUpdate(query);	E.g.: Boolean b= stmt.execute(query);

Q10. Explain Resultset Type and Concurrency

Ans. Resultset Type

ResultSet.*TYPE_FORWARD_ONLY*	The cursor can only move forward in the result set. (Default Type)
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

Concurrency of ResultSet

ResultSet.*CONCUR_READ_ONLY*	Creates a read-only result set. (Default Type)
ResultSet.*CONCUR_UPDATABLE*	Creates an updateable result set.

Example

```
Statement stmt = conn.createStatement(
                         ResultSet.TYPE_FORWARD_ONLY,
                         ResultSet.CONCUR_READ_ONLY);
```

Q11. Explain ResultsetMetaData Interface with Example.

Ans.

- Metadata means data about data.
- If you have to get metadata of a table like
  1. total number of column
  2. column name
  3. column type etc.
- ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

Example: ResultSetMetaData

```
import java.sql.*;
public class MetadataDemo {
public static void main(String[] args) {
    try {Class.forName("com.mysql.jdbc.Driver");
        Connection conn= DriverManager.getConnection
        ("jdbc:mysql://localhost:3306/gtu", "root",“pwd");
Statement stmt = conn.createStatement
(ResultSet.TYPE_FORWARD_ONLY,ResultSet.CONCUR_READ_ONLY);
 ResultSet rs = stmt.executeQuery("SELECT * from gtu");

    ResultSetMetaData rsmd=rs.getMetaData();
    System.out.println("Total columns:
                        "+rsmd.getColumnCount());
    System.out.println("Column Name of 1st column:
                        "+rsmd.getColumnName(1));
    System.out.println("Column Type Name of 1st column:“
                +rsmd.getColumnTypeName(1));
            stmt.close();
            conn.close();
        }catch(Exception e)
    {System.out.println(e.toString());}
       }//PSVM
    }//class
```

OUTPUT:

```
Total columns: 3
Column Name of 1st column:Enr_no
Column Type Name of 1st column:INT
```

Q12. Explain DatabaseMetaData Interface with Example

Ans. DatabaseMetaData interface provides methods to get meta data of a database such as

1. Database product name
2. Database product version
3. Driver name
4. Name of total number of tables etc.

Example: DatabaseMetaData

```
import java.sql.*;
public class DatabaseMetaDataDemo {
 public static void main(String[] args) {
  try {
       Class.forName("com.mysql.jdbc.Driver");
       Connection con=
DriverManager.getConnection("jdbc:mysql://localhost:3306/temp6","r
oot","root");
       DatabaseMetaData dbmd=con.getMetaData();
       System.out.println
      ("getDatabaseProductName:"+dbmd.getDatabaseProductName());
       System.out.println("getDatabaseProductVersion():
                           "+dbmd.getDatabaseProductVersion());

    System.out.println("getDriverName():"+dbmd.getDriverName())
                                                   ;
      System.out.println("getDriverVersion():
                                    "+dbmd.getDriverVersion());
     System.out.println("getURL():"+dbmd.getURL());
System.out.println("getUserName():"+dbmd.getUserName());

           } catch (Exception ex) {
            System.out.println("Exception:"+ex.toString());
        }}}
```

OUTPUT:

```
getDatabaseProductName:MySQL
getDatabaseProductVersion():5.6.16
getDriverName():MySQL-AB JDBC Driver
getDriverVersion():mysql-connector-java-5.1.23 ( Revision:
                                       ${bzr.revision-id} )
getURL():jdbc:mysql://localhost:3306/temp6
getUserName():root@localhost
```

Q13. Explain Transaction Management in JDBC with appropriate example.

Ans.

- Transaction Management in java is required when we are dealing with relational databases.
- By default when we create a database connection, it runs in **auto-commit** mode.
- It means that whenever we execute a query and it's completed, the commit is fired automatically.
- So every SQL query we fire is a transaction and if we are running some DML or DDL queries, the changes are getting saved into database after every SQL statement finishes.
- Sometimes we want a group of SQL queries to be part of a transaction so that we can commit them when all the queries runs fine. If we get any exception, we have a choice of rollback all the queries executed as part of the transaction.

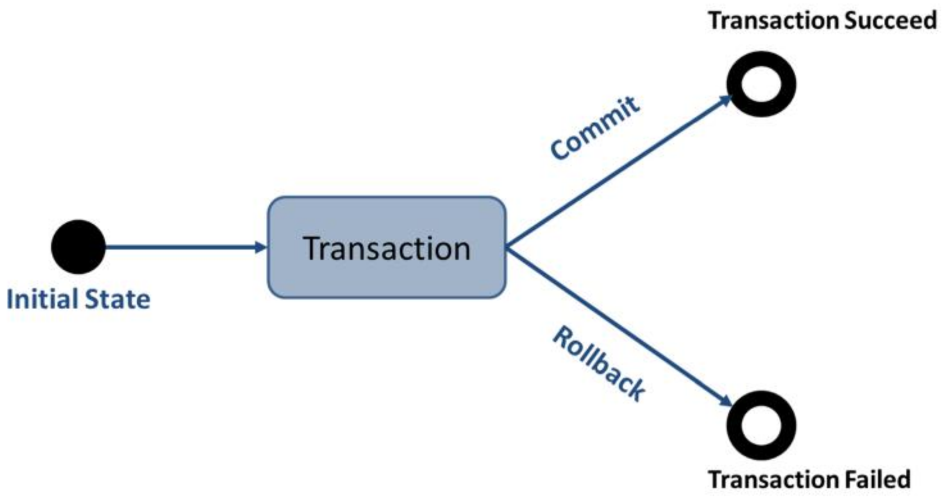

Figure: Transaction Management

Advantage of Transaction Management

- **Fast performance:** It makes the performance fast because database is hit at the time of commit.

In JDBC, **Connection** interface provides methods to manage transaction.

void setAutoCommit(boolean status)	It is true by default means each transaction is committed by default.
void commit()	Commits the transaction.
void rollback()	Cancels the transaction.

Example

```
import java.sql.*;
class RollbackDemo{
public static void main(String args[]){
try{ Class.forName("com.mysql.jdbc.Driver");
  Connection con=DriverManager.getConnection(
             "jdbc:mysql://localhost:3306/GTU","root","root");
  con.setAutoCommit(false);//bydeafault it is true
  Statement stmt=con.createStatement();
  int i=stmt.executeUpdate("insert into diet
                              values(606,'ghi','ee')");
  con.commit(); //Commit Transaction
  i+=stmt.executeUpdate("insert into diet
                              values(607,'mno','ch')");
    System.out.println("no. of rows inserted="+i);
    con.rollback(); //Rollback Transaction
    con.close();
   }catch(Exception e){ System.out.println(e);}
  }}
```

Q14. Explain Transaction Isolation Level in JDBC

Ans.

- JDBC isolation level represents that, how a database maintains its interiority against the problem such as
  1. dirty reads
  2. non-repeatable reads
  3. phantom reads

  that occurs during concurrent transactions.

What is Phantom read?

- At the time of execution of a transaction, if two queries that are identical are executed, and the rows returned are different from one another.
- If you execute a query at time T1 and re-execute it at time T2, additional rows may have been added to the database, which may affect your results. It is stated that a phantom read occurred.

What is Dirty read?

- Dirty read occurs when one transaction is changing the record, and the other transaction can read this record before the first transaction has been committed or rolled back.
- This is known as a dirty read scenario because there is always the possibility that the first transaction may rollback the change, resulting in the second transaction having read an invalid data.

E.g.

Transaction A begins	**Transaction B begins**
UPDATE EMPLOYEE SET SALARY = 10000 WHERE EMP_ID= '123';	**SELECT * FROM EMPLOYEE;** (Transaction B sees data which is updated by transaction A. But, those updates have not yet been committed.)

What is Non-Repeatable Read?

- Non Repeatable Reads happen when in a **same transaction** same query yields to a different result.
- This occurs when one transaction repeatedly retrieves the data, while a difference transactions alters the underlying data.
- This causes the different or non-repeatable results to be read by the first transaction.

Transaction Isolation Level

Initial Val.	**Isolation Level**	**Description**
1	TRANSACTION_READ_UNCOMMITTED	It allows non-repeatable reads, dirty reads and phantom reads to occur.
2	TRANSACTION_READ_COMMITTED	It ensures only those data can be read which is committed.
4	TRANSACTION_REPEATABLE_READ	It is closer to serializable, but phantom reads are also possible.
8	TRANSACTION_SERIALIZABLE	In this level of isolation dirty reads, non-repeatable reads, and phantom reads are prevented.

You can get/set the current isolation level by using method

1. getTransactionIsolation()
2. setTransactionIsolation(int isolationlevelconstant)

Example

```
con.setTransactionIsolation(8);
System.out.println("con.getTransactionIsolation():"
                              +con.getTransactionIsolation());
```

Q15. Explain Batch Processing in JDBC

Ans.

- Instead of executing a single query, we can execute a batch (group) of queries.
- It makes the performance fast.
- The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing.

Methods of Statement Interface

void addBatch(String query)	It adds query into batch.
int[] executeBatch()	It executes the batch of queries.

```
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
   "jdbc:mysql://localhost:3306/GTU","root","root");
con.setAutoCommit(false);
Statement stmt=con.createStatement();
String query1,query2,query3,query4,query5;
query1="create table DietStudent(enr INT PRIMARY
        KEY, name VARCHAR(20),sem INT,branch
                         VARCHAR(10))";
query2="insert into DietStudent
                 values(6001,'java',6,'ce')";
query3="insert into DietStudent
                 values(6002,'php',6,'ce')";
query4="update DietStudent set name='cg' where
                                 enr=6002";
query5="delete from DietStudent where
                           name='java'";
stmt.addBatch(query1);
stmt.addBatch(query2);
stmt.addBatch(query3);
stmt.addBatch(query4);
stmt.addBatch(query5);
int[] i=stmt.executeBatch();
con.commit();
```

GTU Questions

1.	What is JDBC?	[Win -14]
	List out different types of JDBC driver and explain role of each.	[Sum -15]
	Write code snippet for each type of JDBC connection.	[Win -15]
	Explain Thick and Thin driver.	[Sum -16]
	Comment on selection of driver.	[Win -16]
2.	Explain Prepared statements with suitable example	[Win -15]
		[Sum -16]
		[Win -16]
		[Win -17]
3.	Give the use of Statement, PreparedStatement and CallableStatement object. Write code to insert three records into student table using PreparedStatement (assume student table with Name, RollNo, and Branch field).	[Win -14]
4.	What is phantom read in JDBC? Which isolation level prevents it?	[Sum -16]
5.	Discuss CallableStatement with example.	[Win -17]