CO4

Develop Applications Using Database

Q1. Define Database

A database is a structured collection of data that is organised in a way that allows for efficient storage, retrieval, and management of data.

Q2. Define Database Management System (DBMS)

A DBMS (Database Management System) is a software system that is used to create, manage, and maintain databases.

Q3. List the applications of DBMS

1. Railway and Airline reservation system
2. Library management system
3. Banking
4. Education sector
5. Online shopping
6. Human Resource Management
7. Telecommunication Sector
8. Healthcare systems

Q4. Define Relational Database Management System (RDBMS)

RDBMS is a type of DBMS that is based on the relational data model. In an RDBMS, data is organised into tables that consist of rows and columns, where each row represents a unique record and each column represents a specific attribute or field of the record.

Q5. List examples of RDBMS

Oracle, MySQL, Microsoft SQL Server, PostgreSQL, and SQLite

Q6. Define Structured Query Language (SQL)

SQL is a standard language used for managing relational databases. It is used to create, modify, and delete tables and other database objects, as well as to insert, update, delete, and retrieve data from those tables.

Q7. List the features of SQL

1. Data Definition Language (DDL) - SQL provides a set of commands for creating and modifying database objects such as tables, views, indexes, and constraints.

2. Data Manipulation Language (DML) - SQL allows you to insert, update, delete, and retrieve data from a database.

3. Data Control Language (DCL) - SQL provides commands for managing user access to the database, including creating user accounts, granting and revoking privileges, and managing roles.

4. Transaction Control Language (TCL) - SQL supports transactions, which allow you to group multiple operations into a single logical unit that either succeeds or fails as a whole.

5. Query Language - SQL is a powerful query language that allows you to retrieve data from multiple tables, filter and sort data, and perform calculations and aggregations on the data.

6. Security - SQL supports authentication and authorization mechanisms to ensure that only authorised users have access to the data.

7. Portability - SQL is a standard language that is supported by most relational database management systems, allowing you to move your database and applications to different platforms and vendors.

8. Scalability - SQL databases can handle large volumes of data and support high concurrency, making them suitable for large-scale applications.

9. Performance - SQL databases use optimised algorithms for indexing, searching, and sorting data, providing fast and efficient data access.

Q8. List the basic terms in RDBMS

1. **Table :** A table is a collection of related data that is organised into rows and columns

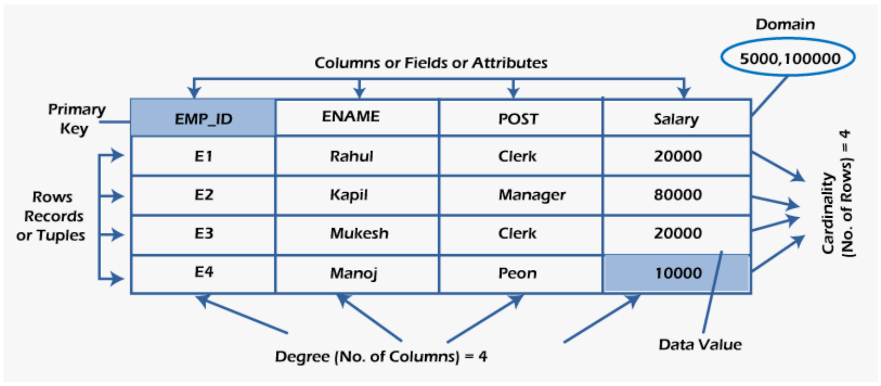
2. **Attribute (Field):** A column in a table represents a specific attribute of the data, such as a person’s name, address or age.

3. Record: A record is a row in a table which represents a particular object having specific values for fields.

4. Primary Key: A primary key is a column or combination of columns that uniquely identifies each row in a table. Eg: regno in student table

6. Foreign Key: A foreign key is a column or combination of columns in one table that refers to the primary key of another table.

7. Query: A query is a request for data from one or more tables in a database. It is used to retrieve and manipulate data based on specific criteria.



EMP_ID	ENAME	POST	Salary
E1	Rahul	Clerk	20000
E2	Kapil	Manager	80000
E3	Mukesh	Clerk	20000
E4	Manoj	Peon	10000

Domain
5000,100000
Columns or Fields or Attributes
Primary Key
Rows Records or Tuples
Cardinality (No. of Rows) = 4
Degree (No. of Columns) = 4
Data Value

Q9. List the basic types of SQL commands with examples

1. **Data Definition Language (DDL) Commands -** These commands are used to define the structure of a database. Examples are CREATE, ALTER, DROP, RENAME and TRUNCATE.
2. **Data Manipulation Language (DML) commands -** These commands are used to manipulate data stored in a database. Examples are SELECT, INSERT, UPDATE and DELETE
3. **Data Control Language (DCL) commands -** These commands are used to control access to the data stored in a database. Examples are GRANT and REVOKE
4. **Transaction Control Language (TCL) commands -** These commands are used to control the transactions that occur within a database. Examples are COMMIT, ROLLBACK and SAVEPOINT.

Q10. Explain DDL commands

1. **CREATE** - This command is used to create a table.

   Syntax:

```
CREATE TABLE table_name (
        column1 datatype,
        column2 datatype,
        column3 datatype,
  ....
);
```

   Example: CREATE TABLE EMPLOYEE (Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

2. **ALTER -** This command is used to modify the structure of an existing table. We can

   a) Add a new column to the table

Syntax: ALTER TABLE table_name ADD column_name datatype;

Example: ALTER TABLE Customers ADD Email varchar(255);

b) Delete a column

Syntax: ALTER TABLE table_name DROP COLUMN column_name;

Example: ALTER TABLE Customers DROP COLUMN Email;

c) Rename a column

Syntax: ALTER TABLE table_name RENAME COLUMN old_name to new_name;

Example: ALTER TABLE Customers RENAME COLUMN Email to Mail

d) Modify datatype of an attribute

Syntax: ALTER TABLE table_name MODIFY COLUMN column_name datatype;

Example: ALTER TABLE Customers MODIFY COLUMN Email varchar (50);

3. DROP - It is used to delete both the structure and record stored in the table.

Syntax: DROP TABLE table_name;

Example: DROP TABLE Employee;

4. TRUNCATE - It is used to delete all the rows from the table and free the space containing the table.

Syntax: TRUNCATE TABLE table_name

Example: TRUNCATE TABLE Employee;

Q11. List the datatypes in SQL

1. INTEGER/INT - for storing whole numbers (e.g. -2147483648 to 2147483647)
2. TINYINT - for storing very small integers (-128 to 127)
3. SMALLINT - for storing small integers (e.g. -32768 to 32767)
4. DECIMAL: for storing fixed-point numbers with a specified precision (e.g. -99.67)
5. CHAR: for storing fixed-length strings (e.g. up to 255 characters)
6. VARCHAR: for storing variable-length strings (e.g. up to 65,535 characters)
7. TEXT: for storing large amounts of text data (e.g. up to 65,535 bytes)
8. DATE: for storing dates (e.g. 'YYYY-MM-DD')
9. TIME: for storing times (e.g. 'HH:MM:SS')
10. DATETIME: for storing dates and times (e.g. 'YYYY-MM-DD HH:MM:SS')
11. BOOLEAN: for storing boolean values (e.g. TRUE or FALSE)

Q12. Explain DML Commands

1. **INSERT** - It is used to insert a row into a table

Syntax:

INSERT INTO TABLE_NAME (col1, col2, col3,.... col N) VALUES (value1, value2, value3, ... valueN);

Or

INSERT INTO TABLE_NAME VALUES (value1, value2, value3, ... valueN);

Example:

INSERT INTO STUDENT(rollno, name, dob) VALUES(1,'Ajith', '20/10/1992');

Or

INSERT INTO STUDENT VALUES(1,'Ajith', '20/10/1992');

2. **UPDATE:** This command is used to update or modify the value of a column in the table

Syntax:

UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]

Example:

UPDATE STUDENT SET name="Ajith Kumar" WHERE rollno=1;

3. DELETE: It is used to remove one or more rows from a table.

Syntax:

DELETE FROM table_name [WHERE condition];

Example:

DELETE FROM STUDENT WHERE rollno=1;

4.SELECT: This command is used to retrieve records from a table

Syntax:

SELECT column1, column2, ...

FROM table1

WHERE condition1

ORDER BY column1 ASC/DESC, column2 ASC/DESC, . . .

- SELECT specifies the columns to be retrieved from the table(s).
- FROM specifies the table(s) to retrieve data from.
- WHERE specifies the condition(s) that must be met for a row to be retrieved.
- ORDER BY sorts the results based on the specified column(s), in ascending (ASC) or descending (DESC) order.

Example:

SELECT emp_name FROM employee WHERE age > 20;

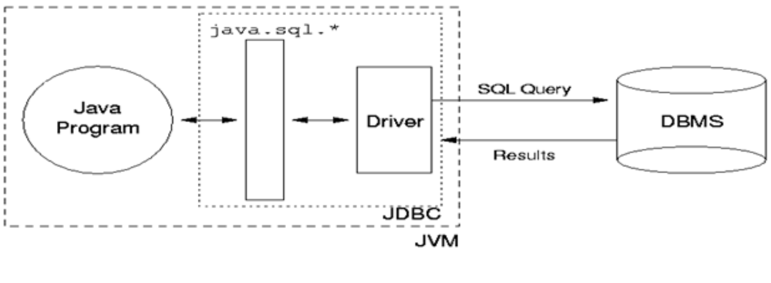
To retrieve all columns from the table we use '*'

SELECT * FROM employee WHERE age > 20;

Q13. What is JDBC and how does it work

Java Database Connectivity (JDBC) is a Java API that provides a standard way for Java programs to access relational databases. It allows Java programs to connect to a database, execute SQL statements, and retrieve the results.

. JDBC provides a set of classes and interfaces, including Connection, Statement, PreparedStatement, CallableStatement, ResultSet that allow developers to write data-independent code. Java programs can connect to any relational database that supports JDBC, without changing the code.



Q14. Name the Java package to be imported for JDBC connection establishment

java.sql package using the statement import java.sql.*;

Q15. Explain the steps for creating a JDBC connection

1. **Load the JDBC driver :** Before connecting to a database we need to load the appropriate JDBC driver for the database. We load the driver class by calling

Class.forName() with the driver class name as an argument. For connecting to MySQL database, we write

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

2. **Establish a connection:** The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. Its getConnection() method is used to establish a connection to a database. It uses a username, password, and a JDBC URL to establish a connection to the database and returns a connection object.

```
Connection  conn
= DriverManager.getConnection("jdbc:mysql://localhost/mydatabase",
                                             "username", "password");
```

3. **Create a statement:** After establishing a connection, you can create a Statement object using the **Connection.createStatement()** method. The Statement object is used to execute SQL statements.

Statement stmt = conn.createStatement();

4. **Execute SQL statements:**You can execute SQL statements using the **Statement.executeQuery()** method. The **executeQuery()** method takes a sql query string as an argument and returns the result as a **ResultSet** object.

ResultSet rs = stmt.executeQuery("SELECT * FROM mytable");

The **executeUpdate()** method executes the SQL **INSERT,DELETE,UPDATE** statements

5. **Process the results:** We can retrieve the data from the ResultSet using the **getXXX()** methods, where **XXX** is the data type of the column.

```
while (rs.next()) {
        int id = rs.getInt("id");
        String name = rs.getString("name");
        System.out.println("ID: " + id + ", Name: " + name);
}
```

6. **Close the connection:** After all database operations are finished, the connection can be closed using the **Connection.close()** method.

conn.close();

Q16. Illustrate the interfaces in JDBC

The JDBC API provides a set of interfaces and classes that can be used to perform database operations such as creating, updating, and retrieving data. The main interfaces in JDBC are:

1. **Driver:** This interface provides methods for connecting to a database. It is implemented by the database driver vendors and registered with the DriverManager class. The implementation of this interface is specific to each database vendor.
2. **Connection:** This interface represents a connection to a database. It provides methods for creating statements, managing transactions, and getting metadata about the database.
3. **Statement**: This interface represents an SQL statement that is executed against a database. It provides methods for executing SQL statements, retrieving results, and managing the statement's properties.
4. **PreparedStatement:** This interface extends the Statement interface and provides support for precompiled SQL statements. It allows the developer to create a parameterized SQL statement that can be executed multiple times with different parameters.
5. **CallableStatement:** This interface extends the PreparedStatement interface and provides support for calling stored procedures in a database.
6. **ResultSet:** This interface represents the results of a query executed against a database. It provides methods for navigating through the result set and retrieving data from the database.
7. **ResultSetMetaData:** This interface provides metadata about the columns in a ResultSet. It allows the developer to get information about the data types, column names, and other properties of the result set.

Q17. Write notes on PreparedStatement with an example

The **PreparedStatement** interface is derived from the Statement interface. **prepareStatement()** method is used to create **PreparedStatementObject.** PreparedStatement is used to execute queries with unknown parameters or where the parameters are provided at run-time. Java JDBC Prepared statements are pre-compiled SQL statements. Precompiled SQL is useful if the same SQL is to be executed repeatedly

Example:

PreparedStatement pstmt = con.prepareStatement("update Orders set pname = ? where Prod_Id = ?");

```
pstmt.setInt(2, 100);
pstmt.setString(1, "Bob");
pstmt.executeUpdate();
```

Q18. Write notes on ResultSet with an example

In JDBC (Java Database Connectivity), a **ResultSet** is an interface that represents the result of a database query. It provides methods to traverse and manipulate the data returned by the query. The ResultSet object is created when the executeQuery() method of the Statement object is called with an SQL query.

Example:

```
import java.sql.*;
public class Example {
 public static void main(String[] args) {
        try {
        // Create a connection to the database
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydatabase", "root", "password");
        // Create a statement object
        Statement stmt = conn.createStatement();
        // Execute a SQL query and get the result set
        ResultSet rs = stmt.executeQuery("SELECT * FROM mytable");
        // Loop through the result set and print the data
        while (rs.next()) {
        int id = rs.getInt("id");
        String name = rs.getString("name");
        int age = rs.getInt("age");
     System.out.println("ID: " + id + ", Name: " + name + ", Age: " + age);
        }
        // Close the resultset, statement, and connection
   rs.close();
```

```
    stmt.close();
    conn.close();
        } catch (SQLException e) {
    System.out.println("Error: " + e.getMessage()); }}}
```

Q18. Write a Java program to create a table ‘students’ and store id, name and age using JDBC

```
import java.sql.*;
public class StudentApplication {
   public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase"; // replace "mydatabase" with the name of your database
      String username = "root"; // replace with your MySQL username
      String password = "password"; // replace with your MySQL password
      try {
         // create a connection to the database
         Connection conn = DriverManager.getConnection(url, username, password);
         // create a statement object
         Statement stmt = conn.createStatement();
         // create a table to store student details
            String createTableSQL = "CREATE TABLE students (id INT PRIMARY KEY, name VARCHAR(50), age INT)";
         stmt.execute(createTableSQL);
         // insert some data into the table
            String insertDataSQL = "INSERT INTO students (id, name, age) VALUES (1, 'John Doe', 20), (2, 'Jane Smith', 21)";
         stmt.execute(insertDataSQL);
         // close the statement and connection
         stmt.close();
         conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }}
```

Q19. Write a Java program to retrieve employee details from a table using JDBC

```
import java.sql.*;
public class EmployeeDetails{
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase"; // replace "mydatabase" with the name of your database
        String username = "root"; // replace with your MySQL username
        String password = "password"; // replace with your MySQL password
        try {
            // create a connection to the database
            Connection conn = DriverManager.getConnection(url, username, password);
            // create a statement object
            Statement stmt = conn.createStatement();
            String sql;
            sql = "SELECT id, name, salary FROM employee";
            ResultSet rs = stmt.executeQuery(sql);
            while(rs.next()){
                    // Retrieve by column name
                     int id  = rs.getInt("id");
                    String name = rs.getString("name");
                    double salary = rs.getDouble("salary");
                    // Display values
                    System.out.println("ID: " + id);
                    System.out.println(", Name: " + name);
                    System.out.println(", Salary: " + salary);
     }
                  // close the statement and connection
                    stmt.close();
                    conn.close();
                    } catch (SQLException e) {
                              e.printStackTrace();
                     }
                     }}
```