**SR 22**

**SET - 1**

**PART – A (10x1M = 10M)**
**Note: Answer all Questions. Each Question carries equal marks.**

| Q. No | Question (s) | Marks |
|---|---|---|
| 1 | **a) What is Data Redundancy?**<br>**Data Redundancy** in DBMS refers to the unnecessary duplication of data within a database. It occurs when the same piece of data is stored in multiple places, leading to increased storage usage and potential inconsistencies. | **1M** |
| | **b) What are the different levels of abstraction in DBMS?**<br><br>The three levels of abstraction in DBMS are:<br>1. **Physical Level**: Describes how data is actually stored.<br>2. **Logical Level**: Describes what data is stored in the database and the relationships among those data.<br>3. **View Level**: Describes only part of the entire database, which is a tailored representation for users or applications. | **1M** |
| | **c) Define Query Processor?**<br><br>A **Query Processor** in DBMS is a component responsible for interpreting and executing database queries, optimizing query performance, and converting high-level queries into low-level operations. | **1M** |
| | **d) What is Database Design?**<br>**Database Design** is the process of specifying the structure, storage, and retrieval of data within a database to ensure data integrity, efficiency, and accessibility. | **1M** |
| | **e) Define Foreign Key.**<br>A **Foreign Key** is a column or a set of columns in a database table that creates a link between two tables by referencing the primary key in another table, ensuring referential integrity. | **1M** |
| | **f) What is RDBMS?**<br><br>A **Relational Database Management System (RDBMS)** is software that manages databases based on a relational model, where data is stored in tables with rows and columns, enabling easy retrieval, updating, and management of data through SQL queries. | **1M** |
| | **g) What is SQL Aliases?**<br>**SQL Aliases** are temporary names given to tables or columns in a SQL query to make them easier to read and write. They are created using the `AS` keyword.<br>For Column:<br>`SELECT first_name AS fname, last_name AS lname`<br>`FROM employee`<br><br>`For Table:`<br>`SELECT e.first_name, e.last_name, d.department_name`<br>`FROM employees AS e`<br>`JOIN departments AS d ON`<br>`e.department_id = d.department_id;` | **1M** |

| | | |
|---|---|---|
| | **h) What are the types of Views?**<br><br>**Types of Views in SQL**:<br>1. **Simple View**: Created from a single table without any functions or groups.<br>2. **Complex View**: Created from multiple tables or includes functions, groups, and joins.<br>**Ex**:<br>CREATE VIEW SimpleView AS<br>SELECT first_name, last_name<br>FROM employees;<br>**<br>CREATE VIEW ComplexView AS<br>SELECT e.first_name, e.last_name, d.department_name<br>FROM employees e<br>JOIN departments d ON e.department_id = d.department_id; | **1M** |
| | **i) What is Null Values?**<br>**Null Values** in a database represent missing or unknown data. A null value is different from a zero or an empty string, as it signifies the absence of any value.<br><br>INSERT INTO students (name, age, email)<br>VALUES ('John Doe', 20, NULL); | **1M** |
| | **j) What are the types of SQL Server Triggers?**<br><br>**Types of SQL Server Triggers**:<br>1. **DML Triggers**: Triggered by Data Manipulation Language (INSERT, UPDATE, DELETE) events.<br>2. **DDL Triggers**: Triggered by Data Definition Language (CREATE, ALTER, DROP) events.<br>3. **Logon Triggers**: Triggered by LOGON events when a user session is established.<br>These triggers help in automating tasks and enforcing business rules within SQL Server.<br><br>a **trigger** is a database object that automatically executes a specified set of actions (such as SQL statements) in response to certain events (like INSERT, UPDATE, or DELETE) occurring on a table or view.<br><br>CREATE TRIGGER UpdateEmployeeSalary<br>AFTER UPDATE ON employees<br>FOR EACH ROW<br>BEGIN<br>IF OLD.salary <> NEW.salary THEN<br>INSERT INTO EmployeeLog (emp_id, old_salary, new_salary)<br>VALUES (OLD.emp_id, OLD.salary, NEW.salary);<br>END IF;<br>END; | **1M** |

**PART – B (20M)**

<table>
<tr><th>Q. No</th><th>Question (s)</th><th>Marks</th></tr>
<tr><td>2</td><td>a) Differentiate between database Management system and File systems?
<table>
<tr><th>Feature</th><th>Database Management System (DBMS)</th><th>File Systems</th></tr>
<tr><td>Data Organization</td><td>Organized in tables, schemas, and relationships for structured storage and easy retrieval.</td><td>Data stored in files and folders, often leading to unstructured storage.</td></tr>
<tr><td>Data Integrity</td><td>Enforces data integrity and consistency through constraints, triggers, and transactions.</td><td>Lacks built-in mechanisms to ensure data integrity, making consistency harder to maintain.</td></tr>
<tr><td>Data Security</td><td>Provides robust security features like access control, authentication, and encryption.</td><td>Basic security features, limited to file permissions and user access controls.</td></tr>
<tr><td>Data Redundancy and Consistency</td><td>Minimizes data redundancy and ensures consistency through normalization and centralized management.</td><td>Higher chances of data redundancy and inconsistency due to lack of centralized control and data duplication.</td></tr>
</table></td><td>4M</td></tr>
<tr><td></td><td>b) Briefly explain about different types of Data Model?<br><br>Different Types of Data Models in DBMS<br><br>1. Hierarchical Data Model:<br>o Structure: Organizes data in a tree-like structure using parent-child relationships.<br>o Usage: Ideal for applications with hierarchical data, such as organizational structures.<br>2. Network Data Model:<br>o Structure: Uses a graph-like structure with nodes representing entities and edges representing relationships.<br>o Usage: Suitable for complex relationships and many-to-many associations, like telecommunication networks.<br>3. Relational Data Model:<br>o Structure: Represents data in tables (relations) with rows (tuples) and columns (attributes).<br>o Usage: Most widely used model, ideal for a wide range of applications and supports SQL for data manipulation.<br>4. Entity-Relationship (ER) Model:<br>o Structure: Uses entities and relationships to represent data and its associations.<br>o Usage: Commonly used for database design, providing a conceptual framework for data modeling.<br>5. Object-Oriented Data Model:<br>o Structure: Combines object-oriented programming principles with database concepts.<br>o Usage: Suitable for applications with complex data and relationships, such as multimedia databases.<br>6. Document Data Model:</td><td>4M</td></tr>
</table>

| | | |
|---|---|---|
| | - **Structure**: Organizes data in document-like structures, typically JSON or XML formats.<br>- **Usage**: Commonly used in NoSQL databases, ideal for applications with flexible and hierarchical data.<br>These data models provide different ways to represent, organize, and manage data in a database system, catering to various application needs. | |
| **3** | **a) Explain about different types of attributes in Entity Relationship Model.**<br><br>**Types of Attributes in Entity-Relationship (ER) Model in DBMS**<br><br>1. **Simple Attribute**:<br>- **Definition**: Cannot be divided into smaller parts.<br>- **Example**: Age, Salary.<br>2. **Composite Attribute**:<br>- **Definition**: Can be divided into smaller subparts, each representing a more basic attribute with independent meaning.<br>- **Example**: Name (composed of First Name, Middle Name, Last Name).<br>3. **Single-Valued Attribute**:<br>- **Definition**: Holds a single value for a particular entity.<br>- **Example**: Social Security Number (SSN), Employee ID.<br>4. **Multi-Valued Attribute**:<br>- **Definition**: Can hold multiple values for a particular entity.<br>- **Example**: Phone numbers, Email addresses.<br>5. **Derived Attribute**:<br>- **Definition**: Derived from other attributes and does not physically exist in the database.<br>- **Example**: Age (derived from Date of Birth).<br>6. **Key Attribute**:<br>- **Definition**: Uniquely identifies an entity in an entity set.<br>- **Example**: Student ID, Passport Number.<br>These attributes help in defining and distinguishing entities within the ER model, allowing for a clearer and more structured database design. | **4M** |
| | **b) Explain in brief about Data Independence?**<br><br>**Data Independence in DBMS**<br><br>**Data Independence** is the capacity to modify the schema at one level of a database system without affecting the schema at the next higher level. It ensures that changes in the data storage or structure do not impact the application programs that interact with the data.<br>1. **Types of Data Independence**:<br>- **Physical Data Independence**: Refers to the ability to change the physical storage structure or devices without affecting the logical schema. This means changes in data storage devices, file structures, or indices can be made without modifying the logical structure of the database. | **4M** |

| | | |
|---|---|---|
| | - **Logical Data Independence**: Refers to the ability to change the logical schema without altering the external schema or application programs. This means changes in the data model, such as adding new fields or entities, do not require changes to the applications that use the data.<br>2. **Importance**:<br>- **Flexibility**: Allows for easy adaptation to changes in data requirements or technology without affecting the entire system.<br>- **Maintenance**: Simplifies database maintenance by decoupling the data structure from the application logic, reducing the risk of errors during updates.<br>- **Efficiency**: Enhances system performance by allowing for optimizations in storage and retrieval without impacting user interactions.<br>Data independence is a crucial aspect of DBMS, providing a more robust, adaptable, and manageable database environment. | |
| **4** | **a) Discuss about Enforcing Integrity Constraints?**<br><br>**Enforcing Integrity Constraints in DBMS**<br><br>**Integrity Constraints** are rules that ensure the accuracy and consistency of data within a database. They are crucial for maintaining the integrity and reliability of the database. Enforcing these constraints prevents invalid data from being entered into the database and ensures that the data adheres to the defined rules.<br>1. **Types of Integrity Constraints**:<br>- **Domain Constraints**: Ensure that the data values in a column must be from a predefined domain. For example, an age column should only have values within a certain range (e.g., 1 to 120).<br>- **Entity Integrity**: Ensures that each table has a primary key, and that primary key values are unique and not null, ensuring the uniqueness of each record.<br>- **Referential Integrity**: Ensures that a foreign key value always refers to an existing record in another table, maintaining consistent and valid relationships between tables.<br>- **Unique Constraints**: Ensure that the values in a column or a set of columns are unique across all rows in the table.<br>- **Check Constraints**: Enforce specific conditions that must be met by the data in a column. For example, a check constraint can ensure that a salary column cannot have negative values.<br>2. **Importance of Enforcing Integrity Constraints**:<br>- **Data Accuracy**: Ensures that the data entered into the database is accurate and valid.<br>- **Consistency**: Maintains consistency across the database by enforcing rules and relationships between tables.<br>- **Data Integrity**: Protects against data anomalies and ensures that the data adheres to business rules. | **4M** |

<table>
<tr><td></td><td>
<ul>
<li>Reliability: Enhances the reliability of the database by preventing the insertion of invalid or inconsistent data.</li>
</ul>
3. Implementation:
<ul>
<li>Database Schema Definition: Constraints are defined during the creation or alteration of database schemas using SQL statements.</li>
<li>Triggers and Stored Procedures: Custom business rules can be enforced using triggers and stored procedures to check for constraints before data modification.</li>
</ul>
Enforcing integrity constraints is essential for maintaining the quality and trustworthiness of the data in a database system, ensuring that it serves as a reliable source of information for decision-making.
</td><td></td></tr>
<tr><td></td><td>
b) What are the advantages & disadvantages of Relational Database Model?
<table>
<tr><th>Feature</th><th>Advantages</th><th>Disadvantages</th></tr>
<tr><td>Simplicity</td><td>Easy to understand and use due to the tabular structure.</td><td>Handling complex relationships can be challenging.</td></tr>
<tr><td>Data Integrity and Accuracy</td><td>Enforces data integrity constraints, ensuring data accuracy and consistency.</td><td>Performance can degrade with very large datasets or complex queries.</td></tr>
<tr><td>Flexibility</td><td>Supports a wide range of operations, providing flexibility in data management.</td><td>Scalability issues in handling high volumes of data and transactions.</td></tr>
<tr><td>Security</td><td>Implements robust security measures, including user authentication, access control, and encryption.</td><td>Resource-intensive, requiring significant computational resources and memory.</td></tr>
</table>
</td><td>4M</td></tr>
<tr><td>5</td><td>
a) Briefly explain in detail about relational Algebra.
<h2>Relational Algebra in DBMS</h2>
Relational Algebra is a procedural query language that operates on relations (tables) in a database. It provides a set of operations that take one or two relations as input and produce a new relation as output, allowing for the manipulation and retrieval of data.
1. Basic Operations:
<ul>
<li>Selection (σ): Filters rows based on a specified condition.
<ul>
<li>Example: σ_age > 30 (Employees) retrieves all employees older than 30.</li>
</ul></li>
<li>Projection (π): Selects specific columns from a relation, removing duplicates.
<ul>
<li>Example: π_name, age (Employees) retrieves the name and age columns from the Employees table.</li>
</ul></li>
<li>Union (U): Combines two relations, including all distinct tuples from both.
<ul>
<li>Example: Employees_U_Managers combines employees and managers into a single relation.</li>
</ul></li>
<li>Set Difference (−): Returns tuples in one relation but not in the other.
<ul>
<li>Example: Employees − Managers returns employees</li>
</ul></li>
</ul>
</td><td>4M</td></tr>
</table>

| | | |
|---|---|---|
| | who are not managers.<br>o **Cartesian Product (×)**: Combines tuples from two relations in all possible ways.<br>▪ **Example**: Employees × Departments pairs each employee with every department.<br>o **Rename (ρ)**: Renames the relation or its attributes.<br>▪ **Example**: ρ(EmpName → name, EmpAge → age) (Employees) renames the columns in the Employees relation.<br>2. **Advanced Operations**:<br>o **Join (⋈)**: Combines tuples from two relations based on a related attribute.<br>▪ **Example**: Employees ⋈ Departments combines employees with their respective departments.<br>o **Intersection (∩)**: Returns common tuples from both relations.<br>▪ **Example**: Employees ∩ Managers returns common tuples in both employees and managers.<br>o **Division (÷)**: Divides one relation by another, finding tuples that match all values in the divisor.<br>▪ **Example**: Projects ÷ Skills returns projects requiring all skills listed.<br>3. **Importance**:<br>o **Foundation**: Provides a theoretical foundation for relational database operations.<br>o **Query Optimization**: Helps optimize and translate high-level queries (SQL) into efficient execution plans.<br>o **Data Manipulation**: Enables complex queries and data retrieval operations in a systematic manner.<br>Relational Algebra is fundamental to understanding how database queries are structured and executed, offering a clear and concise way to describe data manipulation and retrieval tasks in a relational database system. | |
| | **b) Explain any FIVE commands in SQL?**<br><br>five essential SQL commands in DBMS explained in text format:<br>1. **SELECT**: Retrieves data from one or more tables. Example: `SELECT name, age FROM students;`<br>2. **INSERT**: Adds new rows of data to a table. Example: `INSERT INTO students (name, age) VALUES ('John Doe', 20);`<br>3. **UPDATE**: Modifies existing data in a table. Example: `UPDATE students SET age = 21 WHERE name = 'John Doe';`<br>4. **DELETE**: Removes rows from a table. Example: `DELETE FROM students WHERE name = 'John Doe';`<br>5. **CREATE TABLE**: Creates a new table in the database. Example:<br>CREATE TABLE students (<br>id INT PRIMARY KEY,<br>name VARCHAR(50),<br>age INT<br>); | **4M** |

| | | |
|---|---|---|
| 6 | **Explain the concept of Active Databases.**<br><br>**Active Databases in DBMS**<br><br>**Active Databases** are databases that can automatically respond to specific conditions or events through the use of triggers and rules. This event-driven architecture enhances the database's functionality by allowing it to execute predefined actions when certain criteria are met.<br>1. **Triggers**:<br>o **Definition**: Procedural code that automatically executes in response to certain events (such as inserts, updates, or deletions) on a particular table or view.<br>o **Example**: A trigger could update an inventory count whenever a new order is placed.<br>2. **Rules**:<br>o **Definition**: More complex conditions and actions that can define business logic within the database.<br>o **Example**: A rule might ensure that a new employee record can only be added if the corresponding department exists in the department table.<br>3. **Events**:<br>o **Definition**: Specific actions or changes that occur within the database, like data modifications.<br>o **Example**: Inserting a new row, updating a record, or deleting a row can trigger predefined actions.<br>4. **Applications**:<br>o **Automation**: Reduces the need for manual intervention by automating repetitive tasks and enforcing business rules consistently.<br>o **Consistency**: Maintains data consistency and integrity by ensuring that defined actions occur automatically in response to specified events.<br>o **Efficiency**: Enhances operational efficiency by automatically handling conditions and actions that would otherwise require custom code in application programs.<br>Active databases are especially useful in scenarios where timely responses to data changes are critical, such as in real-time monitoring systems, automated transaction systems, and complex business applications. This built-in responsiveness makes them powerful tools for managing dynamic and reactive data environments. | **4M** |
| 7 | **Briefly explain about types of SQL commands.**<br><br>**Types of SQL Commands in DBMS**<br><br>1. **Data Definition Language (DDL)**:<br>o **Purpose**: Defines and manages the structure of the database.<br>o **Key Commands**:<br>▪ **CREATE**: Creates new database objects such as ta- | **4M** |

bles, indexes, and views.
        - **ALTER**: Modifies existing database objects.
        - **DROP**: Deletes existing database objects.
2. **Data Manipulation Language (DML)**:
    - **Purpose**: Manages and manipulates data within the database.
    - **Key Commands**:
        - **INSERT**: Adds new rows of data to a table.
        - **UPDATE**: Modifies existing rows of data.
        - **DELETE**: Removes rows of data from a table.
3. **Data Query Language (DQL)**:
    - **Purpose**: Retrieves data from the database.
    - **Key Command**:
        - **SELECT**: Fetches data from one or more tables.
4. **Data Control Language (DCL)**:
    - **Purpose**: Controls access to the data within the database.
    - **Key Commands**:
        - **GRANT**: Provides user access privileges to database objects.
        - **REVOKE**: Removes user access privileges.
5. **Transaction Control Language (TCL)**:
    - **Purpose**: Manages transactions to ensure data integrity and consistency.
    - **Key Commands**:
        - **COMMIT**: Saves all changes made during the current transaction.
        - **ROLLBACK**: Reverts the database to the state it was in before the current transaction.

These SQL commands are essential for defining, manipulating, querying, controlling, and ensuring the integrity of data within a relational database management system (DBMS).

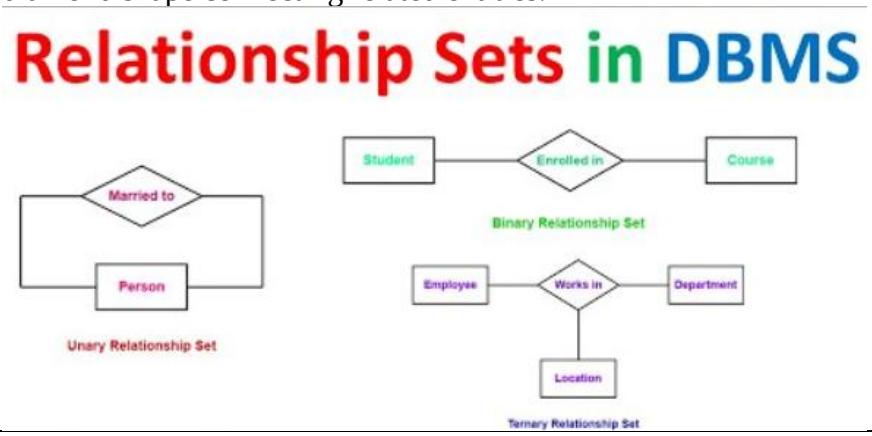
*****

**SR 22**

**SET-2**

**PART – A (10x1M = 10M)**

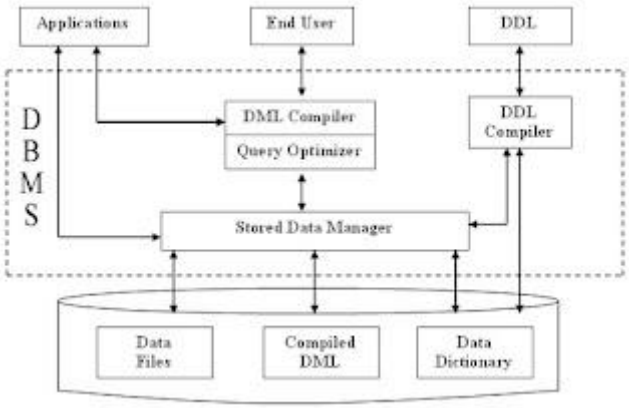
**Note: Answer all Questions. Each Question carries equal marks.**

| Q. No | Question (s) | Marks |
|---|---|---|
| 1 | **a) What are the types of Data Independence?**<br><br>The two types of Data Independence in DBMS are:<br>1. **Logical Data Independence**: The ability to change the logical schema without affecting the external schema or application programs.<br>2. **Physical Data Independence**: The ability to change the physical schema without affecting the logical schema or application programs | **1M** |
| | **b) Explain Requirements Analysis.**<br>**Requirements Analysis** in DBMS is the process of identifying and documenting the data and functional needs of the users and the system to design an effective database. | **1M** |
| | **c) What are the components of an ER diagram?**<br><br>The components of an ER diagram are:<br>1. **Entities**: Represented by rectangles.<br>2. **Attributes**: Represented by ovals.<br>3. **Relationships**: Represented by diamonds.<br>4. **Keys**: Represented by underlined attributes. | **1M** |
| | **d) What is Relationship Set?**<br>A **Relationship Set** in DBMS is a collection of similar relationships between entities, represented in an ER diagram. It is depicted by a diamond shape connecting related entities.<br>  | **1M** |
| | **e) What are the fundamental operations of Relational Algebra?**<br><br>The fundamental operations of Relational Algebra in DBMS are:<br>1. **Selection (σ)**: Retrieves rows that satisfy a given predicate.<br>2. **Projection (π)**: Retrieves specific columns from a relation.<br>3. **Union (∪)**: Combines tuples from two relations.<br>4. **Set Difference (−)**: Finds tuples in one relation but not in another.<br>5. **Cartesian Product (×)**: Combines tuples from two relations in a | **1M** |

| | | |
|---|---|---|
| | pairwise manner.<br>6. **Renaming (ρ)**: Changes the name of a relation or its attributes.<br>7. **Intersection (∩)**: Retrieves tuples that are common to both relations (derived from union and set difference).<br>8. **Join (⋈)**: Combines related tuples from two relations based on a condition.<br>These operations provide a foundation for querying and manipulating data in relational databases. | |
| | **f) What is Data Integrity?**<br>**Data Integrity** in DBMS refers to the accuracy, consistency, and reliability of data throughout its lifecycle. It ensures that data remains unaltered and accurate during operations like storage, retrieval, and transfer. | **1M** |
| | **g) What is BCNF?**<br>**BCNF (Boyce-Codd Normal Form)** is a type of database normalization form that ensures there are no non-trivial functional dependencies of attributes on anything other than a superkey. | **1M** |
| | **h) What are the different forms of Relational Calculus?**<br><br>The different forms of Relational Calculus in DBMS are:<br>1. **Tuple Relational Calculus (TRC)**: Specifies what data to retrieve by describing the desired properties of the tuples.<br>2. **Domain Relational Calculus (DRC)**: Uses domain variables that take on values from an attribute's domain, specifying what data to retrieve by describing the desired properties of the attribute values.<br>These forms provide a declarative way to query data in relational databases. | **1M** |
| | **i) What is Active Databases?**<br>**Active Databases** in DBMS are databases that automatically respond to events and conditions using triggers and rules. They enhance functionality by executing specified actions when certain criteria are met, providing an event-driven architecture. | **1M** |
| | **j) What are the types of SQL Aggregation Function?**<br><br>The types of SQL Aggregation Functions in DBMS are:<br>1. **COUNT()**: Counts the number of rows.<br>2. **SUM()**: Calculates the total sum of a numeric column.<br>3. **AVG()**: Computes the average value of a numeric column.<br>4. **MIN()**: Finds the minimum value.<br>5. **MAX()**: Finds the maximum value.<br>These functions perform calculations on multiple rows of a table to give a single result | **1M** |

**PART – B (20M)**

| Q. No | Question (s) | Marks |
|---|---|---|
| 2 | **a) Briefly explain about Database Design in DBMS.**<br><br>Database Design in a Database Management System (DBMS) is the process of creating a detailed data model of a database. Here's a brief overview:<br>**1. Requirements Analysis:**<br>• Understanding what data needs to be stored and how it will be used.<br>**2. Conceptual Design:**<br>• Creating an Entity-Relationship Diagram (ERD) to outline the entities, attributes, and relationships.<br>**3. Logical Design:**<br>• Translating the ERD into a logical data model (like tables and relationships in an RDBMS).<br>**4. Physical Design:**<br>• Determining how the database will be stored on physical media, considering factors like indexes and partitions.<br>**5. Normalization:**<br>• Ensuring the database design is free from redundancy and update anomalies by organizing data into logical groups.<br>**6. Optimization:**<br>• Enhancing the performance of the database for faster query responses and better resource management. | **4M** |
| | **b) Explain about different types of relationship in Entity Relationship Model.**<br><br>Entity-Relationship Model (ERM) in DBMS, relationships define how entities interact with each other. Here are the different types of relationships:<br><br>**1. One-to-One (1:1)**<br><br>• **Description:** Each entity in the relationship will have exactly one related entity.<br>• **Example:** A person has one passport, and each passport is issued to only one person.<br><br>**2. One-to-Many (1:M)**<br><br>• **Description:** One entity can have multiple related entities, but each of those related entities can only be associated with one primary entity.<br>• **Example:** A single teacher can teach many students, but each student is assigned to one teacher.<br><br>**3. Many-to-One (M:1)** | **4M** |

| | | |
|---|---|---|
| | • **Description:** Multiple entities can be associated with a single primary entity.<br>• **Example:** Many employees work in one department, but each department can have many employees.<br><br>**4. Many-to-Many (M:N)**<br><br>• **Description:** Multiple entities from one set can be associated with multiple entities from another set.<br>• **Example:** Students enroll in many courses, and each course can have many students.<br><br>**5. Self-Referencing Relationship**<br><br>• **Description:** An entity has a relationship with itself.<br>• **Example:** An employee can be a manager of another employee, indicating a hierarchical structure within the same entity set. | |
| **3** | **a) Explain about three major components in a structure of DBMS with a neat sketch.**<br>  <br><br><br>A Database Management System (DBMS) typically consists of three major components:<br><br>**1. Database Engine**<br><br>• **Description:** The core service for storing, processing, and securing data. It provides controlled access and rapid transaction processing.<br>• **Functions:** Data storage, query processing, and transaction management.<br><br>**2. Database Schema**<br><br>• **Description:** The blueprint of how the database is structured. It defines the tables, fields, relationships, views, and indexes. | **4M** |

| | | |
|---|---|---|
| | • **Functions:** Data organization, integrity constraints, and schema definition.<br><br>### 3. Database Management Tools<br><br>• **Description:** Software tools that enable administrators and developers to interact with the database.<br>• **Functions:** Database design, administration, performance monitoring, and user management. | |
| | **b) Discuss about levels of abstraction in DBMS.**<br><br>The levels of abstraction in DBMS help to simplify database design by separating the database into three levels. Here's a concise overview for your exam:<br><br>### 1. Physical Level<br><br>• **Description:** The lowest level of abstraction, detailing how the data is actually stored in the database.<br>• **Focus:** Storage details, like file structures and access methods.<br><br>### 2. Logical Level<br><br>• **Description:** The middle level that describes what data is stored and the relationships among those data.<br>• **Focus:** Database schema, including tables, relationships, and constraints.<br><br>### 3. View Level<br><br>• **Description:** The highest level that describes only part of the entire database. This level involves user interfaces and how users interact with data.<br>• **Focus:** User-specific views, which are subsets of the logical level.<br><br>**Summary:**<br><br>• **Physical Level:** How data is stored.<br>• **Logical Level:** What data is stored.<br>• **View Level:** How users see and interact with data.<br>This abstraction provides a clear separation, ensuring that changes in one level do not affect others, thus enhancing database management and usability. | **4M** |
| **4** | **a) Discuss about Integrity Constraint Over Relations.**<br><br>Integrity constraints in DBMS ensure the accuracy and consistency of data | **4M** |

within a relational database. Here are the key types of integrity constraints:

### 1. Entity Integrity

- **Description:** Ensures that each table has a primary key and that the columns defined as the primary key are unique and not null.
- **Example:** In a table `Students`, the `StudentID` (primary key) must be unique for each student and cannot be null.

### 2. Referential Integrity

- **Description:** Ensures that a foreign key value always points to an existing, valid record in another table.
- **Example:** In a table `Enrollments`, the `StudentID` (foreign key) must match an existing `StudentID` in the `Students` table.

### 3. Domain Integrity

- **Description:** Ensures that all columns in a database contain valid data and that data type, format, and range constraints are adhered to.
- **Example:** A column `Age` in the `Students` table should only contain integer values within a specified range, e.g., 18 to 25.

### 4. User-Defined Integrity

- **Description:** Custom constraints specified by users to enforce specific business rules.
- **Example:** In a table `Employees`, a constraint can be set to ensure that `Salary` values are within a certain range or that `Department` names follow a specific format.

**b) Explain in brief about the Views in SQL.**

**4M**

**Views** are virtual tables that provide a way to present data from one or more tables. They do not store data themselves but derive their data from the underlying tables. Here's a brief overview:

### Purpose of Views

1. **Data Abstraction**: Simplify complex queries by presenting data in a specific format.
2. **Security**: Restrict access to specific data, showing only the necessary information.
3. **Convenience**: Provide a reusable way to represent data for various queries and reports.

### Types of Views

| | | |
|---|---|---|
| | 1. **Simple View**: Based on a single table, without any aggregation or complex calculations.<br>2. **Complex View**: Derived from multiple tables and can include functions, joins, and groupings.<br><br>**Creating a View**<br>Ex:<br>CREATE VIEW EmployeeView AS<br>SELECT EmployeeID, EmployeeName, Department<br>FROM Employees<br>WHERE Department = 'HR';<br><br>Views enhance the flexibility and security of a database, making them a powerful tool in SQL and DBMS. | |
| **5** | **a) Briefly explain the concept of Logical Database.**<br><br>**Logical Database** is a representation of how data is logically organized and interrelated without considering how it is physically stored. Here are the key points:<br><br>**Concepts of Logical Database**<br><br>1. **Logical Schema:**<br>o **Description:** It defines the logical structure of the database, including tables, views, indexes, and relationships.<br>o **Focus:** The arrangement and organization of data elements and how they interact with each other.<br>2. **Data Abstraction:**<br>o **Description:** Logical databases provide an abstract view of the data, hiding the complexities of physical storage.<br>o **Focus:** Simplifies database interaction by focusing on data models and user requirements rather than storage details.<br>3. **Data Independence:**<br>o **Description:** It ensures that changes to the physical storage do not affect the logical structure and vice versa.<br>o **Focus:** Enhances flexibility and maintenance by separating logical data representation from physical storage.<br><br>**Example:**<br><br>Imagine a logical database schema that includes:<br><br>• **Tables:** Employees, Departments, Projects<br>• **Relationships:** Employee works in a Department, Department manages Projects<br><br>This logical representation allows users to interact with the data meaning- | **4M** |

| | | |
|---|---|---|
| | fully without worrying about where and how the data is stored on disk.<br><br>Logical databases are crucial for designing, managing, and interacting with databases efficiently, ensuring that user requirements are met without delving into storage complexities. | |
| | **b) Discuss about Enforcing Integrity Constraints?**<br><br>Enforcing integrity constraints in a Database Management System (DBMS) ensures that the data remains accurate and consistent. Here are the key types of integrity constraints and how they are enforced:<br><br>**1. Entity Integrity**<br><br>• **Description:** Ensures that each table has a primary key and that the columns defined as the primary key are unique and not null.<br>• **Enforcement:**<br>o **Primary Key Constraint:** Defined at the table creation stage or by altering the table.<br><br>**2. Referential Integrity**<br><br>• **Description:** Ensures that a foreign key value always points to an existing, valid record in another table.<br>• **Enforcement:**<br>o **Foreign Key Constraint:** Defined during table creation or by altering the table.<br><br>**3. Domain Integrity**<br><br>• **Description:** Ensures that all columns in a database contain valid data and that data type, format, and range constraints are adhered to.<br>• **Enforcement:**<br>o **Check Constraint:** Ensures that data entered into a column meets a specific condition.<br><br>CREATE TABLE Employees (<br>EmployeeID INT,<br>Age INT,<br>CHECK (Age >= 18 AND Age <= 65)<br>); | **4M** |

| | | |
|---|---|---|
| | **4. User-Defined Integrity**<br><br>• **Description:** Custom constraints specified by users to enforce specific business rules.<br>• **Enforcement:**<br>  ○ **Custom Triggers and Procedures:** Enforced through the creation of database triggers or procedures.<br><br>CREATE TRIGGER CheckSalary<br><br>BEFORE INSERT ON Employees<br><br>FOR EACH ROW<br><br>BEGIN<br><br>IF NEW.Salary < 0 THEN<br><br>SIGNAL SQLSTATE '45000'<br><br>SET MESSAGE_TEXT = 'Salary cannot be negative';<br><br>END IF;<br><br>END;<br><br>**Why Enforcing Integrity Constraints Matters**<br><br>• **Data Accuracy:** Maintains the correctness and validity of data.<br>• **Data Consistency:** Ensures uniformity of data across the database.<br>• **Data Reliability:** Prevents insertion of invalid or corrupt data.<br>• **Data Integrity:** Keeps relationships between tables valid and consistent.<br><br>By rigorously enforcing these integrity constraints, databases maintain high-quality data, essential for reliable and efficient database operations. | |
| **6** | **What is Trigger? Explain how to Implement Triggers in SQL?**<br><br>A **Trigger** in SQL is a special type of stored procedure that automatically executes or fires when certain events occur in the database. These events can include `INSERT`, `UPDATE`, or `DELETE` operations on a table.<br><br>**Types of Triggers**<br><br>1. **Before Trigger**: Executes before an event.<br>2. **After Trigger**: Executes after an event. | **4M** |

| | | |
|---|---|---|
| | 3. **Instead of Trigger**: Used with views to perform operations instead of the triggering event.<br><br>**Implementing Triggers in SQL**<br><br>Here's a step-by-step guide to create a simple trigger:<br><br>**Step 1: Define the Trigger**<br><br>You start by defining what the trigger will do and when it will execute.<br><br>**Step 2: Create the Trigger**<br><br>You use the `CREATE TRIGGER` statement to create the trigger.<br><br>**Example:** Suppose you have a table `Employees` and you want to create a trigger that logs any `INSERT` operation into a `Logs` table.<br><br>**Employees Table:**<br><br>CREATE TABLE Employees (<br>EmployeeID INT PRIMARY KEY,<br>EmployeeName VARCHAR(100),<br>Salary DECIMAL(10, 2)<br>);<br>CREATE TABLE Logs (<br>LogID INT PRIMARY KEY AUTO_INCREMENT,<br>LogMessage VARCHAR(255),<br>LogTime DATETIME<br>);<br>CREATE TRIGGER BeforeInsertEmployee<br>BEFORE INSERT ON Employees<br>FOR EACH ROW<br>BEGIN<br>INSERT INTO Logs (LogMessage, LogTime)<br>VALUES (CONCAT('New employee added: ', NEW.EmployeeName), NOW());<br>END;\<br><br>**Using the Trigger**<br><br>Now, whenever a new row is inserted into the `Employees` table, the trigger automatically inserts a log entry into the `Logs` table. | |
| **7** | **Explain in detail about Nested Queries in SQL.**<br><br>**Nested Queries in SQL**<br><br>Nested queries, also known as subqueries, are queries within another query. | **4M** |

They provide a powerful way to perform complex operations in SQL. Here's a detailed explanation:

## Types of Nested Queries

1. **Single-Row Subqueries**: Return only one row.
2. **Multiple-Row Subqueries**: Return multiple rows.
3. **Multiple-Column Subqueries**: Return multiple columns.

## Usage of Nested Queries

- **In `SELECT` Clause**: Used to fetch data based on another query.
- **In `WHERE` Clause**: Commonly used for filtering based on another query's results.
- **In `FROM` Clause**: Acts as a temporary table.

## Examples

**1. Single-Row Subquery**

Find employees whose salary is higher than the average salary.

```
SELECT EmployeeName
FROM Employees
WHERE Salary > (SELECT AVG(Salary) FROM Employees);
```

**2. Multiple-Row Subquery**

Find employees who work in departments that are in New York.

```
SELECT EmployeeName
FROM Employees
WHERE DepartmentID IN (SELECT DepartmentID FROM Departments WHERE Location = 'New York');
```

**3. Multiple-Column Subquery**

Find employees who have the same job title and salary as an employee named 'John Doe'.

```
SELECT EmployeeName
FROM Employees
WHERE (JobTitle, Salary) = (SELECT JobTitle, Salary FROM Employees WHERE EmployeeName = 'John Doe');
```

## Correlation in Nested Queries

A correlated subquery depends on the outer query for its values and exe-

| | | |
|---|---|---|
| | cutes once for each row in the outer query.<br><br>**Example of Correlated Subquery**<br><br>Find employees who earn more than the average salary of their department.<br><br>**SELECT EmployeeName, Salary**<br>**FROM Employees e1**<br>**WHERE Salary > (SELECT AVG(Salary) FROM Employees e2 WHERE e1.DepartmentID = e2.DepartmentID);**<br><br>**Advantages of Nested Queries**<br><br>• **Simplicity**: Break complex queries into smaller, manageable parts.<br>• **Flexibility**: Combine multiple operations in a single query.<br>• **Readability**: Makes the query logic easier to understand by isolating different parts. | |

*****

**SET-3**

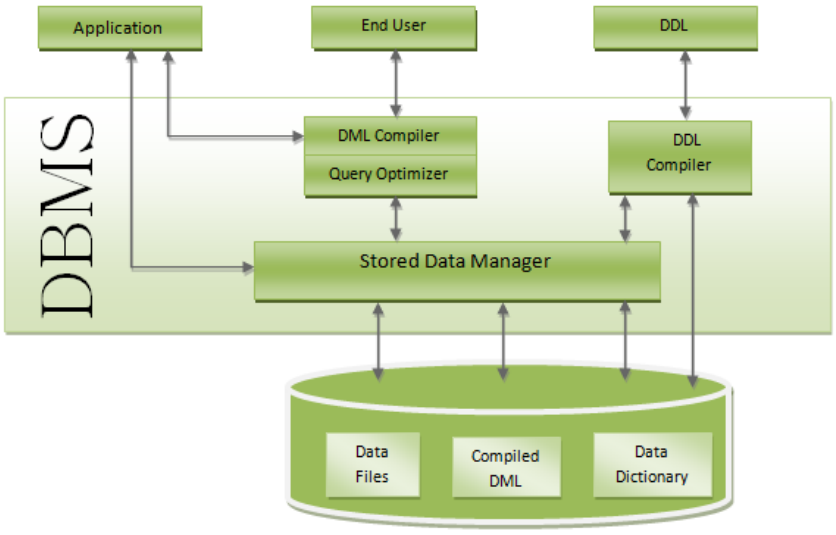
**PART – A (10x1M = 10M)**
**Note: Answer all Questions. Each Question carries equal marks.**

| Q. No | Question (s) | Marks |
|---|---|---|

| | | |
|---|---|---|
| **1** | **a) What is dbms?**<br>A **Database Management System (DBMS)** is software that enables the creation, management, and manipulation of databases. It provides tools for data storage, retrieval, and ensuring data integrity and security. | **1M** |
| | **b) Define entity?**<br>In DBMS, an **entity** is a distinct, identifiable object or concept, such as a person, place, thing, or event, that can have data stored about it in the database. Entities are represented by rectangles in ER diagrams. | **1M** |
| | **c) List out any 2 advantages of dbms?**<br><br>Two advantages of DBMS are:<br>1. **Data Redundancy Control**: Minimizes data duplication by centralizing data storage.<br>2. **Data Security**: Ensures data protection through access controls and authorization mechanisms. | **1M** |
| | **d) Define relationships in ER model?**<br>In an ER model in DBMS, **relationships** represent associations between entities. They are depicted as diamonds connecting related entities in an ER diagram. | **1M** |
| | **e) What is the use of SELECT in relational algebra?**<br>In relational algebra, **SELECT (σ)** is used to retrieve rows from a relation that satisfy a given predicate or condition, effectively filtering the data based on specific criteria. | **1M** |
| | **f) Define a primary key?**<br>A **primary key** in DBMS is a unique identifier for each record in a table. It ensures that each record is distinct and is typically used to establish relationships between tables. A primary key cannot contain NULL values. | **1M** |
| | **g) Short note on projection.**<br>**Projection** in DBMS is a relational algebra operation that retrieves specific columns from a relation (table). It is used to eliminate unwanted columns, returning only the desired attributes, and can also remove duplicate values in the result. This operation helps in focusing on particular data attributes while querying the database. | **1M** |
| | **h) Write any 2 advantages while creating a view.**<br><br>Two advantages of creating a view in DBMS are:<br>1. **Data Security**: Views can restrict access to specific data by displaying only selected columns or rows.<br>2. **Simplified Querying**: Views can simplify complex queries by encapsulating them into a single virtual table, making data retrieval easier. | **1M** |
| | **i) Describe the use of GROUP BY with a query.**<br><br>The `GROUP BY` clause in DBMS is used to group rows that have the same values in specified columns into summary rows. It is commonly used with aggregation functions like `COUNT()`, `SUM()`, `AVG()`, etc., to perform operations on each group.<br><br>**Example Query** | **1M** |

| | | |
|---|---|---|
| | SELECT dept_id, COUNT(emp_id) AS employee_count<br>FROM Employees<br>GROUP BY dept_id; | |
| | **j) What is ORDER BY? Explain with an example.**<br>**ORDER BY** in DBMS is a clause used in SQL to sort the result set of a query by one or more columns. It can sort data in ascending (default) or descending order.<br>Ex:<br>SELECT name, age<br>FROM students<br>ORDER BY age DESC; | **1M** |

**PART – B (20M)**

| Q. No | Question (s) | Marks |
|---|---|---|
| **2** | **a) A neat diagram of structure of dbms?**<br>  <br>A **Database Management System (DBMS)** is a complex software system that allows users to define, create, maintain, and control access to databases. The structure of a DBMS can be divided into | **8M** |

| | | |
|---|---|---|
| | several layers, each responsible for specific functions.<br><br>**Components of DBMS Structure:**<br><br>1. **User Interface:**<br>o **Applications:** Programs that interact with the database using query languages like SQL.<br>o **End Users:** Users who interact with the DBMS through applications or directly via query languages.<br>2. **Query Processor:**<br>o **DML Compiler:** Converts high-level queries into low-level instructions.<br>o **DDL Compiler:** Converts data definition language statements into a set of tables with metadata.<br>o **Query Optimizer:** Optimizes the execution of queries by choosing the most efficient strategy.<br>3. **Database Engine:**<br>o **Storage Manager:** Manages the allocation of space on disk and keeps track of data storage.<br>o **Transaction Manager:** Ensures that all database transactions are processed reliably and adhere to ACID properties.<br>o **Concurrency Control:** Manages the simultaneous operations without conflicting.<br>o **Recovery Manager:** Ensures the database can recover from crashes or errors.<br>4. **Database:**<br>o **Data Files:** Store the actual data.<br>o **Data Dictionary:** Metadata repository storing information about the database schema, constraints, users, etc.<br>o **Indices:** Improve the speed of data retrieval operations.<br><br>**Neat Diagram:** | |
| | **a) Explain about database manager and data manager from structure of DBMS?**<br><br>he **Database Manager** provides overall management and control, while the **Data Manager** focuses on efficient physical storage and access of data, ensuring the DBMS operates effectively. | |

| | | |
|---|---|---|
| | | |

| Component | Description | Responsibilities |
|---|---|---|
| **Database Manager** | Oversees the entire database environment. | - Data storage, retrieval, and updating |
| | | - Transaction management |
| | | - Concurrency control |
| | | - Backup and recovery |
| | | - Ensuring data integrity and security |
| **Data Manager (Storage Manager)** | Handles physical storage aspects of the database. | - Space allocation |
| | | - File organization |
| | | - Buffer management |
| | | - Disk management |
| | | - Index management |

| | | |
|---|---|---|
| **3** | **a) Explain conceptual design with ER Model**<br><br>## Conceptual Design with Entity-Relationship (ER) Model in DBMS<br><br>**Conceptual design** is a high-level representation of a database structure, primarily concerned with identifying entities, attributes, and relationships. The **Entity-Relationship (ER) Model** is a widely used conceptual data model that outlines these components in a structured and visual way. Here's a detailed explanation:<br><br>### 1. Identifying Entities<br><br>Entities are objects or things that have an independent existence in the database. They can be physical objects (like a `Student` or `Car`) or abstract concepts (like a `Course`).<br><br>**Example:**<br><br>• Entities: `Student`, `Course`, `Instructor`<br><br>### 2. Defining Attributes<br><br>Attributes are properties or characteristics of entities. They describe specific details about the entity.<br><br>**Example:**<br><br>• `Student` Entity: Attributes include `StudentID`, `Name`, `DOB`<br>• `Course` Entity: Attributes include `CourseID`, `CourseName`, | **4M** |

| | | |
|---|---|---|
| | `Credits`<br>• `Instructor` Entity: Attributes include `InstructorID`, `Name`, `Department`<br><br>**3. Establishing Relationships**<br><br>Relationships define how entities are related to each other. They can be one-to-one, one-to-many, or many-to-many.<br><br>**Example:**<br><br>• **One-to-Many**: A `Course` can have many `Students`, but a `Student` can enroll in multiple `Courses`.<br>• **Many-to-Many**: An `Instructor` can teach multiple `Courses`, and each `Course` can be taught by multiple `Instructors`.<br><br>**4. Creating the ER Diagram**<br><br>The ER Diagram visually represents entities, attributes, and relationships.<br><br>**Components:**<br><br>• **Entities**: Represented by rectangles.<br>• **Attributes**: Represented by ovals connected to their respective entities.<br>• **Relationships**: Represented by diamonds connecting related entities.<br><br>**Benefits of Conceptual Design with ER Model**<br><br>1. **Clarity**: Provides a clear and visual representation of the database structure.<br>2. **Simplicity**: Helps in understanding and communicating the database design to stakeholders.<br>3. **Flexibility**: Facilitates easier modifications and enhancements in the design phase.<br>4. **Normalization**: Ensures data redundancy is minimized and integrity is maintained. | |
| | **b) Explain the three levels of abstraction in a DBMS**<br><br>**1. Physical Level: How data is stored.**<br><br>• **Description:** The lowest level of abstraction, which details how the data is actually stored in the database. | **4M** |

| | | |
|---|---|---|
| | • **Focus:** Storage details, such as file structures, indices, and access methods.<br>• **Example:** Information about how data files are stored on disks, including details about file organization, indexing, and partitioning.<br><br>**2. Logical Level: What data is stored and the relationships among the data**<br><br>• **Description:** The middle level that describes what data is stored in the database and the relationships among the data. It hides the complexity of the physical storage.<br>• **Focus:** The structure of the database, including tables, views, and constraints.<br>• **Example:** A logical schema that defines the tables, columns, data types, and relationships, such as a schema for a `Students` table with columns like `StudentID`, `Name`, and `DOB`.<br><br>**3. View Level: How users see and interact with data.**<br><br>• **Description:** The highest level of abstraction, which describes only part of the entire database. It focuses on how users interact with data and provides different views of the same database.<br>• **Focus:** User-specific views and interfaces.<br>• **Example:** A view that shows only the `Name` and `DOB` columns of the `Students` table, tailored for a specific user group, like administrative staff or students.<br><br>**Advantages:**<br><br>• **Data Abstraction:** Helps in simplifying the database design by separating the details of how data is stored, structured, and presented.<br><br>• **Flexibility:** Allows changes at one level without affecting other levels. For example, changes in the physical storage do not impact the logical or view levels.<br><br>• **Security:** Provides controlled access to data by defining different views for different user groups, ensuring that users see only the data they need. | |
| **4** | **a) Explain different key constraints with syntax and examples?** | **4M** |

Key constraints are rules applied to keys to ensure data integrity and uniqueness in a database. Here are three key constraints:

1. **Primary Key Constraint:**
   - **Description:** Ensures that each row in a table has a unique and non-null value.
   - **Syntax:**

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    salary DECIMAL(10, 2)
);
```

   - **Example:** The `emp_id` column is the primary key, ensuring unique and non-null employee IDs.
2. **Foreign Key Constraint:**
   - **Description:** Ensures referential integrity by linking one table to another through a common field.
   - **Syntax:**

```
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    emp_id INT,
    FOREIGN KEY (emp_id) REFERENCES em-
ployees(emp_id)
);
```

   - **Example:** The `emp_id` column in the `orders` table references the `emp_id` column in the `employees` table, ensuring that each order is linked to a valid employee.
3. **Unique Key Constraint:**
   - **Description:** Ensures that all values in a column are unique, allowing null values.
   - **Syntax:**

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

   - **Example:** The `email` column is unique, ensuring no two users can have the same email address.

These key constraints help maintain data integrity and consistency in a relational database.

| | | |
|---|---|---|
| | **b)Explain any 3 relational operators with syntax and example?**<br><br>**Relational operators** are used to compare values in SQL queries, returning true or false based on the comparison.<br><br>1. **Equal To (=)**<br>o **Syntax:** `column_name = value`<br>o **Example:**<br><br>`SELECT * FROM employees WHERE salary = 50000;`<br><br>▪ Returns employees with a salary of 50,000.<br>2. **Greater Than (>)**<br>o **Syntax:** `column_name > value`<br>o **Example:**<br><br>`SELECT * FROM employees WHERE salary > 50000;`<br><br>▪ Returns employees with a salary greater than 50,000.<br>3. **Less Than (<)**<br>o **Syntax:** `column_name < value`<br>o **Example:**<br><br>`SELECT * FROM employees WHERE salary < 50000;`<br><br>▪ Returns employees with a salary less than 50,000.<br><br>These operators are fundamental for filtering data based on specific conditions in SQL queries, enabling efficient data retrieval. | |
| **5** | **a) Write 4 different DDL commands and results on student table.**<br><br>**1. CREATE TABLE**<br><br>Creates the `Students` table with columns `StudentID`, `Name`, `DOB`, and `Email`.<br><br>CREATE TABLE Students (<br>StudentID INT PRIMARY KEY,<br>Name VARCHAR(100),<br>DOB DATE,<br>Email VARCHAR(100) | **4M** |

| | | |
|---|---|---|
| | );<br><br>**2. ALTER TABLE (Add Column)**<br><br>Adds a new column `PhoneNumber` to the `Students` table.<br><br>ALTER TABLE Students<br>ADD PhoneNumber VARCHAR(15);<br><br>**3. DROP TABLE**<br><br>Deletes the `Students` table and all its data.<br><br>**DROP TABLE Students;**<br><br>**4. ALTER TABLE (Modify Column)**<br><br>Modifies the `Email` column to increase its length to 150 characters.<br><br>**ALTER TABLE Students**<br>**MODIFY Email VARCHAR(150);** | |
| | **b) What is a view Create a view on a table and how to destroy the view.**<br><br>A **view** in SQL is a virtual table that provides a way to present data from one or more tables. Views do not store data themselves but fetch data from the underlying tables whenever queried.<br><br>**Creating a View**<br><br>Let's say you have a `Students` table, and you want to create a view that shows only the `StudentID` and `Name`.<br><br>**Example:**<br><br>CREATE VIEW StudentNames AS<br>SELECT StudentID, Name<br>FROM Students;<br><br>**Using the View**<br><br>You can query the view just like a regular table.<br><br>**Example:**<br><br>SELECT * FROM StudentNames; | **4M** |

| | | |
|---|---|---|
| | **Destroying the View**<br><br>To delete or destroy a view, you use the `DROP VIEW` statement.<br><br>**Example:**<br><br>DROP VIEW StudentNames; | |
| **6** | **Define aggregate functions and their uses with the help of example queries.**<br><br>**ggregate Functions in SQL**<br><br>Aggregate functions in SQL perform calculations on a set of values and return a single value. They are commonly used in conjunction with the `GROUP BY` clause to summarize data. Here are some of the most commonly used aggregate functions, along with example queries:<br><br>**1. SUM()**<br><br>Calculates the total sum of a numeric column.<br><br>**SELECT SUM(Salary) AS TotalSalary**<br>**FROM Employees;**<br><br>**2. AVG()**<br><br>Calculates the average value of a numeric column.<br><br>**Example:**<br><br>**SELECT AVG(Salary) AS AverageSalary**<br>**FROM Employees;**<br><br>**3. COUNT()**<br><br>Counts the number of rows that match a specified condition.<br><br>**Example:**<br><br>**SELECT COUNT(*) AS NumberOfEmployees**<br>**FROM Employees;**<br><br>**4. MAX()**<br><br>Finds the maximum value in a numeric column. | **4M** |

| | | |
|---|---|---|
| | **SELECT MAX(Salary) AS HighestSalary**<br>**FROM Employees;**<br><br>**5. MIN()**<br><br>Finds the minimum value in a numeric column.<br><br>**Example:**<br><br>**SELECT MIN(Salary) AS LowestSalary**<br>**FROM Employees;**<br><br>**Using Aggregate Functions with GROUP BY**<br><br>Aggregate functions can be used with the `GROUP BY` clause to group rows that have the same values in specified columns.<br><br>**Example:** Calculate the total salary for each department.<br><br>**SELECT DepartmentID, SUM(Salary) AS TotalSalary**<br>**FROM Employees**<br>**GROUP BY DepartmentID;** | |
| **7** | **Explain TRIGGERS and how to create a trigger while data insert into a table.**<br><br>**TRIGGERS in DBMS**<br><br>A **trigger** is a special kind of stored procedure that automatically executes when certain events occur in the database. Triggers are used to enforce business rules, maintain audit trails, and manage data integrity by automating actions in response to changes in the database.<br><br>**Types of Triggers**<br><br>1. **Before Trigger**: Executes before an event (such as an `INSERT`, `UPDATE`, or `DELETE` operation) occurs.<br>2. **After Trigger**: Executes after an event has occurred.<br>3. **Instead of Trigger**: Used with views to perform operations instead of the triggering event.<br><br>**Creating a Trigger for Data Insert**<br><br>Let's create a trigger that logs each `INSERT` operation on a `Students` table into a `Logs` table. | **4M** |

**Example:**

**1. Define the Tables**

**Students Table:**

```
CREATE TABLE Students (
   StudentID INT PRIMARY KEY,
   Name VARCHAR(100),
   DOB DATE,
   Email VARCHAR(100)
);
Log Table"
CREATE TABLE Logs (
   LogID INT PRIMARY KEY AUTO_INCREMENT,
   LogMessage VARCHAR(255),
   LogTime DATETIME
);
Trigger Definition:
CREATE TRIGGER BeforeInsertStudent
BEFORE INSERT ON Students
FOR EACH ROW
BEGIN
   INSERT INTO Logs (LogMessage, LogTime)
   VALUES (CONCAT('New student added: ', NEW.Name),
NOW());
END;
```

## Benefits of Triggers

- **Automated Actions:** Automatically execute predefined actions in response to specific events.
- **Data Integrity:** Enforce complex business rules and maintain data consistency.
- **Audit Trails:** Record changes and maintain a history of database operations.

*****