DBMS QUESTION BANK

Module 1

1. What is DBMS? Explain advantages and disadvantages of DBMS.

A **Database Management System (DBMS)** is software that allows users to store, retrieve, manage, and manipulate data efficiently. It acts as an interface between the database and users or applications.

Advantages of DBMS:

- **Data Redundancy Control** – Eliminates data duplication.
- **Data Consistency** – Ensures uniform and accurate data.
- **Data Security** – Provides authentication and authorization.
- **Data Integrity** – Enforces constraints to maintain accuracy.
- **Concurrency Control** – Supports multiple users accessing data simultaneously.

Disadvantages of DBMS:

- **High Cost** – Expensive software and hardware requirements.
- **Complexity** – Requires skilled personnel for management.
- **Performance Issues** – Overhead due to security, integrity checks.
- **Data Loss Risk** – If the system crashes, recovery is difficult.
- **Regular Maintenance** – Requires continuous monitoring and updates.

---

2. What is Database? Explain different types of Database.

A **Database** is an organized collection of data that allows for efficient storage, retrieval, and management of information.

Types of Databases:

- **Hierarchical Database** – Data is organized in a tree-like structure (e.g., IBM IMS).
- **Network Database** – Uses a graph structure with multiple relationships (e.g., CODASYL).
- **Relational Database (RDBMS)** – Stores data in tables with relationships (e.g., MySQL, PostgreSQL).
- **Object-Oriented Database** – Stores data in the form of objects (e.g., ObjectDB).
- **NoSQL Database** – Designed for unstructured data (e.g., MongoDB, Cassandra).
- **Distributed Database** – Data is stored across multiple locations (e.g., Google Spanner).
- **Cloud Database** – Hosted on cloud platforms (e.g., Amazon RDS, Google Firebase).

---

3. Explain DBMS Architecture with a Neat and Clean Diagram.

DBMS architecture defines how data is stored, accessed, and managed in a structured way.

Types of DBMS Architecture:

- **One-Tier Architecture** – The database and application are on the same system.
- **Two-Tier Architecture** – Client-server model where applications communicate with the database via SQL.
- **Three-Tier Architecture** – Separates the application into three layers:
  - **Presentation Layer** (UI)
  - **Application Layer** (Logic)
  - **Database Layer** (Storage)

(Insert a diagram for better understanding in Notion.)

---

4. What is Entity-Relationship (ER) modeling? Explain its Components in Detail.

ER modeling is a conceptual framework for structuring data in databases using entities, attributes, and relationships.

Components of ER Model:

- **Entity** – Objects that represent real-world data (e.g., Student, Employee).
- **Attributes** – Characteristics of an entity (e.g., Name, Age).
- **Primary Key** – A unique identifier for an entity.
- **Relationships** – Associations between entities (e.g., Student-Enrolls-Course).
- **Cardinality** – Defines the number of entity instances that can be associated (1:1, 1:M, M:N).
- **Generalization & Specialization** – Hierarchical relationships among entities.

---

5. What is meant by Database Model? Explain its Types.

A **Database Model** defines how data is structured, stored, and manipulated in a DBMS.

Types of Database Models:

- **Hierarchical Model** – Data organized in a parent-child relationship.
- **Network Model** – Uses a graph-like structure with multiple relationships.
- **Relational Model** – Data stored in tables with keys (used in RDBMS).
- **Object-Oriented Model** – Data stored as objects similar to OOP concepts.
- **Document-Oriented Model** – Stores data as JSON or XML (e.g., MongoDB).

---

6. Draw E-R Diagram for Bank Management System.

An **E-R Diagram** for a **Bank Management System** includes:

Entities:

- Customer, Account, Transaction, Loan.

Attributes:

- CustomerID, Name, Account Number, Balance, Loan Amount.

Relationships:

- **Customer → Owns → Account**
- **Account → Has → Transaction**
- **Customer → Applies → Loan**

(Insert an E-R Diagram in Notion for better visualization.)

---

7. Explain the Concept of Cardinality and Participation Constraints in ER Modeling.

1. Cardinality Constraints

Defines the number of entity instances that can be associated with another entity.

- **1:1 (One-to-One)** – A manager manages one department.
- **1:M (One-to-Many)** – A teacher teaches multiple students.
- **M:N (Many-to-Many)** – Students enroll in multiple courses.

2. Participation Constraints

Defines whether all entities must participate in a relationship.

- **Total Participation** – Every entity must be involved (e.g., Every employee has a department).
- **Partial Participation** – Some entities may not be involved (e.g., Some students do not take extra courses).

Module 2

1. What is a Command? Explain Types of Commands in MySQL.

A **command** in MySQL is an instruction given to perform database operations such as creating, modifying, retrieving, or managing data.

Types of Commands in MySQL:

1. **DDL (Data Definition Language)** – Defines database structures. (Commands: `CREATE`, `ALTER`, `DROP`)

2. **DML (Data Manipulation Language)** – Modifies data. (Commands: `INSERT`, `UPDATE`, `DELETE`)
3. **DCL (Data Control Language)** – Manages permissions. (Commands: `GRANT`, `REVOKE`)
4. **TCL (Transaction Control Language)** – Controls transactions. (Commands: `COMMIT`, `ROLLBACK`)
5. **DQL (Data Query Language)** – Retrieves data. (Command: `SELECT`)

---

2. What is a DDL Statement? Explain with Syntax and Example.

DDL (**Data Definition Language**) is used to define database structures such as tables and schemas.

Syntax:

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

Example:

```
CREATE TABLE students (
    id INT PRIMARY KEY,
    name VARCHAR(50)
);
```

This command creates a `students` table with `id` as an integer primary key and `name` as a text field.

---

3. What is a DML Statement? Explain with Syntax and Example.

DML (**Data Manipulation Language**) is used to insert, update, and delete data.

Syntax:

```
INSERT INTO table_name (column1, column2) VALUES (value1, value2);
```

Example:

```
INSERT INTO students (id, name) VALUES (1, 'Rahul');
```

This inserts a new record into the `students` table.

---

4. Explain the Use of the SELECT Statement with Suitable Examples.

The **SELECT** statement retrieves data from a database.

Syntax:

```
SELECT column1, column2 FROM table_name WHERE condition;
```

Example:

```
SELECT name FROM students WHERE id = 1;
```

This retrieves the `name` of the student whose `id` is 1.

---

5. What is an Aggregate Function in DBMS? Explain with Example.

Aggregate functions perform calculations on multiple rows and return a single value.

Example:

- **SUM:**
  Returns the total salary of all employees.

  ```
  SELECT SUM(salary) FROM employees;
  ```

- **AVG:**
  Returns the average age of students.

  ```
  SELECT AVG(age) FROM students;
  ```

---

6. Describe the Use of Conditional Operators in SQL with Examples.

Conditional operators filter data based on conditions.

1. **LIKE:** Finds patterns.

```
SELECT * FROM students WHERE name LIKE 'A%';
```

2. **BETWEEN:** Selects values in a range.

```
SELECT * FROM students WHERE age BETWEEN 18 AND 25;
```

3. **OR:** Selects records if any condition is true.

```
SELECT * FROM students WHERE age < 18 OR age > 25;
```

4. **AND:** Selects records if all conditions are true.

```
SELECT * FROM students WHERE age >= 18 AND age <= 25;
```

5. **IN:** Checks if a value exists in a set.

```
SELECT * FROM students WHERE age IN (18, 20, 22);
```

6. **IS NULL:** Checks for NULL values.

```
SELECT * FROM students WHERE email IS NULL;
```

---

7. Differentiate Between DDL and DML with Examples.

DDL (Data Definition Language):

- Defines database structure.
- Examples: CREATE, ALTER, DROP.
- **Example:**

```
CREATE TABLE employees (id INT, name VARCHAR(50));
```

DML (Data Manipulation Language):

- Modifies database data.
- Examples: INSERT, UPDATE, DELETE.
- **Example:**

```
INSERT INTO employees (id, name) VALUES (1, 'John');
```

---

8. Discuss the Importance of the WHERE Clause with an Example.

The **WHERE** clause filters records based on conditions.

Example:

- Without WHERE:
  Retrieves all student records.

```
SELECT * FROM students;
```

- With WHERE:
  Retrieves only students older than 18.

```
SELECT * FROM students WHERE age > 18;
```

---

9. Differentiate Between ORDER BY, GROUP BY, and HAVING Clauses with Examples.

ORDER BY: Sorts results.

```
SELECT * FROM students ORDER BY age DESC;
```

Sorts students in descending order by age.

GROUP BY: Groups rows with the same values.

```
SELECT age, COUNT(*) FROM students GROUP BY age;
```

Groups students by age and counts them.

HAVING: Filters grouped results.

```
SELECT age, COUNT(*) FROM students GROUP BY age HAVING COUNT(*) > 2;
```

Displays only age groups where more than 2 students exist.

Module 3

1. What is Join? Explain Different Types of Joins with Examples.

A **JOIN** in SQL is used to combine records from two or more tables based on a related column.

Types of Joins:

1. **INNER JOIN** – Returns only matching records from both tables.

   ```
   SELECT employees.name, departments.department_name
   FROM employees
   INNER JOIN departments ON employees.dept_id = departments.id;
   ```

2. **LEFT JOIN (OUTER JOIN)** – Returns all records from the left table and matching records from the right table.

   ```
   SELECT employees.name, departments.department_name
   FROM employees
   LEFT JOIN departments ON employees.dept_id = departments.id;
   ```

3. **RIGHT JOIN (OUTER JOIN)** – Returns all records from the right table and matching records from the left table.

   ```
   SELECT employees.name, departments.department_name
   FROM employees
   RIGHT JOIN departments ON employees.dept_id = departments.id;
   ```

4. **FULL JOIN (FULL OUTER JOIN)** – Returns all records when there is a match in either table.

```
SELECT employees.name, departments.department_name
FROM employees
FULL JOIN departments ON employees.dept_id = departments.id;
```

5. **CROSS JOIN** – Returns the Cartesian product of both tables.

```
SELECT employees.name, departments.department_name
FROM employees
CROSS JOIN departments;
```

---

2. What are Subqueries? Explain Nested Subqueries in Detail.

A **subquery** is a query within another SQL query that provides results for the main query.

Types of Subqueries:

1. **Single-row subquery** – Returns a single value.

```
SELECT name FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees);
```

2. **Multi-row subquery** – Returns multiple values.

```
SELECT name FROM employees
WHERE department_id IN (SELECT id FROM departments WHERE locati
on = 'Mumbai');
```

3. **Nested Subqueries** – A subquery within another subquery.

```
SELECT name FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees WHERE depar
tment_id =
                (SELECT id FROM departments WHERE department_name = 'I
T'));
```

3. What is the Difference Between Trigger and Stored Procedure?

Feature	Trigger	Stored Procedure
Execution	Automatically on event	Manually invoked
Use Case	Enforces rules & integrity	Complex operations
Parameters	Cannot accept parameters	Accepts parameters
Execution Time	Runs before/after DML	Runs when called
Example	`AFTER INSERT` trigger	`CREATE PROCEDURE` function

Example of Trigger:

```
CREATE TRIGGER after_employee_insert
AFTER INSERT ON employees
FOR EACH ROW
INSERT INTO audit_log (action, timestamp) VALUES ('Employee Added', NO
W());
```

Example of Stored Procedure:

```
CREATE PROCEDURE GetEmployee(IN emp_id INT)
BEGIN
  SELECT * FROM employees WHERE id = emp_id;
END;
```

4. What is Normalization? Explain Different Forms of Normalization.

Normalization is the process of organizing data to reduce redundancy and improve integrity.

Forms of Normalization:

1. **1NF (First Normal Form)** – Eliminates duplicate columns and ensures atomicity.

2. **2NF (Second Normal Form)** – Removes partial dependencies; all non-key attributes must be fully dependent on the primary key.
3. **3NF (Third Normal Form)** – Removes transitive dependencies (i.e., no non-key attribute should depend on another non-key attribute).
4. **BCNF (Boyce-Codd Normal Form)** – Ensures every determinant is a candidate key.
5. **4NF (Fourth Normal Form)** – Eliminates multi-valued dependencies.
6. **5NF (Fifth Normal Form)** – Deals with complex join dependencies.

Example of Normalization:

Before Normalization (1NF Violation):

OrderID	Customer	Products
1	Alice	Laptop, Mouse
2	Bob	Keyboard

After 1NF:

OrderID	Customer	Product
1	Alice	Laptop
1	Alice	Mouse
2	Bob	Keyboard

---

5. What is the Isolation Level? Explain in Detail.

Isolation levels define how transactions interact with each other in a database.

Types of Isolation Levels:

1. **Read Uncommitted** – Allows dirty reads (reading uncommitted changes).
2. **Read Committed** – Prevents dirty reads but allows non-repeatable reads.
3. **Repeatable Read** – Prevents dirty and non-repeatable reads but allows phantom reads.
4. **Serializable** – Highest level; ensures full transaction isolation.

Example of Isolation Levels:

- **Read Uncommitted:**

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

- **Serializable:**

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

6. Explain the ACID Properties of a Database Transaction.

The **ACID properties** ensure that database transactions are processed reliably.

1. Atomicity

Ensures that a transaction is **all or nothing**. If one part fails, the entire transaction is rolled back.

```
START TRANSACTION;
UPDATE accounts SET balance = balance - 500 WHERE id = 1;
UPDATE accounts SET balance = balance + 500 WHERE id = 2;
COMMIT;
```

2. Consistency

Ensures data remains **valid** before and after a transaction.

3. Isolation

Ensures transactions **do not interfere** with each other.

4. Durability

Once committed, a transaction's changes are **permanently saved**, even after a system failure.

---

7. What is SQL View? Explain in Detail.

A **view** is a virtual table based on a query. It does not store data but displays results dynamically.

Syntax:

```
CREATE VIEW student_view AS
SELECT id, name FROM students WHERE age > 18;
```

Example:

```
SELECT * FROM student_view;
```

This retrieves all students older than 18 from the view.

Benefits of Views:

- Simplifies complex queries
- Enhances security by restricting access
- Provides a consistent data representation

---

8. Describe the Concept of Concurrency Control and Its Significance.

Concurrency control ensures that multiple transactions execute **simultaneously without conflicts** in a database.

Issues in Concurrency:

1. **Dirty Reads** – Reading uncommitted data from another transaction.
2. **Non-Repeatable Reads** – Same query returns different results in one transaction.
3. **Phantom Reads** – New records appear during a transaction.

Concurrency Control Methods:

- **Lock-Based Protocols** (Shared and Exclusive Locks)
- **Timestamp-Based Protocols** (Assigns unique timestamps)
- **Optimistic Concurrency Control** (Validates at commit time)

---

9. What is a Trigger in SQL? Explain Its Types.

A **trigger** is an automatic action executed before or after a database event like INSERT, UPDATE, or DELETE.

Types of Triggers:

1. **Before Trigger** – Executes **before** the event.

```
CREATE TRIGGER before_insert_student
BEFORE INSERT ON students
FOR EACH ROW
SET NEW.name = UPPER(NEW.name);
```

2. **After Trigger** – Executes **after** the event.

```
CREATE TRIGGER after_delete_student
AFTER DELETE ON students
FOR EACH ROW
INSERT INTO audit_log(action, timestamp) VALUES ('Student Deleted', NO
W());
```

3. **Instead Of Trigger** – Used on views instead of performing an action directly.

---

10. What is a Stored Procedure? What Types of Parameters Are Available in Stored Procedure?

A **stored procedure** is a precompiled SQL block stored in the database that can be executed multiple times.

Syntax:

```
CREATE PROCEDURE GetStudent(IN student_id INT)
BEGIN
  SELECT * FROM students WHERE id = student_id;
END;
```

Types of Parameters in Stored Procedures:

1. **IN Parameter** – Passes input to the procedure.
2. **OUT Parameter** – Returns an output value.
3. **INOUT Parameter** – Acts as both input and output.

Example:

```
CREATE PROCEDURE UpdateBalance(INOUT balance INT, IN amount INT)
BEGIN
```

```
    SET balance = balance + amount;
END;
```

Module 4

1. What is User Management? Explain Types of Users and How to Create a User?

User Management in DBMS refers to managing database users, their roles, and access permissions to ensure data security and controlled access.

Types of Users in DBMS:

1. **Database Administrator (DBA)** – Manages the entire database system.
2. **End Users** – Access the database for querying and reports.
3. **Application Programmers** – Develop applications that interact with the database.
4. **System Analysts** – Design and optimize database structures.
5. **Security Officers** – Handle user access and security.

Creating a User in SQL:

```
CREATE USER 'new_user'@'localhost' IDENTIFIED BY 'password';
```

To grant permissions:

```
GRANT ALL PRIVILEGES ON database_name.* TO 'new_user'@'localhost';
```

To remove a user:

```
DROP USER 'new_user'@'localhost';
```

---

2. What is Role and Privileges? Explain with an Example?

A **role** in DBMS is a collection of privileges assigned to users to control their access levels. **Privileges** define the specific actions a user can perform on a

database.

Types of Privileges:

1. **System Privileges** – Allow user management and administrative tasks (e.g., `CREATE USER`, `GRANT` privileges).
2. **Object Privileges** – Control actions on specific database objects (e.g., `SELECT`, `INSERT`, `DELETE`).

Example of Assigning a Role and Privileges:

```
CREATE ROLE manager;
GRANT SELECT, INSERT, UPDATE ON employees TO manager;
GRANT manager TO 'john'@'localhost';
```

To revoke privileges:

```
REVOKE INSERT ON employees FROM manager;
```

---

3. Describe the Importance of Backup and Restore Operations in a DBMS.

Backup and restore operations are essential in DBMS to prevent **data loss, corruption, or accidental deletion** and to recover data in case of a system failure.

Importance of Backup and Restore:

1. **Data Recovery** – Protects against accidental deletion or hardware failure.
2. **Disaster Recovery** – Helps in restoring data after system crashes.
3. **Security & Compliance** – Ensures data integrity and meets regulatory requirements.
4. **Business Continuity** – Ensures minimal downtime during failures.

---

4. How to Backup and Restore Database Using DBMS?

Backup a Database in MySQL:

To create a backup of a MySQL database:

```
mysqldump -u root -p database_name > backup.sql
```

To take a full backup including all databases:

```
mysqldump -u root -p --all-databases > full_backup.sql
```

Restore a Database in MySQL:

To restore the database from a backup file:

```
mysql -u root -p database_name < backup.sql
```

For full database restoration:

```
mysql -u root -p < full_backup.sql
```

These operations ensure **data security and recovery** in case of failures.

Module 5

1. What is Indexing in a Database? Explain Its Types.

Indexing is a technique used in databases to improve the speed of data retrieval. It works like an index in a book, allowing the database to locate data quickly without scanning entire tables.

Types of Indexing:

1. **Primary Index** – Created on a primary key column for faster access.
2. **Clustered Index** – Determines the physical order of data storage in a table.
3. **Non-Clustered Index** – Maintains a separate structure to store index data, pointing to actual data.
4. **Unique Index** – Ensures that all values in the indexed column are unique.
5. **Composite Index** – Created on multiple columns for optimized multi-column queries.

6. **Full-Text Index** – Used for searching text-based data efficiently.

Example:

```
CREATE INDEX idx_student_name ON students(name);
```

This creates an index on the `name` column of the `students` table to speed up searches.

---

2. Explain the Role of Database Monitoring in Database Performance Improvement.

Database monitoring is the process of tracking and analyzing database performance to optimize efficiency and prevent issues.

Importance of Database Monitoring:

1. **Identifies Slow Queries** – Helps detect inefficient queries that slow down performance.
2. **Detects Resource Utilization Issues** – Monitors CPU, memory, and disk usage.
3. **Prevents Deadlocks** – Identifies and resolves transaction conflicts.
4. **Ensures Security** – Tracks unauthorized access and security threats.
5. **Automates Alerts** – Sends notifications for performance issues.

Common Database Monitoring Tools:

- MySQL Performance Schema
- Oracle Enterprise Manager
- SQL Server Profiler
- Prometheus + Grafana for real-time monitoring

---

3. What is Query Optimization and How to Do Query Optimization?

Query Optimization is the process of improving the execution speed and efficiency of SQL queries.

Techniques for Query Optimization:

1. **Use Indexes** – Indexing speeds up search operations.

```
CREATE INDEX idx_employee_salary ON employees(salary);
```

2. *Avoid SELECT *** – Fetch only required columns.

```
SELECT name, age FROM students;
```

3. **Use Joins Instead of Subqueries** – Joins execute faster than nested queries.

```
SELECT e.name, d.department_name FROM employees e
JOIN departments d ON e.dept_id = d.id;
```

4. **Optimize WHERE Clauses** – Use indexed columns in conditions.

```
SELECT * FROM orders WHERE order_date > '2023-01-01';
```

5. **Limit the Number of Rows Returned** – Use `LIMIT` for faster results.

```
SELECT * FROM customers LIMIT 10;
```

6. **Use EXPLAIN Plan** – Analyze query execution steps.

```
EXPLAIN SELECT * FROM orders WHERE status = 'Pending';
```

By implementing these optimization techniques, database performance can be significantly improved.