Module 4 (Theory)
Introduction to DBMS

1. What is SQL, and why is it essential in database management?

- SQL (Structured Query Language) is a computer language used to store, manage, and retrieve data in a relational database.
- It allows us to **create databases, insert, update, delete, and fetch records**.
- SQL is essential because it is the **standard language** used in almost all relational database systems like MySQL, Oracle, and SQL Server, and it makes working with data easier and faster

2. Explain the difference between DBMS and RDBMS.

- **DBMS (Database Management System):** A software system that stores and manages data. It provides ways to store, retrieve, and manipulate data. Example: dBase, FoxPro.

- **RDBMS (Relational Database Management System):** A type of DBMS that stores data in **tables (relations)** and uses **SQL** for managing data. It also supports **relationships between tables, data integrity, and normalization**. Examples: MySQL, Oracle, PostgreSQL

3. Describe the role of SQL in managing relational databases.

SQL plays a central role in relational databases by allowing users to:

- Define and organize data (create databases and tables).
- Manipulate data (insert, update, delete, retrieve records).
- Control access (set permissions).
- Create advanced objects like views, stored procedures, and functions.

In short, SQL helps users **communicate with the database** and manage data effectively

4. What are the key features of SQL?

According to the document, SQL features include:

- Standard language for relational databases.

- Can **create and drop** databases and tables.
- Can **insert, update, delete, and select** data.
- Can create **views, stored procedures, and functions**.
- Can set **permissions** for tables and views.
- Easy to learn and widely supported

5. What are the basic components of SQL syntax?

The main SQL statement types are:

- **DDL (Data Definition Language):** Defines database structure (e.g., CREATE, ALTER, DROP).
- **DML (Data Manipulation Language):** Manages data inside tables (e.g., INSERT, UPDATE, DELETE).
- **DQL (Data Query Language):** Retrieves data (e.g., SELECT).
- **DCL (Data Control Language):** Controls access (e.g., GRANT, REVOKE).
- **TCL (Transaction Control Language):** Manages transactions (e.g., COMMIT, ROLLBACK).

6. Write the general structure of an SQL `SELECT` statement.

The basic structure is:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column_name
HAVING condition
ORDER BY column_name ASC|DESC;
```

It means: choose columns from a table, filter rows with `WHERE`, group them with `GROUP BY`, apply conditions with `HAVING`, and finally sort with `ORDER BY`

7. Explain the role of clauses in SQL statements.

Clauses are special keywords used in SQL statements to **add conditions and rules**.
Examples:

- `WHERE` → filter rows.
- `ORDER BY` → sort results.
- `GROUP BY` → group rows.

- `HAVING` → set condition on groups.
  Clauses make SQL queries **more specific and powerful**

8. What are constraints in SQL? List and explain the different types of constraints.

constraints.
Constraints are **rules applied to table columns** to ensure correct and valid data.
Types:

- **PRIMARY KEY** → uniquely identifies each row; no duplicate or NULL.
- **FOREIGN KEY** → links one table to another using the primary key.
- **UNIQUE** → ensures all values in a column are different (but allows one NULL).
- **NOT NULL** → column cannot have empty values.
  They maintain **accuracy and consistency** of data

9. How do `PRIMARY KEY` and `FOREIGN KEY` constraints differ?

• **PRIMARY KEY**: uniquely identifies each record in a table. No duplicates or NULLs allowed.

• **FOREIGN KEY**: creates a link between two tables by referencing the primary key of another table. It ensures **relationship and data integrity**

10. What is the role of `NOT NULL` and `UNIQUE` constraints?

• **NOT NULL** → makes sure a column cannot have empty values.

• **UNIQUE** → makes sure all values are different (but allows one NULL).
Together, they prevent **missing or duplicate data**

11. Define the SQL Data Definition Language (DDL).

DDL is the part of SQL used to **define and change the structure of database objects** like tables, views, and indexes.
Examples:

- `CREATE` → make new objects.
- `ALTER` → change objects.
- `DROP` → delete objects

12. Explain the `CREATE` command and its syntax.

The `CREATE` command is used to make a **new database or table**.

Syntax for table:

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   ...
   PRIMARY KEY(column_name)
);
```

It defines table columns, data types, and constraints

13. What is the purpose of specifying data types and constraints during table creation?

- **Data types** → decide what kind of data (number, text, date) can be stored in a column.

- **Constraints** → ensure correctness (like no duplicates, no nulls, valid relationships).

This guarantees that the table stores **only valid and accurate data**

14. What is the use of the `ALTER` command in SQL?

The `ALTER` command is used to **change the structure of a table** without deleting it.
Examples: rename a table, add a new column, or change column type

15. How can you add, modify, and drop columns from a table using `ALTER`?

- Add column:

```
ALTER TABLE table_name ADD column_name datatype;
```

- Modify column:

```
ALTER TABLE table_name MODIFY column_name new_datatype;
```

- Drop column:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

16. What is the function of the `DROP` command in SQL?

The `DROP` command is used to **delete an entire table, database, view, or index** permanently from the database

17. What are the implications of dropping a table from a database?

- The table and all its data are **deleted permanently**.

- All relationships (like foreign keys) and indexes linked to it are removed.
- Once dropped, the table **cannot be recovered** unless a backup exists

18. Define the `INSERT`, `UPDATE`, and `DELETE` commands in SQL.

- **INSERT** → Adds new records into a table.
- `INSERT INTO table_name (col1, col2) VALUES (val1, val2);`

- **UPDATE** → Modifies existing records.
- `UPDATE table_name SET col1 = val1 WHERE condition;`

- **DELETE** → Removes records from a table.
- `DELETE FROM table_name WHERE condition;`

19. What is the importance of the `WHERE` clause in `UPDATE` and `DELETE` operations?

The `WHERE` clause is very important because it **restricts which rows are updated or deleted**.

- Without `WHERE`, all rows in the table will be changed or deleted

20. What is the `SELECT` statement, and how is it used to query data?

The `SELECT` statement is used to **retrieve data** from one or more tables.
Syntax:

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

It is the most common command for **viewing and analyzing data**

21. Explain the use of the `ORDER BY` and `WHERE` clauses in SQL queries.

- **WHERE** → Filters rows based on conditions. Example:
- `SELECT * FROM Students WHERE Age > 18;`

- **ORDER BY** → Sorts results in ascending (`ASC`) or descending (`DESC`) order. Example:
- `SELECT * FROM Students ORDER BY Name ASC;`

22. What is the purpose of `GRANT` and `REVOKE` in SQL?

- **GRANT** → Gives a user certain permissions (like SELECT, INSERT, UPDATE).
- **REVOKE** → Takes back those permissions

23. How do you manage privileges using these commands?

- Give privilege:
- `GRANT SELECT, INSERT ON table_name TO user_name;`

- Remove privilege:
- `REVOKE INSERT ON table_name FROM user_name;`

This helps in **controlling access and security** in the database

24. What is the purpose of the `COMMIT` and `ROLLBACK` commands in SQL?

- **COMMIT** → Saves all changes permanently in the database.
- **ROLLBACK** → Cancels changes and restores the database to its previous state

25. Explain how transactions are managed in SQL databases.

A **transaction** is a group of SQL operations treated as a single unit.

- **BEGIN** → Starts the transaction.
- **COMMIT** → Saves all changes.
- **ROLLBACK** → Cancels changes if something goes wrong.
- **SAVEPOINT** → Creates checkpoints inside a transaction to rollback partly if needed.

Transactions ensure **data consistency, accuracy, and recovery**

26. Explain the concept of `JOIN` in SQL. What is the difference between `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, and `FULL OUTER JOIN`?

- A **JOIN** is used to combine rows from two or more tables based on a common field.

Types:

- **INNER JOIN** → returns only matching rows from both tables.

- **LEFT JOIN** → returns all rows from the left table + matching rows from the right table (NULL if no match).

- **RIGHT JOIN** → returns all rows from the right table + matching rows from the left table (NULL if no match).

- **FULL OUTER JOIN** → returns all rows from both tables, filling NULL where no match exists

27. How are joins used to combine data from multiple tables?

Joins allow us to link tables together using a **common field (like an ID)**.
For example:

```
SELECT Students.name, Courses.course_name
FROM Students
INNER JOIN Courses ON Students.course_id = Courses.id;
```

This combines student names with their course details

28. What is the `GROUP BY` clause in SQL? How is it used with aggregate functions?

- `GROUP BY` groups rows that have the same values in a column.
- It is mostly used with **aggregate functions** like `COUNT`, `SUM`, `AVG`, `MAX`, `MIN`.
  Example:

```
SELECT department, COUNT(*)
FROM Employees
GROUP BY department;
```

This counts employees in each department

29. Explain the difference between `GROUP BY` and `ORDER BY`.

- **GROUP BY** → groups rows based on column values (used with aggregates).

- **ORDER BY** → sorts rows in ascending or descending order.

Example:

- `GROUP BY` → "How many students in each class."

- `ORDER BY` → "List students sorted by name."

30. What is a stored procedure in SQL, and how does it differ from a standard SQL query?

- A **stored procedure** is a saved SQL code that can be executed repeatedly.
- Unlike a normal SQL query, it can have **parameters, logic, and reusability**.
  Syntax:

```
CREATE PROCEDURE proc_name AS sql_statement;
EXEC proc_name;
```

31. Explain the advantages of using stored procedures.

- Reusable (write once, use many times).
- Better **performance** (runs faster as precompiled).
- Improves **security** (controls access).
- Reduces **network traffic** (less code sent).
- Makes code **modular and maintainable**

32. What is a view in SQL, and how is it different from a table?

- A **view** is a virtual table created from a query on one or more tables.
- It does **not store data**, only shows data from base tables.
- A **table** physically stores data, but a **view** only represents it.
  Syntax:

```
CREATE VIEW view_name AS SELECT columns FROM table;
```

33. Explain the advantages of using views in SQL databases.

- Simplifies complex queries.
- Provides **security** (restricts access to certain columns/rows).
- Offers a consistent way of presenting data.
- Easier for users to work with instead of writing long queries

34. What is a trigger in SQL? Describe its types and when they are used.

- A **trigger** is a stored procedure that runs **automatically** when a specific event happens (INSERT, UPDATE, DELETE).

Types:

- **BEFORE Trigger** → runs before the event.
- **AFTER Trigger** → runs after the event

35. Explain the difference between `INSERT`, `UPDATE`, and `DELETE` triggers.

- **INSERT Trigger** → activates when a new row is added.
- **UPDATE Trigger** → activates when a row is modified.
- **DELETE Trigger** → activates when a row is removed.

They help in maintaining **data integrity, logging changes, and automation**

36. What is PL/SQL, and how does it extend SQL's capabilities?

PL/SQL (Procedural Language/SQL) is Oracle's extension to SQL.
It adds **programming features** like variables, loops, conditions, and error handling.
This allows us to create **procedures, functions, triggers, and packages** for complex database operations.
So, PL/SQL makes SQL more powerful by allowing **logic + SQL together**

37. List and explain the benefits of using PL/SQL.

- **Performance**: Batch processing reduces database calls.
- **Error handling**: Has strong exception-handling system.
- **Modularity**: Supports reusable procedures and functions.
- **Security**: Users can only access data through controlled procedures.
- **Portability**: Works smoothly with Oracle in different environments

38. What are control structures in PL/SQL? Explain the `IF-THEN` and `LOOP` control structures.

Control structures help in **decision-making and repetition**.

- **IF-THEN** → Runs a block if condition is true.
- `IF age >= 18 THEN`

```
  DBMS_OUTPUT.PUT_LINE('Adult');
END IF;
```

LOOP (WHILE/ FOR)

→ Repeats a block until a condition is false.

Example (WHILE):

```
WHILE counter <= 5 LOOP
  DBMS_OUTPUT.PUT_LINE(counter);
  counter := counter + 1;
END LOOP;
```

39. How do control structures in PL/SQL help in writing complex queries?

- Add **logic** (if conditions).

- Perform **repeated tasks** (loops).

- Handle **multiple scenarios** in one block.

This makes queries **flexible, dynamic, and capable of solving complex problems**

40. What is a cursor in PL/SQL? Explain the difference between implicit and explicit cursors.

A cursor is a **pointer** that retrieves query results **row by row**.

- **Implicit cursor**: Created automatically for single SQL statements (INSERT, UPDATE, SELECT INTO). No need to declare.
- **Explicit cursor**: Declared by programmer, must be opened, fetched, and closed manually. Used for queries returning multiple rows

41. When would you use an explicit cursor over an implicit one?

Use explicit cursor when:

- The query returns **multiple rows**.
- You need **step-by-step control** over result processing.
  Implicit is fine for simple, single-row queries

42. Explain the concept of `SAVEPOINT` in transaction management. How do `ROLLBACK` and `COMMIT` interact with savepoints?

- **SAVEPOINT** → Marks a point in a transaction.
- **ROLLBACK TO savepoint** → Undo only changes made after that point.
- **COMMIT** → Saves everything permanently, including changes after savepoint

43. When is it useful to use savepoints in a database transaction?

- A transaction has **multiple steps**, and you only want to undo part of it.
- For example: Insert records in three tables, rollback only the second insert if error occurs, but keep first and third.

This gives **finer control** during complex operations