Module-4 introduction to dbms

Introduction to sql

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

Ans->**What is SQL?**

- **SQL** stands for **Structured Query Language**. It is a standard programming language specifically designed for **managing and manipulating relational databases**. SQL allows users to perform a variety of operations such as querying data, updating records, deleting data, and creating or modifying database structures.
- **Key points about SQL:**
- SQL is **declarative**, meaning you specify *what* you want, not *how* to get it.
- It is supported by almost all relational database management systems (RDBMS) like **MySQL, Oracle, SQL Server, PostgreSQL**, etc.
- SQL provides commands for multiple purposes, broadly categorized as:
    - **Data Definition Language (DDL):** Commands like `CREATE`, `ALTER`, `DROP` to define or modify database structures.
    - **Data Manipulation Language (DML):** Commands like `SELECT`, `INSERT`, `UPDATE`, `DELETE` to handle data in tables.
    - **Data Control Language (DCL):** Commands like `GRANT`, `REVOKE` to manage permissions.
    - **Transaction Control Language (TCL):** Commands like `COMMIT`, `ROLLBACK`, `SAVEPOINT` to control transactions.

- **Why is SQL essential in database management?**
- SQL is fundamental in database management for several reasons:
- **Efficient Data Retrieval:**
  SQL allows users to quickly and efficiently query large amounts of data using commands like `SELECT` with conditions, sorting, and filtering.
- **Data Integrity and Accuracy:**
  SQL enforces **constraints** such as `PRIMARY KEY`, `FOREIGN KEY`, `UNIQUE`, and `NOT NULL` to maintain accurate and reliable data.
- **Data Manipulation:**
  With SQL, you can insert new records, update existing ones, or delete outdated data safely and consistently.
- **Database Structure Management:**
  SQL lets you create, alter, and drop tables and databases, giving you full control over the structure of your data storage.
- **Security Management:**
  SQL provides mechanisms to manage user access and permissions, ensuring that only authorized users can view or modify data.
- **Standardization and Compatibility:**
  Since SQL is a widely accepted standard, knowledge of SQL makes it easier to work with multiple database systems without learning a new language each time.

❖ **Supports Complex Operations:**
SQL can handle **joins, subqueries, aggregations, and transactions**, enabling complex data analysis and business operations.

2. Explain the difference between DBMS and RDBMS.

Ans->

Feature	DBMS (Database Management System)	RDBMS (Relational Database Management System)
Definition	Software to store, manage, and retrieve data.	A type of DBMS that stores data in tables and supports relationships.
Data Storage	Data may be stored as files (hierarchical, network, or flat files).	Data is stored in **tables** (rows & columns).
Data Relationships	Limited or no support for relationships between data.	Fully supports relationships via **primary keys** and **foreign keys**.
Normalization	Not supported; data redundancy may exist.	Supported; reduces redundancy and ensures consistency.
Query Language	Proprietary commands; SQL may not be supported.	Supports **SQL** for querying and managing data.
Multi-User Access	Usually single-user or limited multi-user support.	Designed for **multi-user access** with concurrency control.
Data Integrity	Weak; may need application-level enforcement.	Strong; enforces **ACID properties** (Atomicity, Consistency, Isolation, Durability).
Examples	Microsoft Access, IBM IMS, FileMaker	MySQL, Oracle, SQL Server, PostgreSQL

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

-> Definition of SQL

- SQL (Structured Query Language) is a standard programming language used to interact with relational databases. It allows users to create, manipulate, and manage data stored in tables, ensuring proper organization, retrieval, and modification of information.

➢ Roles of SQL in Managing Relational Databases

Role	Explanation
Data Definition	SQL provides commands to define the database structure, such as creating, altering, or deleting tables, indexes, and constraints. Example commands: `CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`.

Role	Explanation
Data Manipulation	SQL allows inserting, updating, deleting, and retrieving data from tables. This is done using commands like `INSERT`, `UPDATE`, `DELETE`, and `SELECT`.
Data Querying	SQL enables users to extract specific information using queries. It supports filtering (`WHERE`), sorting (`ORDER BY`), grouping (`GROUP BY`), and joining multiple tables (`JOIN`).
Data Integrity	SQL helps enforce rules and constraints to maintain data accuracy. Examples include `PRIMARY KEY`, `FOREIGN KEY`, `UNIQUE`, `CHECK`, and `NOT NULL`.
Transaction Management	SQL allows handling multiple operations as a single transaction. Commands like `COMMIT` and `ROLLBACK` ensure consistency and reliability in multi-user environments.
Access Control and Security	SQL provides commands to manage user access and permissions. `GRANT` and `REVOKE` allow defining who can read, modify, or delete data.
Data Analysis & Reporting	SQL supports aggregation (`SUM`, `COUNT`, `AVG`) and complex queries, which help in analyzing and generating reports from relational databases.

4.What are the key features of SQL?

Ans->

Feature	Description
1. Standardized Language	SQL is a standard language recognized by ISO and ANSI for managing relational databases, ensuring compatibility across different RDBMS systems.
2. Data Definition	SQL allows users to define the database structure, including creating, modifying, and deleting tables, schemas, and relationships using commands like `CREATE`, `ALTER`, and `DROP`.
3. Data Manipulation	SQL provides commands to insert, update, delete, and retrieve data in tables using commands such as `INSERT`, `UPDATE`, `DELETE`, and `SELECT`.
4. Data Retrieval (Querying)	SQL allows complex queries to extract specific data. Features include filtering (`WHERE`), sorting (`ORDER BY`), grouping (`GROUP BY`), and joining tables (`JOIN`).

Feature	Description
5. Transaction Management	SQL supports transactions, ensuring multiple operations are executed as a single unit. Commands like `COMMIT` and `ROLLBACK` help maintain consistency and reliability.
6. Data Integrity & Constraints	SQL enforces rules and constraints to maintain data accuracy, such as `PRIMARY KEY`, `FOREIGN KEY`, `UNIQUE`, `CHECK`, and `NOT NULL`.
7. Security & Access Control	SQL provides commands to control access to data. Users can be granted or revoked permissions using `GRANT` and `REVOKE`.
8. Data Aggregation & Analysis	SQL allows summary and analytical operations like `SUM`, `AVG`, `COUNT`, `MAX`, `MIN` for generating reports and insights.
9. Multi-User Support	SQL supports concurrent access by multiple users without compromising data consistency, using locking and isolation mechanisms.
Portability	SQL is platform-independent; the same SQL commands can be used across different RDBMS systems with minimal changes.

2.SQL Syntax

1.What are the basic components of SQL syntax?

Ans->

SQL Statements

SQL commands are written as **statements**, each performing a specific task in the database. A statement usually ends with a **semicolon (;)**.

Categories of SQL statements:

DDL (Data Definition Language): Defines database structure (`CREATE`, `ALTER`, `DROP`).

DML (Data Manipulation Language): Manipulates data (`INSERT`, `UPDATE`, `DELETE`).

DQL (Data Query Language): Retrieves data (`SELECT`).

DCL (Data Control Language): Manages permissions (`GRANT`, `REVOKE`).

TCL (Transaction Control Language): Handles transactions (`COMMIT`, `ROLLBACK`).

2. Keywords

Reserved words that define actions in SQL.

Examples: `SELECT`, `FROM`, `WHERE`, `INSERT`, `UPDATE`, `DELETE`.

Keywords are **not case-sensitive** in most RDBMS, but usually written in uppercase for clarity.

3. Clauses

Clauses are parts of SQL statements that perform specific functions.

Examples:

`SELECT` clause – specifies columns to retrieve.

`FROM` clause – specifies the table(s).

`WHERE` clause – filters rows based on conditions.

`ORDER BY` clause – sorts the result.

`GROUP BY` clause – groups rows for aggregation.

4. Expressions

Combines **operators, constants, and column names** to produce values.

Examples:

```
price * 0.9
age + 5
"John" || " Doe" (concatenation)
```

5. Predicates

Conditions that evaluate to **TRUE, FALSE, or UNKNOWN** in a `WHERE` clause.

Examples:

`=, <, >, <=, >=, <>`

`BETWEEN`, `LIKE`, `IN`, `IS NULL`

6. Operators

Used to perform **arithmetic, comparison, or logical operations**.

Arithmetic: +, −, *, /

Comparison: =, <>, <, >

Logical: `AND`, `OR`, `NOT`

7. Comments

Used to **explain SQL code** and are ignored by the database engine.

Single-line comment: `-- This is a comment`

Multi-line comment: `/* This is a comment */`

2.Write the general structure of an SQL SELECT statement.

Ans->

Purpose

- The `SELECT` statement is used to retrieve data from one or more tables in a relational database. It can also filter, sort, and group the data as needed.

General Syntax

```
SELECT column1, column2, ..., columnN
FROM table_name
[WHERE condition]
[GROUP BY column1, column2, ...]
[HAVING condition]
[ORDER BY column1 [ASC|DESC], column2 [ASC|DESC], ...]
[LIMIT number];
```

Explanation of Each Clause

Clause	Description
SELECT	Specifies the columns to retrieve. Use `*` to select all columns.
FROM	Specifies the table(s) from which to retrieve data.
WHERE	Filters rows based on a specified condition. Optional.
GROUP BY	Groups rows that have the same values in specified columns, often used with aggregate functions (`SUM`, `COUNT`, `AVG`). Optional.
HAVING	Filters groups after `GROUP BY` based on a condition. Optional.
ORDER BY	Sorts the result set by one or more columns in ascending (ASC) or descending (DESC) order. Optional.
LIMIT	Restricts the number of rows returned. Optional (syntax may vary in some databases like SQL Server using `TOP`).

Example

```
SELECT student_name, age, grade
FROM students
WHERE age >= 18
GROUP BY grade
HAVING COUNT(*) > 5
ORDER BY age DESC
LIMIT 10;
```

3. Explain the role of clauses in SQL statements.

What is a Clause in SQL?

A **clause** is a component of an SQL statement that performs a specific function. SQL statements are made up of one or more clauses, each helping to define **what data to retrieve, how to filter it, how to group it, or how to sort it**.

In short, **clauses give structure and meaning to an SQL statement**.

Clauses and Their Roles

Clause	Role / Purpose	Example
SELECT	Specifies the columns or expressions to retrieve from a table.	`SELECT student_name, age`
FROM	Specifies the table(s) from which to retrieve the data.	`FROM students`
WHERE	Filters rows based on a condition; only rows that satisfy the condition are returned.	`WHERE age >= 18`
GROUP BY	Groups rows that have the same values in one or more columns; often used with aggregate functions.	`GROUP BY grade`
HAVING	Filters groups after grouping with `GROUP BY`; used for conditions on aggregates.	`HAVING COUNT(*) > 5`
ORDER BY	Sorts the result set in ascending (ASC) or descending (DESC) order.	`ORDER BY age DESC`
LIMIT / TOP	Restricts the number of rows returned by the query.	`LIMIT 10`

How Clauses Work Together

Clauses work **hierarchically** in SQL:

FROM → Identify tables

WHERE → Filter rows

GROUP BY → Group filtered rows

HAVING → Filter groups

SELECT → Choose columns or expressions

LIMIT → Restrict the output

This order is **important** for understanding query execution, as it affects how data is processed.

3. SQL Constraints

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

Ans->

❖ What are Constraints in SQL?

- Constraints are rules applied to columns in a table to enforce data integrity and accuracy. They ensure that the data entered into the database follows specific rules, preventing invalid or inconsistent data.
- Constraints can be applied when creating a table using the `CREATE TABLE` statement or added later using `ALTER TABLE`.

2. Types of Constraints in SQL

Constraint	Description / Role	Example
PRIMARY KEY	Uniquely identifies each row in a table. A table can have only one primary key.	`student_id INT PRIMARY KEY`
FOREIGN KEY	Ensures referential integrity by linking a column in one table to the primary key of another table.	`FOREIGN KEY (class_id) REFERENCES classes(class_id)`
UNIQUE	Ensures that all values in a column are distinct.	`email VARCHAR(50) UNIQUE`
NOT NULL	Ensures that a column cannot have NULL values.	`name VARCHAR(50) NOT NULL`
CHECK	Ensures that values in a column satisfy a specific condition.	`CHECK (age >= 18)`
DEFAULT	Provides a default value for a column when no value is specified.	`status VARCHAR(10) DEFAULT 'Active'`

2. How do PRIMARY KEY and FOREIGN KEY constraints differ?

Ans->

1.Definition

Constraint	Definition
PRIMARY KEY	A column (or a set of columns) that uniquely identifies each row in a table. Each table can have only one primary key, and it cannot contain NULL values.
FOREIGN KEY	A column (or a set of columns) in one table that refers to the primary key in another table. It is used to enforce referential integrity between two tables.

---

2. Key Differences

Feature	PRIMARY KEY	FOREIGN KEY
Purpose	Uniquely identifies each record in a table.	Maintains a relationship between two tables.
Uniqueness	Must be unique for each row.	Values may repeat (not necessarily unique).
NULL Values	Cannot contain NULL values.	Can contain NULL values (unless specified NOT NULL).
Table Limit	Only one primary key per table.	A table can have multiple foreign keys.
Relationship Enforcement	Ensures row uniqueness within the table.	Ensures referential integrity with another table.
Example	`student_id INT PRIMARY KEY`	`class_id INT FOREIGN KEY REFERENCES classes(class_id)`

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

NOT NULL Constraint

Definition:

The `NOT NULL` constraint **ensures that a column cannot have NULL values**. Every row must have a valid (non-empty) value in this column.

Role / Purpose:

Ensures **mandatory data entry** for important fields.

Helps maintain **data completeness and integrity**.

Often used for columns that are critical to a table, like `id`, `name`, or `email`.

Example:

```
CREATE TABLE students (
student_id INT NOT NULL,
```

```
student_name VARCHAR(50) NOT NULL,
age INT
);
```

Explanation: `student_id` and `student_name` **cannot be left empty**, while `age` can be NULL.

2. UNIQUE Constraint

Definition:
The `UNIQUE` constraint **ensures that all values in a column are distinct**. No two rows can have the same value in that column.

Role / Purpose:

Prevents **duplicate data** in important fields like email, username, or product codes.

Supports **data integrity** by enforcing uniqueness, though NULL values are usually allowed unless combined with NOT NULL.

Example:

```
CREATE TABLE students (
student_id INT NOT NULL,
email VARCHAR(50) UNIQUE,
student_name VARCHAR(50)
);
```

4. Main SQL Commands and Sub-commands (DDL)

1.Define the SQL Data Definition Language (DDL)

Ans->

Definition of DDL

- **Data Definition Language (DDL)** is a subset of SQL used to **define, modify, or remove database structures** such as **tables, indexes, schemas, and constraints**.
- DDL commands do **not manipulate the data itself** (that's DML's job); instead, they define the **structure and schema** of the database.

In short, **DDL is used to create and manage the "skeleton" of the database**.

2. Key Features of DDL

- **Defines database objects:** tables, indexes, views, schemas.
- **Controls structure and constraints:** like PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL.

- **Changes are permanent:** DDL commands are automatically **committed**, so the changes cannot be rolled back in most RDBMS.
- **Supports database management:** allows creating, altering, and dropping objects easily.

3. Common DDL Commands

Command	Purpose	Example
CREATE	Creates a new database object (table, index, schema).	`CREATE TABLE students (id INT PRIMARY KEY, name VARCHAR(50));`
ALTER	Modifies an existing database object (add, modify, or delete columns or constraints).	`ALTER TABLE students ADD COLUMN age INT;`
DROP	Deletes an existing database object permanently.	`DROP TABLE students;`
TRUNCATE	Deletes all rows in a table **without removing the table structure**.	`TRUNCATE TABLE students;`
RENAME	Renames a database object (table or column).	`ALTER TABLE students RENAME TO learners;`

2.Explain the CREATE command and its syntax ?

Ans->

Definition of CREATE Command

The **CREATE** command in SQL is part of the **Data Definition Language (DDL)**. It is used to **create new database objects** such as:

Databases

Tables

Views

Indexes

Schemas

The CREATE command **defines the structure and constraints** of the object at the time of creation.

2. Syntax of CREATE Command

a) Creating a Database

CREATE `DATABASE database_name;`

Example:

CREATE `DATABASE SchoolDB;`

Explanation: Creates a new database named `SchoolDB`.

b) Creating a Table

```
CREATE TABLE table_name (
    column1 datatype [constraint],
    column2 datatype [constraint],
    ...,
    columnN datatype [constraint]
);
```

Example:

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    student_name VARCHAR(50) NOT NULL,
    age INT,
    email VARCHAR(50) UNIQUE
);
```

Explanation:

Creates a table `Students` with columns: `student_id`, `student_name`, `age`, and `email`.

Adds **constraints**: `PRIMARY KEY`, `NOT NULL`, and `UNIQUE`.

c) Creating a View

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Example:

```
CREATE VIEW AdultStudents AS
SELECT student_name, age
FROM Students
WHERE age >= 18;
```

Explanation: Creates a **virtual table** `AdultStudents` showing only students who are 18 or older.

d) Creating an Index

```
CREATE INDEX index_name
ON table_name (column_name);
```

Example:

```
CREATE INDEX idx_student_name
ON Students(student_name);
```

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

Specifying Data Types

Definition:
A **data type** defines the **kind of data** that a column can store, such as numbers, text, dates, or boolean values.

Purpose / Role:

Data Integrity: Ensures that only valid data of the correct type is stored in a column.

Example: `INT` column cannot store text.

Efficient Storage: Helps the database optimize **storage space** based on the type of data.

Example: Using `VARCHAR(50)` instead of `TEXT` for short strings saves space.

Data Validation: Automatically enforces basic type rules.

Example: You cannot insert letters into a column defined as `INT`.

Query Optimization: Knowing the data type helps the database **optimize searches, indexing, and sorting**.

Example:

```
student_id INT,
student_name VARCHAR(50),
dob DATE
```

2. Specifying Constraints

Definition:
A **constraint** is a rule applied to a column or table to enforce **data integrity, uniqueness, and consistency**.

Purpose / Role:

Maintain Data Accuracy: Prevents invalid or inconsistent data from being entered.

Example: `CHECK (age >= 0)` ensures no negative ages.

Enforce Uniqueness: Ensures that certain columns have unique values.

Example: `UNIQUE` or `PRIMARY KEY`.

Ensure Mandatory Fields: Forces certain columns to always have a value.

Example: `NOT NULL` ensures that essential fields are filled.

Maintain Relationships: Helps in **referential integrity** between tables.

Example: `FOREIGN KEY` ensures a valid link to another table.

Example:

```
student_id INT PRIMARY KEY,
email VARCHAR(50) UNIQUE,
student_name VARCHAR(50) NOT NULL
```

3. Summary

Data types define **what kind of data** a column can hold.

Constraints define **rules that data must follow**.

Together, they **ensure data integrity, accuracy, consistency, and efficient storage** in a relational database.

5. ALTER Command

1.What is the use of the ALTER command in SQL?

Ans->

Primary Uses of ALTER Command

Add a new column
You can add a new column to an existing table.

```
ALTER TABLE table_name
ADD column_name datatype [constraint];
```

Example:

```
ALTER TABLE Students
ADD email VARCHAR(50);
```

Modify an existing column
You can change the data type, size, or constraints of a column.

```
ALTER TABLE table_name
MODIFY column_name new_datatype [constraint];
```

Example:

```
ALTER TABLE Students
MODIFY phone_number VARCHAR(15);
```

Drop (delete) a column
You can remove a column from a table if it's no longer needed.

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

Example:

```
ALTER TABLE Students
DROP COLUMN email;
```

Rename a table or column (depending on SQL variant)

Rename table:

```
ALTER TABLE old_table_name
RENAME TO new_table_name;
```

Rename column (syntax may vary by SQL database):

```
ALTER TABLE table_name
RENAME COLUMN old_column_name TO new_column_name;
```

Add or drop constraints

Add a primary key or foreign key constraint:

```
ALTER TABLE table_name
ADD CONSTRAINT pk_name PRIMARY KEY (column_name);
```

Drop a constraint:

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

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

Ans->

Overview of ALTER TABLE

The `ALTER TABLE` command in SQL is part of **Data Definition Language (DDL)**.
It is used to **modify the structure of an existing table** without deleting its data.

Common operations include:

Adding a new column

Modifying an existing column

Dropping (deleting) a column

2. Adding a Column

Syntax:

```
ALTER TABLE table_name
ADD column_name data_type [constraint];
```

Example:

```
ALTER TABLE Students
ADD phone_number VARCHAR(15);
```

Explanation: Adds a new column `phone_number` of type `VARCHAR(15)` to the `Students` table.

3. Modifying a Column

Syntax (general):

```
ALTER TABLE table_name
MODIFY column_name new_data_type [constraint];
```

Example (MySQL):

```
ALTER TABLE Students
MODIFY student_name VARCHAR(100) NOT NULL;
```

Explanation: Changes the `student_name` column to **increase its length to 100 characters** and makes it **NOT NULL**.

Note: Some RDBMS (like SQL Server) use `ALTER COLUMN` instead of `MODIFY`:

```
ALTER TABLE Students
ALTER COLUMN student_name VARCHAR(100) NOT NULL;
```

4. Dropping a Column

Syntax:

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

Example:

```
ALTER TABLE Students
DROP COLUMN phone_number;
```

6. DROP Command

1.What is the function of the DROP command in SQL?

Ans->Definition of DROP Command

The **DROP** command in SQL is a **Data Definition Language (DDL)** command used to **permanently delete database objects** such as:

Tables

Databases

Views

Indexes

Schemas

Important: Once executed, the object and all its data are **permanently removed** and cannot be recovered unless a backup exists.

2. Purpose / Function of DROP

Remove Database Objects: Deletes the table, database, or other object from the system completely.

Free Up Space: Removes unused or obsolete objects, freeing storage in the database.

Reset Structures: Allows dropping objects before recreating them with a new structure or definition.

3. Syntax of DROP Command

a) Dropping a Table

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE Students;
```

Explanation: Permanently deletes the `Students` table and all its data.

b) Dropping a Database

```
DROP DATABASE database_name;
```

Example:

```
DROP DATABASE SchoolDB;
```

Explanation: Deletes the `SchoolDB` database along with all its tables and data.

c) Dropping a View

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW AdultStudents;
```

Explanation: Removes the view `AdultStudents` from the database.

d) Dropping an Index

DROP `INDEX index_name` ON `table_name;`

Example:

DROP `INDEX idx_student_name` ON `Students;`

Explanation: Deletes the index `idx_student_name` from the `Students` table.

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

Ans->

Dropping a table in SQL is a **permanent action** that removes the table and all its data from the database. It's done using the **`DROP TABLE`** command:

DROP TABLE `table_name;`

Here's a detailed look at the **implications** of dropping a table:

1. Complete Data Loss

When you drop a table, **all rows in the table are permanently deleted**.

Unlike `DELETE`, which can remove data row by row, `DROP TABLE` removes the entire table **including its data**.

This action **cannot be undone** (unless you have a backup).

Example:

DROP TABLE `Students;`

The table `Students` and all its records are gone permanently.

2. Removal of Table Structure

Dropping a table deletes the **table schema**, including:

Column definitions

Data types

Constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK)

Indexes

After dropping, the table no longer exists in the database.

3. Impact on Dependent Objects

If other tables have **foreign key references** to this table, dropping it can cause:

Errors (in databases enforcing referential integrity)

Or automatic cascading deletion if `ON DELETE CASCADE` is defined.

Views, triggers, or stored procedures depending on the table may **fail** or become invalid.

4. Loss of Indexes and Triggers

All **indexes**, **triggers**, and **constraints** associated with the table are automatically removed.

5. Permissions and Grants

Any user permissions granted on the table are lost after dropping it.