

Module 4 – Introduction to DBMS

1. Introduction to SQL

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

- Ans:**
1. **Data Manipulation** – Provides commands like SELECT, INSERT, UPDATE, and DELETE to manage records in databases.
 2. **Data Definition** – Defines the structure of databases using commands like CREATE, ALTER, and DROP.
 3. **Data Control** – Manages access and security with commands like GRANT and REVOKE.
 4. **Standardization** – SQL is widely used across almost all relational database systems (MySQL, Oracle, SQL Server, PostgreSQL, etc.).
 5. **Data Integrity** – Ensures accuracy and consistency of data using constraints (e.g., PRIMARY KEY, FOREIGN KEY).
 6. **Efficient Querying** – Allows filtering, sorting, and aggregating data quickly, even from large datasets.
 7. **Integration** – Easily integrates with other programming languages (like Java, Python, C#) for application development.

2. Explain the difference between DBMS and RDBMS.

Ans: DBMS (Database Management System)

- A software system that allows users to create, store, retrieve, and manage data in databases.
- Data is usually stored in files without strict relationships.
- Example: Microsoft Access, FoxPro

Features:

- Data is stored as files (hierarchical, network, or non-relational models).
- Relationships between data are not well-defined.
- Does not enforce data integrity strongly.
- Suitable for small-scale applications.

RDBMS (Relational Database Management System)

- An advanced type of DBMS that stores data in the form of tables (rows and columns).

- Uses SQL for defining and managing data.
- Example: MySQL, Oracle, SQL Server, PostgreSQL

Features:

- Data is stored in tables with relationships (primary key, foreign key).
- Enforces data integrity through constraints.
- Supports normalization to reduce redundancy.
- Suitable for large-scale, complex applications.

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

Ans: Role of SQL in Managing Relational Databases

SQL (Structured Query Language) is the **standard language** used to **interact with and manage relational databases (RDBMS)**. Since relational databases store data in **tables (rows and columns)**, SQL provides all the necessary tools to work with that data.

1. Data Definition (DDL – Data Definition Language)

SQL helps define the structure of a database.

- Create and modify tables, schemas, and relationships.
- Example:

```
CREATE TABLE Students (
```

```
    StudentID INT PRIMARY KEY,
```

```
    Name VARCHAR(50),
```

```
    Age INT,
```

```
    Course VARCHAR(50)
```

```
);
```

2. Data Manipulation (DML – Data Manipulation Language)

SQL allows inserting, updating, deleting, and retrieving records.

- Example:

```
INSERT INTO Students VALUES (1, 'Rahul', 20, 'Computer Science');
```

```
UPDATE Students SET Age = 21 WHERE StudentID = 1;
```

```
DELETE FROM Students WHERE StudentID = 1;
```

3. Data Querying (Retrieval)

SQL makes it easy to search, filter, and analyze data using queries.

- Example:

```
SELECT Name, Course  
FROM Students  
WHERE Age > 18;
```

4. Data Control (DCL – Data Control Language)

Manages **permissions** and **access control** to keep data secure.

- Example:

```
GRANT SELECT ON Students TO User1;  
REVOKE UPDATE ON Students FROM User1;
```

5. Transaction Control (TCL – Transaction Control Language)

Ensures **data integrity** during multiple operations.

- Example:

```
BEGIN TRANSACTION;  
UPDATE Students SET Course = 'AI' WHERE StudentID = 2;  
COMMIT; -- saves changes permanently  
ROLLBACK; -- undo if error occurs
```

6. Maintaining Relationships

SQL enforces **primary keys**, **foreign keys**, and **constraints** to maintain **relationships between tables**.

- Example:

```
ALTER TABLE Students  
ADD CONSTRAINT fk_course  
FOREIGN KEY (Course) REFERENCES Courses(CourseName);
```

4. What are the key features of SQL?

Ans: Key Features of SQL

1. Data Definition
 - SQL lets you define the structure of databases (tables, schemas, relationships).
 - Commands: CREATE, ALTER, DROP.
2. Data Manipulation
 - You can add, modify, delete, and retrieve data.
 - Commands: INSERT, UPDATE, DELETE, SELECT.
3. Data Querying
 - SQL provides powerful querying capabilities to filter, sort, and aggregate data.
 - Example: SELECT Name, Age FROM Students WHERE Age > 18;
4. Data Control (Security)
 - SQL allows controlling user access and permissions.
 - Commands: GRANT, REVOKE.
5. Transaction Control
 - Ensures data consistency and reliability during multiple operations.
 - Commands: COMMIT, ROLLBACK, SAVEPOINT.
6. Relational Database Support
 - SQL works with tables, primary keys, and foreign keys to maintain relationships.
7. Data Integrity
 - SQL enforces constraints like PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, and CHECK to maintain accuracy.
8. Standardized Language
 - SQL is a universal standard (ANSI/ISO) supported by most RDBMS (MySQL, Oracle, PostgreSQL, SQL Server).
9. Scalability & Portability
 - Can handle small-scale (single-user) databases as well as large enterprise-level databases.
 - SQL queries can be used across multiple systems with minimal changes.
10. Interactive & User-Friendly
 - SQL is easy to learn and use; it is English-like, making it beginner-friendly.

2. SQL Syntax

1. What are the basic components of SQL syntax?

Ans: Basic Components of SQL Syntax

SQL syntax is made up of keywords, clauses, expressions, and statements. The main components are:

1. Keywords (Commands)

- Reserved words in SQL that perform specific actions.
- Example: SELECT, INSERT, UPDATE, DELETE, CREATE, DROP.

2. Clauses

- Part of SQL statements that define conditions or limits.
- Example:
 - WHERE → filter records
 - ORDER BY → sort results
 - GROUP BY → group records

3. Expressions

- Produce values using columns, constants, and operators.
- Example: salary * 0.1 (to calculate bonus).

4. Predicates

- Conditions that evaluate to TRUE, FALSE, or UNKNOWN.
- Example: age > 18, name LIKE 'A%'.

5. Statements

- Complete SQL instructions made up of keywords + clauses.
- Example:
 - SELECT Name, Age
 - FROM Students
 - WHERE Age > 18
 - ORDER BY Age;

6. Identifiers

- Names of database objects like tables, columns, views, etc.
- Example: Students, EmployeeID.

7. Operators

- Symbols or keywords used in conditions and calculations.
- Example: =, <, >, AND, OR, LIKE.

2. Write the general structure of an SQL SELECT statement

Ans: General Structure of SQL SELECT Statement

SELECT column1, column2, ...

FROM table_name

WHERE condition

GROUP BY column

HAVING condition

ORDER BY column [ASC | DESC];

Explanation of Each Clause

1. SELECT → Specifies which columns (or * for all columns) to display.
 2. FROM → Specifies the table(s) to retrieve data from.
 3. WHERE → Filters rows based on a condition.
 4. GROUP BY → Groups rows that have the same values in specified columns.
 5. HAVING → Applies conditions to groups (like WHERE but for groups).
 6. ORDER BY → Sorts the result (ascending ASC by default, or descending DESC).
-

Example

SELECT Name, Course, COUNT(*) AS StudentsCount

FROM Students

WHERE Age > 18

GROUP BY Course

HAVING COUNT(*) > 5

ORDER BY StudentsCount DESC;

3. Explain the role of clauses in SQL statements.

Ans: Role of Clauses in SQL Statements

In SQL, clauses are building blocks of statements.

They define conditions, filtering, grouping, or sorting rules that control how data is selected, updated, or deleted.

Think of clauses as the “rules or conditions” inside an SQL command.

Common SQL Clauses and Their Roles

1. WHERE Clause

- Filters records based on conditions.
- Example:
- `SELECT * FROM Students WHERE Age > 18;`
- Role: Restricts rows returned.

2. ORDER BY Clause

- Sorts query results in ascending (ASC) or descending (DESC) order.
- Example:
- `SELECT Name, Age FROM Students ORDER BY Age DESC;`
- Role: Controls sorting.

3. GROUP BY Clause

- Groups rows that have the same values in specific columns.
- Example:
- `SELECT Course, COUNT(*)`
- `FROM Students`
- `GROUP BY Course;`
- Role: Used with aggregate functions (SUM, COUNT, AVG, etc.).

4. HAVING Clause

- Filters groups (like WHERE but applied after grouping).
- Example:
- `SELECT Course, COUNT(*)`
- `FROM Students`
- `GROUP BY Course`
- `HAVING COUNT(*) > 5;`
- Role: Applies conditions on aggregated data.

5. FROM Clause

- Defines the table(s) from which to retrieve data.
- Example:
- SELECT Name FROM Students;
- Role: Specifies data source.

6. JOIN Clause

- Combines rows from two or more tables based on related columns.
- Example:
- SELECT Students.Name, Courses.CourseName
- FROM Students
- JOIN Courses ON Students.CourseID = Courses.ID;
- Role: Retrieves data from multiple tables.

3. SQL Constraints

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

Ans: What are Constraints in SQL?

In SQL, constraints are rules applied to table columns to ensure the accuracy, integrity, and reliability of the data stored in a database.

They restrict the type of data that can be inserted into a column and prevent invalid data from entering the database.

👉 Think of them as "conditions" or "checks" placed on data.

Types of Constraints in SQL

1. NOT NULL
 - Ensures that a column cannot store NULL values.
 - Example: A student's roll number should always be provided.
2. CREATE TABLE Students (
3. RollNo INT NOT NULL,

4. Name VARCHAR(50)
5.);
6. UNIQUE
 - Ensures that all values in a column are unique (no duplicates).
 - Example: Email IDs of users must be unique.
7. CREATE TABLE Users (
8. UserID INT UNIQUE,
9. Email VARCHAR(100) UNIQUE
10.);
11. PRIMARY KEY
 - A combination of NOT NULL + UNIQUE.
 - Uniquely identifies each row in a table.
 - Only one primary key is allowed per table.
12. CREATE TABLE Employees (
13. EmpID INT PRIMARY KEY,
14. Name VARCHAR(50)
15.);
16. FOREIGN KEY
 - Establishes a relationship between two tables.
 - The foreign key in one table refers to the primary key in another table.
 - Ensures referential integrity.
17. CREATE TABLE Orders (
18. OrderID INT PRIMARY KEY,
19. CustomerID INT,
20. FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
21.);
22. CHECK
 - Ensures that values in a column meet a specific condition.
 - Example: Age should always be greater than 18.
23. CREATE TABLE Persons (
24. PersonID INT PRIMARY KEY,

```

25.    Age INT CHECK (Age >= 18)
26. );
27. DEFAULT
      ○ Provides a default value for a column if no value is specified.
      ○ Example: If no country is entered, default should be "India".
28. CREATE TABLE Customers (
29.    CustomerID INT PRIMARY KEY,
30.    Country VARCHAR(50) DEFAULT 'India'
);

```

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

Ans: Difference between PRIMARY KEY and FOREIGN KEY

Feature	PRIMARY KEY	FOREIGN KEY
Definition	A column (or set of columns) that uniquely identifies each row in a table.	A column (or set of columns) in one table that refers to the PRIMARY KEY of another table.
Uniqueness	Must contain unique values (no duplicates).	Can contain duplicate values (many rows can reference the same key).
NULL Values	Cannot contain NULL (NOT NULL is part of it).	Can contain NULL (if allowed).
Purpose	Ensures entity integrity (each record is unique in a table).	Ensures referential integrity (valid link between two tables).
Number per Table	Only one PRIMARY KEY is allowed per table.	A table can have multiple FOREIGN KEYS.
Location	Defined in the same table where uniqueness is required.	Defined in a different table, pointing to the primary key of another table.
Relationship	Defines a record uniquely within a table.	Establishes a relationship between two tables.

Example

-- Parent Table

```
CREATE TABLE Customers (
```

```
CustomerID INT PRIMARY KEY,
```

```

    Name VARCHAR(50) NOT NULL
);

-- Child Table

CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

```

Explanation:

- CustomerID in Customers is a PRIMARY KEY → uniquely identifies each customer.
- CustomerID in Orders is a FOREIGN KEY → ensures that every order is linked to a valid customer.

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

Ans: NOT NULL Constraint

- Role: Ensures that a column cannot have NULL (empty) values.
- Used when a field is mandatory and must always have data.

Example:

```

CREATE TABLE Students (
    RollNo INT NOT NULL,
    Name VARCHAR(50) NOT NULL
);

```

- Here, both RollNo and Name must have values (no blank entries).

UNIQUE Constraint

- Role: Ensures that all values in a column are different (no duplicates).
- Used when a field should not repeat (like email IDs, usernames, etc.).

Example:

```

CREATE TABLE Users (
    UserID INT UNIQUE,

```

```
Email VARCHAR(100) UNIQUE  
);
```

- No two users can have the same UserID or the same Email.
-

Key Difference Between NOT NULL and UNIQUE

Feature	NOT NULL	UNIQUE
Purpose	Prevents missing values	Prevents duplicate values
NULL Allowed?	Not allowed	Allowed (but only once, since multiple NULLs are not equal in SQL)
Guarantee	Every row has a value	Every row has a different value

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

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

Ans: SQL Data Definition Language (DDL)

Definition:

- DDL stands for Data Definition Language in SQL.
 - It consists of commands used to define, modify, and manage the structure of database objects such as tables, schemas, views, indexes, and constraints.
 - In short → DDL deals with how data is stored, not the data itself.
-

Key Features of DDL

- Creates and modifies database schema (structure).
 - Defines relationships between tables.
 - Controls how data is stored and accessed.
 - Changes made using DDL commands are usually auto-committed (permanent).
-

Main DDL Commands

1. CREATE → Used to create new database objects (table, database, view, index).

2. CREATE TABLE Students (
3. RollNo INT PRIMARY KEY,
4. Name VARCHAR(50),
5. Age INT
6.);
7. ALTER → Used to modify an existing database object (e.g., add/remove a column).
8. ALTER TABLE Students ADD Email VARCHAR(100);
9. DROP → Deletes a database object permanently.
10. DROP TABLE Students;
11. TRUNCATE → Removes all rows from a table but keeps the structure for future use.
12. TRUNCATE TABLE Students;
13. RENAME → Renames an existing database object.

RENAME TABLE Students TO Learners;

2. Explain the CREATE command and its syntax.

Ans: CREATE Command

- The CREATE command in SQL is a part of DDL (Data Definition Language).
- It is used to create new database objects such as:
 - Databases
 - Tables
 - Views
 - Indexes
 - Schemas

Most commonly, we use it to create tables.

General Syntax

```
CREATE TABLE table_name (
  column1 datatype [constraint],
  column2 datatype [constraint],
  column3 datatype [constraint],
```

```
...  
);
```

Explanation:

- `table_name` → Name of the table you want to create.
 - `column1, column2, ...` → Names of the columns.
 - `datatype` → Type of data allowed in that column (e.g., INT, VARCHAR, DATE).
 - `constraint` → Rules applied on the column (e.g., PRIMARY KEY, NOT NULL, UNIQUE).
-

Example 1: Creating a Table

```
CREATE TABLE Students (  
    StudentID INT PRIMARY KEY,  
    Name VARCHAR(50) NOT NULL,  
    Age INT CHECK (Age >= 18),  
    Email VARCHAR(100) UNIQUE,  
    Country VARCHAR(50) DEFAULT 'India'  
);
```

What this does:

- `StudentID` → primary key (unique + not null).
 - `Name` → cannot be NULL.
 - `Age` → must be at least 18.
 - `Email` → must be unique.
 - `Country` → if not provided, defaults to "India".
-

Example 2: Creating a Database

```
CREATE DATABASE SchoolDB;
```

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

Ans: Purpose of Specifying Data Types

When creating a table, assigning a data type to each column ensures:

1. Defines the kind of data → e.g., INT for numbers, VARCHAR for text, DATE for dates.

2. Memory/storage efficiency → Database allocates just the right amount of space.
 - Example: CHAR(5) always uses 5 bytes, INT uses 4 bytes.
3. Data consistency → Prevents invalid data from being inserted.
 - Example: You can't insert "abc" into an INT column.
4. Performance optimization → Helps database engines work faster (better indexing, searching, sorting).

Example:

Age INT,

Name VARCHAR(50),

JoinDate DATE

- Age → only numbers allowed.
 - Name → up to 50 characters.
 - JoinDate → only valid dates allowed.
-

Purpose of Specifying Constraints

Constraints add rules on top of data types to maintain accuracy and integrity of data.

1. Ensure Valid Data Entry
 - Example: NOT NULL makes sure a required field isn't left empty.
2. Maintain Uniqueness
 - Example: UNIQUE or PRIMARY KEY prevents duplicate IDs.
3. Protect Relationships
 - Example: FOREIGN KEY ensures that orders are linked to valid customers.
4. Control Values
 - Example: CHECK (Age >= 18) ensures only eligible ages are stored.
5. Provide Defaults
 - Example: DEFAULT 'India' fills in a value automatically if nothing is provided.

Example with constraints:

CREATE TABLE Employees (

EmpID INT PRIMARY KEY,

Name VARCHAR(50) NOT NULL,

Age INT CHECK (Age >= 18),

```
Email VARCHAR(100) UNIQUE,  
Country VARCHAR(50) DEFAULT 'India'  
);
```

5. ALTER Command

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

Ans: ALTER Command in SQL

Definition:

- The ALTER command in SQL is a part of DDL (Data Definition Language).
 - It is used to modify the structure of an existing database object, most commonly a table, without losing the data stored in it.
-

Uses of ALTER Command

1. Add a new column to an existing table.
 2. Modify an existing column (change data type, size, or constraints).
 3. Drop (delete) a column from a table.
 4. Rename a table or a column.
 5. Add or drop constraints (like PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK).
-

Syntax Examples

1. Add a Column

ALTER TABLE Employees

ADD Salary DECIMAL(10,2);

- Adds a new column Salary to the Employees table.

2. Modify a Column

ALTER TABLE Employees

MODIFY Name VARCHAR(100) NOT NULL;

- Changes the size of Name to 100 characters and makes it NOT NULL.

3. Drop a Column

```
ALTER TABLE Employees
```

```
DROP COLUMN Salary;
```

- Removes the Salary column from the table.

4. Rename a Table

```
ALTER TABLE Employees
```

```
RENAME TO Staff;
```

- Renames the table from Employees to Staff.

5. Add a Constraint

```
ALTER TABLE Employees
```

```
ADD CONSTRAINT PK_EmpID PRIMARY KEY (EmpID);
```

- Adds a primary key constraint on the EmpID column.

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

Ans: 1. Add a Column

- Use ALTER TABLE ... ADD to add a new column to an existing table.
- You can also specify data type and constraints for the new column.

Syntax:

```
ALTER TABLE table_name
```

```
ADD column_name datatype [constraint];
```

Example:

```
ALTER TABLE Employees
```

```
ADD Salary DECIMAL(10,2);
```

- Adds a Salary column with decimal values.

2. Modify a Column

- Use ALTER TABLE ... MODIFY (in MySQL) or ALTER TABLE ... ALTER COLUMN (in SQL Server/PostgreSQL) to change the data type, size, or constraints of a column.

Syntax (MySQL):

```
ALTER TABLE table_name
```

```
MODIFY column_name new_datatype [constraint];
```

Example:

```
ALTER TABLE Employees
```

```
MODIFY Name VARCHAR(100) NOT NULL;
```

- Changes the Name column to 100 characters and makes it NOT NULL.

Syntax (SQL Server/PostgreSQL):

```
ALTER TABLE table_name
```

```
ALTER COLUMN column_name TYPE new_datatype;
```

3. Drop a Column

- Use ALTER TABLE ... DROP COLUMN to remove a column from the table.

Syntax:

```
ALTER TABLE table_name
```

```
DROP COLUMN column_name;
```

Example:

```
ALTER TABLE Employees
```

```
DROP COLUMN Salary;
```

- Deletes the Salary column from Employees.

6. DROP Command

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

Ans: DROP Command in SQL

Definition:

- The DROP command is a part of DDL (Data Definition Language).
 - It is used to delete database objects permanently from the database.
 - Unlike DELETE or TRUNCATE, DROP removes the object itself, not just the data inside it.
-

Function / Purpose

1. Remove a table along with all its data and structure.
2. Delete a database completely.

3. Remove other objects like views, indexes, or constraints.
4. Frees up space and cleans up unused objects from the database.

Caution: Once a table or database is dropped, it cannot be recovered unless you have a backup.

Syntax

1. Drop a Table

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE Employees;
```

- Deletes the Employees table and all its data permanently.

2. Drop a Database

```
DROP DATABASE database_name;
```

Example:

```
DROP DATABASE SchoolDB;
```

- Deletes the SchoolDB database entirely.

3. Drop a View

```
DROP VIEW view_name;
```

Example:

```
DROP VIEW StudentView;
```

- Deletes the view StudentView.

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

Ans: Implications of Dropping a Table

When you use the DROP TABLE command, the table is permanently deleted from the database. Here's what happens:

1. All Data is Lost

- Every row stored in the table is completely deleted.
- Cannot be recovered unless you have a backup.

2. Table Structure is Removed

- Column definitions, data types, and constraints are deleted.
- You cannot insert new data because the table no longer exists.

3. Indexes and Constraints are Deleted
 - o Any PRIMARY KEY, FOREIGN KEY, UNIQUE, or CHECK constraints on the table are removed.
 - o Indexes created on the table are also deleted.
 4. Dependent Objects May Break
 - o Views, triggers, stored procedures, or foreign key references that depend on the table may become invalid or cause errors.
 5. Cannot Be Rolled Back
 - o Unlike DELETE or TRUNCATE, DROP TABLE is usually irreversible.
 - o You can only restore it using a database backup.
 6. Frees Up Space
 - o Disk space used by the table is released to the database system.
-

Example

```
DROP TABLE Employees;
```

Consequences:

- All employee data is gone.
- Table structure, indexes, and constraints are gone.
- Any queries or views using Employees will fail.

7. Data Manipulation Language (DML)

1. Define the INSERT, UPDATE, and DELETE commands in SQL

Ans: 1. INSERT Command

Definition:

- The INSERT command is used to add new rows of data into a table.

Syntax (Basic):

```
INSERT INTO table_name (column1, column2, ...)
```

```
VALUES (value1, value2, ...);
```

Example:

```
INSERT INTO Students (StudentID, Name, Age)
```

```
VALUES (1, 'Shlok', 20);
```

- Adds a new student record with ID 1, name "Shlok", and age 20.
-

2. UPDATE Command

Definition:

- The UPDATE command is used to modify existing data in a table.
- Usually combined with a WHERE clause to update specific rows.

Syntax:

```
UPDATE table_name
```

```
SET column1 = value1, column2 = value2, ...
```

```
WHERE condition;
```

Example:

```
UPDATE Students
```

```
SET Age = 21
```

```
WHERE StudentID = 1;
```

- Updates the age of the student with ID 1 to 21.

Without a WHERE clause, all rows in the table will be updated.

3. DELETE Command

Definition:

- The DELETE command is used to remove rows from a table.
- Can delete specific rows using WHERE, or all rows if WHERE is omitted.

Syntax:

```
DELETE FROM table_name
```

```
WHERE condition;
```

Example:

```
DELETE FROM Students
```

```
WHERE StudentID = 1;
```

- Deletes the student record with ID 1.

Without a WHERE clause, all records in the table will be deleted.

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

Ans: Importance of the WHERE Clause

1. Targets Specific Rows
 - WHERE lets you specify which rows should be updated or deleted.
 - Without it, the operation affects all rows in the table.
 2. Prevents Accidental Data Loss
 - Using UPDATE or DELETE without WHERE can change or remove all records, which is often disastrous.
 3. Controls Data Accuracy
 - Ensures only the intended data is modified or removed, keeping the database consistent and reliable.
-

Examples

UPDATE with WHERE

UPDATE Students

SET Age = 21

WHERE StudentID = 1;

- Only updates the age of the student with StudentID = 1.
- Safe and targeted update.

Without WHERE:

UPDATE Students

SET Age = 21;

- Updates every student in the table.
 - Often unintended.
-

DELETE with WHERE

DELETE FROM Students

WHERE Age < 18;

- Deletes only students younger than 18.

Without WHERE:

```
DELETE FROM Students;
```

- Deletes all students in the table.
- Very dangerous if done by mistake.

8. Data Query Language (DQL)

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

Ans: SELECT Statement in SQL

Definition:

- The SELECT statement is used to retrieve data from one or more tables in a database.
 - It is the most commonly used SQL command for querying and analyzing data.
-

Basic Syntax

```
SELECT column1, column2, ...
```

```
FROM table_name
```

```
WHERE condition
```

```
ORDER BY column_name
```

```
LIMIT number;
```

Explanation of Clauses:

- SELECT → Specifies the column(s) you want to retrieve.
 - FROM → Specifies the table from which to retrieve data.
 - WHERE → Filters rows based on a condition (optional).
 - ORDER BY → Sorts the result set (optional).
 - LIMIT → Limits the number of rows returned (optional).
-

Examples

1. Select All Columns

```
SELECT *
```

```
FROM Students;
```

- Retrieves all columns for every row in the Students table.

2. Select Specific Columns

```
SELECT Name, Age
```

```
FROM Students;
```

- Retrieves only the Name and Age columns.

3. Select with WHERE Clause

```
SELECT Name, Age
```

```
FROM Students
```

```
WHERE Age >= 18;
```

- Retrieves students whose age is 18 or older.

4. Select with ORDER BY

```
SELECT Name, Age
```

```
FROM Students
```

```
ORDER BY Age DESC;
```

- Retrieves data sorted by age in descending order.

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

Ans: 1. WHERE Clause

Purpose:

- The WHERE clause is used to filter rows in a table based on a specific condition.
- Only rows that meet the condition are included in the result set.

Syntax:

```
SELECT column1, column2, ...
```

```
FROM table_name
```

```
WHERE condition;
```

Example:

```
SELECT Name, Age
```

```
FROM Students
```

```
WHERE Age >= 18;
```

- Retrieves only students aged 18 or older.

Key Points:

- Can use comparison operators: =, <, >, <=, >=, <>
 - Can combine conditions with AND, OR, NOT.
 - Helps in precise data retrieval.
-

2. ORDER BY Clause

Purpose:

- The ORDER BY clause is used to sort the result set by one or more columns.
- Can sort in ascending (ASC) or descending (DESC) order.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column_name [ASC|DESC];
```

Example:

```
SELECT Name, Age
FROM Students
ORDER BY Age DESC;
```

- Retrieves students sorted by age from oldest to youngest.

Key Points:

- Default sort is ascending (ASC) if not specified.
- Can sort by multiple columns:

```
SELECT Name, Age
FROM Students
ORDER BY Age ASC, Name DESC;
```

- First sorts by Age ascending, then by Name descending.
-

Combined Example

```
SELECT Name, Age
FROM Students
WHERE Age >= 18
ORDER BY Age DESC;
```

- Retrieves students 18 or older, sorted from oldest to youngest.

9. Data Control Language (DCL)

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

Ans: GRANT and REVOKE in SQL

Both GRANT and REVOKE are part of SQL's Data Control Language (DCL), which is used to control access and permissions on database objects.

1. GRANT

Purpose:

- The GRANT command is used to give specific privileges to a user or role on database objects (like tables, views, procedures).
- It allows a user to perform actions such as SELECT, INSERT, UPDATE, DELETE, etc.

Syntax:

```
GRANT privilege_name
```

```
ON object_name
```

```
TO user_name;
```

Example:

```
GRANT SELECT, INSERT
```

```
ON Students
```

```
TO user1;
```

- Gives user1 permission to read and add data in the Students table.

Key Points:

- Helps control who can access or modify data.
 - Can grant multiple privileges at once.
 - Privileges can also be granted to roles for easier management.
-

2. REVOKE

Purpose:

- The REVOKE command is used to remove previously granted privileges from a user or role.

Syntax:

```
REVOKE privilege_name
```

```
ON object_name
```

```
FROM user_name;
```

Example:

```
REVOKE INSERT
```

```
ON Students
```

```
FROM user1;
```

- Removes INSERT permission from user1 on the Students table, but they can still SELECT.

Key Points:

- Ensures security by controlling access.
- Can revoke one or multiple privileges.
- Important for maintaining data integrity and privacy.

2. How do you manage privileges using these commands?

Ans: Managing Privileges in SQL

Privileges determine what actions a user can perform on database objects (tables, views, procedures, etc.). Using GRANT and REVOKE, you can control access securely.

1. Granting Privileges

Steps:

1. Identify the user/role to give access.
2. Decide which privileges they need:
 - SELECT → Read data
 - INSERT → Add data
 - UPDATE → Modify data
 - DELETE → Remove data
 - ALL → All available privileges
3. Use the GRANT command to assign privileges.

Examples:

- Grant single privilege

```
GRANT SELECT
```

```
ON Students
```

```
TO user1;
```

- user1 can now read data from the Students table.
- Grant multiple privileges

GRANT SELECT, INSERT, UPDATE

ON Students

TO user2;

- user2 can read, add, and modify data.
- Grant ALL privileges

GRANT ALL

ON Students

TO admin_user;

- admin_user can perform any action on the table.
-

2. Revoking Privileges

Steps:

1. Identify the user/role whose privileges need to be revoked.
2. Use the REVOKE command to remove specific privileges.

Examples:

- Revoke a single privilege

REVOKE INSERT

ON Students

FROM user2;

- user2 can no longer add data, but can still SELECT and UPDATE.
- Revoke multiple privileges

REVOKE SELECT, UPDATE

ON Students

FROM user2;

- Removes both SELECT and UPDATE privileges.
-

3. Best Practices for Managing Privileges

1. Grant only necessary privileges → Follow the principle of least privilege.
2. Use roles for groups of users → Easier to manage privileges collectively.

3. Regularly audit privileges → Ensure no unnecessary access remains.
 4. Combine GRANT and REVOKE carefully → Avoid accidentally removing critical access.
-

10. Transaction Control Language (TCL)

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

Ans: COMMIT and ROLLBACK in SQL

Both are part of Transaction Control Language (TCL) in SQL, which is used to manage transactions in a database.

A transaction is a sequence of one or more SQL statements executed as a single unit.

1. COMMIT

Purpose:

- The COMMIT command is used to save all changes made by the transaction permanently to the database.
- Once committed, the changes cannot be undone.

Use Case:

- After inserting, updating, or deleting data, when you are sure that all operations were successful, use COMMIT to make the changes permanent.

Example:

```
INSERT INTO Students (StudentID, Name, Age)
```

```
VALUES (1, 'Shlok', 20);
```

```
COMMIT;
```

- The new student record is saved permanently.
-

2. ROLLBACK

Purpose:

- The ROLLBACK command is used to undo all changes made in the current transaction.
- Useful if an error occurs or if you want to cancel the operation.

Use Case:

- If a mistake is detected during a transaction, use ROLLBACK to revert the database to its previous state.

Example:

```
INSERT INTO Students (StudentID, Name, Age)
```

```
VALUES (2, 'Rahul', 17);
```

```
ROLLBACK;
```

- The record for Rahul is not saved, and the database remains unchanged.

2. Explain how transactions are managed in SQL databases.

Ans: COMMIT and ROLLBACK in SQL

Both are part of Transaction Control Language (TCL) in SQL, which is used to manage transactions in a database.

A transaction is a sequence of one or more SQL statements executed as a single unit.

1. COMMIT

Purpose:

- The COMMIT command is used to save all changes made by the transaction permanently to the database.
- Once committed, the changes cannot be undone.

Use Case:

- After inserting, updating, or deleting data, when you are sure that all operations were successful, use COMMIT to make the changes permanent.

Example:

```
INSERT INTO Students (StudentID, Name, Age)
```

```
VALUES (1, 'Shlok', 20);
```

```
COMMIT;
```

- The new student record is saved permanently.
-

2. ROLLBACK

Purpose:

- The ROLLBACK command is used to undo all changes made in the current transaction.
- Useful if an error occurs or if you want to cancel the operation.

Use Case:

- If a mistake is detected during a transaction, use ROLLBACK to revert the database to its previous state.

Example:

```
INSERT INTO Students (StudentID, Name, Age)
```

```
VALUES (2, 'Rahul', 17);
```

```
ROLLBACK;
```

- The record for Rahul is not saved, and the database remains unchanged.
-

Key Points

Command	Purpose
---------	---------

COMMIT	Makes all changes in the current transaction permanent
--------	--

ROLLBACK	Undoes all changes in the current transaction
----------	---

When to Use Use COMMIT after successful operations

In short:

- COMMIT = “Save these changes permanently.”
 - ROLLBACK = “Undo these changes and go back to the previous state.”
-

I can also create a small example showing a transaction where some changes are committed and some are rolled back, so it's easier to visualize. Do you want me to do that?

You said:

2. Explain how transactions are managed in SQL databases.

ChatGPT said:

Perfect! Let's go step by step and explain transaction management in SQL.

What is a Transaction?

A transaction is a sequence of one or more SQL statements that are executed as a single unit of work.

- Either all operations succeed (commit) or none are applied (rollback).

- Ensures data integrity in the database.
-

ACID Properties of Transactions

Transactions in SQL are managed to ensure ACID properties:

Property	Description
Atomicity	All operations in a transaction succeed or all fail. No partial updates.
Consistency	Database remains in a valid state before and after the transaction.
Isolation	Transactions are independent; intermediate changes are not visible to other transactions until committed.
Durability	Once a transaction is committed, changes are permanent, even if the system crashes.

Transaction Control Commands

1. START TRANSACTION / BEGIN
 - Marks the beginning of a transaction.
 2. START TRANSACTION;
 3. COMMIT
 - Saves all changes made in the transaction permanently.
 4. COMMIT;
 5. ROLLBACK
 - Reverts all changes made in the transaction.
 6. ROLLBACK;
 7. SAVEPOINT (optional)
 - Sets a point within a transaction to which you can roll back.
 8. SAVEPOINT sp1;
 9. ROLLBACK TO sp1;
-

Example of Transaction Management

START TRANSACTION;

INSERT INTO Accounts (AccountID, Balance) VALUES (1, 5000);

INSERT INTO Accounts (AccountID, Balance) VALUES (2, 3000);

```
-- Suppose an error occurs here  
UPDATE Accounts SET Balance = Balance - 1000 WHERE AccountID = 3;
```

```
ROLLBACK; -- Undo all above changes
```

```
-- Or if everything was fine  
COMMIT; -- Make all changes permanent
```

Explanation:

- If all operations succeed → COMMIT saves the changes.
- If an error occurs → ROLLBACK undoes all changes, maintaining data integrity.

11. SQL Joins

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

Ans: 1. Concept of JOIN in SQL

- A JOIN is used to combine rows from two or more tables based on a related column between them.
- It allows you to retrieve data from multiple tables in a single query.
- Commonly used when tables have relationships, often via primary key and foreign key.

Basic Syntax:

```
SELECT columns  
FROM table1  
JOIN table2  
ON table1.column = table2.column;
```

2. Types of JOINS and Their Differences

JOIN Type	Description	Example/Explanation
INNER JOIN	Returns only rows that have matching values in both tables.	Only records with matches in both tables appear.
LEFT JOIN (or LEFT OUTER JOIN)	Returns all rows from the left table, and matching rows from the right table. Non-matching rows from the right table appear as NULL.	Retrieves all left table data, even if there's no match in the right table.
RIGHT JOIN (or RIGHT OUTER JOIN)	Returns all rows from the right table, and matching rows from the left table. Non-matching rows from the left table appear as NULL.	Retrieves all right table data, even if there's no match in the left table.
FULL OUTER JOIN	Returns all rows from both tables. Non-matching rows from either table appear as NULL.	Combines LEFT and RIGHT JOIN results.

3. Example with Two Tables

Tables:

Students

StudentID	Name
1	Shlok
2	Rahul
3	Meera

Courses

StudentID	Course
1	Math
2	Science
4	History

INNER JOIN

SELECT Students.Name, Courses.Course

FROM Students

INNER JOIN Courses

ON Students.StudentID = Courses.StudentID;

Result:

Name	Course
Shlok	Math
Rahul	Science

- Only shows students with a matching course.
-

LEFT JOIN

```
SELECT Students.Name, Courses.Course  
FROM Students  
LEFT JOIN Courses  
ON Students.StudentID = Courses.StudentID;
```

Result:

Name	Course
Shlok	Math
Rahul	Science
Meera	NULL

- Shows all students; Meera has no course, so NULL.
-

RIGHT JOIN

```
SELECT Students.Name, Courses.Course  
FROM Students  
RIGHT JOIN Courses  
ON Students.StudentID = Courses.StudentID;
```

Result:

Name	Course
Shlok	Math
Rahul	Science
NULL	History

- Shows all courses; StudentID 4 has no matching student, so NULL.
-

FULL OUTER JOIN

```

SELECT Students.Name, Courses.Course
FROM Students
FULL OUTER JOIN Courses
ON Students.StudentID = Courses.StudentID;

```

Result:

Name	Course
Shlok	Math
Rahul	Science
Meera	NULL
NULL	History

- Shows all rows from both tables, with NULL where no match exists.

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

Ans: How Joins Are Used to Combine Data

- Purpose of Joins:
Joins allow you to retrieve related data from two or more tables in a single query.
Instead of storing all information in one table, relational databases split data into multiple tables to avoid duplication and maintain consistency.
 - Key Idea:
You join tables on related columns, usually using primary key–foreign key relationships.
-

Steps to Combine Data Using Joins

- Identify the tables you want to combine.
 - Identify the related columns (keys) between the tables.
 - Choose the type of join depending on which rows you want to include:
 - INNER JOIN → Only matching rows
 - LEFT JOIN → All rows from left table + matches from right
 - RIGHT JOIN → All rows from right table + matches from left
 - FULL OUTER JOIN → All rows from both tables
 - Write the JOIN query using ON to specify the condition.
-

Example

Tables:

Students

StudentID	Name
1	Shlok
2	Rahul
3	Meera

Courses

StudentID	Course
1	Math
2	Science
4	History

INNER JOIN Example

```
SELECT Students.Name, Courses.Course  
FROM Students  
INNER JOIN Courses  
ON Students.StudentID = Courses.StudentID;
```

Result:

Name	Course
Shlok	Math
Rahul	Science

- Combines data from both tables where StudentID matches.
-

LEFT JOIN Example

```
SELECT Students.Name, Courses.Course  
FROM Students  
LEFT JOIN Courses  
ON Students.StudentID = Courses.StudentID;
```

Result:

Name	Course
Shlok	Math
Rahul	Science
Meera	NULL

- Combines all students, even if they don't have a course.

12. SQL Group By

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

Ans: 1. GROUP BY Clause in SQL

Definition:

- The GROUP BY clause is used to group rows that have the same values in one or more columns.
- It is commonly used with aggregate functions to perform calculations on each group instead of the entire table.

Purpose:

- Summarizes data.
- Helps answer questions like:
 - “How many students are in each course?”
 - “What is the total sales per region?”

2. Aggregate Functions

Common aggregate functions used with GROUP BY:

Function	Description
COUNT()	Counts the number of rows
SUM()	Adds up numeric values
AVG()	Calculates average
MAX()	Finds the maximum value

MIN()

Finds the minimum value

3. Syntax

```
SELECT column1, aggregate_function(column2)  
FROM table_name  
GROUP BY column1;
```

4. Example

Table: Students

StudentID	Course	Marks
1	Math	85
2	Science	90
3	Math	75
4	Science	80

Query: Count how many students are in each course

```
SELECT Course, COUNT(StudentID) AS NumStudents  
FROM Students  
GROUP BY Course;
```

Result:

Course	NumStudents
Math	2
Science	2

Query: Average marks per course

```
SELECT Course, AVG(Marks) AS AvgMarks  
FROM Students  
GROUP BY Course;
```

Result:

Course	AvgMarks
Math	80
Science	85

2. Explain the difference between GROUP BY and ORDER BY.

Ans: 1. GROUP BY

Purpose:

- Groups rows that have the same values in one or more columns.
- Often used with aggregate functions (COUNT, SUM, AVG, MAX, MIN) to summarize data per group.

Key Points:

- Groups data logically.
- Returns one row per group when using aggregate functions.
- Does not control the order of the result set unless combined with ORDER BY.

Example:

```
SELECT Course, COUNT(StudentID) AS NumStudents  
FROM Students  
GROUP BY Course;
```

- Groups students by Course and counts them per course.
-

2. ORDER BY

Purpose:

- Sorts the result set based on one or more columns in ascending (ASC) or descending (DESC) order.

Key Points:

- Controls the display order of rows.
- Can be used with or without GROUP BY.
- Does not summarize or group data.

Example:

```
SELECT Name, Marks  
FROM Students  
ORDER BY Marks DESC;
```

- Shows all students sorted from highest to lowest marks.
-

3. Key Differences

Feature	GROUP BY	ORDER BY
Purpose	Groups rows with the same value	Sorts rows in a specific order
Used With	Aggregate functions	Any column or expression
Output	One row per group	Same number of rows as original (unless combined with aggregate)
Effect on Data	Summarizes/group data	Only changes display order, not data
Mandatory?	Needed for aggregates per group	Optional for sorting

Example: Combined Usage

```
SELECT Course, AVG(Marks) AS AvgMarks
FROM Students
GROUP BY Course
ORDER BY AvgMarks DESC;
```

- Groups students by course (GROUP BY)
- Calculates average marks per course
- Sorts the result by average marks from highest to lowest (ORDER BY)

13. SQL Stored Procedure

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

Ans: 1. Stored Procedure in SQL

Definition:

- A stored procedure is a precompiled set of SQL statements stored in the database.
- It can be executed multiple times without rewriting the SQL commands.
- Can include logic such as conditions (IF, CASE), loops, and parameters.

Purpose:

- To perform repetitive tasks efficiently.

- To encapsulate business logic in the database.
 - To improve performance and reduce network traffic.
-

2. Syntax (Example in MySQL)

```
CREATE PROCEDURE GetStudentMarks(IN student_id INT)
BEGIN
    SELECT Name, Marks
    FROM Students
    WHERE StudentID = student_id;
END;
```

Explanation:

- CREATE PROCEDURE → Creates a new stored procedure.
- IN student_id INT → Parameter passed to the procedure.
- BEGIN ... END → Encapsulates the SQL statements.
- Can be executed using:

```
CALL GetStudentMarks(1);
```

3. Difference Between Stored Procedure and Standard SQL Query

Feature	Stored Procedure	Standard SQL Query
Definition	Precompiled set of SQL statements with optional logic	Single SQL statement executed once
Execution	Can be executed multiple times with parameters	Runs once per execution
Logic	Can include conditions, loops, variables	Usually a single operation (SELECT, INSERT, UPDATE, DELETE)
Performance	Faster for repeated tasks (precompiled)	Slower if repeatedly sending same query
Reusability	Highly reusable	Less reusable, usually ad hoc
Security	Can restrict access and hide implementation	No encapsulation; user sees the full query

Example Usage

- Standard SQL Query:

```
SELECT Name, Marks FROM Students WHERE StudentID = 1;
```

- Stored Procedure:

```
CALL GetStudentMarks(1);
```

2. Explain the advantages of using stored procedures.

Ans: Advantages of Using Stored Procedures

1. Reusability

- A stored procedure can be written once and executed multiple times by different applications or users.
- Reduces repetition of SQL code.

Example:

A procedure GetStudentMarks(student_id) can be reused for any student without rewriting the query.

2. Improved Performance

- Stored procedures are precompiled and stored in the database.
 - The database doesn't need to parse and optimize the query each time it runs.
 - This results in faster execution for repetitive tasks.
-

3. Reduced Network Traffic

- Instead of sending multiple SQL statements over the network, you can call a procedure once.
 - Reduces data transfer and load on the network.
-

4. Enhanced Security

- You can restrict direct access to tables and allow users to execute stored procedures only.
 - Users don't need permissions on underlying tables; they just call the procedure.
 - Helps prevent SQL injection attacks.
-

5. Maintainability

- Business logic is centralized in the database.
 - Any change in the logic requires updating the stored procedure once, not modifying multiple queries in the application.
-

6. Support for Complex Logic

- Stored procedures can include control-of-flow statements like IF, WHILE, LOOP, and variables.
 - Can handle complex operations that are hard to do in a single SQL query.
-

Example

```
CREATE PROCEDURE UpdateMarks(IN student_id INT, IN new_marks INT)
```

```
BEGIN
```

```
    IF new_marks >= 0 AND new_marks <= 100 THEN
        UPDATE Students
        SET Marks = new_marks
        WHERE StudentID = student_id;
```

```
    ELSE
```

```
        SELECT 'Invalid marks';
```

```
    END IF;
```

```
END;
```

- Handles logic and validation inside the database, making it reusable and safe.

14. SQL View

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

Ans: 1. View in SQL

Definition:

- A view is a virtual table in SQL that is based on the result of a SELECT query.
- It does not store data physically; it only stores the SQL query that defines it.
- When you query a view, SQL runs the underlying SELECT statement to return the data.

Purpose:

- Simplifies complex queries.
 - Provides a layer of abstraction and security.
 - Can hide certain columns or rows from users.
-

2. Creating a View

Syntax:

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

Example:

```
CREATE VIEW TopStudents AS  
SELECT Name, Marks  
FROM Students  
WHERE Marks >= 85;
```

- Creates a view TopStudents that shows only students with marks ≥ 85 .
- Querying the view:

```
SELECT * FROM TopStudents;  
• Returns the filtered data from Students.
```

3. Differences Between a View and a Table

Feature	View	Table
Definition	Virtual table based on a query	Physical table storing data
Data Storage	Does not store data	Stores data on disk
Purpose	Simplifies queries, hides data, adds security	Stores actual data
Modification	Can be read-only or updatable (depending on DB)	Always updatable (unless constraints prevent it)
Dependency	Depends on underlying table(s)	Independent
Performance	Slight overhead because query runs at runtime	Direct access to stored data

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

Ans: Advantages of Using Views in SQL

1. Simplifies Complex Queries

- Views can encapsulate complex joins, filters, and calculations into a single virtual table.
- Users can query the view without writing complex SQL repeatedly.

Example:

```
CREATE VIEW TopStudents AS
```

```
SELECT Name, Marks
```

```
FROM Students
```

```
WHERE Marks >= 85;
```

- Users can simply do `SELECT * FROM TopStudents`; instead of repeating the WHERE condition every time.
-

2. Data Security

- Views can restrict access to certain columns or rows.
- Users can query the view without having direct access to the underlying tables.

Example:

- A view can show Name and Marks but hide sensitive information like Address or Salary.
-

3. Consistency and Reusability

- Common queries can be stored as views and reused in multiple places.
 - Ensures consistent results across applications and reports.
-

4. Logical Data Independence

- Views decouple the database structure from applications.
 - If the underlying table structure changes, the view can be updated without affecting applications that use it.
-

5. Simplifies Maintenance

- Changes in business logic (like filtering certain rows) can be applied to the view without modifying application code.
 - Reduces errors and redundancy.
-

6. Supports Aggregation and Summarization

- Views can precompute aggregated data like sums, averages, counts, etc.
- Useful for reports and dashboards.

Example:

```
CREATE VIEW AvgMarksPerCourse AS  
SELECT Course, AVG(Marks) AS AvgMarks  
FROM Students  
GROUP BY Course;
```

15. SQL Triggers

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

Ans: 1. Trigger in SQL

Definition:

- A trigger is a special kind of stored procedure that is automatically executed (fired) by the database when a specific event occurs on a table or view.
- Events can be INSERT, UPDATE, or DELETE operations.

Purpose:

- Enforce business rules automatically.
 - Maintain data integrity.
 - Perform automatic actions like logging changes or updating related tables.
-

2. Types of Triggers

Triggers can be classified based on timing and event:

A. Based on Timing

1. BEFORE Trigger

- Executed before the triggering event occurs.
- Often used to validate or modify data before it is inserted/updated/deleted.
- Example: Ensure a new employee's salary is not negative before insertion.

2. AFTER Trigger

- Executed after the triggering event occurs.
 - Used for logging, auditing, or cascading changes.
 - Example: After a student record is deleted, log the deletion in an audit table.
-

B. Based on Event

1. INSERT Trigger

- Fires when a new row is inserted into the table.

2. UPDATE Trigger

- Fires when an existing row is updated.

3. DELETE Trigger

- Fires when a row is deleted from the table.
-

C. Row-Level vs Statement-Level Triggers

1. Row-Level Trigger

- Executes once for each affected row.

2. Statement-Level Trigger

- Executes once per SQL statement, regardless of the number of affected rows.
-

3. Example: AFTER INSERT Trigger

```
CREATE TRIGGER AfterStudentInsert
```

```
AFTER INSERT ON Students
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    INSERT INTO StudentAudit(StudentID, Action, ActionDate)
```

```
    VALUES (NEW.StudentID, 'Inserted', NOW());
```

```
END;
```

- Whenever a new student is added to the Students table, a record is automatically inserted into StudentAudit.
-

4. When Triggers Are Used

- Enforcing business rules: e.g., auto-update stock when an order is placed.
- Auditing: Keep track of changes in data.
- Data validation: Check constraints beyond normal column constraints.
- Cascading changes: Automatically update related tables.
- Synchronizing tables: Keep summary tables up-to-date.

2. Explain the difference between INSERT, UPDATE, and DELETE triggers.

Ans: 1. INSERT Trigger

Definition:

- Fires automatically when a new row is inserted into a table.

Purpose:

- Can validate or modify the data before or after insertion.
- Useful for auditing inserted data.

Example (AFTER INSERT):

```
CREATE TRIGGER AfterStudentInsert
```

```
AFTER INSERT ON Students
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    INSERT INTO StudentAudit(StudentID, Action, ActionDate)
```

```
    VALUES (NEW.StudentID, 'Inserted', NOW());
```

```
END;
```

- Logs every new student inserted into the StudentAudit table.
-

2. UPDATE Trigger

Definition:

- Fires automatically when an existing row is updated in a table.

Purpose:

- Can validate updated data or log changes.
- Useful for tracking changes to critical fields.

Example (BEFORE UPDATE):

```
CREATE TRIGGER BeforeMarksUpdate
```

```
BEFORE UPDATE ON Students
```

```
FOR EACH ROW
```

```
BEGIN
```

```
IF NEW.Marks < 0 THEN
```

```
    SET NEW.Marks = 0;
```

```
END IF;
```

```
END;
```

- Ensures marks never go below 0 before updating.
-

3. DELETE Trigger

Definition:

- Fires automatically when a row is deleted from a table.

Purpose:

- Can log deleted records, enforce cascading deletes, or prevent deletion in some cases.

Example (AFTER DELETE):

```
CREATE TRIGGER AfterStudentDelete
```

```
AFTER DELETE ON Students
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    INSERT INTO StudentAudit(StudentID, Action, ActionDate)
```

```
    VALUES (OLD.StudentID, 'Deleted', NOW());
```

```
END;
```

- Logs every student deletion in StudentAudit.

16. Introduction to PL/SQL

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

Ans: 1. What is PL/SQL?

Definition:

- PL/SQL stands for Procedural Language/Structured Query Language.
- It is Oracle Corporation's procedural extension to SQL.
- Combines the data manipulation power of SQL with the programming capabilities of procedural languages (like loops, conditions, variables).

Purpose:

- Allows developers to write complex programs that interact with the database.
 - Enables control-of-flow logic, error handling, and modular programming inside the database.
-

2. How PL/SQL Extends SQL

Feature	SQL	PL/SQL
Nature	Declarative (what data to retrieve)	Procedural (how to process data)
Variables	No support	Supports variables, constants, and data types
Control Structures	Limited (e.g., CASE in queries)	Full programming constructs: IF, LOOP, WHILE, FOR
Error Handling	Minimal	Exception handling using EXCEPTION block
Modularity	No procedures or functions	Supports procedures, functions, packages, triggers
Complex Operations	Limited to single SQL statements	Can perform complex logic and multi-step operations
Performance	Executes SQL statements individually	Reduces network traffic; executes blocks of statements at once

3. Example of a PL/SQL Block

DECLARE

```
v_studentName VARCHAR2(50);  
v_marks NUMBER;
```

BEGIN

```

SELECT Name, Marks
INTO v_studentName, v_marks
FROM Students
WHERE StudentID = 1;

IF v_marks >= 85 THEN
    DBMS_OUTPUT.PUT_LINE(v_studentName || ' is a Top Student!');
ELSE
    DBMS_OUTPUT.PUT_LINE(v_studentName || ' needs Improvement.');
END IF;
END;

```

Explanation:

- DECLARE → Declares variables.
 - BEGIN ... END → Main execution block.
 - IF ... ELSE → Conditional logic.
 - DBMS_OUTPUT.PUT_LINE → Prints output.
 - Combines SQL queries with procedural logic.
-

4. Key Advantages of PL/SQL Over SQL

1. Procedural Capabilities: Can write loops, conditions, and variables.
2. Modularity: Use procedures, functions, triggers, and packages.
3. Error Handling: Handle exceptions gracefully.
4. Improved Performance: Reduces network traffic by processing multiple SQL statements in a single block.
5. Integration with SQL: Can embed SQL inside procedural code seamlessly.

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

Ans: Benefits of Using PL/SQL

1. Combines SQL and Procedural Features

- PL/SQL allows you to use SQL statements (SELECT, INSERT, UPDATE, DELETE) along with procedural programming features like variables, loops, and conditions.
 - This makes it more powerful than plain SQL.
-

2. Improved Performance

- PL/SQL allows processing multiple SQL statements in a single block, reducing the number of trips between application and database.
 - Less network traffic → faster execution for complex operations.
-

3. Modularity

- Supports procedures, functions, packages, and triggers.
 - Code can be organized, reused, and maintained easily.
 - Example: A procedure to calculate student grades can be used by multiple applications.
-

4. Error Handling

- Provides exception handling to manage runtime errors gracefully.
- Prevents crashes or incorrect data processing.

Example:

EXCEPTION

```
WHEN NO_DATA_FOUND THEN  
    DBMS_OUTPUT.PUT_LINE('No record found!');
```

5. Security

- Allows you to hide complex logic inside procedures and functions.
 - Users can execute procedures without direct access to underlying tables.
-

6. Support for Loops and Conditions

- Can perform repetitive tasks with loops (FOR, WHILE) and make decisions with IF...ELSE.
 - Example: Updating marks of all students above a certain score in one block.
-

7. Maintainability

- Changes in business logic can be updated in a single place (procedure, function, or package) instead of multiple SQL statements.
 - Makes applications easier to maintain and less error-prone.
-

8. Portability

- PL/SQL code is stored in the database and can be executed from different applications without modification.

17. PL/SQL Control Structures

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

Ans: 1. Control Structures in PL/SQL

Definition:

- Control structures are constructs that control the flow of execution in a PL/SQL program.
- They allow the program to make decisions, repeat actions, or choose between multiple paths.

Types of Control Structures in PL/SQL:

1. Conditional Control – IF...THEN...ELSE
 2. Looping Control – LOOP, WHILE, FOR
 3. Sequential Control – GOTO (less commonly used)
-

2. IF-THEN Control Structure

Purpose:

- Used to execute a block of code only if a certain condition is true.
- Can include optional ELSE and ELSIF for multiple conditions.

Syntax:

IF condition THEN

 -- statements if condition is true

ELSIF another_condition THEN

```
-- statements if another_condition is true  
ELSE  
    -- statements if all conditions are false  
END IF;
```

Example:

```
DECLARE
```

```
    v_marks NUMBER := 75;
```

```
BEGIN
```

```
    IF v_marks >= 85 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Excellent');
```

```
    ELSIF v_marks >= 60 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Good');
```

```
    ELSE
```

```
        DBMS_OUTPUT.PUT_LINE('Needs Improvement');
```

```
    END IF;
```

```
END;
```

Output:

Good

- Checks the conditions in order and executes the first true branch.
-

3. LOOP Control Structure

Purpose:

- Used to repeat a block of code multiple times until a certain condition is met.

Types of Loops in PL/SQL

1. Simple LOOP – repeats indefinitely until an EXIT condition is met.
 2. WHILE LOOP – repeats while a condition is true.
 3. FOR LOOP – repeats a fixed number of times.
-

A. Simple LOOP

```
DECLARE
```

```
    v_counter NUMBER := 1;
```

```
BEGIN  
  LOOP  
    DBMS_OUTPUT.PUT_LINE('Count: ' || v_counter);  
    v_counter := v_counter + 1;  
    EXIT WHEN v_counter > 5;  
  END LOOP;
```

```
END;
```

Output:

```
Count: 1  
Count: 2  
Count: 3  
Count: 4  
Count: 5
```

B. WHILE LOOP

```
DECLARE  
  v_counter NUMBER := 1;  
  
BEGIN  
  WHILE v_counter <= 5 LOOP  
    DBMS_OUTPUT.PUT_LINE('Count: ' || v_counter);  
    v_counter := v_counter + 1;  
  END LOOP;  
  
END;
```

C. FOR LOOP

```
BEGIN  
  FOR i IN 1..5 LOOP  
    DBMS_OUTPUT.PUT_LINE('Count: ' || i);  
  END LOOP;  
  
END;
```

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

Ans: How Control Structures in PL/SQL Help Write Complex Queries

Definition:

- Control structures in PL/SQL allow you to control the flow of execution in your program.
 - Unlike plain SQL, which only retrieves or manipulates data, PL/SQL can process data logically, iteratively, and conditionally.
-

1. Conditional Processing (IF-THEN / CASE)

- Use: Execute different SQL statements or logic based on conditions.
- Benefit: Allows queries to adapt to different situations without writing multiple separate queries.

Example:

```
DECLARE
    v_marks NUMBER := 75;
BEGIN
    IF v_marks >= 85 THEN
        DBMS_OUTPUT.PUT_LINE('Excellent');
    ELSIF v_marks >= 60 THEN
        DBMS_OUTPUT.PUT_LINE('Good');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Needs Improvement');
    END IF;
END;
```

- Decides which message to display based on student marks.
 - Without control structures, you would need multiple SQL statements to handle different conditions.
-

2. Iterative Processing (LOOP, FOR, WHILE)

- Use: Execute SQL statements repeatedly for multiple rows or calculations.
- Benefit: Automates repetitive tasks, such as updating multiple records, generating reports, or computing aggregates manually.

Example:

```
BEGIN  
    FOR i IN 1..5 LOOP  
        UPDATE Students  
        SET Marks = Marks + 5  
        WHERE StudentID = i;  
    END LOOP;  
  
END;
```

- Adds 5 marks to each student with ID 1 to 5.
- Avoids writing multiple UPDATE statements.

3. Exception Handling

- Use: Handle errors gracefully during query execution.
- Benefit: Prevents program crashes and ensures data integrity.

Example:

```
BEGIN  
    DELETE FROM Students WHERE StudentID = 10;  
  
EXCEPTION  
    WHEN NO_DATA_FOUND THEN  
        DBMS_OUTPUT.PUT_LINE('No student found with ID 10.');
```

END;

- Safely handles cases when the record doesn't exist.

4. Modularity with Procedures and Functions

- Control structures allow you to write reusable blocks with logic.
 - Example: A procedure to calculate grades for all students using IF-THEN and LOOP.
 - Makes complex queries maintainable and reusable.
-

18. SQL Cursors

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

Ans: 1. What is a Cursor in PL/SQL

Definition:

- A cursor is a pointer or handle that allows PL/SQL to process query results row by row.
- It is used when a query returns multiple rows, and you want to manipulate or process each row individually.

Purpose:

- Retrieve and process one row at a time from the result set of a query.
 - Useful for row-by-row processing when a single SQL statement is insufficient.
-

2. Types of Cursors

PL/SQL supports two types of cursors:

A. Implicit Cursor

- Automatically created by PL/SQL when any SQL statement (SELECT INTO, INSERT, UPDATE, DELETE) is executed.
- No need to declare it explicitly.
- Useful for single-row queries.

Example (Implicit Cursor with SELECT INTO):

```
DECLARE
    v_name VARCHAR2(50);
    v_marks NUMBER;
BEGIN
    SELECT Name, Marks
    INTO v_name, v_marks
    FROM Students
    WHERE StudentID = 1;
    DBMS_OUTPUT.PUT_LINE('Name: ' || v_name || ', Marks: ' || v_marks);
END;
```

- PL/SQL automatically handles the cursor for the single row returned.
 - Use %FOUND, %NOTFOUND, %ROWCOUNT attributes to check the execution result.
-

B. Explicit Cursor

- Declared manually by the programmer.
- Used when a query returns multiple rows and row-by-row processing is required.
- Steps for using an explicit cursor:
 1. Declare the cursor with a SELECT statement.
 2. Open the cursor.
 3. Fetch each row into variables.
 4. Close the cursor.

Example (Explicit Cursor):

```
DECLARE
    CURSOR student_cursor IS
        SELECT Name, Marks FROM Students;
        v_name VARCHAR2(50);
        v_marks NUMBER;
BEGIN
    OPEN student_cursor; -- Step 2
    LOOP
        FETCH student_cursor INTO v_name, v_marks; -- Step 3
        EXIT WHEN student_cursor%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE('Name: ' || v_name || ', Marks: ' || v_marks);
    END LOOP;
    CLOSE student_cursor; -- Step 4
END;
```

- Processes all rows one by one.
-

3. Key Differences Between Implicit and Explicit Cursors

Feature	Implicit Cursor	Explicit Cursor
Declaration	Automatic by PL/SQL	Must be declared manually
Use Case	Single-row queries	Multi-row queries needing row-by-row processing
Control	Limited	Full control: open, fetch, close
Attributes	%FOUND, %NOTFOUND, %ROWCOUNT, %ISOPEN	Same attributes plus manual handling
Complexity	Simple	More complex but flexible

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

Ans: When to Use an Explicit Cursor Over an Implicit Cursor

Implicit cursors are simple and automatic, but they only handle single-row queries easily. Explicit cursors give you full control over query results, especially when you need row-by-row processing.

You would use an explicit cursor when:

1. Processing Multiple Rows

- If a query returns more than one row, implicit cursors are not sufficient.
- Explicit cursors allow you to fetch and process each row individually.

Example: Processing all students' marks:

CURSOR student_cursor IS

SELECT Name, Marks FROM Students;

2. Row-by-Row Processing

- When you need to perform calculations, validations, or updates for each row individually.

Example:

```
FETCH student_cursor INTO v_name, v_marks;
IF v_marks < 50 THEN
  UPDATE Students SET Status = 'Fail' WHERE Name = v_name;
END IF;
```

-
- 3. More Control Over Execution
 - o Explicit cursors let you open, fetch, and close at your own pace.
 - o Can combine with loops, conditions, or exception handling for complex logic.

 - 4. Need to Use Cursor Attributes
 - o %FOUND, %NOTFOUND, %ROWCOUNT, %ISOPEN can be used more flexibly with explicit cursors.

 - 5. Dynamic or Complex Queries
 - o For queries that cannot be handled in a single SELECT INTO, like multiple joins or aggregates, explicit cursors are ideal.

19. Rollback and Commit Savepoint

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

Ans: 1. What is a SAVEPOINT in SQL

Definition:

- A SAVEPOINT is a marker within a transaction that allows you to partially rollback to that point without affecting the entire transaction.
- Think of it as a checkpoint in your transaction.

Purpose:

- Provides fine-grained control over transactions.
- Allows you to undo specific parts of a transaction if an error occurs, without losing all work.

2. Syntax for SAVEPOINT

SAVEPOINT savepoint_name;

Example:

```
BEGIN;  
INSERT INTO Students (StudentID, Name) VALUES (1, 'Shlok');  
SAVEPOINT sp1;  
INSERT INTO Students (StudentID, Name) VALUES (2, 'Meera');  
• sp1 is a savepoint within the transaction.
```

3. Interaction with ROLLBACK and COMMIT

A. ROLLBACK

- ROLLBACK can undo changes up to a savepoint instead of the entire transaction.
- Syntax:

```
ROLLBACK TO savepoint_name;
```

Example:

```
ROLLBACK TO sp1;  
• Undoes the insertion of Meera but keeps the insertion of Shlok.
```

B. COMMIT

- COMMIT makes all changes since the start of the transaction permanent.
- After a COMMIT, all savepoints are automatically released.
- You cannot rollback to a savepoint after committing.

Example:

```
COMMIT;  
• Now both insertions (Shlok and Meera) are permanent, and sp1 no longer exists.
```

4. Key Points

Feature	Description
SAVEPOINT	Creates a checkpoint inside a transaction
ROLLBACK TO	Undo changes up to a specific savepoint
COMMIT	Makes all changes permanent, releases all savepoints
Usage	Allows partial undo for safer transaction management

Illustrative Flow

1. Start transaction
2. Insert Shlok → Savepoint sp1
3. Insert Meera → Savepoint sp2 (optional)
4. Error occurs → ROLLBACK TO sp1 → Only Shlok remains
5. COMMIT → Shlok's insertion is now permanent

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

Ans: When Savepoints Are Useful in a Database Transaction

SAVEPOINTS provide fine-grained control over transactions, allowing you to undo specific parts without rolling back the entire transaction. They are especially useful in the following situations:

1. Partial Rollbacks

- When a transaction performs multiple operations, a savepoint allows you to undo only some of them instead of the whole transaction.
- Example:
 - Insert multiple student records.
 - One record fails validation → Rollback only that record, keep the others.

2. Complex Transactions

- In transactions with many steps or dependent operations, savepoints help manage errors without losing all work.
- Example:
 - Transfer money from one account to another (deduct from A, add to B, update logs).
 - If adding to B fails, rollback only that step instead of undoing the deduction from A.

3. Error Recovery and Exception Handling

- Useful in PL/SQL or procedural scripts for handling exceptions gracefully.
- Example:
- SAVEPOINT sp1;
- BEGIN

- -- Insert student
 - EXCEPTION
 - WHEN OTHERS THEN
 - ROLLBACK TO sp1; -- undo only this insertion
 - END;
 - Prevents entire transaction failure while handling errors locally.
-

4. Testing and Validation

- When performing data migrations, batch updates, or bulk operations, savepoints allow you to test intermediate steps and rollback if needed.
 - Example: Validate data in intermediate steps before committing the final changes.
-

5. Nested Transactions Simulation

- Some databases do not support true nested transactions, but savepoints can simulate nested transactions by marking points within a larger transaction.