❖Module 5 :– Introduction to DBMS

Introduction to SQL:

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

Ans:-

- SQL (**Structured Query Language**) is the language used to **communicate with databases**.
- SQL helps you work with data in a database – easily, quickly, and safely**.**

➢ Why is SQL Important?

- **Stores and manages data** in tables.
- **Finds** specific data.
- **Adds, updates, or deletes** data.
- **Creates tables and databases.**
- **Controls access** to data.
- Keeps data **organized and accurate.**

Example:

Query:

SELECT * FROM students;

Output:

Shows all student data from the students table.

2.Explain the difference between DBMS and RDBMS.

Ans:-

Feature	DBMS (Database Management System)	RDBMS (Relational Database Management System)
Data Storage	Stores data as files or hierarchical/tree structures.	Stores data in **tables** (rows and columns).
Relationship	Does **not** support relationships between data.	Supports **relationships** using **foreign keys**.
Normalization	Not required or not supported.	Supports **data normalization** to reduce redundancy.
Data Integrity	Low – no rules to maintain data accuracy.	High – uses constraints like **primary key**, **foreign key**, etc.
Security	Less security features.	Advanced security at row, column, and user level.
Examples	File systems, XML-based storage, MS Access (basic)	MySQL, PostgreSQL, Oracle, MS SQL Server, etc.
Complexity	Suitable for small-scale applications.	Suitable for large, complex applications.

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

Ans:-

- **SQL** is the **standard language** used to **communicate with and manage relational databases**.

- It plays a central role in all database operations.

1. DDL – Data Definition Language:

- Used to **define and modify** database structure.

Command		Purpose
CREATE	:	Create a new table or database
ALTER	:	Modify existing table
DROP	:	Delete table or database
TRUNCATE	:	Remove all records (faster than DELETE)

2. DML – Data Manipulation Language:

- Used to **change the data** in tables (insert, update, delete).

Command		Purpose
INSERT	:	Add new data
UPDATE	:	Modify existing data
DELETE	:	Remove data

3. DQL – Data Query Language:

- Used to **fetch data** from the database.

Command		Purpose
SELECT	:	Retrieve data from table

4. TCL – Transaction Control Language:

- Used to **manage transactions** (group of operations) and maintain data integrity.

Command	Purpose
COMMIT	: Save changes
ROLLBACK	: Undo changes
SAVEPOINT	: Set a point to roll back to

5. DCL – Data Control Language:

- Used to **control access** and **permissions** on the data.

Command	Purpose
GRANT	: Give access to users
REVOKE	: Remove user access

4. What are the key features of SQL?

Ans:-

1. **Easy to Use:**
    - SQL is simple and uses English-like commands (like SELECT, INSERT).
2. **Used to Get Data:**
    - You can use it to find or view data from the database.
3. **Create and Manage Tables:**
    - You can make new tables, change them, or delete them.
4. **Add, Change, or Delete Data:**
    - You can add new data, update old data, or remove data.
5. **Control Access:**
    - You can give permission to others to use the database or take it away.
6. **Safe Transactions:**
    - You can save your work, undo changes, or set save points.
7. **Used with Programming:**
    - SQL can be used inside other languages like Python, Java, etc.

SQL Syntax

1. What are the basic components of SQL syntax?

Ans:-

1. Keywords:

- Reserved words used to perform actions.
- Example:
  SELECT, INSERT, UPDATE, DELETE, CREATE, WHERE, etc.

2. Clauses:

- Parts of SQL statements that define conditions or rules.
- Example:
  - WHERE – filters records
  - ORDER BY – sorts records
  - GROUP BY – groups records by column

3. Expressions:

- Combines values, operators, and functions to return a result.
- Example:
  price * quantity

4. Predicates:

- Conditions used to limit results.
- Example:
  WHERE age > 18

5. Identifiers:

- Names of database objects (tables, columns, etc.)
- Example:
  students, student_id, name

6. Operators:

- Used to perform operations.
- Types:
  - Arithmetic: +, -, *, /
  - Comparison: =, >, <, !=, <>
  - Logical: AND, OR, NOT

7. Literals:

- Constant values in a query.
- Example:
  'John', 100, '2025-07-29'

8. Functions:

- Built-in routines that return values.
- Example:
  - COUNT(), SUM(), AVG(), NOW(), UPPER()

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

Ans:-

SYNTAX:

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

Explanation:

Clause	Description
SELECT	Specifies the columns to retrieve
FROM	Specifies the table to retrieve data from
WHERE	Filters rows based on a condition (optional)
GROUP BY	Groups rows with the same values (optional)
HAVING	Filters grouped rows (used with GROUP BY)
ORDER BY	Sorts the result (optional: ASC or DESC)
LIMIT	Limits the number of rows returned (optional)

3. Explain the role of clauses in SQL statements.

Ans:-

Role of Clauses in SQL Statements:

- Clauses in SQL are **building blocks** of SQL queries.
- They define **what data to select**, **from where**, and **how to filter, group, or sort** that data.

➢ **Key SQL Clauses and Their Roles:**

Clause	Role / Purpose
SELECT	Specifies the columns to be retrieved.
FROM	Specifies the table(s) from which to fetch the data.
WHERE	Filters rows based on a condition (before grouping).
GROUP BY	Groups rows with the same values in specified columns.
HAVING	Filters groups created by GROUP BY (like WHERE, but after grouping).
ORDER BY	Sorts the result set in ascending (ASC) or descending (DESC) order.
LIMIT	Restricts the number of rows returned.
JOIN	Combines rows from two or more tables based on a related column.

SQL Constraints:

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

Ans:-

- Constraints are rules used in SQL to limit the type of data that can be stored in a table.
- They help keep the data accurate and reliable.

Types of SQL Constraints:

- ✓ **NOT NULL** – Data must not be empty.

  Example: Name cannot be blank.

- ✓ **UNIQUE** – All values in a column must be different.

  Example: Email must be unique.

- ✓ **PRIMARY KEY** – Uniquely identifies each record and cannot be null.

  Example: ID in a student table.

- ✓ **FOREIGN KEY** – Links two tables using a key from another table.

  Example: Student ID in a marks table refers to ID in student table.

- ✓ **CHECK** – Sets a condition that values must meet.

  Example: Age must be more than 18.

- ✓ **DEFAULT** – Sets a default value if none is given.

  Example: Default country = 'India'.

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

Ans :-

PRIMARY KEY		FOREIGN KEY
Purpose	Uniquely identifies each row	Links one table to another
Uniqueness	Not allowed	Can be duplicate
Location	Location	In main (parent) table
Relation	Does not refer to another table	Refers to a PRIMARY KEY in another table

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

Ans:-

role of NOT NULL and UNIQUE constraints:

Constraint	Role/Function	Example
NOT NULL	Makes sure a column **cannot have empty values**	Name must always be filled
UNIQUE	Makes sure **all values are different** in column	Email must not repeat

Main SQL Commands and Sub-commands (DDL)

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

Ans:-

- DDL is a part of SQL used to **define and manage database structures** like tables, schemas, and indexes.
- DDL helps you **create, modify, and delete** database objects.

Common DDL Commands:

CREATE : To create a new table or database

ALTER : To change an existing table structure

DROP : To delete a table or database

TRUNCATE : To delete all data from a table

RENAME : To rename a table

2.Explain the CREATE command and its syntax.

Ans:-

- The CREATE command is used to **create new database objects** like tables, databases, views, etc.

Used For:

- Creating a new **table**
- Creating a new **database**

Syntax to Create a Table:

```
CREATE  TABLE  table_name (
   column1  datatype  constraint,
   column2  datatype  constraint,
   ...
)
```

Example:

```
CREATE  TABLE  Students (
   stud_id  INT  PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   age INT,
   email VARCHAR(100) UNIQUE
);
```

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

Ans:-

1. Data Types:

Purpose:

- To define what kind of data can be stored in each column.

Additional Theory:

- **Memory Optimization:** Helps the database engine allocate the right amount of memory. For example, using TINYINT instead of INT for small numbers saves space.
- **Indexing Efficiency:** Some data types index faster and more efficiently than others.
- **Data Validation at DB Level:** Reduces chances of invalid or inconsistent data being stored (e.g., rejecting text in a numeric column).

2. Constraints:

Purpose:

- To enforce rules and conditions on the data.

Additional Theory:

- **Improves Data Integrity:** For example, a FOREIGN KEY ensures relationships between tables remain valid.
- **Reduces Application Logic:** Some validation is handled by the database, reducing the need for extra code in applications.
- **Prevents Duplicate or Missing Data:** With constraints like UNIQUE, NOT NULL, or CHECK, the system prevents invalid entries.

ALTER Command:

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

Ans:-

- The ALTER command is used to **change the structure of an existing table**.

Uses of ALTER Command:

Action	Example
Add a new column	ADD
Change column data type	MODIFY / ALTER COLUMN
Rename a column or table	RENAME
Delete a column	DROP

Example:

1.Add a new column:

```
ALTER TABLE Students ADD age INT;
```

2. Change data type:

```
ALTER TABLE Students MODIFY age VARCHAR(3);
```

3.Rename table:

```
ALTER TABLE Students RENAME TO Learners;
```

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

Ans:-

1. Add a Column:

```
ALTER TABLE table_name

ADD column_name datatype;
```

Example:

```
ALTER TABLE Students
ADD age INT;
```

2. Modify a Column:

```
ALTER TABLE table_name

MODIFY column_name new_datatype;
```

Example:

```
ALTER TABLE Students
MODIFY age VARCHAR(3);
```

3. Drop (Delete) a Column:

```
ALTER TABLE table_name

DROP COLUMN column_name;
```

Example:

```
ALTER TABLE Students
DROP COLUMN age;
```

DROP Command

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

Ans:-

- The DROP command is used to **permanently delete** a **database object** like a **table**, **database**, or **column**.
- The DROP command **deletes the entire object** from the database — **no recovery**.

Function:

- Completely **removes the structure and data.**
- **Cannot be undone**, so use with care.

SYNTAX:

1. Drop a table:

```
DROP TABLE  TABLE_NAME;
```

2.Drop a database:

```
DROP DATABASE DATABASE_NAME;
```

Examples:

1. Drop a table:

```
DROP TABLE Students;
```

2.Drop a database:

```
DROP DATABASE School;
```

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

Ans:-

- When you use the DROP TABLE command, the following happens:

Effect	Explanation
Table is deleted	The table structure and all its data are **permanently removed**
Data loss	All rows in the table are gone **forever**
Constraints removed	Primary keys, foreign keys, and indexes are also deleted
No rollback	Cannot be undone unless you have a backup
Dependencies break	Other tables depending on it (via foreign key) may cause errors

Data Manipulation Language (DML):

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

Ans:-

1. INSERT:

Use: To add new data (records) into a table.

Syntax:

INSERT INTO table_name (column1, column2, ...) VALUES (value1, value2, ...)

Example:

INSERT INTO Students (Name, Age, Class)VALUES ('Hensi', 21, 'BCA')

2. UPDATE:

Use: To change or modify existing data in a table.

Syntax:

UPDATE table_name SET column1 = value1, column2 = value2, ...WHERE condition;

Example:

UPDATE Students SET Age = 20 WHERE Name = 'Hensi';

3. DELETE:

Use: To remove existing data (records) from a table.

Syntax:

DELETE FROM table_name WHERE condition;

Example:

DELETE FROM Students WHERE Name = ''Hensi'';

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

Ans:-

- The **WHERE clause** is **very important** in **UPDATE** and **DELETE** operations because it **specifies which records** should be changed or removed.

- Without a WHERE clause, the operation will affect **all rows** in the table — which can lead to **accidental data loss or wrong updates**.

Importance of WHERE Clause:

Operation	Without WHERE Clause	With WHERE Clause
UPDATE	Updates **all rows** in the table	Updates **only matching rows**
DELETE	Deletes **all rows** in the table	Deletes **only matching rows**

Additional Explanation:

- **Data Safety:**

  Prevents unintentional changes or deletions.

- **Accuracy:**

  Targets specific records based on conditions.

- **Performance:**

  Speeds up the query by working on fewer rows.

- **Business Logic Enforcement:**

  Ensures only valid, necessary changes are made.

Data Query Language (DQL):

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

Ans:-

- The **SELECT** statement is used to **fetch (query) data** from a table in a database.
- It helps you **see only the data** you are interested in.

Syntax:

```
SELECT column1, column2, ...
FROM table_name;
```

Example:

```
SELECT Name, Age
FROM Students;
```

Useful Variations:

Use	Example
All columns	SELECT * FROM Students;
Filter rows	SELECT * FROM Students WHERE Age > 18;
Sort results	SELECT * FROM Students ORDER BY Name ASC;
Limit rows	SELECT * FROM Students LIMIT 5;

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

Ans:-

1. WHERE Clause:

- **Use:**
  Filters rows in a table **based on a condition**.

Syntax:

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

Example:

```
SELECT * FROM Students
WHERE Age > 18;
```

➡Shows only students **older than 18**.

2. ORDER BY Clause:

- **Use:**
  Sorts the result **in ascending (default) or descending order**.

Syntax:

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

Example:

```
SELECT * FROM Students
ORDER BY Age DESC;
```

➡Shows all students sorted by **Age from highest to lowest**.

Data Control Language (DCL):

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

Ans:-

GRANT and **REVOKE** are used in SQL to control user access to the database:

- **GRANT**:
    - Gives specific permissions (like SELECT, INSERT, UPDATE) to a user.
- **REVOKE**:
    - Takes back those permissions from a user.

Example:

```
GRANT  SELECT  ON students TO user1;

REVOKE  SELECT ON students FROM user1;
```

Purpose:

- To **manage security** by deciding **who can do what** in the database.

  GRANT = Allow access.

  REVOKE = Remove access.

2. How do you manage privileges using these commands?

Ans:-

You manage privileges in SQL using **GRANT** and **REVOKE** like this:

- Use **GRANT** to **allow** actions.
- Use **REVOKE** to **deny** or **remove** actions.
- Helps keep your database **safe and controlled**.

To Give Privileges (using GRANT):

- GRANT privilege_name ON table_name TO user_name;

Example:

```
GRANT SELECT, INSERT ON students TO user1;
```

➡This gives **user1** permission to **view** and **add** data in the students table.

To Remove Privileges (using REVOKE):

- REVOKE privilege_name ON table_name FROM user_name;

Example:

```
REVOKE INSERT ON students FROM user1;
```

➡This removes **user1**'s permission to **add data** to the students table.

Transaction Control Language (TCL)

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

Ans:-

- The **COMMIT** and **ROLLBACK** commands are used to manage **transactions** in SQL.

Purpose:

- **COMMIT** = Save changes
- **ROLLBACK** = Cancel changes

1. **COMMIT:**

- Saves all changes made during the current transaction **permanently** in the database.

Example:

```
INSERT INTO students
VALUES (1, 'Amit');
COMMIT;
```

2.ROLLBACK:

- Undoes all changes made in the current transaction, **restoring** the database to its previous state.

Example:

```
INSERT INTO students
VALUES (2, 'Neha');
ROLLBACK;
```

2. Explain how transactions are managed in SQL databases.

Ans:-

- A **transaction** is a group of SQL operations (like INSERT, UPDATE, DELETE) that are treated as a **single unit of work**.
- Transactions make sure the database remains **accurate and consistent**, even if something goes wrong.

➢ **Key Concepts of Transaction Management:**

1. **BEGIN / START TRANSACTION:**

   Starts a new transaction.

   START TRANSACTION;

2. **COMMIT:**

   Saves all the changes made in the transaction permanently.

   COMMIT;

3. **ROLLBACK:**

   Cancels the transaction and undoes all changes.

   ROLLBACK;

➢ **ACID Properties (rules every transaction follows):**

- **A**: Atomicity – All or nothing.
- **C**: Consistency – Keeps data correct.
- **I**: Isolation – One transaction doesn't affect another.
- **D**: Durability – Changes remain after COMMIT.

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:-

- A **JOIN** in SQL is used to **combine rows from two or more tables** based on a related column (usually a foreign key).

➢ **Why use JOIN?**

To get **complete information** from multiple tables in one result.

Types of JOINs:

Type	Description
INNER JOIN	Returns only **matching rows** from both tables
LEFT JOIN	Returns **all rows from the left table**, and matching rows from the right table (NULL if no match)
RIGHT JOIN	Returns **all rows from the right table**, and matching rows from the left table (NULL if no match)
FULL OUTER JOIN	Returns **all rows from both tables**, with NULL where there's no match (Not supported directly in MySQL, but can be simulated using UNION)

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

Ans:-

- **Joins** are used to **merge rows from two or more tables** based on a **related column**.

How it works:

- SQL looks for **matching values** in the columns you specify.
- Then it combines the columns from both tables into one result.

Example:

Table 1: students:

student_id	name	teacher_id
1	Ravi	101
2	Neha	102

Table 2: teachers:

teacher_id	teacher_name
101	Amit
102	Sneha

Using JOIN to combine:

```
SELECT s.name, t.teacher_name
FROM students s
JOIN teachers t ON s.teacher_id = t.teacher_id;
```

SQL Group By:

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

Ans:-

- The **GROUP BY** clause is used to **group rows that have the same values** in specified columns.
- GROUP BY is used to **organize data into groups**.
- It works **with aggregate functions** to summarize data per group.
- Always make sure selected columns are **either in GROUP BY** or **inside an aggregate function**.

➢ It's most often used **with aggregate functions** like:

- COUNT() – number of rows
- SUM() – total value
- AVG() – average
- MAX() – maximum
- MIN() – minimum

EXAMPLE:

```
SELECT class, MAX(marks) AS top_score

FROM students

GROUP BY class;
```

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

Ans:-

- **Difference Between GROUP BY and ORDER BY in SQL:**

- **GROUP BY:**

1. Used to **group rows** that have the same values in specified columns.
2. Always used with **aggregate functions** like SUM(), COUNT(), AVG(), etc.
3. Reduces multiple rows into **one row per group**.
4. Syntax: GROUP BY column_name
5. Example: Get total marks **per class**.

- **ORDER BY:**

1. Used to **sort** the result set by one or more columns.
2. Can be used **with or without** aggregate functions.
3. Does **not group** rows, only **changes the display order**.
4. Syntax: ORDER BY column_name [ASC | DESC]
5. Example: Sort students by **marks in descending order**.

SQL Stored Procedure:

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

Ans:-

- A **stored procedure** is a **predefined set of SQL statements** that are **stored in the database** and can be executed as a unit by calling its name.

Difference Between Stored Procedure and Standard SQL Query

Feature	Stored Procedure	Standard SQL Query
Definition	Saved, reusable SQL block	One-time SQL command
Reusability	Can be reused multiple times	Must be retyped/run again
Logic	Can include logic (IF, LOOP, etc.)	Just a single SQL statement
Parameters	Can accept input/output parameters	No parameters
Performance	Faster for repeated tasks (compiled once)	Parsed every time it runs
Security	Access can be restricted	Less control over permissions

2. Explain the advantages of using stored procedures.

Ans:-

1. Reusability

- You write the SQL logic once and reuse it many times by just calling the procedure.
- Makes development faster and reduces duplicate code.

2. Better Performance

- Stored procedures are **compiled and cached** in the database.
- They execute faster than sending multiple SQL queries from the application.

3. Improved Security

- You can **control access** using privileges: users can run the procedure without seeing or changing the underlying table.
- Sensitive logic stays hidden from the application.

4. Maintainability

- If you need to update logic, you only change it in the stored procedure.
- No need to update the same SQL code across multiple apps or scripts.

5. Supports Programming Logic

- Stored procedures allow:
  - **IF**, **CASE**, **LOOP**
  - **Variables**, **error handling**, and **transactions**
- Helps implement complex business logic directly in the database.

6. Reduces Network Traffic

- Only the procedure call is sent from the application to the server, not all the SQL code.
- Especially useful for batch operations.

7. Easier Debugging and Testing

- You can test stored procedures independently from your app.
- Makes it easier to isolate and fix issues.

SQL View :

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

Ans:-

- A **view** is a **virtual table** created by a SQL query.
- It **doesn't store data** itself — it **displays data** from one or more tables.

Example:

```
CREATE VIEW course_view AS
SELECT course_id, course_name
FROM courses
WHERE credits >= 4;
```

Difference Between a View and a Table:

Feature	View	Table
Definition	Virtual table based on a SELECT query	Physical storage of actual data
Storage	Does **not** store data	**Stores** data permanently
Creation	Created with CREATE VIEW	Created with CREATE TABLE
Updatable	Sometimes (read-only by default)	Fully updatable
Data source	Based on one or more tables or views	Base object for storing raw data
Use case	Simplify complex queries, limit access	Store and manage actual data

In Simple Words:

- A **table** is like a **real book** — it stores the content.
- A **view** is like a **bookmark** — it shows selected pages, but doesn't store anything itself.

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

Ans:-

1. Simplifies Complex Queries:

- You can save a complex SELECT query as a view.
- Next time, just use:
  SELECT * FROM view_name;

2. Improves Security:

- You can **hide sensitive columns** (like salary) by only showing selected data in the view.
- Users see only what they need.

3. Easy to Use:

- Views make it easy for beginners to use complicated data.
- Acts like a shortcut or a filtered table.

4. No Duplicate Code:

- If many queries need the same logic, just use a view instead of writing the same query again and again.

5. Dynamic Updates:

- Views show the **latest data** from the table every time — no need to refresh it.

6. Logical Data Separation:

- You can organize and present data differently without changing the original table.

SQL Triggers :

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

Ans:-

- A **trigger** is a special type of stored program in SQL that **automatically runs** when a specific **event** happens on a table — like an **INSERT**, **UPDATE**, or **DELETE**.
- You don't run it manually. The database triggers it when needed.

Example:

```
CREATE TRIGGER before_insert_student
BEFORE INSERT ON students
FOR EACH ROW
BEGIN
  SET NEW.created_at = NOW();
END;
```

This trigger **auto-sets the date** before inserting a student.

Types of Triggers:

Trigger Type	When It Runs	Use Case Example
BEFORE INSERT	Before inserting new data	Auto-fill values (e.g., timestamps)
AFTER INSERT	After inserting new data	Log the new entry
BEFORE UPDATE	Before updating a row	Validate or change values before update
AFTER UPDATE	After updating a row	Track changes, audit history
BEFORE DELETE	Before deleting a row	Check permissions, prevent deletion
AFTER DELETE	After deleting a row	Log deletion, cleanup related data

When Are Triggers Used?

- **Auto-update fields** like timestamps or status
- **Validate data** before saving
- **Maintain logs** for insert/update/delete actions
- **Enforce business rules** (e.g., prevent deleting important records)
- **Automatically sync or update related table**

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

Ans:-

➢ **INSERT Trigger**

- Runs **when new data is added** to a table.
- Used to **check, modify, or log** the inserted data.

Example:

BEFORE INSERT ON students

➢ **UPDATE Trigger**

- Runs **when existing data is changed**.
- Used to **validate, track, or restrict** updates.

Example:

AFTER UPDATE ON courses

➢ **DELETE Trigger**

- Runs **when a row is deleted** from a table.
- Used to **log, prevent, or clean up** related data.

Example:

BEFORE DELETE ON students

Comparison Table:

Trigger Type	When It Runs	Common Uses
INSERT	On new row insert	Auto-fill values, validate data, log
UPDATE	On data change	Audit trail, prevent unwanted changes
DELETE	On row delete	Log deletions, prevent data loss

Introduction to PL/SQL

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

Ans:-

- **PL/SQL** stands for **Procedural Language extensions to SQL**.
- It is a **programming language** used in **Oracle Database** (not MySQL).

Difference from SQL:

- **SQL** is used to **query and manipulate data**.
- **PL/SQL** is used to **write programs** (with logic, loops, conditions).

How PL/SQL extends SQL:

SQL Can Do	PL/SQL Adds
SELECT, INSERT, UPDATE, DELETE	Variables, Loops, If-Else
Just one query at a time	Multiple queries in one block
No logic control	Full program control

Example:

```
DELIMITER  $$

CREATE PROCEDURE insert_numbers()
BEGIN

 DECLARE i INT DEFAULT 1;

 WHILE i <= 5 DO
  INSERT INTO demo_table (value) VALUES (i);
  SET i = i + 1;
 END WHILE;

END  $$

DELIMITER  ;
```

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

Ans:-

1. Combines SQL and Programming Logic:

- You can use **SQL commands** with **programming features** (like loops, if-else).
- Example: Process multiple rows with a loop.

2. Code Reusability:

- You can create **procedures, functions, and packages** and reuse them again and again.
- This saves time and reduces mistakes.

3. Better Performance

- PL/SQL code runs **directly on the database server**, so it's faster than sending many SQL queries from an app.

4. Error Handling

- PL/SQL supports **exception handling** to catch and manage errors smoothly.
- Example: You can write custom error messages when something fails.

5. Security

- You can **control access** to your code using **procedures and functions** without giving full table access.
- Helps protect sensitive data.

6. Easy Maintenance

- Logic is stored in one place (like stored procedures), so if changes are needed, update it once and it's done everywhere.

7. Support for Complex Business Logic

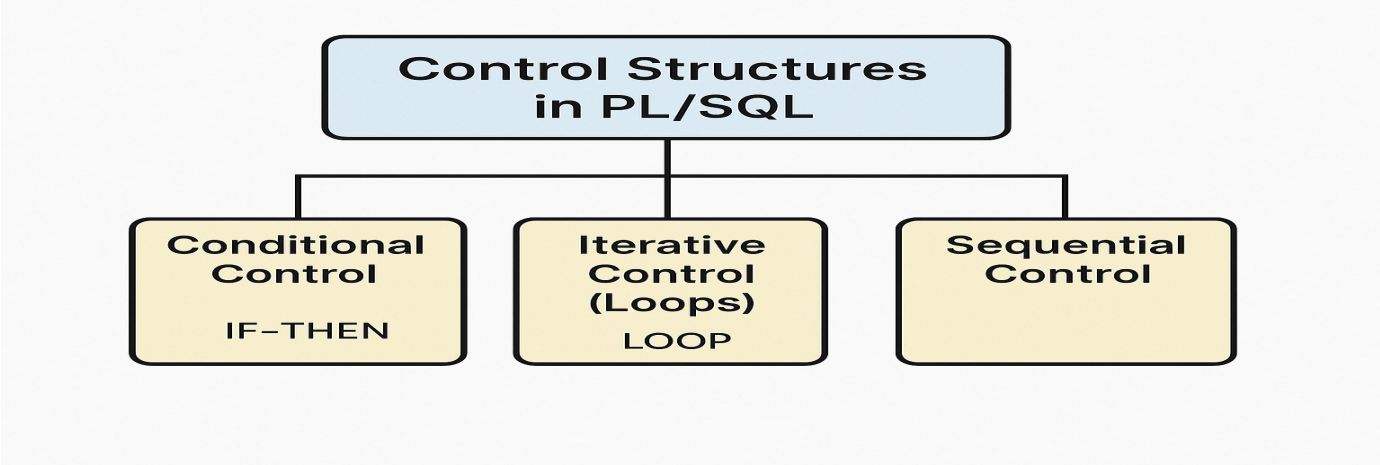
- You can write full programs with **conditions, loops, variables**, etc., making it easier to handle complex tasks inside the database.

PL/SQL Control Structures:

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

Ans:-

- **Control structures** are used to control the **flow of execution** in a PL/SQL program — like decision-making and repeating tasks.



1. IF-THEN (Conditional Control):

- Used to run code only if a condition is **true**.

Syntax:

```
IF condition THEN
  -- statements;
END IF;
```

2. LOOP (Iterative Control):

- Used to **repeat** a block of code **multiple times**.

Syntax:

```
LOOP
  -- statements;
  EXIT WHEN condition;
END LOOP
```

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

Ans:-

- **Control structures = SQL + Logic + Flow = Complex rules handled easily**

1. Decision-Making (IF-THEN):

- You can **execute specific SQL** statements **only when certain conditions are met**.
- Helps handle different business rules in one program.

Example:

```
IF salary > 50000 THEN
  UPDATE emp SET grade = 'A' WHERE emp_id = 101;
END IF;
```

2. LOOPs:

- You can **repeat SQL statements** for multiple values or conditions without writing them multiple times.

Example:

```
FOR i IN 1..5 LOOP
  INSERT INTO temp_table VALUES (i);
END LOOP;
```

3. Error Handling (EXCEPTION)

- You can catch and handle errors **without stopping the program**, which is very useful in complex logic.

4. Combining Multiple SQLs

- You can **group multiple SQL queries** with logic, making the code cleaner and more manageable.

5. Modular Programming

- With **procedures/functions**, you can split complex logic into smaller, reusable parts.

SQL Cursors:

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

Ans:-

- A **cursor** is a pointer that helps you **process query results row by row**.
- When a **SELECT query returns multiple rows**, a cursor is used to handle them **one at a time**.

Two Types of Cursors:

Type	Created By	Used When	User Control
Implicit	PL/SQL engine	Automatically for simple queries (like SELECT INTO, INSERT, UPDATE, DELETE)	No
Explicit	Manually by programmer	For complex SELECT queries that return multiple rows	Yes (OPEN, FETCH, CLOSE)

1. Implicit Cursor:

- PL/SQL creates and manages it **automatically**.

Example:

```
BEGIN
  UPDATE emp SET salary = salary + 1000 WHERE dept_id = 10;
  -- PL/SQL creates an implicit cursor here
END;
```

You can check its result using built-in attributes like:

```
SQL%ROWCOUNT  -- shows how many rows were affected
```

2. Explicit Cursor:

- Created by you to **fetch multiple rows one by one**.

Steps:

1. **DECLARE** the cursor
2. **OPEN** the cursor
3. **FETCH** each row
4. **CLOSE** the cursor

Example:

```
DECLARE
  CURSOR emp_cursor IS SELECT emp_name FROM emp;
  emp_name_var emp.emp_name%TYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_name_var;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(emp_name_var);
  END LOOP;
  CLOSE emp_cursor;
END;
```

Differences:

Feature	Implicit Cursor	Explicit Cursor
Created By	Automatically by PL/SQL	Manually by the programmer
Used For	Single-row queries	Multi-row SELECT queries
Control	No manual control	Needs OPEN, FETCH, CLOSE
Attributes	SQL%FOUND, SQL%ROWCOUNT	cursor_name%FOUND, %ROWCOUNT

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

Ans:-

When to Use an **Explicit Cursor** over an **Implicit Cursor**:

1. Query Returns Multiple Rows:

- If your SELECT statement returns **more than one row**, and you need to **process each row one by one**, use an explicit cursor.

Example:

```
SELECT emp_name FROM employee;  -- multiple rows
```

2. You Need More Control:

- Explicit cursors let you **OPEN**, **FETCH**, and **CLOSE** rows manually.
- Useful when you want to **skip, filter, or apply custom logic** while looping through rows.

3. Using Cursor Attributes:

- You may need attributes like:
  - cursor_name%ROWCOUNT
  - cursor_name%FOUND
  - cursor_name%NOTFOUND

These give you more info about the fetch status.

4. Complex Business Logic:

- When your logic involves **looping through rows**, checking conditions for each row, or performing multiple actions per row.

Rollback and Commit Savepoint:

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

Ans:-

➢ **What is a SAVEPOINT?**

- A **SAVEPOINT** is a marker or label set **inside a transaction** to which you can **roll back** without affecting the entire transaction.

Why Use SAVEPOINT?

- To **partially undo** changes in a transaction.
- Useful when you want to **test** or **validate** part of a transaction before completing everything.

Syntax:

SAVEPOINT savepoint_name;

How COMMIT & ROLLBACK Interact with SAVEPOINT:

Command	What It Does
SAVEPOINT	Sets a named point in the transaction
ROLLBACK TO	Undoes changes **after the savepoint** only
ROLLBACK	Undoes **entire** transaction (ignores savepoints)
COMMIT	Saves **all changes** and removes all savepoints

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

Ans:-

- SAVEPOINTs are helpful when you want to **control parts of a transaction** — especially in large or complex operations.

1. Partially Roll Back Changes

- If something goes wrong in **one part** of a transaction, you can undo **just that part** using a savepoint — instead of canceling the whole transaction.

2. Error Handling in Steps

- You can test part-by-part:
  - Insert → Savepoint
  - Update → Savepoint
  - If error: Roll back to last safe point

3. Complex Business Logic

- In operations involving **multiple steps**, SAVEPOINT lets you **go back** if only a specific condition fails — without restarting everything.

4. Nested Operations

- In stored procedures or triggers that perform many steps, savepoints let you keep valid changes and undo only the failed step.

5. User Confirmation

- In multi-step processes where user approval is needed in between steps, you can:
  - Perform step 1 → SAVEPOINT sp1
  - Wait for approval
  - If rejected → ROLLBACK TO sp1