Module 4 – Introduction to DBMS

(Theory practical)

1. Introduction to SQL

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

SQL stands for Structured Query Language.
It is a special language used to communicate with databases.

Why it is important:

- It helps you store, change, delete, and read data in a database.
- It allows developers and users to manage and organize large amounts of data easily.
- It is used in almost all modern database systems, like MySQL, Oracle, SQL Server, etc.

---

2. Explain the difference between DBMS and RDBMS.

Feature	DBMS (Database Management System)	RDBMS (Relational Database Management System)
Data Storage	Stores data in files or tables	Stores data in related tables (rows & columns)
Relationship	No relationship between data	Supports relationships using keys (primary, foreign)
Example	MS Access, XML	MySQL, Oracle, SQL Server
Data Integrity	Less secure	More secure with constraints
Normalization	Not supported	Supported

In simple words:

- DBMS is a general system for handling data.
- RDBMS is an advanced version that follows relational models and uses SQL.

---

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

SQL helps manage relational databases by:

- Creating tables and setting rules (using CREATE, ALTER, etc.)
- Adding or updating data (using INSERT, UPDATE, DELETE)
- Finding specific data using SELECT
- Setting permissions for users

- Creating relationships between tables using primary and foreign keys

4. What are the key features of SQL?

Here are the main features of SQL:

1. ✅ Data Querying – Get data from the database using SELECT.
2. 🖍️ Data Manipulation – Add, update, or delete data using INSERT, UPDATE, DELETE.
3. 📐 Data Definition – Create and modify tables using CREATE, ALTER, DROP.
4. 🔐 Data Control – Control access using GRANT and REVOKE.
5. 🔍 Filtering and Searching – Use WHERE, ORDER BY, GROUP BY, LIKE, etc.
6. 📊 Supports Relational Databases – Works with tables having relationships.
7. 🗃️ Handles Large Data – Can manage huge amounts of data efficiently.
8. 🌐 Standard Language – Used in all major RDBMS like MySQL, Oracle, PostgreSQL.

---

2. SQL Syntax

1. What are the basic components of SQL syntax?

The basic components of SQL syntax are the keywords and rules used to write SQL commands.

Here are the main components:

1. SQL Keywords – Words like SELECT, FROM, WHERE, INSERT, UPDATE, etc.
2. Identifiers – Names of tables, columns, and databases.
3. Operators – Symbols like =, >, <, AND, OR used in conditions.
4. Clauses – Parts of SQL statements (e.g., WHERE, GROUP BY, ORDER BY).
5. Statements – Complete SQL commands (e.g., SELECT * FROM students;)
6. Literals – Actual values (like 'Jay', 25, 'Ahmedabad')
7. Comments – Notes for developers (e.g., -- This is a comment)

---

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

The basic structure of an SQL SELECT statement is:

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

Example :-

```sql
SELECT name, age
FROM students
WHERE age > 18
ORDER BY name ASC;
```

---

3. Explain the role of clauses in SQL statements.

Clauses are the building blocks of SQL statements.
They help to filter, group, and sort data in a more useful way.

Common clauses and their roles:

Clause	Use / Role
SELECT	Choose which columns to show
FROM	Select the table to get data from
WHERE	Filter rows based on a condition
GROUP BY	Group rows with the same value
HAVING	Filter groups (used after GROUP BY)
ORDER BY	Sort the result by one or more columns

Example:

```sql
SELECT city, COUNT(*)
FROM customers
GROUP BY city
HAVING COUNT(*) > 5
ORDER BY city;
```

This example:

- Selects cities,
- Counts how many customers are in each city,
- Only shows cities with more than 5 customers,
- Then sorts them.

3. SQL Constraints

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

Constraints in SQL are rules applied to table columns to make sure the data is correct and reliable.

They help in:

- Preventing wrong data from being entered.
- Maintaining accuracy and consistency in the database.

Types of SQL Constraints:

Constraint	Meaning
NOT NULL	The column must have a value, it cannot be empty.
UNIQUE	All values in the column must be different.
PRIMARY KEY	Uniquely identifies each row. Combines NOT NULL + UNIQUE.
FOREIGN KEY	Connects two tables. It refers to the primary key of another table.
CHECK	Ensures that values meet a specific condition.
DEFAULT	Sets a default value if no value is given during insert.

---

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

Feature	PRIMARY KEY	FOREIGN KEY
Purpose	Uniquely identifies each row in a table	Links one table to another
Value	Must be unique and not null	Can have duplicate or null values
Location	Defined in the main (parent) table	Defined in the related (child) table
Example	Student ID in students table	Student ID in marks table refers to students table

In simple words:

- Primary Key = Identity of a record in its own table.
- Foreign Key = Connects two tables using the primary key of the other table.

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

- NOT NULL:
  Ensures that a column must always have a value.
  You cannot leave it empty when inserting data.

Example:
If a column name is NOT NULL, you must enter a name when inserting data.

- UNIQUE:
  Ensures that all values in a column are different.
  No two rows can have the same value in this column.

Example:
If the column email is UNIQUE, two users cannot register with the same email.

---

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

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

DDL stands for Data Definition Language.
It includes SQL commands that are used to create, change, or delete the structure of database objects like tables, views, indexes, etc.

Main DDL commands:

Command	Purpose
CREATE	To create a new table or database
ALTER	To change the structure of an existing table (like adding/removing columns)
DROP	To delete a table or database
TRUNCATE	To remove all data from a table, but keep the structure

---

2. Explain the CREATE command and its syntax.

The CREATE command is used to create new tables, databases, or other objects in the database.

Syntax to create a table:

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

Example :-

```
CREATE TABLE students (
  student_id INT PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  age INT CHECK (age >= 5),
  email VARCHAR(100) UNIQUE
);
```

This command will:

- Create a table called students
- Add columns with names, data types, and constraints

---

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

Purpose of data types:

- Data types tell the database what kind of data can be stored in each column.
    - Example: INT for numbers, VARCHAR for text, DATE for dates.
- Helps to save storage space and prevent wrong data.

Purpose of constraints:

- Constraints help control the values entered into the table.
- They protect data accuracy and integrity.

Examples:

- NOT NULL ensures no empty values.
- UNIQUE avoids duplicate values.
- PRIMARY KEY uniquely identifies each record.

5. ALTER Command

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

The ALTER command in SQL is used to change the structure of an existing table without deleting the table or its data.

You can use it to:

- Add a new column
- Change (modify) the data type or size of a column
- Rename a column
- Delete (drop) a column
- Add or remove constraints
    - In short, ALTER lets you update the design of a table after it has already been created.

---

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

Here are the basic SQL commands for using ALTER to add, modify, or drop columns:

A. Add a column

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

Example :-

```sql
ALTER TABLE students
ADD phone_number VARCHAR(15);
```

B. Modify a column (change data type or size)

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

Example :-

```sql
ALTER TABLE students
MODIFY name VARCHAR(150);
```

C. Drop (delete) a column

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

Example :-

```sql
ALTER TABLE students
DROP COLUMN phone_number;
```

---

6. DROP Command

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

The DROP command in SQL is used to permanently delete:

- A table
- A database
- Or any other database object (like views, indexes, etc.)

➢ Once you use DROP, the object is completely removed from the database.

Example:

```sql
DROP TABLE students;
```

This command will delete the entire students table, including all its data and structure.

---

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

Dropping a table has serious and permanent effects:

Effect	Description
❌ Data Loss	All the data inside the table is deleted permanently.
❌ Structure Lost	The table structure (columns, constraints, etc.) is also deleted.
❌ Cannot be Recovered	Once dropped, you cannot undo it unless a backup exists.
⚠️ Related Tables Affected	If other tables use it (via foreign keys), you may get errors or break relationships.

---

7. Data Manipulation Language (DML)

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

DML stands for Data Manipulation Language.
It includes commands used to work with data stored in the database (inside tables).

- A. INSERT – Add new data to a table

Used to add a new row (record) into a table.

Syntax:

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

Example :-

```sql
INSERT INTO students (student_id, name, age)
VALUES (1, 'Jay', 20);
```

B. UPDATE – Change existing data

Used to modify data in one or more columns of a table.

Syntax:

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

Example :-

```sql
UPDATE students
SET age = 21
WHERE student_id = 1;
```

C. DELETE – Remove data from a table

Used to delete one or more rows from a table.

Syntax:

```sql
DELETE FROM table_name
WHERE condition;
```

Example :-

```sql
DELETE FROM students
WHERE student_id = 1;
```

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

The WHERE clause is very important when using UPDATE and DELETE because:

Reason	Explanation
🎯 Targets specific rows	It lets you choose which rows to update or delete.
🚫 Prevents unwanted changes	Without WHERE, all rows will be affected.
🔐 Protects data	Helps avoid accidental data loss or wrong updates.

Examples:

- Correct use:

```sql
DELETE FROM students WHERE student_id = 5;
```

Only deletes one student with ID 5.

- ❌ Without WHERE:

```sql
DELETE FROM students;
```

Deletes all rows in the table!

➢ In short:
Always use WHERE carefully to control which data is changed or removed.

---

8. Data Query Language (DQL)

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

The SELECT statement is the main command used to retrieve (query) data from one or more tables in a database.

❖ It is part of DQL (Data Query Language).

Basic Syntax:

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

Example :-

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

- This command shows the name and age of all students.

You can also use it with filters, sorting, grouping, etc., like:

- WHERE (to filter rows)
- ORDER BY (to sort)
- GROUP BY (to group rows)

---

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

- A. WHERE Clause – Filter specific rows
  - Used to show only those rows that match a condition.
  - Helps in getting only needed data.

Syntax:

```sql
SELECT * FROM table_name
WHERE condition;
```

Example:

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

Shows only students whose age is more than 18.

◇ B. ORDER BY Clause – Sort the result

- Used to sort rows in ascending (ASC) or descending (DESC) order.
- Can sort by one or more columns.

Syntax:

```sql
SELECT * FROM table_name
ORDER BY column_name ASC|DESC;
```

Example :-

```sql
SELECT * FROM students
ORDER BY name ASC;
```

Combined Example:

```sql
SELECT name, age
FROM students
WHERE age > 18
ORDER BY age DESC;
```

This command:

- Shows names and ages of students older than 18
- And sorts them by age from highest to lowest

9.Data Control Language (DCL)

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

DCL (Data Control Language) is used to control access (permissions) to data in a database.

- ➢ GRANT – Gives permission to users

- Used to allow a user to do specific actions like SELECT, INSERT, UPDATE, or DELETE on a table.

Example:

```sql
GRANT SELECT, INSERT ON students TO jay;
```

Gives user jay the permission to view and add data to the students table.

- ➢ REVOKE – Takes away permission

- Used to remove previously given permissions.

Example:

```sql
REVOKE INSERT ON students FROM jay;
```

- ❖ Removes the insert permission from user jay on the students table.

---

2. How do you manage privileges using these commands?

You can manage user privileges in two ways:

- ◇ A. Giving privileges (Using GRANT)

- Decide which user should have access
- Choose what actions they can perform
- Use GRANT to allow actions like SELECT, UPDATE, DELETE, etc.

Example:

```sql
GRANT SELECT, UPDATE ON employee TO 'admin_user';
```

◇ B. Removing privileges (Using REVOKE)

- If you no longer want a user to perform an action
- Use REVOKE to remove that permission

Example:

```sql
REVOKE UPDATE ON employee FROM 'admin_user';
```

In short:

- GRANT = Give access
- REVOKE = Remove access

These are useful for data security and user control in multi-user environments.

---

10. Transaction Control Language (TCL)

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

TCL (Transaction Control Language) commands are used to manage changes made during a transaction in SQL.

◇ COMMIT – Saves the changes

- Used to permanently save all the changes made by INSERT, UPDATE, or DELETE.
- Once you use COMMIT, the changes cannot be undone.

Example:

```sql
UPDATE students SET age = 20 WHERE student_id = 1;
COMMIT;
```

◇ ROLLBACK – Undo the changes

- Used to cancel or undo changes made during a transaction before they are committed.
- It returns the database to the previous safe state.

Example:

```sql
UPDATE students SET age = 25 WHERE student_id = 1;
ROLLBACK;
```

- This will undo the age update and keep the original value

2. EXPLAIN HOW TRANSACTIONS ARE MANAGED IN SQL DATABASES.

A transaction is a group of one or more SQL commands that are executed together as a single unit.

A transaction must follow 4 properties (ACID):

Property	Meaning
A – Atomicity	All commands in the transaction must complete successfully, or none at all.
C – Consistency	Keeps the database in a valid state before and after the transaction.
I – Isolation	Transactions do not affect each other until they are complete.
D – Durability	Once committed, changes are permanent, even after a system crash.

Transaction Flow Example:

```sql
START TRANSACTION;  -- or BEGIN;
UPDATE account SET balance = balance - 500 WHERE acc_no = 1;
UPDATE account SET balance = balance + 500 WHERE acc_no = 2;
COMMIT;  -- if both successful
-- or ROLLBACK;  -- if any error
```

- This ensures that money is transferred safely, and if anything fails, all changes are rolled back.

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?

- What is a JOIN?

A JOIN is used in SQL to combine data from two or more tables based on a related column (usually a primary key–foreign key relationship).

Types of Joins:

Join Type	Description	Visual Meaning
INNER JOIN	Returns only matching rows from both tables.	Only the common part
LEFT JOIN (or LEFT OUTER JOIN)	Returns all rows from the left table, and matching rows from the right table. If no match, right table columns show NULL.	All from left, matched from right
RIGHT JOIN (or RIGHT OUTER JOIN)	Returns all rows from the right table, and matching rows from the left table. If no match, left table columns show NULL.	All from right, matched from left
FULL OUTER JOIN	Returns all rows from both tables. Shows NULL where no match is found.	All from both sides

Examples:

Suppose you have two tables:

```sql
-- Table: students
student_id | name
-----------|-------
1          | Jay
2          | Rahul
3          | Priya

-- Table: marks
student_id | marks
-----------|------
1          | 85
2          | 78
4          | 90
```

- INNER JOIN:

```sql
SELECT s.name, m.marks
FROM students s
INNER JOIN marks m ON s.student_id = m.student_id;
```

  - Only students with matching IDs in both tables (Jay, Rahul)

- LEFT JOIN:

```sql
SELECT s.name, m.marks
FROM students s
LEFT JOIN marks m ON s.student_id = m.student_id;
```

All students + matching marks
Priya will have NULL in marks.

- RIGHT JOIN:

```sql
SELECT s.name, m.marks
FROM students s
RIGHT JOIN marks m ON s.student_id = m.student_id;
```

All marks entries + matching names
ID 4 will show NULL for name.

- FULL OUTER JOIN: (not supported in MySQL directly without UNION)

➡ Combines all rows from both tables
Missing values are shown as NULL.

---

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

Joins combine rows from two or more tables using a related column (like a primary key and foreign key).

- ◇ Why we use joins:
  - To see complete information spread across multiple tables.
  - To link related data (e.g., students with their marks or customers with orders).
  - To perform complex queries from different sources in the database.
- ◇ How it works:

```sql
SELECT students.name, marks.marks
FROM students
JOIN marks ON students.student_id = marks.student_id;
```

This JOIN:

- Takes data from students table and marks table
- Matches rows where student_id is the same in both
- Returns combined result in one table view

---

12. SQL Group By

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

- ◇ What is GROUP BY?

The GROUP BY clause is used to group rows that have the same values in a column.
It is mainly used with aggregate functions like:

- COUNT() – total number
- SUM() – total sum
- AVG() – average
- MAX() – highest value
- MIN() – lowest value

- ◇ How it works:
  - It collects rows into groups based on one or more columns.

- Then it applies the aggregate function to each group.

◇ Example:

Table: orders

customer	product	amount
Jay	Pen	10
Jay	Book	50
Priya	Pencil	5
Priya	Pen	15

Query:

```sql
SELECT customer, SUM(amount)
FROM orders
GROUP BY customer;
```

This groups the data by customer name, and then shows the total amount per customer.

Result:

customer	SUM(amount)
Jay	60
Priya	20

---

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

Feature	GROUP BY	ORDER BY
Purpose	Groups rows based on column values	Sorts the result rows

Feature	GROUP BY	ORDER BY
Used With	Aggregate functions like SUM(), COUNT()	Can be used with any columns
Output	One result per group	All rows, just sorted
Order Control	Does not sort the data unless combined with ORDER BY	Controls sorting (ASC/DESC)

- Example of GROUP BY:

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

  - Counts how many students are in each city

- Example of ORDER BY:

```sql
SELECT name, age
FROM students
ORDER BY age DESC;
```

  - Sorts students by age from highest to lowest.

- Use Both Together:

```sql
SELECT city, COUNT(*)
FROM students
GROUP BY city
ORDER BY COUNT(*) DESC;
```

  - Groups students by city, then sorts by student count in descending order.

13. SQL Stored Procedure

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

Definition:

A stored procedure is a group of SQL statements that are saved in the database and can be executed as a single unit.

It is like a function that performs a task, and we can call it many times whenever we need it.

Syntax Example:

```sql
CREATE PROCEDURE ShowAllStudents()
BEGIN
    SELECT * FROM Students;
END;
```

Difference from standard SQL query:

Standard SQL Query	Stored Procedure
One-time command	Saved in database
Run manually each time	Can be reused easily
Can’t accept parameters	Can accept input/output parameters
Less secure	More secure with controlled access

---

2. Explain the advantages of using stored procedures.

Advantages:

1. Reusability: You can use the same stored procedure many times without writing the same code again.
2. Faster Execution: Stored procedures are precompiled, so they run faster than normal queries.
3. Security: Permissions can be given to users to execute the procedure without allowing direct table access.
4. Reduced Network Traffic: Since logic is executed in the database, fewer commands are sent from the application.
5. Easy Maintenance: If you want to update logic, change it in the procedure instead of updating every part of the application.

14. SQL View

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

Definition:

A view in SQL is a virtual table. It is based on the result of a SQL query.

- It does not store data itself.
- It shows data from one or more tables.
- You can use it like a table in SELECT queries.

Syntax Example:

```sql
CREATE VIEW ActiveStudents AS
SELECT name, class
FROM Students
WHERE status = 'Active';
```

This view will always show only active students.

Difference between View and Table:

Feature	Table	View
Storage	Stores actual data	Does not store data (virtual only)
Data Update	You can insert/update/delete	Some views allow, some don't
Performance	Fast (direct access to data)	Slightly slower (runs query every time)
Based On	Independent	Based on table(s)
Usage	Core data source	Simplifies complex queries

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

Advantages of Views:

1. Simplify Complex Queries
    - You can write a long query once and save it as a view. Then use it like a table.
2. Security
    - Hide specific columns or rows from users. Give access to view, not the actual table.
3. Reusability
    - Use the same view in multiple reports or queries.
4. Logical Separation
    - Keep data logic (filters, joins) in views to keep queries clean.
5. Data Abstraction
    - Users can work with simplified data models without knowing table relationships.

---

15. SQL Triggers

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

Definition:

A trigger in SQL is a special stored program that automatically runs (fires) when a specific action (like INSERT, UPDATE, or DELETE) happens on a table.

- You don't call a trigger directly.
- It works automatically when conditions are met.

Types of Triggers in SQL:

Trigger Type	Description	When it is Used
BEFORE INSERT	Runs before new data is inserted into a table	To validate or modify data before saving it
AFTER INSERT	Runs after new data is inserted	To log the insert or update another table
BEFORE UPDATE	Runs before data is updated	To check new values or restrict unwanted updates
AFTER UPDATE	Runs after data is updated	To log changes or update audit tables

Trigger Type	Description	When it is Used
BEFORE DELETE	Runs before data is deleted	To prevent deletion based on some condition
AFTER DELETE	Runs after data is deleted	To keep records of deleted data (e.g., in an archive table)

Example (AFTER INSERT Trigger):

```sql
CREATE TRIGGER log_student_insert
AFTER INSERT ON Students
FOR EACH ROW
INSERT INTO StudentLogs (name, action)
VALUES (NEW.name, 'Inserted');
```

---

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

Trigger Type	When it Fires	Example Use Case
INSERT	When new row(s) are added	Log who added data, assign default values
UPDATE	When existing row(s) are changed	Keep track of old and new values, validate input
DELETE	When row(s) are removed from a table	Save deleted data into backup table

Summary Table:

Feature	INSERT Trigger	UPDATE Trigger	DELETE Trigger
Action Triggered	Row is added	Row is changed	Row is removed
Typical Usage	Logging, defaults	Validation, auditing	Archiving, restrictions

16. Introduction to PL/SQL

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

Answer: PL/SQL (Procedural Language for SQL) is an extension of SQL used mainly in Oracle databases. It allows you to write programming logic (like conditions, loops, and variables) along with SQL statements.

- How it extends SQL:
  - SQL is used only to query and manipulate data, but it doesn't support complex programming logic.
  - PL/SQL supports features like:
    - Variables
    - Loops (for, while)
    - Conditions (if-else)
    - Procedures and Functions
    - Error handling

This makes PL/SQL powerful for creating complex applications in the database itself.

---

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

Answer:

Here are some key benefits of PL/SQL:

Benefit	Explanation
Block structure	PL/SQL uses blocks, which helps in organizing code in logical sections like declaration, execution, and exception handling.
Better performance	PL/SQL code runs directly on the database server, so it's faster than sending many SQL statements one by one.
Supports procedural logic	You can use loops, if-else, and functions like in other programming languages.
Error handling	PL/SQL allows you to catch and handle errors gracefully using EXCEPTION blocks.
Reusable code	You can write stored procedures and functions, which can be reused in different parts of your application.
Security	PL/SQL allows you to give restricted access using procedures instead of giving full access to data.

17. PL/SQL Control Structures

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

Control Structures in PL/SQL:

Control structures in PL/SQL are programming constructs that control the flow of execution of statements based on conditions, loops, or branches. They allow developers to:

- Execute code conditionally.
- Repeat code multiple times.
- Branch program logic based on decisions.

There are three main types:

1. Conditional Control (IF statements)
2. Iterative Control (LOOP statements)
3. Sequential Control (GOTO, EXIT, etc.)

IF-THEN Control Structure:

The IF-THEN statement is used to execute a block of code only if a condition is true.

Syntax:

```
IF condition THEN
   -- statements to execute if condition is true
END IF;
```

Example:

```
DECLARE
   v_salary NUMBER := 3000;
BEGIN
   IF v_salary > 2500 THEN
      DBMS_OUTPUT.PUT_LINE('High Salary');
   END IF;
END;
```

Output:

```
High Salary
```

LOOP Control Structure:

A LOOP repeatedly executes a block of code until an exit condition is met.

Syntax:

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

Example:

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

Output:

```
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
```

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

Control structures **add procedural logic** to SQL, making it possible to:

- **Handle conditional execution:** Different actions can be taken depending on data (e.g., bonuses only for employees above a certain salary).
- **Perform repetitive tasks:** Loops allow iterating over records, performing updates, or calculations.
- **Implement decision-making:** Helps in branching logic (e.g., IF…ELSE to choose different paths).
- **Automate complex business rules:** They make it easier to embed business logic directly in the database.
- **Reduce manual work:** Instead of writing multiple queries manually, loops and conditions automate them.

---

18. SQL Cursors

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

What is a Cursor?

A **cursor** in PL/SQL is a pointer (or a handle) to a memory area that stores the result set of a query. It allows the program to **fetch rows one by one** and process them.

Types of Cursors:

There are **two main types**:

(A) Implicit Cursor:

- Automatically created by PL/SQL **when a single SQL statement (SELECT, INSERT, UPDATE, DELETE) is executed**.
- It is used for **simple queries** that return only one row or do not need to be processed row-by-row.
- Managed by Oracle; the programmer does **not** explicitly declare or control it.

Example:

```
DECLARE
   v_count NUMBER;
BEGIN
   SELECT COUNT(*) INTO v_count FROM employees;
   DBMS_OUTPUT.PUT_LINE('Total employees: ' || v_count);
END;
```

Here, PL/SQL internally creates an **implicit cursor**.

(B) Explicit Cursor:

- Declared and managed **manually** by the programmer.
- Used when the query **returns multiple rows**, and we need to **fetch and process each row individually**.
- Requires four steps: **DECLARE → OPEN → FETCH → CLOSE**.

Example:

```
DECLARE
   CURSOR emp_cursor IS
       SELECT first_name, salary FROM employees;
   v_name employees.first_name%TYPE;
   v_salary employees.salary%TYPE;
BEGIN
   OPEN emp_cursor;
   LOOP
      FETCH emp_cursor INTO v_name, v_salary;
      EXIT WHEN emp_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(v_name || ' earns ' || v_salary);
   END LOOP;
   CLOSE emp_cursor;
END;
```

Feature	Implicit Cursor	Explicit Cursor
Creation	Automatically by Oracle	Manually by programmer
Row Processing	For single-row queries	For multi-row queries
Control	No manual control	Full control (open, fetch, close)
Performance	Faster for simple queries	Needed for complex processing

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

You use an **explicit cursor** when:

- The query **returns multiple rows**.
- You need to **process each row individually** (e.g., calculations, conditional logic).
- You want **manual control** over the fetching process.
- You need to use **cursor attributes** like %ROWCOUNT, %FOUND, %NOTFOUND in a loop.

---

19. Rollback and Commit Savepoint

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

SAVEPOINT:

A **SAVEPOINT** is a marker within a database transaction that allows partial rollbacks.
It helps you set intermediate points in a transaction so you can undo (ROLLBACK) changes only up to that point instead of undoing the entire transaction.

How ROLLBACK works with SAVEPOINT:

- ROLLBACK TO savepoint_name; undoes all changes **after** the specified savepoint but **keeps changes made before it**.
- If you do a simple ROLLBACK; without specifying a savepoint, the entire transaction is undone.

How COMMIT works with SAVEPOINT:

- COMMIT; **makes all changes permanent** in the database.
- Once committed, **all savepoints are erased** because the transaction is finished.

Example:

```
BEGIN
   INSERT INTO employees VALUES (1, 'John');
   SAVEPOINT sp1;

   INSERT INTO employees VALUES (2, 'Sara');
   SAVEPOINT sp2;

   DELETE FROM employees WHERE id = 1;

   ROLLBACK TO sp2; -- Undo delete but keep insert of John and Sara
   COMMIT;          -- Permanently save both inserts
END;
```

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

Savepoints are useful when:

- You want to **partially undo** changes without rolling back the entire transaction.
- You are executing **complex transactions** involving multiple steps and want the ability to revert only the last few steps if something goes wrong.
- You want **better control** over error handling in long scripts.
- You need to implement **conditional rollbacks** depending on business logic.
