Module-4: Introduction to DBMS

Theory exercise

❖ 1.INTRODUCTION TO SQL:-

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

Ans.

SQL (Structured Query Language) is a standard language used to store, manage, and retrieve data from a database.

It is essential in database management because it helps in:

- Adding, updating, and deleting data
- Creating and managing database structures
- Querying data quickly and efficiently

→ 2. Explain the difference between DBMS and RDBMS.

Ans.

DBMS (Database Management System) is software used to store and manage data. It does not support relationships between data.

RDBMS (Relational Database Management System) is a type of DBMS that stores data in tables and supports relationships using primary and foreign keys.

Key Differences:

- DBMS: Data stored in files or simple formats

- RDBMS: Data stored in related tables
- DBMS: No data integrity rules
- RDBMS: Follows relational rules (ACID properties)

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

Ans.

SQL (Structured Query Language) is used to create, manage, and interact with relational databases.

It helps in:

- Creating tables and structures
- Inserting, updating, deleting data
- Retrieving data using queries
- Controlling access to data (permissions)

→ 4. What are the key features of SQL?

Ans.

Here are the key features of SQL:

1. Data Querying – Retrieve data using SELECT statements
2. Data Manipulation – Add, update, or delete data (INSERT, UPDATE, DELETE)

3. Data Definition – Create or modify tables and schemas (CREATE, ALTER, DROP)
4. Data Control – Control access with GRANT and REVOKE
5. Transaction Control – Ensure safe and reliable changes (COMMIT, ROLLBACK)
6. Standardized Language – Used across all major relational databases (e.g., MySQL, Oracle)

❖ 2.SQL SYNTAX-

→1. What are the basic components of SQL syntax?

Ans.

The basic components of SQL syntax include:

1. Statements – Instructions like SELECT, INSERT, UPDATE, DELETE
2. Clauses – Parts of a statement, like WHERE, FROM, ORDER BY
3. Expressions – Conditions or calculations (e.g., price > 100)
4. Predicates – Used in conditions to filter rows (e.g., WHERE age > 18)
5. Identifiers – Names of tables, columns, or databases
6. Operators – Symbols like =, >, <, AND, OR used in expressions.

→2.

Ans.

General Structure of an SQL SELECT Statement:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
GROUP BY column
HAVING condition
ORDER BY column ASC|DESC;
-- Explanation of each part:
```

- SELECT – Specifies the columns to retrieve
- FROM – Specifies the table to get data from
- WHERE – Filters rows based on a condition (optional)
- GROUP BY – Groups rows that have the same values (optional)
- HAVING – Filters groups (used with GROUP BY, optional)
- ORDER BY – Sorts the result in ascending or descending order (optional)

Example:

```
SELECT name, age
FROM students
```

WHERE age > 18

ORDER BY age DESC;

This will select students older than 18 and sort them by age in descending order.

→3. Explain the role of clauses in SQL statements

Ans.

Clauses are parts of an SQL statement that help define what to do with the data.

Common Clauses and Their Roles:

- SELECT – Chooses which columns to show
- FROM – Specifies the table to get data from
- WHERE – Filters rows based on a condition
- GROUP BY – Groups rows that have the same values
- HAVING – Filters grouped data
- ORDER BY – Sorts the result

In short, clauses control how data is selected, filtered, grouped, and sorted in SQL queries.

❖ Que-3. SQL CONSTRAINTS:-

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

Ans.

Constraints in SQL are rules applied to table columns to control the type of data that can be inserted. They ensure data accuracy, consistency, and integrity.

→ Types of SQL Constraints:

1. NOT NULL
    - Ensures a column cannot have NULL values
    - Example: name VARCHAR(50) NOT NULL
2. UNIQUE
    - Ensures all values in a column are different
    - Example: email VARCHAR(100) UNIQUE
3. PRIMARY KEY
    - Uniquely identifies each row in a table
    - Combines NOT NULL + UNIQUE
    - Example: id INT PRIMARY KEY
4. FOREIGN KEY
    - Links one table to another
    - Ensures data in one table matches values in another

- Example: student_id INT REFERENCES students(id)

5. CHECK
   - Ensures that a value meets a specific condition
   - Example: age INT CHECK (age >= 18)
6. DEFAULT
   - Sets a default value if no value is provided
   - Example: status VARCHAR(10) DEFAULT 'active'

These constraints help maintain data quality and enforce business rules in a database

---

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

Ans.

→ PRIMARY KEY:

- Uniquely identifies each row in a table
- No duplicate or NULL values allowed
- Only one primary key per table
- Example: id INT PRIMARY KEY

→ FOREIGN KEY:

- Links one table to another
- Refers to the primary key in another table
- Can have duplicate values and NULLs (if allowed)

- Ensures referential integrity (data must exist in the referenced table)
- Example: student_id INT REFERENCES students(id)

Summary:

- Primary Key = Unique ID within a table
- Foreign Key = Reference to another table's primary key

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

Ans.

→ NOT NULL:

- Ensures that a column cannot have NULL (empty) values
- It forces data to be entered in that field
- Example: name VARCHAR(50) NOT NULL → name must be provided

→ UNIQUE:

- Ensures that all values in a column are different
- No duplicate values are allowed
- Example: email VARCHAR(100) UNIQUE → each email must be unique.

In short:

- NOT NULL = Value must be present
- UNIQUE = Value must not repeat

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

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

Ans.

DDL (Data Definition Language) is a set of SQL commands used to define and manage database structures, such as tables, schem

as, and indexes.

→ Common DDL Commands:

- CREATE – To create a new table or database
- ALTER – To modify an existing table
- DROP – To delete a table or database
- TRUNCATE – To delete all data from a table quickly

In short: DDL helps in defining, changing, or removing the structure of database objects.

→2. Explain the CREATE command and its syntax.

Ans. The CREATE command in SQL is used to create new database objects like tables, databases, views, etc.

→ Most common usage: CREATE TABLE

- Syntax:

```
CREATE TABLE table_name (
    column1 datatype constraint,
```

column2 datatype constraint,

...

);

→Example:

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

In short:
The CREATE command defines the structure of a new table or database in SQL.

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

Ans.

→Purpose of Data Types:

- Define what kind of data a column can hold (e.g., INT, VARCHAR, DATE)
- Helps in efficient storage and data validation

→ Purpose of Constraints:

- Apply rules to data (e.g., NOT NULL, UNIQUE, PRIMARY KEY)
- Ensure data accuracy, integrity, and consistency

In short:
Specifying data types and constraints helps to store valid, correct, and meaningful data in the table.

❖ 5. ALTER COMMAND:-

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

Ans.

The ALTER command is used to modify the structure of an existing table in a database.

It can be used to:

- Add new columns
- Modify existing columns
- Rename columns or the table itself
- Drop (delete) columns

→Example:

ALTER TABLE students ADD gender VARCHAR(10);

→ Adds a new column named gender to the students table.

In short:
ALTER helps to change the structure of a table without deleting it.

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

Ans.

The ALTER TABLE command is used to add, modify, or drop columns in an existing table.

→1. Add a Column:

ALTER TABLE table_name ADD column_name datatype;

Example:

ALTER TABLE students ADD gender VARCHAR(10);

→2. Modify a Column:

ALTER TABLE table_name MODIFY column_name new_datatype;

Example:

ALTER TABLE students MODIFY age INT;

→3. Drop a Column:

ALTER TABLE table_name DROP COLUMN column_name;

Example:

ALTER TABLE students DROP COLUMN gender;

In short:
Use ALTER TABLE to change the structure of a table by adding, changing, or removing columns.

❖ 6. DROP COMMAND:-

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

Ans.

The DROP command in SQL is used to permanently delete a database object like a table, database, view, or index.

→ It:

- Removes the structure and all data in the object
- Cannot be undone, so use it carefully

→Example:

DROP TABLE students;

→ This deletes the entire students table and all its data.

In short:
DROP is used to completely remove tables or databases from the system.

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

Ans.

When you drop a table, it is permanently deleted from the database along with:

- All data stored in the table
- The table structure (columns, data types, etc.)
- Any constraints, indexes, or relationships linked to the table

→ Important Notes:

- The action cannot be undone
- Other tables that depend on it (via foreign keys) may also be affected
- You may lose important data if not backed up

In short:
Dropping a table means permanent loss of both data and structure, so use it with caution.

❖7. Data Manipulation Language (DML):-

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

Ans.

These are Data Manipulation Language (DML) commands used to manage data in tables.

→ INSERT

- Adds new data (rows) into a table.
  Example:

```
INSERT INTO students (id, name, age) VALUES (1, 'Yogesh', 21);
```

→ UPDATE

- Modifies existing data in a table.
  Example:

```
UPDATE students SET age = 22 WHERE id = 1;
```

DELETE

- Removes data from a table.
  Example:

DELETE FROM students WHERE id = 1;

In short:

- INSERT = Add data
- UPDATE = Change data
- DELETE = Remove data

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

Ans.

The WHERE clause is used to specify which rows should be updated or deleted.

→Importance:

- It prevents all rows from being affected by mistake
- Ensures that only the correct records are changed or removed

→Example:

UPDATE students SET age = 22 WHERE id = 1;

→ Only updates the row where id is 1

DELETE FROM students WHERE age < 18;

→ Deletes only students younger than 18

→Without WHERE:

DELETE FROM students;

→ Deletes all rows in the table!

In short:
WHERE clause is essential for safe and targeted changes to data.

❖ 8. Data Query Language (DQL):-

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

Ans.

The SELECT statement is used in SQL to retrieve data from one or more tables in a database.

→Basic Syntax:

SELECT column1, column2 FROM table_name;

Example:

SELECT name, age FROM students;

→ This gets the name and age columns from the students table.

→You can also use:

- WHERE – to filter rows
- ORDER BY – to sort results
- GROUP BY – to group similar data

In short:
The SELECT statement is the main tool to view data stored in a database.

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

Ans.

→WHERE Clause:

- Used to filter records based on a condition
- Returns only the rows that match the condition

Example:

SELECT * FROM students WHERE age > 18;

→ Shows only students older than 18

→ORDER BY Clause:

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

Example:

SELECT * FROM students ORDER BY age DESC;

→ Shows students sorted by age, highest first

In short:

- WHERE = Filters data
- ORDER BY = Sorts data

❖ 9. Data Control Language (DCL):-

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

Ans.

→ GRANT

- Used to give permissions to users to perform specific actions (like SELECT, INSERT, UPDATE) on database objects. Example:

GRANT SELECT, INSERT ON students TO user1;

→ REVOKE

- Used to take back permissions previously given to a user. Example:

REVOKE INSERT ON students FROM user1;

In short:

- GRANT = Allows user actions
- REVOKE = Removes user permissions

→ 2. How do you manage privileges using these commands?

Ans.

You manage user access and permissions in SQL using the GRANT and REVOKE commands.

→Using GRANT:

- To give permissions to a user on a database object
  Syntax:

GRANT permission(s) ON object TO user;

Example:

GRANT SELECT, INSERT ON students TO user1;

→ Allows user1 to read and insert data into the students table

→Using REVOKE:

- To remove permissions from a user
  Syntax:

REVOKE permission(s) ON object FROM user;

Example:

REVOKE INSERT ON students FROM user1;

→ Removes the ability of user1 to insert data into the students table

In short:
Use GRANT to assign privileges and REVOKE to remove them, helping to control who can access or modify data in the database.

❖ 10. Transaction Control Language (TCL):-

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

Ans.

→COMMIT

- Saves all the changes made during the current transaction
- Changes become permanent in the database

Example:

```
UPDATE students SET age = 20 WHERE id = 1;
COMMIT;
```

→ROLLBACK

- Cancels all changes made during the current transaction
- Returns the database to its previous state

Example:

```
UPDATE students SET age = 20 WHERE id = 1;
ROLLBACK;
```

In short:

- COMMIT = Save changes permanently
- ROLLBACK = Undo changes if something goes wrong

→ 2. Explain how transactions are managed in SQL databases.

Ans.

A transaction in SQL is a group of operations that are treated as a single unit of work. It ensures that either all operations succeed or none do, maintaining data accuracy.

→Transaction Management Commands:

1. BEGIN / START TRANSACTION – Starts a new transaction
2. COMMIT – Saves all changes made in the transaction
3. ROLLBACK – Cancels all changes if there's an error
4. SAVEPOINT – Sets a point to roll back to (optional use)

→Example:

START TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;

UPDATE accounts SET balance = balance + 100 WHERE id = 2;

COMMIT;

→ Both updates succeed and changes are saved. If any update fails, use ROLLBACK.

In short:
SQL manages transactions to keep data consistent, accurate, and safe, especially when multiple changes happen together.

❖ 11. SQL JOIN:-

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

Ans.

→ JOIN in SQL:

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

→Types of JOINs:

JOIN Type	Description
INNER JOIN	Returns only the matching rows from both tables
LEFT JOIN	Returns all rows from the left table, and matching rows from the right
RIGHT JOIN	Returns all rows from the right table, and matching rows from the left
FULL OUTER JOIN	Returns all rows from both tables, with NULLs where there's no match

→ Example:

Two tables:

- students(id, name)
- marks(student_id, score)

→ INNER JOIN

SELECT * FROM students

INNER JOIN marks ON students.id = marks.student_id;

→ Returns students who have marks.

→ LEFT JOIN

SELECT * FROM students

LEFT JOIN marks ON students.id = marks.student_id;

→ Returns all students, even if they don't have marks

In short:

- INNER JOIN = Only matches
- LEFT JOIN = All from left + matches from right
- RIGHT JOIN = All from right + matches from left
- FULL OUTER JOIN = All from both sides + NULL where no match

---

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

Ans.

JOINs are used in SQL to combine data from two or more tables based on a related column, usually a primary key and foreign key.

→How JOINs work:

- Match rows from one table with rows in another table
- Use a common column to link them (like id, student_id, etc.)

→Example:

Two tables:

- students(id, name)
- marks(student_id, score)

SELECT students.name, marks.score

FROM students

JOIN marks ON students.id = marks.student_id;

→ This combines the students and marks tables where id = student_id.

In short:
JOINs are used to fetch related data from different tables in one result.

❖12. SQL Group By :-

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

Ans.

→ GROUP BY clause:

The GROUP BY clause is used to group rows that have the same values in specified columns.

It is commonly used with aggregate functions like COUNT(), SUM(), AVG(), MAX(), and MIN() to perform calculations on each group.

→Example:

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

→ This counts how many employees are in each department.

In short:
GROUP BY groups data, and aggregate functions calculate summary results for each group.

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

Ans.

Feature	GROUP BY	ORDER BY
Purpose	Groups rows with the same values	Sorts the result in ascending/descending order
Used with	Aggregate functions (COUNT, SUM, etc.)	Any column or expression
Output	One row per group	All rows, just sorted

→Examples:

→ GROUP BY (for grouping):

SELECT department, COUNT(*)

FROM employees

GROUP BY department;

→ ORDER BY (for sorting):

SELECT name, salary

FROM employees

ORDER BY salary DESC;

In short:

- GROUP BY = Group data for summaries
- ORDER BY = Sort data for display

❖ 13. SQL Stored Procedure:-

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

Ans.

Stored Procedure:

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

→Difference from standard SQL query:

Feature	Stored Procedure	Standard SQL Query
Definition	A reusable block of code saved in DB	A single SQL command or short script
Reusability	Can be executed multiple times easily	Must be written and executed every time
Parameters	Can accept input/output parameters	Usually has fixed values unless dynamic
Logic	Can include logic (IF, LOOP, etc.)	Typically just one operation (SELECT, etc.)

→Example:

```
-- Creating a stored procedure

CREATE PROCEDURE GetEmployees()
```

```
BEGIN
  SELECT * FROM employees;
END;


-- Calling it
CALL GetEmployees();
```

In short:
A stored procedure is like a saved function in SQL that lets you run complex operations quickly and repeatedly, unlike a one-time SQL query.

→2. Explain the advantages of using stored procedures.

Ans.

Advantages of Stored Procedures:

1. Reusability
   – Write once, use many times by just calling it.
2. Improved Performance
   – Executes faster since it's precompiled and stored in the database.
3. Security
   – You can control access using permissions, and hide the code from users.

4. Reduced Network Traffic
   – One call can perform multiple SQL operations, reducing client-server communication.
5. Modularity
   – Complex logic can be organized into manageable, reusable blocks.
6. Maintainability
   – Easy to update logic in one place without changing application code.

In short:
Stored procedures help make SQL faster, secure, reusable, and easier to manage.

❖ 14. SQL View:-

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

Ans.

→ View in SQL:

A view is a virtual table based on the result of a SELECT query. It does not store data itself, but shows data from one or more real tables.

→Difference between View and Table:

Feature	View	Table
Storage	Doesn't store data (virtual)	Stores actual data

Definition	Created from a SELECT query	Created with CREATE TABLE
Updatable	Limited (some views can't be updated)	Fully updatable
Use case	Used for simplifying complex queries, security	Used for storing and managing data

→Example:

-- View creation

CREATE VIEW high_salary AS

SELECT name, salary FROM employees WHERE salary > 50000;

In short:
A view is a saved SQL query that looks like a table but doesn’t store data itself. It helps simplify queries and control data access.

---

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

Ans.

Advantages of Views:

1. Simplifies Complex Queries
   – You can save a complex SELECT query as a view and use it like a table.

2. Improves Security
   – Views can limit access to specific columns or rows in a table.
3. Data Abstraction
   – Hides the complexity of the database structure from users.
4. Reusability
   – Views can be reused in multiple queries without rewriting the logic.
5. Logical Data Independence
   – You can change the underlying table without affecting how users access data via views.
6. Easier Maintenance
   – If business logic changes, update the view instead of updating multiple queries.

In short:
Views make SQL easier to use, safer, and more organized, especially when working with complex or sensitive data.

❖15. SQL Triggers :-

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

Ans.

A trigger is a special SQL object that automatically runs (fires) when a specific event occurs in a table, like INSERT, UPDATE, or DELETE.

Types of Triggers in SQL:

Trigger Type	Description
BEFORE Trigger	Executes before the data modification (INSERT/UPDATE/DELETE).
AFTER Trigger	Executes after the data modification.
INSTEAD OF Trigger	Used in views, replaces the usual action with a custom one.

When to Use:

- To automatically log changes to a table
- To enforce business rules (like validating data before insert)
- To prevent invalid actions (like blocking certain deletes)
- To sync data between related tables

Example:

```
CREATE TRIGGER log_insert

AFTER INSERT ON employees

FOR EACH ROW

INSERT INTO log_table(emp_id, action) VALUES (NEW.id, 'Inserted');
```

In short:
A trigger is like an automatic action that runs when data changes, helping enforce rules and automate tasks in the database.

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

Ans.

Triggers are activated automatically when a specific action occurs on a table. Based on the action, there are three main types:

→INSERT Trigger

- Fires when a new row is added to a table.
- Used to check values, log inserts, or update related tables.

Example Use: Log every new employee added to a system.

→UPDATE Trigger

- Fires when existing data is modified.
- Used to track changes, validate updates, or maintain history.

Example Use: Save old and new salary values when an employee's salary is updated.

→DELETE Trigger

- Fires when a row is removed from a table.
- Used to prevent deletion, log deleted records, or archive data.

Example Use: Move deleted records to a backup table before deletion.

→Summary Table:

Trigger Type	When It Fires	Common Uses
INSERT	On inserting a row	Logging, validation, update related tables
UPDATE	On updating a row	Auditing, validation, history tracking
DELETE	On deleting a row	Backup, restriction, audit deletion

→In short:
Each trigger type is tied to a specific data action — INSERT, UPDATE, or DELETE — and helps you automate and control what happens during those changes.

❖ 16. Introduction to PL/SQL:-

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

Ans.

PL/SQL (Procedural Language/Structured Query Language)

PL/SQL is Oracle's procedural extension of SQL. It adds programming features like loops, conditions, and variables to standard SQL.

→How PL/SQL Extends SQL's Capabilities:

Feature	SQL	PL/SQL Extension
Procedural logic	Not supported	Supports IF, LOOP, CASE, etc.
Variables	Not available	Can declare and use variables
Error handling	Limited	Provides powerful EXCEPTION handling
Modular code	No blocks	Supports procedures, functions, packages
Performance	Query-by-query	Can run multiple SQLs together in one block

Example:

```
BEGIN
  IF salary < 3000 THEN
    UPDATE employees SET salary = 3000 WHERE emp_id = 101;
```

END IF;

END;

In short:

PL/SQL combines SQL + programming to let you write powerful, secure, and efficient database programs

---

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

Ans.

PL/SQL offers several advantages that make database programming more powerful and efficient.

→Benefits of PL/SQL:

1. Procedural Programming Support
   ➤ Lets you use loops, conditions, and variables — like a real programming language.
2. Improved Performance
   ➤ Multiple SQL statements can be executed in one block — reduces database calls.
3. Code Reusability
   ➤ You can write procedures, functions, and packages once and reuse them anywhere.
4. Error Handling
   ➤ Built-in EXCEPTION blocks make it easy to handle run-time errors gracefully.

5. Security

   ➤ Business logic can be stored in the database — prevents tampering and enhances control.

6. Better Integration with SQL

   ➤ PL/SQL works closely with SQL, so you can combine queries and logic easily.

7. Modular Approach

   ➤ Programs can be organized into logical blocks, improving readability and maintenance.

→In short:

PL/SQL improves performance, reliability, and maintainability by combining the power of SQL with procedural features — ideal for complex database operations.

❖17. PL/SQL Control Structures:-

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

Ans.

Control structures in PL/SQL allow you to control the flow of your program using decisions and repetitions — just like in other programming languages.

PL/SQL has three main control structures:

- Conditional (IF-THEN)
- Iterative (LOOP)

- Sequential (basic block flow)

→IF-THEN Structure

Used for decision-making. Executes code only if a condition is true.

Syntax:

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

Example:

```
IF salary < 5000 THEN
  salary := 5000;
END IF;
```

→LOOP Structure

Used to repeat a block of code until manually exited.

Syntax:

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

→ Example:

```
i := 1;
LOOP
```

```
  DBMS_OUTPUT.PUT_LINE(i);
  i := i + 1;
  EXIT WHEN i > 5;
END LOOP;
```

→In Short:

- IF-THEN is used to make decisions.
- LOOP is used to repeat tasks until a condition is met.
  These structures help control logic flow in PL/SQL programs.

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

Ans.

Control structures in PL/SQL help add logic, decision-making, and repetition to your SQL queries, making them more flexible, powerful, and dynamic.

How They Help:

1. Conditional Execution (IF-THEN)
   ➤ Lets you run different queries or actions based on conditions.
   Example: Update salary only if it's below a certain amount.
2. Looping (LOOP, WHILE, FOR)
   ➤ Lets you run queries repeatedly (like inserting or updating multiple rows in a loop).

3. Better Error Handling

➤ Combined with control structures, EXCEPTION blocks handle errors smartly inside complex operations.

4. Dynamic Query Flow

➤ You can adjust what SQL statements run based on logic, like user input or calculations.

5. Improved Code Modularity

➤ Organize your logic into blocks, conditions, and loops, making the program easier to understand and maintain.

In Short:

Control structures allow PL/SQL to go beyond static SQL, letting you write smart, logic-based database programs — ideal for handling real-world business rules.

❖18. SQL Cursors:-

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

Ans.

A cursor in PL/SQL is a pointer to the result set of a SQL query. It is used to retrieve and process multiple rows returned by a query one at a time.

→Types of Cursors:

1. Implicit Cursor

- Automatically created by PL/SQL when a single-row SQL statement (like SELECT INTO, INSERT, UPDATE, DELETE) is executed.
- You don’t need to declare it.
- System handles opening, fetching, and closing.

→ Example:

```
BEGIN
  UPDATE employees SET salary = salary + 1000 WHERE department_id = 10;
  -- Implicit cursor is used internally
END;
```

2. Explicit Cursor

- Manually declared and controlled by the programmer.
- Used for queries that return multiple rows.
- You must DECLARE, OPEN, FETCH, and CLOSE the cursor.

Example:

```
DECLARE
  CURSOR emp_cursor IS SELECT name FROM employees;
  emp_name employees.name%TYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
```

```
    FETCH emp_cursor INTO emp_name;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(emp_name);
  END LOOP;
  CLOSE emp_cursor;
END;
```

→In Short:

Feature	Implicit Cursor	Explicit Cursor
Declaration	Automatic	Manual
Usage	Single-row operations	Multi-row queries
Control	System handles it	You control open, fetch, close
Example Use Case	INSERT, UPDATE, DELETE	SELECT returning multiple rows

→Summary:

Cursors allow row-by-row processing in PL/SQL. Use implicit cursors for simple operations, and explicit cursors when you need more control over queries that return multiple rows.

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

Ans.

You use an explicit cursor when:

→Use Cases for Explicit Cursor:

1. **When a query returns *multiple rows*
   – Implicit cursors can only handle one row.
   – Use explicit cursor to fetch one row at a time in a loop.
2. When you need full control over:
   - Opening the cursor
   - Fetching rows manually
   - Closing the cursor
3. When you want to process each row individually
   – Example: Printing each employee name or applying different logic per row.
4. When using complex logic or calculations
   – Useful in report generation, row-by-row validation, etc.
5. To improve readability and modularity
   – Code is easier to maintain for multi-row operations.

→In Short:

Use an explicit cursor when your query returns multiple rows and you need to process them one at a time with more control and flexibility.

❖19. Rollback and Commit Savepoint :-

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

Ans.

A SAVEPOINT is a marker within a transaction that allows you to roll back part of the transaction without undoing the entire transaction.

→How it works:

- You create a savepoint using:

SAVEPOINT savepoint_name;

- Later, you can roll back to that point using:

ROLLBACK TO savepoint_name;

- This undoes changes made after the savepoint, but keeps earlier changes.

Interaction with ROLLBACK and COMMIT:

- ROLLBACK TO SAVEPOINT → Undo only part of the transaction (to that point).
- ROLLBACK → Undoes the entire transaction.
- COMMIT → Saves all changes in the transaction, including those before and after the savepoint. After commit, savepoints are no longer available.

Example:

```
BEGIN;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;
SAVEPOINT step1;

UPDATE accounts SET balance = balance + 100 WHERE id = 2;
ROLLBACK TO step1; -- Undo only the second update
COMMIT; -- Save the first update
```

→Summary:

Command	Purpose
SAVEPOINT	Marks a point to roll back to
ROLLBACK TO	Undo changes after a savepoint only
ROLLBACK	Undo the entire transaction
COMMIT	Save all changes permanently

In short:
SAVEPOINT gives you partial control in transactions by allowing undo from a specific point, while ROLLBACK and COMMIT control the full or partial application of changes.

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

Ans.

When to Use Savepoints:

1. Partial Rollback Needed
   – If only part of a transaction fails, you can roll back just that part without losing the rest.
2. Complex Transactions
   – In big transactions with multiple steps, savepoints allow you to safely test parts of logic and undo only specific ones.
3. Error Handling
   – When you expect some operations might fail, but don't want to cancel the full transaction, use savepoints for safety.
4. Conditional Logic
   – If you're making decisions based on data while inserting or updating, savepoints let you roll back certain conditions only.
5. Manual Control
   – When you need fine-grained control over what gets saved or discarded during a transaction.

→Example Scenario:

```
BEGIN;

-- Step 1: Deduct balance

UPDATE users SET balance = balance - 500 WHERE id = 1;



-- Set savepoint
```

SAVEPOINT after_deduction;

-- Step 2: Try adding to another account (might fail)

UPDATE users SET balance = balance + 500 WHERE id = 2;

-- If step 2 fails, rollback only to savepoint

ROLLBACK TO after_deduction;

-- Commit successful part

COMMIT;

→Summary:

Savepoints are useful when you want to:

- Undo only part of a transaction
- Handle errors without losing everything
- Maintain data integrity in complex operations

They give you better control and flexibility in transaction management

---