Module-4 introduction to dbms

Introduction to sql

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

Ans->**What is SQL?**

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

- ✞ **Why is SQL essential in database management?**
- ✞ SQL is fundamental in database management for several reasons:
- ✞ **Efficient Data Retrieval:**

SQL allows users to quickly and efficiently query large amounts of data using commands like `SELECT` with conditions, sorting, and filtering. ✞ **Data Integrity and Accuracy:**
SQL enforces **constraints** such as `PRIMARY KEY`, `FOREIGN KEY`, `UNIQUE`, and `NOT NULL` to maintain accurate and reliable data.

- ✞ **Data Manipulation:**
  With SQL, you can insert new records, update existing ones, or delete outdated data safely and consistently.
- ✞ **Database Structure Management:**
  SQL lets you create, alter, and drop tables and databases, giving you full control over the structure of your data storage.
- ✞ **Security Management:**
  SQL provides mechanisms to manage user access and permissions, ensuring that only authorized users can view or modify data.
- ✞ **Standardization and Compatibility:**
  Since SQL is a widely accepted standard, knowledge of SQL makes it easier to work with multiple database systems without learning a new language each time.
- ✞ **Supports Complex Operations:**

SQL can handle **joins, subqueries, aggregations, and transactions**, enabling complex data analysis and business operations.

2. Explain the difference between DBMS and RDBMS.

Ans->

Aspect	DBMS	RDBMS
Full Form	Database Management System	Relational Database Management System
Data Storage	Data stored in files or simple tables	Data stored in structured tables (relations)
Data Relationship	No relationship between data	Relationships supported using keys (Primary, Foreign)
Normalization	Not supported	Supported to reduce redundancy
Data Integrity	Low focus on data integrity	Ensures integrity with constraints (PK, FK, Unique, Not Null)
Transactions (ACID)	May not fully support	Fully supports ACID properties
Security	Limited	Strong security (roles, permissions, authentication)
Examples	dBASE, MS Access	MySQL, Oracle, PostgreSQL, SQL Server

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

Ans-> ➢ **SQL (Structured Query Language)** is the primary tool used to communicate with and manage relational databases. It provides commands to define, manipulate, control, and query data stored in tables. ➢ Roles are:

1. **Data Definition (DDL):**

   ➢ SQL allows users to define and modify the structure of database objects such as tables, views, and indexes. Commands like `CREATE`, `ALTER`, and `DROP` are used to specify schemas and change table structures. This ensures that the database is well-organized before storing data.

2. **Data Manipulation (DML):**
   ➢ SQL provides commands such as `INSERT`, `UPDATE`, `DELETE`, and `SELECT` to add, modify, remove, and retrieve data from tables. This makes it possible to handle large amounts of information in a structured and efficient way.
3. **Data Control (DCL):**
   ➢ SQL controls access to the database using commands like `GRANT` and `REVOKE`. This ensures security by restricting unauthorized access and assigning privileges to specific users or roles.
4. **Transaction Management (TCL):**
   ➢ SQL supports transaction management through commands like `COMMIT`, `ROLLBACK`, and `SAVEPOINT`. These features maintain the **ACID properties** (Atomicity, Consistency, Isolation, Durability) of transactions, which ensure data accuracy and reliability even in cases of system failure.
5. **Data Integrity and Constraints:**
   ➢ SQL enforces rules such as `PRIMARY KEY`, `FOREIGN KEY`, `NOT NULL`, `UNIQUE`, and `CHECK` to maintain accuracy and consistency of data across tables, reducing redundancy and preventing anomalies.

4.What are the key features of SQL?

Ans-> . **Standard Language**

- SQL is an ANSI/ISO standard, supported by almost all RDBMS like MySQL, Oracle, PostgreSQL, and SQL Server.

ii. **Data Definition**

- SQL provides DDL commands (`CREATE`, `ALTER`, `DROP`) to define and modify database structures.

iii. **Data Manipulation**

- DML commands (`INSERT`, `UPDATE`, `DELETE`) allow adding, changing, or removing data.

iv. **Data Retrieval (Querying)**

- `SELECT` is used for retrieving data with conditions, sorting, grouping, and joins.

v. **Data Control & Security**

- DCL commands (`GRANT`, `REVOKE`) manage user permissions and secure the database.

vi. **Transaction Management**

- TCL commands (`COMMIT`, `ROLLBACK`, `SAVEPOINT`) maintain data integrity and ensure ACID properties.

vii. **Support for Constraints**

- SQL enforces `PRIMARY KEY`, `FOREIGN KEY`, `UNIQUE`, `NOT NULL`, and `CHECK` to ensure data consistency.

viii. **Portability**

- SQL works across different platforms and DBMS with minimal modifications.

ix. **User-Friendly**

- SQL uses simple, English-like syntax, making it easier for beginners and professionals.

2.SQL Syntax

1.What are the basic components of SQL syntax?

Ans->

Basic Components of SQL

i. **Data Definition Language (DDL)**

- Used to define and manage the structure of database objects.
- Commands: `CREATE`, `ALTER`, `DROP`, `TRUNCATE`.

ii. **Data Manipulation Language (DML)**

- Used to insert, modify, and delete data in tables.
- Commands: `INSERT`, `UPDATE`, `DELETE`.

iii. **Data Query Language (DQL)**

- Used to retrieve data from the database.
- Command: `SELECT`.

iv. **Data Control Language (DCL)**

- Used to control access and permissions in the database.
- Commands: `GRANT`, `REVOKE`.

v. **Transaction Control Language (TCL)**

- Used to manage transactions and maintain database integrity.
- Commands: `COMMIT`, `ROLLBACK`, `SAVEPOINT`.

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

Ans->

i. **SELECT**

- Specifies the columns to be retrieved.
- Example: `SELECT name, age`

ii. **FROM**

- Identifies the table(s) from which data will be fetched.
- Example: `FROM students`

iii. **WHERE (Optional)**

- Applies conditions to filter rows.
- Example: `WHERE age > 18`

iv. **GROUP BY (Optional)**

- Groups rows that have the same values in specified columns.
- Example: `GROUP BY department`

v. **HAVING (Optional)**

- Applies conditions on grouped data (used with aggregate functions).
- Example: `HAVING COUNT(*) > 5`

vi. **ORDER BY (Optional)**

- Sorts the result in ascending (`ASC`) or descending (`DESC`) order.
- Example: `ORDER BY name ASC`

3. Explain the role of clauses in SQL statements.

Ans-> Clauses in SQL are **building blocks** of a query. Each clause performs a **specific role** to tell the database **what to do with the data**.

Common SQL Clauses & Their Roles:

Clause	Role / Purpose	Example
SELECT	Specifies the columns (or data) you want to retrieve.	SELECT Name, Marks
FROM	Specifies the table(s) from which to retrieve data.	FROM Students
WHERE	Filters rows based on conditions.	WHERE Marks > 80
GROUP BY	Groups rows based on one or more columns.	GROUP BY Department
HAVING	Applies conditions on groups (used with GROUP BY).	HAVING AVG(Marks) > 70
ORDER BY	Sorts the result set in ascending (ASC) or descending (DESC) order.	ORDER BY Marks DESC
LIMIT / TOP (depends on DBMS)	Restricts the number of rows returned.	LIMIT 5

3. SQL Constraints

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

Ans->

- **Constraints** are rules applied to table columns to ensure accuracy, validity, and integrity of data**.**
- Types of Constraints in SQL:

Constraint	Description	Example
PRIMARY KEY	Ensures each row is uniquely identified. A table can have only one primary key (single or composite).	PRIMARY KEY (StudentID)
FOREIGN KEY	Establishes a relationship between two tables; ensures values match a primary key in another table.	FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
UNIQUE	Ensures all values in a column are unique (but allows one NULL).	UNIQUE (Email)
NOT NULL	Ensures a column cannot store NULL values.	Name VARCHAR(50) NOT NULL
CHECK	Ensures that values in a column meet a condition.	CHECK (Age >= 18)
DEFAULT	Assigns a default value to a column if no value is provided.	DEFAULT 'India' for Country column

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

Feature	PRIMARY KEY	FOREIGN KEY
Purpose	Uniquely identifies each record in the table	Establishes a link between two tables (maintains referential integrity)
Uniqueness	Must be unique for each row	Can have duplicate values (but must match a value in the referenced table)
Null Values	Cannot contain NULL	Can contain NULL (unless specified otherwise)
Table	Defined in the same table to identify rows	Defined in child table, references primary key in parent table
Example	StudentID INT PRIMARY KEY	DeptID INT FOREIGN KEY REFERENCES Department(DeptID)

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

Ans-> Role of NOT NULL and UNIQUE Constraints

i. **NOT NULL Constraint**

- Ensures that a column **cannot have NULL values**.
- It forces a field to always contain a valid value.
- Example:

```
CREATE TABLE Students (
    student_id INT NOT NULL,
    name VARCHAR(50) NOT NULL
);
```

.

ii. **UNIQUE Constraint**

- Ensures that all values in a column are **different (no duplicates allowed)**.
- Helps maintain data integrity by avoiding repetition.
- Example:

```
CREATE TABLE Employees(
emp_id INT UNIQUE,
    email VARCHAR(100) UNIQUE
);
```

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

1. NOT NULL Constraint

- **Purpose:** Ensures that a column **cannot have NULL values**.
- **Role:**

  ○ Forces users to provide a value for the column when inserting data. ○ Helps maintain data completeness.

Example:

```
CREATE TABLE Students (
    StudentID      INT    PRIMARY KEY,
    Name VARCHAR(50) NOT NULL             );
```

2. UNIQUE Constraint

- **Purpose:** Ensures that all values in a column are distinct**.**
- **Role:**
  - Prevents duplicate entries in the column.
  - Can be applied to multiple columns (composite unique key).

Example:

```
CREATE     TABLE     Students (
    StudentID      INT    PRIMARY KEY,
    Email     VARCHAR(100)        UNIQUE
);
```

▪ <u>Main SQL Commands and Sub-commands (DDL)</u>

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

- DDL is a subset of SQL used to **define, modify, and remove the structure of database objects** such as tables, indexes, and schemas.
- It **does not manipulate data itself**; it only deals with the **structure** of the database.
- Common DDL Commands:

Command	**Purpose**
CREATE	Creates a new database, table, index, or view.

ALTER	Modifies the structure of an existing database object (e.g., add/remove columns).
DROP	Deletes an existing database object.
TRUNCATE	Removes all records from a table, but keeps the table structure.
RENAME	Renames an existing database object.

2. Explain the CREATE command and its syntax.

- The **CREATE** command is a **DDL (Data Definition Language)** statement used to **create new database objects** such as **databases, tables, views, indexes, or schemas**.
- It defines the **structure and schema** of the object before any data is inserted.

Syntax of CREATE TABLE:

```
CREATE TABLE table_name (

    column1 datatype [constraint],

    column2 datatype [constraint],

    ...) ;
```

Example:

```
CREATE TABLE Students (

    StudentID 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?

➢ When creating a table in SQL, **data types** and **constraints** are specified to ensure **data integrity, accuracy, and efficient storage**.

Purpose of Data Types

- **Define the kind of data** a column can hold (e.g., numbers, text, dates).
- Helps the database **store data efficiently** and **perform operations correctly**.
- **Examples:**
  - INT → Integer numbers
  - VARCHAR(50) → Text up to 50 characters ○ DATE → Date values

Purpose of Constraints

- **Constraints = Rules for data validity and integrity**.
- Ensures the database follows **business rules** and prevents errors.

Common Roles:

Constraint	**Purpose**
PRIMARY KEY	Uniquely identifies each row
FOREIGN KEY	Maintains relationships between tables
NOT NULL	Prevents empty values
UNIQUE	Prevents duplicate values
CHECK	Ensures column values meet a condition
DEFAULT	Assigns a default value if none is provided

▪ ALTER Command

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

➢ The **ALTER** command is a **DDL (Data Definition Language)** statement used to **modify the structure of an existing database object** (usually a table). ➢ It allows changes **without deleting the table or losing data**.

Uses of ALTER Command

- **Add new columns** to a table
- **Modify the data type or size** of existing columns
- **Rename a table or column**
- **Add or drop constraints** (PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL)

Syntax:

- Add a column:

```
ALTER TABLE table_name

ADD column_name datatype [constraint];
```

Modify a column:

```
ALTER TABLE table_name

MODIFY COLUMN column_name datatype
[constraint];
```

- Drop a column:

```
ALTER TABLE table_name

DROP COLUMN column_name;
```

- Rename a table:

```
ALTER TABLE old_table_name

RENAME TO new_table_name;
```

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

1. Add a Column

- Used when you want to insert a new column into an existing table.

Syntax:

```
ALTER TABLE table_name

ADD column_name datatype [constraint];
```

Example:

```
ALTER TABLE Students

ADD Email VARCHAR(100) UNIQUE;
```

2. Modify a Column

- Used to change the **data type, size, or constraints** of an existing column.

Syntax:

```
ALTER TABLE table_name

MODIFY COLUMN column_name
new_datatype [constraint];
```

Example:

```
ALTER TABLE       Students

MODIFY COLUMN Name VARCHAR(80)
NOT NULL;
```

3. Drop a Column

Used to **remove an existing column** from the table.

Syntax:

```
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?

➢ The **DROP** command is a **DDL (Data Definition Language)** statement used to **permanently delete database objects** such as **tables, databases, views, or indexes**.

Functions of DROP Command

- Removes an entire **table structure** (including all rows and constraints).
- Deletes an entire **database** with all its tables.
- Removes **views, indexes, or triggers** from the database.
- Frees up storage space used by the object.

Syntax

```
DROP TABLE table_name;

DROP DATABASE database_name;
```

Example

```
DROP TABLE Students;

DROP DATABASE SchoolDB;
```

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

When you use the DROP TABLE command, it completely removes the table and its structure from the database.

1. Permanent Data Loss

- All the records (rows) in the table are permanently deleted**.**
- Unlike DELETE (which can be rolled back if inside a transaction), DROP usually cannot be undone**.**

2. Loss of Table Structure

- The schema/definition of the table (columns, data types, constraints) is deleted. You cannot insert or query data into that table again unless you recreate it**.**

3. Removal of Constraints and Relationships

- Any primary key, foreign key, unique, or check constraints linked to the table are also deleted.

4. Loss of Indexes, Views, and Triggers

- Any indexes created on the table are removed.

5. Storage Space Freed

- The storage (disk space) used by the table and its data is released back to the database system.

---

4. ▪ Data Manipulation Language (DML)

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

1. INSERT Command

➢ Used to **add new records (rows)** into a table.

Syntax:

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

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

Example:

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

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

2. UPDATE Command

- Used to **modify existing records** in a table.

Syntax:

```
UPDATE table_name

SET column1 = value1, column2 = value2

WHERE condition;
```

Example:

```
UPDATE Students

SET Age = 21

WHERE StudentID = 1;
```

3. DELETE Command

- **Definition:** Used to **remove records (rows)** from a table.

Syntax:

```
DELETE FROM table_name

WHERE condition;
```

Example:

```
DELETE FROM Students

WHERE StudentID = 1;
```

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

- The **WHERE clause** in SQL is **crucial** for controlling which rows are affected during UPDATE and DELETE operations.

 Purpose of the WHERE clause

- It **specifies the condition** that determines which rows should be updated or deleted.
- Without the WHERE clause, **all rows** in the table will be affected.

1. In UPDATE

- UPDATE is used to modify existing data.
- **Importance of WHERE**: Ensures that only the rows matching the condition are updated.

Example:

```
UPDATE employees

SET salary = salary + 5000

WHERE department =          'Sales';
```

- Only employees in the **Sales department** get a salary increment.

2. In DELETE

- DELETE is used to remove rows from a table.
- **Importance of WHERE**: Prevents accidental deletion of all data.

Example :

```
DELETE FROM employees

WHERE retirement_age          = 60;
```

- Only employees aged **60** will be deleted.

---

5. ▪ Data Query Language (DQL)

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

- The **SELECT statement** is one of the most important SQL commands. It is used to **retrieve data from a database**.

1. Purpose of SELECT

- Fetches data from one or more tables.
- Can return **all columns**, **specific columns**, or even **calculated values**.
- Can filter, sort, and group data using additional clauses. 2. Basic Syntax

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

3. Examples:

a) Select all columns:

```
SELECT * FROM employees;
```

b) Select specific columns:

```
SELECT name, salary FROM employees;
```

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

1. WHERE Clause

- **Purpose:** Filters rows based on a specified condition so that only the matching rows are retrieved, updated, or deleted. □ **Used in:** SELECT, UPDATE, DELETE.

Syntax:

```
SELECT column1, column2

FROM table_name

WHERE condition;
```

2. ORDER BY Clause

- **Purpose:** Sorts the result set based on one or more columns.
- **Default order:** Ascending (ASC).
- Can also sort descending using DESC.

Syntax:

```
SELECT column1, column2

FROM table_name

ORDER BY column1 [ASC|DESC], column2 [ASC|DESC];
```

---

6. ▪ Data Control Language (DCL)

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

They are used to **control user access and permissions** in a database.

1. GRANT

- **Purpose:** Gives (grants) specific privileges/permissions to a user or role.
- Allows a user to perform actions like SELECT, INSERT, UPDATE, DELETE, etc. on database objects (tables, views, procedures).

Syntax:

```
GRANT privilege_list

ON object_name

TO user_name;
```

2. **REVOKE**

- **Purpose:** Removes (takes back) previously granted privileges from a user or role.
- Used when you want to restrict access.

Syntax:

```
REVOKE privilege_list

ON object_name

FROM user_name;
```

2. How do you manage privileges using these commands?

Privileges = permissions given to users for performing actions on database objects.
We manage them using **GRANT** and **REVOKE**.

1. Granting Privileges (Giving Access)

- Use the **GRANT** command to allow users to perform specific tasks.
- You can grant privileges like SELECT, INSERT, UPDATE, DELETE, ALL PRIVILEGES.

Example:

```
GRANT SELECT, INSERT

ON employees

TO user1;
```

2. Revoking Privileges (Removing Access)

- Use the **REVOKE** command to take back permissions already granted.

Example:

```
REVOKE INSERT

ON employees

FROM user1;
```

Revoking all privileges:

```
REVOKE ALL PRIVILEGES

ON employees

FROM    user1;
```

3. Managing Multiple Users

- You can grant/revoke privileges to multiple users at once.

Example:

```
GRANT SELECT

ON employees

TO user1, user2, user3;
```

7. ▪ Transaction Control Language (TCL)

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

- COMMIT
- **Purpose:** Saves all changes (INSERT, UPDATE, DELETE) made in the current transaction **permanently** in the database.
- After COMMIT, you **cannot undo** the changes.

Example:

```
UPDATE employees

SET salary = salary + 5000

WHERE department = 'Sales';


COMMIT;
```

- ROLLBACK
- **Purpose:** Undoes (reverses) all changes made in the current transaction if they haven't been committed yet.
- Used when you make a mistake and want to go back to the original state.

Example:

```
DELETE FROM employees

WHERE department = 'HR';


ROLLBACK;
```

2. Explain how transactions are managed in SQL databases.

- A **transaction** is a sequence of one or more SQL operations (like INSERT, UPDATE, DELETE) performed as a **single logical unit of work**.
- Either **all operations succeed** (saved) or **none are applied** (undone).
- Transactions ensure **data consistency, reliability, and integrity**.

- SQL provides commands (TCL – Transaction Control Language) to manage transactions: **a) START TRANSACTION / BEGIN**
- Marks the beginning of a transaction.

b) COMMIT

- Saves all changes permanently.

```
BEGIN TRANSACTION;
```

c) ROLLBACK

```
COMMIT;
```

- Cancels all changes since the last BEGIN or SAVEPOINT.

d) SAVEPOINT

```
ROLLBACK;
```

- Creates a checkpoint within a transaction.
- You can roll back to this point without affecting the whole transaction.

```
SAVEPOINT sp1;

ROLLBACK TO sp1;
```

e) SET TRANSACTION

- Defines transaction properties like read/write access or isolation level.

```
SET TRANSACTION READ ONLY;
```

- <u>SQL Joins</u>

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

- A **JOIN** is used to combine rows from **two or more tables** based on a related column between them (usually a primary key–foreign key relationship).
- Helps in retrieving **related data** that is stored in different tables

Basic Syntax:

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

▪ Types of JOIN

a) **INNER JOIN**

- Returns only the **matching rows** from both tables. If no match, the row is excluded.

Example:

```
SELECT employees.name, departments.dept_name
FROM employees
INNER JOIN departments
ON employees.dept_id = departments.dept_id;
```

b) **LEFT JOIN (LEFT OUTER JOIN)**

- Returns **all rows from the left table** and the **matching rows from the right table**.
- If no match, right-side columns show **NULL**.

Example:

```
SELECT employees.name, departments.dept_name
FROM employees
LEFT JOIN departments
ON employees.dept_id = departments.dept_id;
```

c) **RIGHT JOIN (RIGHT OUTER JOIN)**

- Opposite of LEFT JOIN.

- Returns **all rows from the right table** and the **matching rows from the left table**.
- If no match, left-side columns show **NULL**.

Example:

```
SELECT employees.name, departments.dept_name

FROM employees

RIGHT JOIN departments

ON employees.dept_id = departments.dept_id;
```

d) **FULL OUTER JOIN**

- Combines results of **LEFT JOIN + RIGHT JOIN**.
- Returns **all rows from both tables**, with NULLs where there is no match.

Example:

```
SELECT employees.name, departments.dept_name

FROM employees

FULL OUTER JOIN departments

ON employees.dept_id = departments.dept_id;
```

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

- In relational databases, data is often stored in **multiple related tables** (normalized structure).
- **Joins** allow us to combine these tables and query them **as if they were one**.
- They use a **common column** (usually a primary key in one table and a foreign key in another).

General Syntax

```
SELECT table1.column, table2.column

FROM table1

JOIN table2

ON table1.common_column = table2.common_column;
```

a) INNER JOIN → Only matching rows

```
SELECT e.name, d.dept_name
FROM employees e
INNER JOIN departments d
ON e.dept_id = d.dept_id;
```

b) LEFT JOIN → All from left + matches from right

```
SELECT e.name, d.dept_name
FROM employees e
LEFT JOIN departments d
ON e.dept_id = d        .dept_id;
```

c) RIGHT JOIN → All from right + matches from left

```
SELECT e.name, d.dept_name
FROM employees e
RIGHT JOIN departments d
ON e.dept_id = d.dept_id;
```

d) FULL OUTER JOIN → All from both sides

```
SELECT e.name, d.dept_name
FROM employees e
FULL OUTER JOIN departments d
ON e.dept_id = d.dept_id;
```

- <u>SQL Group By</u>

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

- The **GROUP BY clause** is used to **arrange rows into groups** based on one or more columns.
- Often used with **aggregate functions** (COUNT, SUM, AVG, MAX, MIN) to perform calculations **for each group** of data.

Basic Syntax:

```
SELECT column1,
aggregate_function(column2)

FROM table_name

WHERE condition

GROUP BY column1;
```

Examples of GROUP BY with Aggregate Functions

a) Count employees in each department

```
SELECT dept_id, COUNT(*) AS
num_employees

FROM employees

GROUP BY dept_id;
```

b) Average salary per department

```
SELECT dept_id, AVG(salary) AS avg_salary

FROM employees

GROUP BY dept_id;
```

c) Maximum salary per department

```
SELECT dept_id, MAX(salary) AS highest_salary
FROM employees
GROUP BY dept_id;
```

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

➢ Difference Between GROUP BY and ORDER BY:

Feature	GROUP BY	ORDER BY
Purpose	Groups rows into categories based on column values.	Sorts rows in ascending (ASC) or descending (DESC) order.
Use with Aggregates	Commonly used with aggregate functions (COUNT, SUM, AVG, MAX, MIN).	Not required, but can be used to sort aggregated or normal results.
Result	Produces **one row per group**.	Produces **all rows**, only rearranged.
Clause Position	Appears **before ORDER BY** in a query.	Appears **after GROUP BY** (if both are used).
Focus	Summarizes data.	Presents data in a specific sequence.

---

8. ▪ SQL Stored Procedure

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

➢ A **Stored Procedure** is a **precompiled collection of SQL statements** (like SELECT, INSERT, UPDATE, etc.) stored in the database.

➢ It can also include **parameters, conditional logic, and loops**.
➢ Think of it as a **function** in programming that performs a specific task.

- Once created, you can **reuse** it multiple times without rewriting the SQL code.

```
CREATE PROCEDURE GetEmployeeDetails

AS

BEGIN

   SELECT * FROM Employees;

END;
```

To execute:

```
EXEC GetEmployeeDetails;
```

Aspect	Stored Procedure	Standard SQL Query
Definition	Precompiled group of SQL statements stored in the database.	A single SQL command written and executed directly.
Reusability	Can be reused many times by calling the procedure name.	Must be retyped or copied every time.
Performance	Faster (compiled once and stored in database).	Slower (parsed and executed each time).
Logic Handling	Can include loops, IF-ELSE conditions, variables.	Only one query at a time, limited logic.
Security	Access can be restricted, hiding actual SQL code.	Query must be written and exposed each time.
Maintenance	Easier (change procedure once, applies everywhere).	Harder (need to update queries in all places).

2. Explain the advantages of using stored procedures.

- The advantages of stored procedures are:

1. **Fast execution** – precompiled and stored in the database.
2. **Reusable** – write once, use many times.
3. **Easy maintenance** – change in one place, applies everywhere.
4. **Secure** – users can run procedure without direct table access.
5. **Less network traffic** – only procedure call is sent, not big queries.
6. **Consistent results** – same logic runs every time.
7. **Organized** – makes database code modular and clean.

- SQL View

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

- A **view** is a **virtual table** created from a SQL query.
- It does not store data physically; it only **shows data** from one or more tables.

Example:

```
CREATE VIEW EmployeeDetails AS

SELECT Name, DepartmentID

FROM Employees

WHERE Status = 'Active';
```

Now you can use it like:

```
SELECT * FROM EmployeeDetails;
```

Aspect	Table	View
Definition	A physical object that stores actual data.	A virtual table created from a query.

Data Storage	Stores data in the database.	Does not store data, only stores the query definition.
Aspect	**Table**	**View**
Data Update	Can insert, update, delete (full CRUD).	Mostly read-only (some updates possible under conditions).
Performance	Faster, as data is stored physically.	Slower, as it fetches data from underlying tables each time.
Purpose	Used to store data permanently.	Used to simplify queries, hide complexity, or provide security.

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

➢ Advantages of Views

1. **Simplifies complex queries** – You can save a long query as a view and reuse it easily.
2. **Provides security** – Users can access only the view, not the actual tables.
3. **Data abstraction** – Hides unnecessary columns and complexity from users.
4. **Consistency** – Same logic/query is reused, so results are uniform.
5. **Easier maintenance** – If the query changes, update the view once instead of changing everywhere.
6. **Logical independence** – Underlying table structure can change, but the view can still provide the same output format.
7. **Reusability** – Use the same view in multiple queries or applications.

▪ SQL Triggers

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

➢ A **trigger** is a special kind of stored procedure that **automatically runs** when a specific event happens in a table (like INSERT, UPDATE, or DELETE).

➢ You don't call it manually – it is **triggered by the database** when the event occurs.

Types of Triggers in SQL

 Based on Timing

1. **BEFORE Trigger** ○ Executes **before** the action (INSERT, UPDATE, or DELETE) happens.
    - ○ Use case**:** Validate or modify data before saving it.

o Example: Prevent negative salary before inserting into table.

2. **AFTER Trigger** o Executes **after** the action happens. o Use case**:** Audit/logging changes, updating related tables.
   o Example: After inserting an order, update the stock table.

When are Triggers Used?

- **Auditing/logging** → Record who changed what and when.
- **Enforcing business rules** → Prevent invalid data entry.
- **Maintaining referential integrity** → Update or delete related records automatically.
- **Automatic actions** → Example: Updating stock after a sale.

Based on Events 1. **INSERT Trigger** – Fires when a new

row is inserted.

o Example: Automatically add entry in a log table when a new employee is added.

2. **UPDATE Trigger** – Fires when a row is updated. o Example: Track changes in salary history table when salary is updated.
3. **DELETE Trigger** – Fires when a row is deleted.
   o Example: Save deleted data into an archive table.

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

Trigger Type	**When it Fires**	**What It Can Do**	**Example Use Case**
INSERT Trigger	When a new row is inserted into a table.	Can validate or log new entries.	Automatically add a record to an audit table when a new employee is added.
UPDATE Trigger	When an existing row is modified.	Can track changes or enforce rules.	Store old salary in a history table whenever salary is updated.
DELETE Trigger	When a row is removed from a table.	Can archive deleted data or prevent deletion.	Save deleted customer details in an archive table before removing them.

- <u>Introduction to PL/SQL</u>

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

➢ **PL/SQL** stands for **Procedural Language/SQL**.

➢ It is **Oracle's procedural extension of SQL**.
➢ Combines **SQL statements** with **procedural programming features** like variables, loops, conditions, and error handling.
➢ Essentially, it allows you to write **programs inside the database**.

Feature	SQL	PL/SQL
Procedural logic	No loops or conditions, only queries.	Supports `IF-ELSE`, loops (`FOR`, `WHILE`), and variables.
Modularity	Cannot create functions/procedures inside the database.	Can create **stored procedures, functions, and packages**.
Error handling	No built-in error handling.	Supports **exception handling** to manage runtime errors.
Control over execution	Executes one query at a time.	Can execute multiple SQL statements together in a **block**.
Reusability	Limited; queries must be repeated.	Code can be **reused** with procedures and functions.

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

➢ Benefits of PL/SQL

1. **Combines SQL and Programming** ○ Allows using **loops, conditions, and variables** along with SQL.
   - You can write more complex logic than plain SQL.
2. **Improves Performance** ○ Blocks of code are **precompiled and stored** in the database.
   - Reduces **network traffic** by executing multiple statements in one block.
3. **Reusability** ○ You can create **procedures, functions, and packages** that can be used multiple times.
4. **Error Handling** ○ Supports **exception handling** to catch and manage runtime errors.

5. **Security** ○ Can hide complex queries and allow users to access **procedures/functions** without direct table access.
6. **Modularity**

   ○ Large programs can be broken into smaller **manageable blocks** for easier maintenance.

7. **Maintainability** ○ Changing logic in a procedure/function automatically affects all programs that use it.
8. **Portability** ○ PL/SQL code can run on any Oracle database with minimal changes.

▪ <u>PL/SQL Control Structures</u>

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

➢ **Control structures** are statements that **control the flow of execution** in a PL/SQL program.

➢ They let you make **decisions**, **repeat actions**, or **handle exceptions**.

 Main types:

1. **Conditional statements** → Make decisions (IF-THEN, CASE)
2. **Looping statements** → Repeat actions (LOOP, WHILE, FOR)

i. IF-THEN Control Structure

➢ Used to **execute code only if a condition is true**.

Syntax:

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

Example:

```
DECLARE
  salary NUMBER := 25000;
BEGIN
  IF salary > 20000 THEN
    DBMS_OUTPUT.PUT_LINE('High salary');
  END IF;
END;
```

LOOP Control Structure

- Used to **repeat a set of statements multiple times**.

Types: LOOP, WHILE LOOP, FOR LOOP.

```
LOOP
  -- statements
  EXIT WHEN condition;  -- exit loop when condition is
true
END LOOP;
```

Example:

```
DECLARE

   i NUMBER := 1;

BEGIN

   LOOP

      DBMS_OUTPUT.PUT_LINE('Count: ' || i);

      i := i + 1;

      EXIT WHEN i > 5;          --  stop loop after 5

   END LOOP;

END;
```

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

1.Decision Making (IF-THEN / CASE)

 Allows executing different SQL statements **based on conditions**. Example: Apply a bonus only if salary > 20000.

2. Repetition (LOOP / WHILE / FOR)

- Lets you **repeat SQL operations** for multiple rows or tasks without writing queries again and again.
- Example: Update all employee salaries in a department using a loop.

3. Modularity

- Combine multiple SQL statements inside loops or conditions into **one logical block**.
- Makes large operations easier to manage.

4. Error Handling (EXCEPTION)

- Allows handling errors during query execution without stopping the entire program.
- Example: Skip invalid records while inserting multiple rows.

5.Automation ➢ Complex multi-step SQL tasks can be automated using loops and

conditions.

- ➢ Example: Automatically calculate taxes, bonuses, and update tables in one PL/SQL block.

▪ <u>SQL Cursors</u>

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

- ➢ A **cursor** is a **pointer** that allows you to **fetch and process rows returned by a SQL query one at a time**.
- ➢ Think of it as a **handle for a query result set**.
- ➢ Useful when a query returns **multiple rows** and you want to process them **row by row**.

▪ Types of Cursors

A. Implicit Cursor

- Automatically created by PL/SQL **when you execute a SQL statement** like SELECT INTO, INSERT, UPDATE, or DELETE.
- You **do not declare** it.
- **Used for:** SQL statements that return **only one row**.
- Example:

```
DECLARE
  emp_name Employees.Name%TYPE;
BEGIN
  SELECT Name INTO emp_name
  FROM Employees
  WHERE EmployeeID = 101;
```

B. Explicit Cursor

- ➢ **Declared by the programmer** for queries that return **multiple rows**.
- ➢ Allows **row-by-row processing** using OPEN, FETCH, and CLOSE.

```
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;
```

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

Use Explicit Cursor when:

- Query returns **multiple rows** (need to fetch one by one).
- You want **full control** (open, fetch, close).
- You need to **use parameters** in the cursor.
- You want to check **cursor attributes** after each fetch. You want to make code **clear and modular**.

Use Implicit Cursor when:

- Query returns **only one row**.
- You are using **simple DML** (INSERT, UPDATE, DELETE).
- You don’t need to control fetching manually.

- Rollback and Commit Savepoint

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

- A **SAVEPOINT** is like a **bookmark** in a transaction.
- It marks a point inside a transaction so you can **roll back** to it later without undoing the whole transaction.
- You can have **multiple savepoints** in one transaction

```
SAVEPOINT sp1;
```

- **How ROLLBACK Works with SAVEPOINT**

  - `ROLLBACK TO sp1;` undoes all changes **after** the savepoint `sp1` but keeps the changes **before** it.
  - It does **not end** the transaction. ➢ You can continue from there.

- **How COMMIT Works with SAVEPOINT**

  - `COMMIT` makes **all changes permanent** (including those before and after savepoints).
  - Once you commit, all savepoints are automatically removed. ➢ You cannot roll back to a savepoint after a commit.

```
BEGIN;
INSERT INTO employees VALUES (1,'A');
SAVEPOINT s1;

INSERT INTO employees VALUES (2,'B');)
SAVEPOINT s2;

ROLLBACK TO s1;  -- Undo only the insert of 'B', keep 'A'

COMMIT;         -- Make all remaining changes permanent
```

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

- **Partial Undo**:

  - When you want to undo only part of a transaction without undoing the whole thing.

- **Complex Transactions**:
  - In long transactions with many steps (inserts/updates/deletes), you can set savepoints at key stages.
- **Error Handling**:
  - If an error happens after a certain step, you can roll back only to the last successful point instead of starting over.
- **Testing Intermediate Results**:
  - When you’re not sure if a step will succeed and you may need to backtrack.
- **Maintaining User Choices**:
  - In apps where the user performs multiple operations before finalizing, savepoints let you roll back some actions but keep others.
