Module 4 – Introduction to DBMS

1.Introduction to SQL.

Theory Questions:

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

Ans:

SQL (Structured Query Language) is a standard programming language specifically designed for managing and manipulating relational databases. It is essential because it allows users to **store, retrieve, update, and delete data** efficiently. SQL is widely used in almost every application that deals with structured data, such as banking systems, e-commerce platforms, and social networks.

Key reasons why SQL is important:

- It provides a **universal standard** for interacting with databases like MySQL, PostgreSQL, Oracle, and SQL Server.
- Enables **data retrieval using queries** with SELECT statements.
- Supports **data manipulation** through INSERT, UPDATE, and DELETE commands.
- Ensures **data integrity and security** with constraints and permissions.
- Facilitates **database schema creation and management** using CREATE, ALTER, and DROP.
- Allows **complex operations** such as joins, subqueries, and aggregations.

2. Explain the difference between DBMS and RDBMS.

Ans:

Feature	**DBMS**	**RDBMS**
Full Form	Database Management System	Relational Database Management System
Data Storage	Stores data as files or hierarchical form	Stores data in **tables (rows & columns)**
Relationships	Does **not** maintain relationships between data	Maintains relationships using **primary & foreign keys**
Data Redundancy	Higher redundancy	Reduces redundancy using **normalization**
ACID Properties	Not fully supported	Fully supports **ACID properties**

Feature	DBMS	RDBMS
Complex Queries	Limited query support	Supports **SQL** for complex queries
Examples	XML DB, File System	MySQL, Oracle, PostgreSQL, SQL Server

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

Ans:

SQL (Structured Query Language) plays a crucial role in managing **Relational Database Management Systems (RDBMS)** by providing a standardized way to interact with relational data. Its primary roles include:

1. **Data Definition (DDL):** SQL allows the creation and management of database structures using commands like CREATE, ALTER, and DROP.
2. **Data Manipulation (DML):** It enables inserting, updating, and deleting data with commands such as INSERT, UPDATE, and DELETE.
3. **Data Retrieval (DQL):** SQL retrieves data from tables using the SELECT statement, supporting filtering, sorting, and aggregation.
4. **Transaction Control (TCL):** SQL ensures data integrity during transactions through commands like COMMIT, ROLLBACK, and SAVEPOINT.
5. **Data Control (DCL):** It provides security by granting or revoking user permissions with GRANT and REVOKE.
6. **Relational Operations:** SQL manages relationships between tables using **joins, keys, and constraints.**

4. What are the key features of SQL?

Ans:

- **Data Retrieval:** Retrieve data from databases using `SELECT` queries.
- **Data Manipulation:** Insert, update, and delete data with `INSERT`, `UPDATE`, and `DELETE`.
- **Data Definition:** Create and manage database structures using `CREATE`, `ALTER`, and `DROP`.
- **Data Control:** Manage permissions with `GRANT` and `REVOKE` commands.
- **Transaction Control:** Maintain data integrity using `COMMIT`, `ROLLBACK`, and `SAVEPOINT`.
- **Supports Joins:** Combine data from multiple tables using joins.

- **Standardized Language:** Works across most relational databases like MySQL, Oracle, and PostgreSQL.
- **ACID Compliance:** Ensures reliable transactions and data integrity.
- **Scalability:** Handles large amounts of data efficiently.
- **Portability:** Can run on different systems without changes.

2. SQL Syntax.

Theory Questions:

1.What are the basic components of SQL syntax?

Ans:

Keywords: Reserved words used for specific tasks like SELECT, INSERT, UPDATE, DELETE, WHERE, etc.

Clauses: Parts of a query that define conditions or operations, such as WHERE, ORDER BY, GROUP BY.

Statements: Complete SQL commands, e.g., SELECT * FROM students;.

Expressions: Combinations of columns, operators, and values that return a result (e.g., salary + bonus).

Predicates: Conditions that return TRUE, FALSE, or UNKNOWN, such as WHERE age > 18.

Operators: Symbols for comparison and arithmetic, e.g., =, >, <, AND, OR.

2. Write the general structure of an SQL SELECT statement

Ans:

Ex:

```
SELECT column1, column2, ...

FROM table_name

WHERE condition

GROUP BY column
```

HAVING condition

ORDER BY column [ASC|DESC];

SELECT – Specifies the columns to retrieve.

FROM – Indicates the table from which to retrieve data.

WHERE – Filters rows based on conditions.

GROUP BY – Groups rows that have the same values in specified columns.

HAVING – Filters groups based on conditions (used with GROUP BY).

ORDER BY – Sorts the result in ascending (ASC) or descending (DESC) order.

3. Explain the role of clauses in SQL statements.

Ans:

1. **SELECT** – Specifies which columns or data to retrieve from the table.
2. **FROM** – Indicates the table(s) from which data will be fetched.
3. **WHERE** – Filters rows based on given conditions.
4. **GROUP BY** – Groups rows sharing the same column values for aggregation.
5. **HAVING** – Applies conditions on grouped data (used with GROUP BY).
6. **ORDER BY** – Sorts the results in ascending or descending order.

3. SQL Constraints.

Theory Questions:

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

Ans:

Types of Constraints:

1. **NOT NULL**
   - Ensures that a column cannot have a NULL value.
   - Example:

name VARCHAR(50) NOT NULL;

2. **UNIQUE**
   - Ensures that all values in a column are unique (no duplicates).
   - Example:

email VARCHAR(100) UNIQUE;

3. **PRIMARY KEY**
   - Uniquely identifies each record in a table; it combines **NOT NULL + UNIQUE**.
   - Example:

```
PRIMARY KEY(id);
```

4. **FOREIGN KEY**
   - Maintains **referential integrity** by linking one table's column to another table's primary key.
   - Example:

```
FOREIGN KEY(department_id) REFERENCES departments(id);
```

5. **CHECK**
   - Ensures that all values in a column satisfy a specific condition.
   - Example:

```
CHECK(age >= 18);
```

6. **DEFAULT**
   - Assigns a default value if no value is provided for the column.
   - Example:

```
status VARCHAR(10) DEFAULT 'Active'
```

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

Ans:

Feature	**PRIMARY KEY**	**FOREIGN KEY**
Definition	Uniquely identifies each record in a table.	Establishes a link between two tables.
Uniqueness	Must have **unique** values.	Can have **duplicate** values.
NULL values	Cannot contain NULL values.	Can contain NULL values.
Number per table	Only **one PRIMARY KEY** allowed per table.	Multiple FOREIGN KEYS can exist in a table.
Function	Ensures **entity integrity**.	Ensures **referential integrity**.
Example	PRIMARY KEY (id)	FOREIGN KEY (dept_id) REFERENCES departments(id)

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

Ans:

Role of NOT NULL Constraint:

- Ensures that a column **cannot store NULL values**.
- It is used when a field must always have a value (mandatory data).
- Example:

name VARCHAR(50) NOT NULL;

Here, the name column cannot be empty.

Role of UNIQUE Constraint:

- Ensures that all values in a column are **unique** (no duplicates).
- Unlike **PRIMARY KEY**, it allows **one NULL value** (depending on the database system).
- Example:

email VARCHAR(100) UNIQUE;

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

Theory Questions:

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

Ans:

SQL Data Definition Language (DDL) is a part of SQL used to define and manage the **structure of a database** rather than its data. It includes commands that create, modify, and delete database objects like tables, schemas, indexes, and views. DDL is essential for setting up and maintaining the **schema** of a database system.

The main characteristics of DDL are:

- It defines how data is stored, organized, and related.
- It works on database objects, not on the actual data.
- DDL commands are automatically **committed**, meaning changes are permanent.

Common DDL Commands:

1. **CREATE:** Used to create new database objects such as tables, views, or databases.
   Example:

```
CREATE TABLE Students (

    ID INT PRIMARY KEY,

    Name VARCHAR(50),

    Age INT );
```

2. **ALTER:** Modifies an existing object, like adding a column to a table.
3. **DROP:** Deletes an existing database object permanently.
4. **TRUNCATE:** Removes all data from a table without deleting the table structure.

2. Explain the CREATE command and its syntax.

Ans:

Purpose of CREATE Command:

- To create a **database** for storing data.
- To create **tables** with specific columns and data types.
- To create other objects like **views, indexes, and schemas**.

General Syntax of CREATE Command:

1. **For Database:**

```
 CREATE DATABASE database_name;
```

2. **For Table:**

```
CREATE TABLE table_name (

    column1 datatype constraint,

    column2 datatype constraint,

    ...

);
```

Example:

```
CREATE TABLE Employees (

    EmpID INT PRIMARY KEY,

    Name VARCHAR(50) NOT NULL,

    Salary DECIMAL(10,2),
    Department VARCHAR(30)

);
```

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

Ans:

Purpose of Data Types:

1. **Defines the kind of data** a column can store (e.g., INT for numbers, VARCHAR for text, DATE for dates).
2. Ensures **accuracy** by preventing invalid data entry (e.g., text in a numeric column).
3. Helps in **efficient storage and performance**, as each data type uses specific memory space.
4. Allows proper **operations and functions** on data (e.g., mathematical operations on numbers only).

Purpose of Constraints:

1. **Maintain data integrity** by enforcing rules on columns.
2. **NOT NULL:** Ensures a column cannot have empty values.
3. **UNIQUE:** Prevents duplicate values in a column.
4. **PRIMARY KEY:** Uniquely identifies each record in the table.
5. **FOREIGN KEY:** Maintains relationships between tables.
6. **CHECK:** Restricts values based on a condition.
7. **DEFAULT:** Assigns a default value if none is provided.

Example:

```
CREATE TABLE Students (

    ID INT PRIMARY KEY,

    Name VARCHAR(50) NOT NULL,

    Age INT CHECK(Age >= 18),

    Status VARCHAR(10) DEFAULT 'Active'

);
```

5. ALTER Command.

Theory Questions:

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

Ans:

Add a new column to a table.

```
ALTER TABLE employees ADD age INT;
```

Modify an existing column (e.g., change data type or size).

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

Rename a column.

```
ALTER TABLE employees RENAME COLUMN age TO emp_age;
```

Drop (delete) a column from a table.

```
ALTER TABLE employees DROP COLUMN emp_age;
```

Add or remove constraints like PRIMARY KEY, FOREIGN KEY, or UNIQUE.

Rename a table.

```
ALTER TABLE employees RENAME TO staff;
```

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

Ans:

1.Add a Column

To add a new column to a table:

```
ALTER TABLE table_name

ADD column_name datatype;
```

Example:

```
ALTER TABLE employees

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

This adds a salary column with a decimal data type.

2. Modify a Column

To change the data type, size, or constraints of an existing column:

```
ALTER TABLE table_name

MODIFY column_name new_datatype;
```

Example:

```
ALTER TABLE employees

MODIFY salary INT;
```

This changes the salary column from DECIMAL to INT.

3. Drop a Column

To remove a column from a table:

```
ALTER TABLE table_name

DROP COLUMN column_name;
```

Example:

```
ALTER TABLE employees

DROP COLUMN salary;
```

6. DROP Command.

Theory Questions:

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

Ans:

1.Delete a table along with all its data and structure

```
DROP TABLE employees;
```

This removes the entire employees table.

2.Delete an entire database

```
DROP DATABASE company_db;
```

3.Delete other objects like views, indexes, or triggers

DROP VIEW view_name;

DROP INDEX index_name;

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

Ans:

Implications of Dropping a Table:

1. **Permanent Deletion of Data:**
   All rows (records) stored in the table are permanently deleted and **cannot be recovered** unless there is a backup.
2. **Loss of Table Structure:**
   The table's schema (columns, data types, constraints) is completely removed. The table no longer exists in the database.
3. **Cascading Effects (if constraints exist):**
   - If the table is referenced by **foreign keys** in other tables, the DROP command may fail unless you first remove or disable those constraints.
   - In some databases, using DROP TABLE ... CASCADE; will drop the table along with all dependent objects (views, constraints, etc.).
4. **Invalidation of Dependent Objects:**
   Views, stored procedures, or triggers that rely on the table will become invalid and stop working.
5. **Loss of Permissions:**
   Any privileges or grants associated with the table are removed.
6. **Cannot be Rolled Back:**
   Once executed, DROP TABLE cannot be undone with a simple transaction rollback (unlike DELETE). A backup is the only way to restore it.
7. **Impact on Applications:**
   Applications or queries that reference the dropped table will fail, leading to potential runtime errors.

Example:

DROP TABLE employees;

7. Data Manipulation Language (DML).

Theory Questions:

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

Ans:

1.INSERT Command

- The **INSERT** command is used to **add new rows of data** into a table.
- It can insert values into **all columns** or specific columns.

Syntax:

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

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

Example:

```
INSERT INTO employees (emp_id, name, salary)

VALUES (101, 'John Doe', 50000);
```

2. UPDATE Command

- The **UPDATE** command is used to **modify existing records** in a table.
- It can update one or more columns based on a condition (WHERE clause).

Syntax:

```
UPDATE table_name

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

WHERE condition;
```

Example:

```
UPDATE employees

SET salary = 60000

WHERE emp_id = 101;
```

3. DELETE Command

- The **DELETE** command is used to **remove rows** from a table based on a condition.
- If the WHERE clause is omitted, **all rows** in the table are deleted, but the table structure remains.

Syntax:

```
DELETE FROM table_name

WHERE condition;
```

Example:

DELETE FROM employees

WHERE emp_id = 101;

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

Ans:

1. It **specifies which rows** in the table should be updated or deleted.
2. Without the WHERE clause, **all rows** in the table will be updated or deleted, which can lead to **unintentional data loss**.
3. It ensures **precision**, allowing only specific records to be changed or removed.
4. It helps maintain **data integrity** by preventing unwanted modifications.
5. It acts as a **filter**, ensuring that the command affects only the rows that meet the condition.

8. Data Query Language (DQL).

Theory Questions:

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

Ans:

The **SELECT statement** in SQL is used to **query and retrieve data** from one or more tables in a database. It allows you to specify **which columns** and **rows** you want to fetch, and you can also filter, sort, and group the data using various clauses.

Key Features of SELECT:

1. It retrieves **specific columns** or all columns (*) from a table.
2. It can filter records using the **WHERE** clause.
3. It can sort data using **ORDER BY**.
4. It can group data using **GROUP BY** and filter grouped results with **HAVING**.
5. It can combine data from multiple tables using **JOINs**.

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

Ans:

- **WHERE Clause:**
  - Filters rows in a table based on a specific condition.
  - Only rows that meet the condition will appear in the result.
  - It is used with operators like =, >, <, >=, <=, LIKE, IN, etc.
  - Example:

SELECT name, salary FROM employees WHERE salary > 50000;

This query returns only employees who earn more than 50,000.

- **ORDER BY Clause:**
  - Sorts the result set in ascending (**ASC**) or descending (**DESC**) order.
  - By default, it sorts in ascending order.
  - You can sort by one column or multiple columns.
  - Example:

```
SELECT name, salary FROM employees ORDER BY salary DESC;
```

This query returns all employees sorted by salary in descending order.

- **Together:**
  - WHERE filters data first, then ORDER BY arranges the filtered data.
  - Example:

```
SELECT name, salary FROM employees

WHERE salary > 50000

ORDER BY salary DESC;
```

9. Data Control Language (DCL).

Theory Questions:

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

Ans:

GRANT and REVOKE are commands used for **managing user permissions** in SQL.

They help in controlling **who can access or modify database objects**.

GRANT is used to give specific privileges to users or roles.

These privileges may include reading data, inserting data, updating data, deleting data, or executing procedures.

It allows database administrators to **assign rights** for secure data handling.

REVOKE is used to **take back or remove** privileges from a user or role.

It ensures that a user can no longer perform operations that were previously allowed.

Both commands are essential for **database security and user access control**.

They help prevent unauthorized access or changes to the database.

Widely used in multi-user environments to manage permissions effectively.

2. How do you manage privileges using these commands?

Ans:

GRANT Command:

- The GRANT command is used to **assign privileges** to users or roles.
- It allows users to perform actions like SELECT, INSERT, UPDATE, DELETE, or even manage other users.

Syntax:

GRANT privilege_name ON object_name TO user_name;

Example:

GRANT SELECT, INSERT ON Employees TO John;

This allows user **John** to select and insert data into the Employees table.

2. REVOKE Command:

- The REVOKE command is used to **remove privileges** that were previously granted to a user.

Syntax:

REVOKE privilege_name ON object_name FROM user_name;

Example:

REVOKE INSERT ON Employees FROM John;.

10. Transaction Control Language (TCL).

Theory Questions:

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

Ans:

Purpose of COMMIT Command:

- The COMMIT command is used to **permanently save** all changes made by a transaction (such as INSERT, UPDATE, or DELETE).
- Once a COMMIT is executed, the changes cannot be undone.

Purpose of ROLLBACK Command:

- The ROLLBACK command is used to **undo changes** made by the current transaction **before COMMIT is executed**.

- It is useful when an error occurs, and you need to revert to the previous stable state.

2. Explain how transactions are managed in SQL databases.

Ans:

1. **Start of a Transaction:**
    - A transaction usually begins automatically when a DML operation (e.g., INSERT, UPDATE) is executed.
2. **ACID Properties:**
    - **A – Atomicity:** All operations within a transaction are treated as one unit.
    - **C – Consistency:** Ensures the database remains in a valid state before and after a transaction.
    - **I – Isolation:** Each transaction runs independently without interfering with others.
    - **D – Durability:** Once committed, changes are permanent, even in case of a system failure.
3. **COMMIT Command:**
    - Used to **save** all changes of the current transaction permanently.
4. **ROLLBACK Command:**
    - Used to **undo** all changes in case of an error or failure before a COMMIT.
5. **SAVEPOINT Command:**
    - Creates a **checkpoint** within a transaction, allowing partial rollbacks.

11. SQL Joins.

Theory Questions:

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 Type	Description	Result
INNER JOIN	Returns only rows where there is a match in both tables.	Common records from both tables.
LEFT JOIN	Returns all rows from the left table and matching rows from the right table.	Non-matching rows from the right table are shown as NULL.
RIGHT JOIN	Returns all rows from the right table and matching rows from the left table.	Non-matching rows from the left table are shown as NULL.
FULL OUTER JOIN	Returns all rows from both tables, whether matched or not.	Non-matching rows from both tables are shown as NULL.

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

Ans:

- Joins use the JOIN keyword along with the ON clause to specify the relationship between tables.
- A common column (e.g., DeptID in Employees and Departments) is used to link rows between tables.
- Different types of joins (INNER, LEFT, RIGHT, FULL OUTER) determine which rows are returned.

General Syntax of a Join:

```
SELECT table1.column1, table2.column2
FROM table1
JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

Suppose we have two tables:

- **Employees (EmpID, Name, DeptID)**

- **Departments (DeptID, DeptName)**

To combine employee names with their department names:

```
SELECT Employees.Name, Departments.DeptName

FROM Employees

INNER JOIN Departments

ON Employees.DeptID = Departments.DeptID;
```

12. SQL Group By.

Theory Questions:

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

Ans:

The **GROUP BY** clause in SQL is used to **group rows that have the same values** in one or more columns, allowing you to perform operations on each group rather than on individual rows. It is commonly used with **aggregate functions** like COUNT(), SUM(), AVG(), MAX(), and MIN() to calculate summaries for each group.

How GROUP BY Works:

- The GROUP BY clause divides the result set into groups based on the values of the specified columns.
- Then, **aggregate functions** are applied to each group to produce summarized results.

Syntax:

```
SELECT column_name, AGGREGATE_FUNCTION(column_name)

FROM table_name

WHERE condition

GROUP BY column_name;
```

Example:

Suppose we have a table **Sales (Product, Quantity)** and we want the total quantity sold for each product:

```
SELECT Product, SUM(Quantity) AS TotalQuantity

FROM Sales

GROUP BY Product;
```

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

Ans:

Aspect	GROUP BY	ORDER BY
Purpose	Groups rows with the same values into summary rows.	Sorts the result set in ascending or descending order.
Use with Functions	Often used with aggregate functions (SUM, COUNT, AVG).	Can be used with or without aggregate functions.
Result	Produces one row per group.	Does not group data; it just arranges rows.
Clause Position	Comes **before ORDER BY** in SQL query.	Comes **after GROUP BY** in SQL query.
Focus	Focuses on **data aggregation**.	Focuses on **data presentation**.
Example	GROUP BY Product (groups sales by product).	ORDER BY Price DESC (sorts sales by price).

13. SQL Stored Procedure.

Theory Questions:

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

Ans:

1. It is stored and executed on the database server.
2. It can accept **parameters (IN, OUT, or INOUT)** to make it dynamic.
3. It improves **performance**, as the SQL code is compiled and cached for reuse.
4. It enhances **security** by restricting direct access to the tables.
5. It allows **modular programming** by reusing the same logic multiple times.

Difference Between Stored Procedure and Standard SQL Query:

Aspect	Stored Procedure	Standard SQL Query
Definition	A precompiled set of SQL statements stored in the database.	A single SQL command executed directly.

Aspect	Stored Procedure	Standard SQL Query
Reusability	Can be executed multiple times with parameters.	Must be written and executed each time.
Performance	Faster due to pre compilation and caching.	Slower for complex queries.
Logic	Can include loops, conditions (IF-ELSE), etc.	Only performs single SQL operations.
Security	Can restrict access and hide underlying tables.	Executes directly on tables.

2. Explain the advantages of using stored procedures.

Ans:

Improved Performance: Precompiled and cached, so they execute faster than individual SQL queries.

Reusability: Can be executed multiple times with different parameters, avoiding code repetition.

Reduced Network Traffic: A single call can execute multiple SQL statements on the server.

Enhanced Security: Restricts direct access to tables and hides business logic.

Modular Code: Allows separation of logic, making applications easier to maintain.

Consistency: Ensures uniform results by centralizing common operations.

Error Handling: Supports mechanisms like TRY-CATCH for handling runtime errors.

Transaction Support: Can group multiple operations with COMMIT and ROLLBACK.

Ease of Maintenance: Changes can *be* made in the procedure without affecting application code.

Scalability: Ideal for large applications where complex logic needs to be reused frequently.

14. SQL View.

Theory Questions:

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

Ans:

Aspect	**View**	**Table**
Definition	A virtual table based on a query.	A physical structure that stores actual data.
Data Storage	Does not store data; shows data from tables.	Stores data permanently in rows and columns.
Modification	Cannot always be updated directly.	Allows direct insert, update, and delete operations.
Purpose	Used for simplifying queries and enhancing security.	Used for storing and managing data.
Execution	Executes the underlying SQL query each time it is accessed.	Data is retrieved directly from storage.

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

Ans:

Simplifies Complex Queries:

- A view can store complex SQL queries and present them as a simple table, making it easier for users to retrieve data without writing long queries.

Enhanced Security:

- Views can restrict access to specific columns or rows of a table, allowing users to see only the required data while hiding sensitive information.

Data Abstraction:

- Users can interact with the view without knowing the underlying table structure or relationships, providing a simplified interface to the database.

Reusability:

- A view can be reused in multiple queries or applications, avoiding repetition of complex SQL code.

Logical Data Independence:

- Changes in the underlying tables (e.g., adding new columns) do not necessarily affect queries that use the view.

Improved Readability:

- Since views can be named and stored, they make queries shorter and easier to read.

Aggregation and Formatting:

- Views can present aggregated data (e.g., totals or averages) or data formatted in a specific way without altering the base tab

15. SQL Triggers.

Theory Questions:

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

Ans:

A **trigger** in SQL is a database object that automatically executes when a specific event occurs on a table or view. It is commonly used for tasks like data validation, auditing, enforcing business rules, or automating system tasks.

Types of SQL Triggers:

1. **BEFORE Trigger:** Runs before an INSERT, UPDATE, or DELETE operation. Useful for checking or modifying data before it is committed.
2. **AFTER Trigger:** Runs after an INSERT, UPDATE, or DELETE operation. Typically used for logging or cascading changes.
3. **INSTEAD OF Trigger:** Runs in place of an INSERT, UPDATE, or DELETE operation, often on views where direct modifications are not possible.
4. **DML Triggers:** Respond to data manipulation events such as INSERT, UPDATE, or DELETE.
5. **DDL Triggers:** Respond to schema-related changes like CREATE, ALTER, or DROP.
6. **LOGON/LOGOFF Triggers:** Fire in response to user session events (in some SQL systems).

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

Ans:

Aspect	INSERT Trigger	UPDATE Trigger	DELETE Trigger
When It Fires	Executes **after or before** a new row is inserted into a table.	Executes **after or before** an existing row is updated.	Executes **after or before** an existing row is deleted.
Purpose	Typically used to validate data, log insert actions, or initialize related tables.	Typically used to track changes, enforce rules, or maintain audit logs.	Typically used to log deletions, enforce referential integrity, or archive data.
Pseudo Tables	Uses the **INSERTED** table (contains new rows being inserted).	Uses both **INSERTED** (new data) and **DELETED** (old data) tables.	Uses the **DELETED** table (contains rows being removed).
Example Action	Could check constraints before adding a new row.	Could track changes by storing old and new values in an audit table.	Could back up data to another table before deletion.

16. Introduction to PL/SQL.

Theory Questions:

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

Ans:

- Query data (SELECT)
- Modify data (INSERT, UPDATE, DELETE)
- Define schema (CREATE, ALTER)

But SQL **cannot**:

- Use **conditional logic** (IF-THEN-ELSE)
- Perform **loops** or **iterations**
- Declare **variables and constants**
- Handle **exceptions (errors)** in a structured way
- Group multiple SQL statements as a **single block** of code

PL/SQL adds all these features, turning SQL into a **full-fledged programming language**.

Key Features of PL/SQL

1. **Block Structure** – Code is organized into blocks (DECLARE, BEGIN, EXCEPTION, END).
2. **Procedural Constructs** – Support for loops (FOR, WHILE), conditional statements, etc.
3. **Variables and Constants** – Store and manipulate intermediate values.
4. **Exception Handling** – Handle runtime errors gracefully.
5. **Cursors** – Work with result sets of queries row by row.
6. **Modularity** – Support for functions, procedures, and packages.

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

Ans:

1. **Combines SQL with Procedural Logic**
   - Adds loops, conditions, and variables to SQL for complex operations.
2. **Improved Performance**
   - Executes multiple SQL statements as one block, reducing network overhead.
3. **Error Handling**
   - Provides robust exception handling to manage runtime errors gracefully.
4. **Reusability & Modularity**

   Supports procedures, functions, and packages for code reuse and maintainability.
5. **Security & Transaction Control**
   - Allows secure access through stored procedures and precise transaction management (COMMIT/ROLLBACK).

17. PL/SQL Control Structures

Theory Questions:

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

Ans:

1. **Conditional Control** – IF-THEN, IF-THEN-ELSE, CASE.
2. **Iterative Control** – LOOP, WHILE LOOP, FOR LOOP.
3. **Sequential Control** – GOTO, NULL.

1. IF-THEN Control Structure

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

- Variants include:
  - IF ... THEN
  - IF ... THEN ... ELSE
  - IF ... THEN ... ELSIF ... ELSE

Syntax (basic IF-THEN):

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

Example:

```
IF salary > 50000 THEN
   DBMS_OUTPUT.PUT_LINE('High salary');
END IF;
```

2. LOOP Control Structure

- A **LOOP** is used to execute a block of statements **repeatedly** until explicitly stopped with EXIT or EXIT WHEN.

Syntax:

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

Example:

```
DECLARE
   counter NUMBER := 1;
BEGIN
   LOOP
      DBMS_OUTPUT.PUT_LINE('Counter: ' || counter);
      counter := counter + 1;
      EXIT WHEN counter > 5;  -- loop ends after 5 iterations
   END LOOP;
END;
```

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

Ans:

1.Add Conditional Logic

- **IF-THEN** and **CASE** allow different SQL statements to execute based on conditions.
- Example: Updating employee bonuses differently based on their department or performance.

2. Enable Iterative Processing

- **LOOP, WHILE, and FOR** allow row-by-row processing of query results using cursors.
- Useful for complex scenarios where multiple calculations or updates must be done per row.

3. Error Handling and Decision Making

- **EXCEPTION blocks** with control structures can manage errors during execution, enabling fallback logic.
- Example: If a record insert fails, a backup insert can be attempted.

4. Dynamic Query Execution

- Control structures allow executing **different SQL queries dynamically** based on runtime conditions (e.g., different reports for different user roles).

5. Modular and Maintainable Logic

- Combining SQL with **procedures, loops, and conditions** results in clean, reusable, and maintainable code for complex business rules.

18. SQL Cursors.

Theory Questions:

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 or a memory area** that stores the result set of a SQL query so that rows can be fetched and processed one at a time.

Whenever a SQL statement (like SELECT) is executed, Oracle uses a cursor to manage the context area containing the query result.

Aspect	Implicit Cursor	Explicit Cursor
Creation	Created automatically by Oracle.	Created manually by the programmer.
Use Case	For single-row queries (INSERT, UPDATE, DELETE, SELECT INTO).	For multi-row SELECT queries.
Control	Limited control (fetch happens automatically).	Full control (OPEN, FETCH, CLOSE explicitly).
Naming	Always named SQL.	Can be given any user-defined name.

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

Ans:

The query returns **more than one row**, and we need to process each row individually.

We require **row-by-row processing**, for example, calculating bonuses for each employee in a department.

We need **more control** over the fetching process using OPEN, FETCH, and CLOSE.

We want to use **cursor attributes** like %FOUND, %NOTFOUND, %ROWCOUNT, and %ISOPEN to track the cursor's status.

We need to **loop through the result set** and perform specific operations on each row.

19. Rollback and Commit Save point.

Theory Questions:

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

Ans:

A **SAVEPOINT** is a marker within a transaction that allows partial rollback of changes without affecting the entire transaction. It acts like a checkpoint, enabling the programmer to undo only a portion of the work done since the start of the transaction.

The **ROLLBACK TO save point name** command undoes all changes made after that save point, but changes made before the save point remain intact. If no save point is mentioned, ROLLBACK undoes the entire transaction.

A **COMMIT** permanently saves all changes made in the transaction and automatically removes all defined save points. Once committed, you cannot roll back to any previous save point.

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

Ans:

A **save point** is useful when we want to **partially rollback** specific operations within a transaction without affecting previous successful operations. It is particularly helpful in **multi-step transactions**, where some steps may fail but others should remain intact.

Save points are often used for **error handling**, allowing the transaction to revert to a known safe point when an exception occurs. They are also beneficial in **complex business logic**, **testing**, or debugging scenarios, where we need fine control over which operations to undo.

For example, while updating salaries of multiple departments, we can create save points after each update and rollback only the changes for a specific department if needed.