1. **Introduction to SQL**

**Theory Questions:**

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

Here’s a precise explanation:

**SQL (Structured Query Language):**

* SQL is a **standard programming language** used to manage and manipulate relational databases.
* It allows users to **store, retrieve, update, and delete** data efficiently.
* SQL also provides commands to **define database structures (DDL), control access (DCL), and manage transactions (TCL).**

**Why SQL is essential in database management:**

* **Standardization:** SQL is the universally accepted language for relational databases (used in MySQL, Oracle, SQL Server, PostgreSQL, etc.).
* **Data Manipulation:** It enables easy querying and manipulation of large datasets.
* **Data Definition:** Provides a way to create and modify database schemas.
* **Security:** Supports user roles, permissions, and access control.
* **Transaction Management:** Ensures data consistency and reliability with commit, rollback, and savepoint features.
* **Portability:** SQL code can often be used across different database systems with minimal changes.

✅ **In short:** SQL is essential because it is the backbone of relational database management, ensuring efficient data storage, retrieval, security, and integrity.

1. **Explain the difference between DBMS and RDBMS.**

**DBMS vs RDBMS**

| **Aspect** | **DBMS (Database Management System)** | **RDBMS (Relational Database Management System)** |
| --- | --- | --- |
| **Definition** | Software that manages data in databases (can be relational, hierarchical, or network-based). | A type of DBMS that specifically manages data in **tabular (row-column) form** using relationships. |
| **Data Storage** | Stores data as **files** (hierarchical, network, or flat-file structure). | Stores data in **tables (relations)**. |
| **Relationships** | Does not enforce relationships among data. | Enforces **relationships via primary keys, foreign keys, and constraints**. |
| **Data Integrity** | Limited support for data integrity. | Strong support for **ACID properties (Atomicity, Consistency, Isolation, Durability)** and constraints. |
| **Examples** | dBase, Microsoft Access (basic versions). | MySQL, Oracle, SQL Server, PostgreSQL. |
| **Normalization** | Not supported. | Supports **normalization** to reduce redundancy. |
| **Multi-user Support** | Limited support for multiple users. | Fully supports multiple concurrent users. |

✅ **In short:**

* **DBMS** is a general system for managing data, without strict structure or relationships.
* **RDBMS** is an advanced DBMS that organizes data into **tables with defined relationships**, ensuring **data integrity and consistency**.

1. **Describe the role of SQL in managing relational databases.**

**Role of SQL in Managing Relational Databases**

SQL plays a **central role** in the creation, manipulation, and control of relational databases. Its functions can be grouped into the following:

1. **Data Definition (DDL – Data Definition Language)**
   * SQL defines the structure of a relational database.
   * Commands: CREATE, ALTER, DROP
   * Example: CREATE TABLE Students (ID INT, Name VARCHAR(50));
2. **Data Manipulation (DML – Data Manipulation Language)**
   * SQL allows inserting, updating, deleting, and retrieving data stored in tables.
   * Commands: INSERT, UPDATE, DELETE, SELECT
   * Example: SELECT \* FROM Students;
3. **Data Control (DCL – Data Control Language)**
   * SQL controls access to data through permissions and roles.
   * Commands: GRANT, REVOKE
   * Example: GRANT SELECT ON Students TO User1;
4. **Transaction Management (TCL – Transaction Control Language)**
   * SQL ensures **data consistency and reliability** in multi-user environments.
   * Commands: COMMIT, ROLLBACK, SAVEPOINT
   * Example: ROLLBACK TO Savepoint1;
5. **Query Processing**
   * SQL enables **complex queries** using JOIN, GROUP BY, ORDER BY, and aggregate functions.
   * Helps extract meaningful information from relational data.

✅ **In short:** SQL acts as the **communication bridge** between users and relational databases. It defines structures, manipulates data, enforces security, manages transactions, and retrieves insights effectively.

1. **What are the key features of SQL?**

**Key Features of SQL**

1. **Data Definition**
   * SQL allows creating, altering, and deleting database structures.
   * Example: CREATE TABLE, ALTER TABLE.
2. **Data Manipulation**
   * Enables insertion, modification, deletion, and retrieval of data.
   * Example: INSERT, UPDATE, DELETE, SELECT.
3. **Data Querying**
   * Provides powerful querying with conditions, sorting, grouping, and joining.
   * Example: SELECT Name FROM Students WHERE Marks > 80;.
4. **Data Control**
   * Manages permissions and access rights for security.
   * Example: GRANT, REVOKE.
5. **Transaction Management**
   * Supports reliable transactions with **ACID properties**.
   * Commands: COMMIT, ROLLBACK, SAVEPOINT.
6. **Portability**
   * SQL works across many relational database systems (MySQL, Oracle, PostgreSQL, SQL Server).
7. **Standards Compliance**
   * SQL is governed by **ANSI and ISO standards**, making it widely accepted.
8. **Support for Functions**
   * Provides aggregate (COUNT, SUM, AVG) and scalar functions for calculations.
9. **Multi-user Support**
   * Handles concurrent access in multi-user environments without data conflicts.

✅ **In short:** SQL is powerful because it is **standardized, portable, secure, and versatile** for defining, manipulating, controlling, and querying relational databases.

**2. SQL Syntax**

**Theory Questions:**

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

SQL statements are made up of several basic components:

* **Keywords** → Reserved words that define the operation (e.g., SELECT, INSERT, UPDATE).
* **Identifiers** → Names of databases, tables, columns, or aliases (e.g., Students, Marks).
* **Operators** → Symbols for comparison or logic (e.g., =, >, <, AND, OR).
* **Clauses** → Subsections that refine the query (e.g., WHERE, ORDER BY, GROUP BY).
* **Expressions** → Combinations of values, operators, and functions (e.g., Marks + 10).
* **Functions** → Predefined operations for calculations (e.g., COUNT(), SUM()).

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

The **SELECT statement** is the most widely used SQL query for data retrieval.

**General Syntax:**

SELECT column1, column2, ...

FROM table\_name

WHERE condition

GROUP BY column

HAVING condition

ORDER BY column ASC|DESC;

**Explanation of parts:**

* SELECT → Specifies the columns to retrieve.
* FROM → Defines the table(s).
* WHERE → Filters rows based on conditions.
* GROUP BY → Groups rows with common values.
* HAVING → Applies conditions on grouped data.
* ORDER BY → Sorts the results.

**3. Explain the role of clauses in SQL statements.**

* **Clauses** refine and control the execution of SQL queries.
* Each clause has a **specific role**:
  + WHERE → Filters rows before grouping.
  + GROUP BY → Organizes rows into groups.
  + HAVING → Applies filters after grouping.
  + ORDER BY → Sorts the final output.
  + LIMIT / TOP (depending on DB) → Restricts the number of rows returned.
* Clauses can be **combined** to form complex queries.

✅ **In short:** Clauses act as **building blocks** that give more precision and flexibility to SQL statements.

**3. SQL Constraints**

**Theory Questions:**

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

**Definition:**

* Constraints in SQL are **rules applied to table columns** to ensure the **accuracy, integrity, and reliability** of data.
* They restrict the type of data that can be stored, preventing invalid or inconsistent entries.

**Types of 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 distinct (no duplicates).
   * Example: Email VARCHAR(100) UNIQUE
3. **PRIMARY KEY**
   * Combines **NOT NULL + UNIQUE**.
   * Each row in a table must have a unique and non-null primary key.
   * Example: StudentID INT PRIMARY KEY
4. **FOREIGN KEY**
   * Establishes a link between two tables by referencing the **primary key** of another table.
   * Maintains **referential integrity**.
   * Example: FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
5. **CHECK**
   * Ensures values meet a specific condition.
   * Example: Age INT CHECK (Age >= 18)
6. **DEFAULT**
   * Assigns a default value if no value is specified.
   * Example: Status VARCHAR(10) DEFAULT 'Active'

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

| **Aspect** | **PRIMARY KEY** | **FOREIGN KEY** |
| --- | --- | --- |
| **Definition** | Uniquely identifies each row in a table. | Links a column in one table to the **primary key** of another table. |
| **Uniqueness** | Must be **unique** and **not null**. | Can have **duplicate values**. |
| **Null values** | Does not allow null values. | Can allow nulls (unless defined otherwise). |
| **Number allowed** | Only **one primary key** per table. | Multiple foreign keys can exist in a table. |
| **Purpose** | Ensures **entity integrity**. | Ensures **referential integrity**. |
| **Example** | StudentID in Students table. | DeptID in Students referencing Department table. |

✅ **In short:**

* **Primary Key = Unique identifier in its own table.**
* **Foreign Key = Connector between two tables.**

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

* **NOT NULL Constraint:**
  + Prevents a column from storing NULL (empty) values.
  + Ensures that every row has a **valid value** for that column.
  + Example: Name VARCHAR(50) NOT NULL → Every student must have a name.
* **UNIQUE Constraint:**
  + Ensures that all values in a column are **different**.
  + Prevents duplicate entries.
  + Example: Email VARCHAR(100) UNIQUE → No two users can have the same email ID.

✅ **Combined Use:**

* Often used together for strict validation.
* Example: Username VARCHAR(50) NOT NULL UNIQUE → Every user must have a username, and it must be different from others.

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

**Theory Questions:**

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

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

* **DDL (Data Definition Language):**
  + A subset of SQL used to **define and manage database structures**.
  + It deals with the **schema (structure) of a database**, not the data itself.
  + DDL commands are **auto-committed** → once executed, changes are permanent.

**Main DDL Commands:**

* CREATE → To create database objects (tables, views, indexes).
* ALTER → To modify existing database structures.
* DROP → To delete database objects permanently.
* TRUNCATE → To remove all rows from a table (structure remains).
* RENAME → To rename database objects.

✅ **In short:** DDL defines **what data structures exist**, not the actual data stored in them.

**2. Explain the CREATE command and its syntax.**

* **Definition:**
  + The CREATE command is used to **create new database objects** like tables, databases, or views.
  + Most commonly, it is used to create **tables**.

**General Syntax (for tables):**

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 CHECK (Age >= 18),

Email VARCHAR(100) UNIQUE,

DeptID INT,

FOREIGN KEY (DeptID) REFERENCES Department(DeptID)

);

**Explanation:**

* StudentID → primary key, uniquely identifies each student.
* Name → cannot be null.
* Age → must be 18 or above.
* Email → unique for every student.
* DeptID → foreign key, linked to another table.

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

**Data Types:**

* Define the **kind of data** a column can store.
* Example: INT, VARCHAR, DATE, FLOAT.
* **Purpose:**
  + Ensures **consistency** (e.g., Age must be an integer).
  + Optimizes **storage and performance**.
  + Prevents invalid data (e.g., cannot insert text into an INT column).

**Constraints:**

* Define **rules** that the data must follow.
* Examples: NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK, DEFAULT.
* **Purpose:**
  + Enforces **data integrity** (e.g., no duplicate IDs).
  + Maintains **relationships** (via foreign keys).
  + Prevents **invalid or missing values**.

✅ **In short:**

* **Data types → Define what kind of data is allowed.**
* **Constraints → Define rules to maintain accuracy, consistency, and validity.**

**5. ALTER Command**

**Theory Questions:**

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

* **Definition:**
  + The ALTER command is a **DDL (Data Definition Language)** command used to **modify the structure of an existing table** without deleting it.
* **Uses of ALTER:**
  + **Add new columns** to a table.
  + **Modify existing columns** (data type, size, or constraints).
  + **Drop (remove) columns** from a table.
  + **Rename columns or the table itself** (in some DBMS).
  + **Add or remove constraints**.

✅ **In short:** ALTER helps **evolve database schema** as requirements change, without losing existing data.

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

**a) Add a column**

* Syntax:

ALTER TABLE table\_name

ADD column\_name datatype constraint;

* Example:

ALTER TABLE Students

ADD Phone VARCHAR(15);

👉 Adds a new column Phone to the Students table.

**b) Modify a column**

* Syntax (varies slightly by DBMS):

ALTER TABLE table\_name

MODIFY column\_name new\_datatype constraint; -- (MySQL/Oracle)

or

ALTER TABLE table\_name

ALTER COLUMN column\_name new\_datatype; -- (SQL Server/PostgreSQL)

* Example:

ALTER TABLE Students

MODIFY Name VARCHAR(100) NOT NULL;

👉 Changes the Name column to hold up to 100 characters and disallow NULLs.

**c) Drop a column**

* Syntax:

ALTER TABLE table\_name

DROP COLUMN column\_name;

* Example:

ALTER TABLE Students

DROP COLUMN Phone;

👉 Removes the Phone column from the Students table.

✅ **Summary:**

* ADD → Introduces new columns.
* MODIFY / ALTER COLUMN → Changes data type or constraints.
* DROP COLUMN → Removes unwanted columns.

**6. DROP Command**

**Theory Questions:**

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

* **Definition:**
  + The DROP command is a **DDL (Data Definition Language)** command in SQL used to **permanently remove a database object**.
  + Once executed, the object (table, database, view, index, etc.) and all the data it contains are **irreversibly deleted**.
* **Syntax (for table):**

DROP TABLE table\_name;

* **Examples:**

DROP TABLE Students;

👉 Permanently deletes the Students table, including its structure, data, and constraints.

DROP DATABASE College;

👉 Permanently deletes the entire College database and all objects inside it.

✅ **In short:** The DROP command is used for **complete deletion** of database objects.

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

When a table is dropped:

1. **Data Loss**
   * All records stored in the table are permanently removed.
   * This action **cannot be rolled back** (unless the DBMS supports flashback features).
2. **Loss of Table Structure**
   * The table’s schema (column definitions, constraints, indexes) is deleted.
   * You cannot access or query the table anymore.
3. **Impact on Relationships**
   * If the table is referenced by **foreign keys** in other tables, dropping it may cause **referential integrity issues** (many DBMS require dropping constraints first).
4. **Dependent Objects Removed**
   * Views, stored procedures, or triggers depending on the table may fail or get invalidated.
5. **System Resources Freed**
   * Storage space occupied by the table and its indexes is released back to the database.

✅ **In short:** Dropping a table means **permanent removal of both data and structure**, which can also impact related objects and relationships.

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

**Theory Questions:**

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

These are **DML (Data Manipulation Language)** commands used to manage data inside tables:

**a) INSERT Command**

* **Definition:** 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, DeptID)

VALUES (101, 'Rahul', 20, 5);

👉 Adds a new student record into the Students table.

**b) UPDATE Command**

* **Definition:** 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 = 101;

👉 Updates the age of student 101 to 21.

**c) DELETE Command**

* **Definition:** Used to **remove records** from a table.
* **Syntax:**

DELETE FROM table\_name

WHERE condition;

* **Example:**

DELETE FROM Students

WHERE StudentID = 101;

👉 Deletes the student with ID 101.

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

* **Purpose of WHERE Clause:**
  + The WHERE clause specifies **which rows** should be updated or deleted.
  + Without WHERE, **all rows** in the table will be affected.

**a) In UPDATE**

* Example with WHERE:

UPDATE Students

SET Age = 22

WHERE StudentID = 102;

👉 Only student 102 has their age updated.

* Example without WHERE:

UPDATE Students

SET Age = 22;

👉 Updates the age of **all students** to 22 (likely an error).

**b) In DELETE**

* Example with WHERE:

DELETE FROM Students

WHERE DeptID = 5;

👉 Deletes only students in department 5.

* Example without WHERE:

DELETE FROM Students;

👉 Deletes **all rows** from the Students table (but keeps the table structure).

✅ **In short:**

* WHERE is **crucial** in UPDATE and DELETE to target specific rows.
* Without it, the command applies to **every row**, which can cause accidental data loss or unwanted changes.

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

**Theory Questions:**

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

* **Definition:**
  + The SELECT statement is part of **DQL (Data Query Language)**.
  + It is used to **retrieve data from one or more tables** in a relational database.
  + SELECT does **not modify data**, it only queries it.
* **General Syntax:**

SELECT column1, column2, ...

FROM table\_name

WHERE condition

GROUP BY column

HAVING condition

ORDER BY column ASC|DESC;

* **Example:**

SELECT Name, Age, DeptID

FROM Students

WHERE Age > 20;

👉 Retrieves the names, ages, and department IDs of students older than 20.

* **Key Points:**
  1. Can retrieve **specific columns** or use \* for all columns.
  2. Can query **one or multiple tables** using joins.
  3. Can filter, sort, and group data using clauses.

✅ **In short:** SELECT is the **main command for fetching data** and analyzing information in relational databases.

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

**a) WHERE Clause**

* **Purpose:** Filters rows based on a **condition**.
* **Position:** Comes after FROM but before GROUP BY / ORDER BY.
* **Example:**

SELECT Name, Age

FROM Students

WHERE DeptID = 5;

👉 Returns only students in department 5.

* **Operators commonly used:**
  + Comparison: =, <>, <, >, <=, >=
  + Logical: AND, OR, NOT
  + Pattern: LIKE, IN, BETWEEN

**b) ORDER BY Clause**

* **Purpose:** Sorts the result set based on **one or more columns**.
* **Default Sorting:** Ascending (ASC).
* **Optional Sorting:** Descending (DESC).
* **Example:**

SELECT Name, Age

FROM Students

ORDER BY Age DESC;

👉 Displays students sorted by age from **highest to lowest**.

* Can combine with WHERE:

SELECT Name, Age

FROM Students

WHERE DeptID = 5

ORDER BY Age ASC;

👉 Filters students in Dept 5 and sorts by age in ascending order.

✅ **In short:**

* WHERE → Filters rows **before output**.
* ORDER BY → Sorts rows **after filtering**

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

**Theory Questions:**

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

* **Definition of DCL (Data Control Language):**
  + DCL is a subset of SQL used to **control access and permissions** on database objects.
  + Main commands: GRANT and REVOKE.

**a) GRANT**

* **Purpose:**
  + Gives **specific privileges** on database objects (tables, views, procedures) to users or roles.
  + Ensures **controlled access**, allowing only authorized operations.
* **Syntax:**

GRANT privilege\_name ON object\_name TO user\_name;

* **Example:**

GRANT SELECT, INSERT ON Students TO User1;

👉 User1 can now **read and insert data** in the Students table.

**b) REVOKE**

* **Purpose:**
  + Removes privileges that were previously granted to a user.
  + Helps maintain **security** by restricting unauthorized operations.
* **Syntax:**

REVOKE privilege\_name ON object\_name FROM user\_name;

* **Example:**

REVOKE INSERT ON Students FROM User1;

👉 User1 can no longer insert data into the Students table but may still read it (if SELECT was granted).

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

**Steps to manage privileges:**

1. **Grant Privileges**
   * Decide the type of access: SELECT, INSERT, UPDATE, DELETE, ALL.
   * Assign privileges only to users who need them (principle of least privilege).
   * Example:

GRANT ALL ON Students TO AdminUser;

👉 AdminUser gets full access to the table.

1. **Revoke Privileges**
   * When a user no longer needs access, remove privileges.
   * Helps **prevent misuse or accidental changes**.

* Example:

REVOKE DELETE ON Students FROM TempUser;

👉 TempUser cannot delete data anymore.

1. **Check Privileges**
   * Use system tables or commands (DBMS dependent) to view user privileges.
   * Example (MySQL):

SHOW GRANTS FOR 'User1';

✅ **In short:**

* GRANT → **Give permissions** to users or roles.
* REVOKE → **Take back permissions** to maintain security.
* Together, they ensure **controlled access, data integrity, and security** in multi-user environments.

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

**Theory Questions:**

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

**Definition:**

* **TCL (Transaction Control Language)** commands are used to **manage transactions** in a database, ensuring data consistency and integrity.
* A **transaction** is a logical unit of work that must be either **fully completed** or **fully undone**.

**a) COMMIT**

* **Purpose:**
  + Permanently saves all changes made by a transaction into the database.
  + Ensures that the changes are **persistent** and cannot be undone by a rollback.
* **Syntax:**

COMMIT;

* **Example:**

UPDATE Students SET Age = 22 WHERE StudentID = 101;

COMMIT;

👉 The age change for student 101 is **permanently saved**.

**b) ROLLBACK**

* **Purpose:**
  + Reverts the database to the **state before the transaction started**.
  + Useful when an error occurs or a transaction needs to be undone.
* **Syntax:**

ROLLBACK;

* **Example:**

UPDATE Students SET Age = 22 WHERE StudentID = 101;

ROLLBACK;

👉 The age change is **discarded**, and the original value is restored.

**c) SAVEPOINT (Optional for finer control)**

* Allows setting intermediate points in a transaction to **rollback partially**.
* **Syntax:**

SAVEPOINT savepoint\_name;

ROLLBACK TO savepoint\_name;

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

**Key Points in Transaction Management:**

1. **ACID Properties** (ensures reliable transactions):
   * **Atomicity:** All operations in a transaction succeed or fail as a unit.
   * **Consistency:** Database moves from one valid state to another.
   * **Isolation:** Transactions are independent; intermediate results are hidden.
   * **Durability:** Committed changes are permanent, even after system failure.
2. **Transaction Flow:**
   * **Start Transaction:** Automatically (in most DBMS) or using BEGIN TRANSACTION.
   * **Execute DML Commands:** INSERT, UPDATE, DELETE.
   * **Set Savepoints (optional):** For partial rollback.
   * **COMMIT:** Save changes permanently.
   * **ROLLBACK:** Undo changes if an error occurs.
3. **Example:**

BEGIN TRANSACTION;

UPDATE Students SET Age = 22 WHERE StudentID = 101;

SAVEPOINT BeforeDeptChange;

UPDATE Students SET DeptID = 5 WHERE StudentID = 101;

-- Something goes wrong

ROLLBACK TO BeforeDeptChange;

COMMIT;

👉 Age change is saved, but DeptID update is undone.

✅ **In short:**

* **COMMIT → Save changes permanently.**
* **ROLLBACK → Undo changes.**
* TCL ensures **transaction integrity, consistency, and reliability** in multi-user database systems.

**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?**

* **Definition:**
  + A **JOIN** in SQL is used to **combine rows from two or more tables** based on a related column between them, usually a **primary key and foreign key**.
  + Joins help retrieve meaningful data from multiple tables in a single query.

**Types of Joins and Differences**

| **Join Type** | **Definition** | **Result** |
| --- | --- | --- |
| **INNER JOIN** | Returns rows where there is a **match in both tables**. | Only matching rows from both tables are included. |
| **LEFT JOIN (LEFT OUTER JOIN)** | Returns all rows from the **left table**, and matching rows from the right table. | If no match exists in the right table, NULLs are shown. |
| **RIGHT JOIN (RIGHT OUTER JOIN)** | Returns all rows from the **right table**, and matching rows from the left table. | If no match exists in the left table, NULLs are shown. |
| **FULL OUTER JOIN** | Returns all rows from **both tables**, with NULLs for non-matching rows. | Combines LEFT JOIN and RIGHT JOIN results. |

**Example Tables:**

**Students**

| **StudentID** | **Name** | **DeptID** |
| --- | --- | --- |
| 101 | Rahul | 1 |
| 102 | Priya | 2 |
| 103 | Aman | 3 |

**Departments**

| **DeptID** | **DeptName** |
| --- | --- |
| 1 | CSE |
| 2 | IT |
| 4 | Mechanical |

**JOIN Examples:**

* **INNER JOIN:**

SELECT Students.Name, Departments.DeptName

FROM Students

INNER JOIN Departments ON Students.DeptID = Departments.DeptID;

Returns only students with matching departments (Rahul & Priya).

* **LEFT JOIN:**

SELECT Students.Name, Departments.DeptName

FROM Students

LEFT JOIN Departments ON Students.DeptID = Departments.DeptID;

Returns all students. Aman shows NULL for DeptName (no match).

* **RIGHT JOIN:**

SELECT Students.Name, Departments.DeptName

FROM Students

RIGHT JOIN Departments ON Students.DeptID = Departments.DeptID;

Returns all departments. Mechanical shows NULL for Student Name.

* **FULL OUTER JOIN:**

SELECT Students.Name, Departments.DeptName

FROM Students

FULL OUTER JOIN Departments ON Students.DeptID = Departments.DeptID;

Returns all students and all departments, with NULLs where there is no match.

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

* **Purpose:**
  + Relational databases often **split data across tables** to reduce redundancy.
  + Joins allow us to **combine related data** in queries for reporting, analysis, or application logic.
* **Key Points:**
  + **Primary Key & Foreign Key Relationship:** Joins typically rely on matching keys.
  + **Multiple Tables:** You can join more than two tables using multiple JOIN clauses.
  + **Filtering:** WHERE or ON clauses specify **conditions for matching rows**.
  + **Aggregation:** Joins can be combined with GROUP BY and HAVING to summarize data across tables.

**Example:** Retrieve students and their department names:

SELECT Students.Name, Departments.DeptName

FROM Students

INNER JOIN Departments ON Students.DeptID = Departments.DeptID;

✅ **In short:**

* **Joins** are used to **combine data** across related tables.
* Choosing **INNER, LEFT, RIGHT, or FULL JOIN** depends on whether you want **only matches or all rows** from one or both tables.

**12. SQL Group By**

**Theory Questions:**

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

* **Definition:**
  + The GROUP BY clause is used to **group rows that have the same values** in one or more columns.
  + It is often used **with aggregate functions** to perform calculations on each group instead of the entire table.
* **Aggregate Functions Commonly Used:**
  + COUNT() → Count of rows
  + SUM() → Total sum of values
  + AVG() → Average value
  + MAX() → Maximum value
  + MIN() → Minimum value
* **Syntax:**

SELECT column1, aggregate\_function(column2)

FROM table\_name

WHERE condition

GROUP BY column1;

* **Example Table: Sales**

| **SalesID** | **Product** | **Amount** | **Region** |
| --- | --- | --- | --- |
| 1 | Pen | 100 | East |
| 2 | Pen | 150 | West |
| 3 | Book | 200 | East |
| 4 | Pen | 120 | East |
| 5 | Book | 180 | West |

* **Example Query:**

SELECT Product, SUM(Amount) AS TotalSales

FROM Sales

GROUP BY Product;

* **Result:**

| **Product** | **TotalSales** |
| --- | --- |
| Pen | 370 |
| Book | 380 |

**Explanation:**

* All rows are grouped by Product.
* SUM(Amount) calculates total sales **for each product group**.

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

| **Aspect** | **GROUP BY** | **ORDER BY** |
| --- | --- | --- |
| **Purpose** | Groups rows based on column values for aggregation. | Sorts rows (or grouped results) in ascending or descending order. |
| **Used With** | Often used with **aggregate functions** (SUM, COUNT, etc.). | Can be used with any query result; not necessarily with aggregates. |
| **Effect on Rows** | Combines multiple rows into **summary rows** per group. | Does not change rows; only rearranges their **display order**. |
| **Clause Position** | Comes **after WHERE** and **before HAVING** (if used). | Comes **at the end of the query**. |
| **Example** | SELECT Product, SUM(Amount) FROM Sales GROUP BY Product; | SELECT \* FROM Sales ORDER BY Amount DESC; |

**Key Notes:**

* You can **use both together**: first GROUP BY to aggregate, then ORDER BY to sort the grouped results.
* Example:

SELECT Product, SUM(Amount) AS TotalSales

FROM Sales

GROUP BY Product

ORDER BY TotalSales DESC;

Groups sales by product and displays **highest total sales first**.

✅ **In short:**

* GROUP BY → **Groups data** for aggregation.
* ORDER BY → **Sorts data** for better readability.
* Together, they help summarize and organize data efficiently.

**13. SQL Stored Procedure**

**Theory Questions:**

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

* **Definition:**
  + A **stored procedure** is a **precompiled collection of one or more SQL statements** stored in the database and executed as a single unit.
  + It can include **queries, DML statements (INSERT, UPDATE, DELETE), control-of-flow statements (IF, LOOP), and parameters**.
* **Difference between Stored Procedure and Standard SQL Query:**

| **Aspect** | **Stored Procedure** | **Standard SQL Query** |
| --- | --- | --- |
| **Definition** | Precompiled set of SQL statements stored in the database. | A single SQL statement executed directly. |
| **Execution** | Executed using EXEC or CALL command. | Executed directly in SQL editor or application. |
| **Performance** | Faster due to **precompilation** and execution plan reuse. | Slower for repeated execution as it is compiled each time. |
| **Reusability** | Can be **reused multiple times** with different parameters. | Each query must be written/executed separately. |
| **Parameters** | Supports **input, output, and input-output parameters**. | Cannot take parameters directly (must construct query dynamically). |
| **Complexity** | Can contain multiple statements, loops, and conditional logic. | Usually handles **simple operations** only. |
| **Security** | Users can be granted permission to execute without direct access to underlying tables. | Users need access to tables for execution. |

* **Example Stored Procedure (MySQL):**

CREATE PROCEDURE GetStudentByDept(IN DeptID INT)

BEGIN

SELECT \* FROM Students

WHERE DeptID = DeptID;

END;

* Execute the procedure:

CALL GetStudentByDept(2);

**2. Explain the advantages of using stored procedures.**

| **Advantage** | **Explanation** |
| --- | --- |
| **Performance** | Precompiled execution reduces parsing and compilation time, speeding up repeated queries. |
| **Reusability** | Can be called multiple times with different parameters, avoiding code duplication. |
| **Maintainability** | Centralized logic in the database makes updates easier without modifying client applications. |
| **Security** | Users can execute procedures without direct access to tables, reducing risk of SQL injection. |
| **Reduced Network Traffic** | Multiple SQL statements are executed on the server side, sending only final results to the client. |
| **Consistency** | Business logic implemented in stored procedures ensures consistent operations across applications. |

✅ **In short:** Stored procedures are **precompiled, reusable, secure, and efficient** blocks of SQL code that provide better performance and maintainability compared to standard SQL queries.

**14. SQL View**

**Theory Questions:**

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

* **Definition:**
  + A **view** in SQL is a **virtual table** that is derived from one or more base tables or other views.
  + It **does not store data physically** but displays data dynamically based on the underlying SQL query.
* **Difference between Table and View:**

| **Aspect** | **Table** | **View** |
| --- | --- | --- |
| **Definition** | Physical storage of data in rows and columns. | Virtual table created using a SQL query; no physical storage. |
| **Data Storage** | Stores actual data. | Does not store data; displays data from underlying tables. |
| **Updates** | Can be inserted, updated, or deleted directly. | Some views are **read-only**, but updatable views allow limited modifications. |
| **Complexity** | Simple structure with columns and rows. | Can combine multiple tables, include joins, aggregations, and filters. |
| **Dependencies** | Independent object; does not rely on other tables. | Depends on base tables; if a base table is dropped, the view becomes invalid. |
| **Security** | All columns are visible to users with table access. | Can **hide specific columns or rows** from users, providing selective access. |

* **Example:** Create a view to show only student names and departments:

CREATE VIEW StudentInfo AS

SELECT Name, DeptID

FROM Students;

* Query the view:

SELECT \* FROM StudentInfo;

Shows Name and DeptID without exposing other table columns.

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

| **Advantage** | **Explanation** |
| --- | --- |
| **Data Security** | Restricts access to sensitive columns or rows; users see only what is allowed. |
| **Simplified Queries** | Encapsulates complex joins or aggregations into a single query; users can query the view easily. |
| **Consistency** | Ensures consistent representation of data across multiple applications. |
| **Reusability** | Can be used in multiple queries without rewriting the underlying SQL. |
| **Logical Abstraction** | Provides a logical representation of data independent of physical table structure. |
| **Updatability (Optional)** | Some views allow **updating data indirectly**, making maintenance easier. |

✅ **In short:**

* Views are **virtual tables** that simplify access, improve security, and provide abstraction, without duplicating data.

**15. SQL Triggers**

**Theory Questions:**

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

* **Definition:**
  + A **trigger** is a special type of stored procedure that **automatically executes** in response to certain events on a table or view.
  + Triggers are used to **enforce rules, maintain integrity, and automate tasks** without manual intervention.
* **Key Features:**
  + Automatically executed when a specified event occurs.
  + Cannot be called directly by a user.
  + Can be associated with **INSERT, UPDATE, or DELETE** operations.
* **Types of Triggers:**

| **Type** | **Definition** | **Use Case / Example** |
| --- | --- | --- |
| **BEFORE Trigger** | Executes **before** the triggering event (INSERT, UPDATE, DELETE). | Validate or modify data before it is inserted or updated. |
| **AFTER Trigger** | Executes **after** the triggering event. | Log changes, update audit tables, or enforce business rules after data change. |
| **INSTEAD OF Trigger** | Executes **instead of** the triggering event (mainly used on views). | Allows updates/deletes on views that are otherwise not directly updatable. |

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

| **Trigger Type** | **Event** | **When Executed** | **Typical Use** |
| --- | --- | --- | --- |
| **INSERT Trigger** | INSERT operation | Fires when new rows are added to a table. | Automatically validate or log newly added data. |
| **UPDATE Trigger** | UPDATE operation | Fires when existing rows are modified. | Track changes, maintain history, or prevent invalid updates. |
| **DELETE Trigger** | DELETE operation | Fires when rows are deleted from a table. | Log deleted records, prevent accidental deletions, or maintain referential integrity. |

* **Example:** Logging changes with an AFTER UPDATE trigger:

CREATE TRIGGER LogStudentUpdate

AFTER UPDATE ON Students

FOR EACH ROW

BEGIN

INSERT INTO StudentAudit (StudentID, OldAge, NewAge, ChangeDate)

VALUES (OLD.StudentID, OLD.Age, NEW.Age, NOW());

END;

Every time a student’s age is updated, the old and new values are stored in StudentAudit.

✅ **In short:**

* **Triggers** automate tasks in response to table events.
* **BEFORE/AFTER/INSTEAD OF** define timing of execution.
* **INSERT/UPDATE/DELETE triggers** correspond to the type of DML operation they monitor.
* They are widely used for **data validation, auditing, and enforcing business rules**.

**16. Introduction to PL/SQL**

**Theory Questions:**

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

* **Definition:**
  + **PL/SQL (Procedural Language/SQL)** is Oracle's **procedural extension of SQL**.
  + It combines the power of **SQL for data manipulation** with **procedural programming constructs** like loops, conditions, and exceptions.
  + PL/SQL is used to write **blocks of code** for complex business logic, automation, and error handling in the database.
* **Key Features / Capabilities:**

| **Feature** | **Description** |
| --- | --- |
| **Procedural Constructs** | Supports IF-ELSE, loops (FOR, WHILE), and CASE statements. |
| **Variables & Constants** | Allows declaration and manipulation of variables, constants, and cursors. |
| **Error Handling** | Supports **exception handling** for runtime errors. |
| **Blocks of Code** | Programs can be organized into **anonymous blocks, procedures, functions, and packages**. |
| **Integration with SQL** | Can execute multiple SQL statements within a PL/SQL block efficiently. |

* **Example:** PL/SQL block to calculate bonus:

DECLARE

salary NUMBER := 50000;

bonus NUMBER;

BEGIN

bonus := salary \* 0.10;

DBMS\_OUTPUT.PUT\_LINE('Bonus: ' || bonus);

END;

This block calculates a 10% bonus and displays it.

✅ **In short:** PL/SQL extends SQL by adding **programming constructs** for automation, procedural control, and complex operations.

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

| **Benefit** | **Explanation** |
| --- | --- |
| **Improved Performance** | Multiple SQL statements can be executed as a **single block**, reducing network traffic. |
| **Procedural Capabilities** | Supports loops, conditions, and variables, enabling **complex business logic** inside the database. |
| **Error Handling** | Provides **exception handling**, allowing developers to manage runtime errors gracefully. |
| **Modularity** | Code can be organized into **procedures, functions, and packages** for reusability and maintenance. |
| **Security** | Sensitive operations can be encapsulated in PL/SQL blocks, restricting direct access to underlying tables. |
| **Integration** | Seamlessly integrates SQL operations with procedural logic, allowing **dynamic and conditional queries**. |
| **Portability** | PL/SQL code runs **inside Oracle database**, ensuring consistent behavior across applications. |

✅ **In short:**

* PL/SQL combines **SQL’s data handling** with **procedural programming**, making it ideal for **complex business logic, automation, and robust applications** in Oracle databases.

**17. PL/SQL Control Structures**

**Theory Questions:**

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

* **Definition:**
  + **Control structures** in PL/SQL are programming constructs that control the **flow of execution** of statements in a PL/SQL block.
  + They allow you to perform **conditional operations, repeated actions, or selective execution**.

**a) IF-THEN Control Structure**

* **Purpose:** Executes a block of code **only if a specified condition is true**.
* **Syntax:**

IF condition THEN

-- statements

ELSIF another\_condition THEN

-- statements

ELSE

-- statements

END IF;

* **Example:**

DECLARE

salary NUMBER := 50000;

BEGIN

IF salary > 40000 THEN

DBMS\_OUTPUT.PUT\_LINE('High salary');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Moderate salary');

END IF;

END;

Output: High salary (since 50000 > 40000)

* **Notes:**
  + Supports ELSIF for multiple conditions.
  + ELSE handles cases where no conditions are true.

**b) LOOP Control Structure**

* **Purpose:** Repeats a block of code **multiple times** until a certain condition is met.
* **Types of LOOPs:**

| **Loop Type** | **Description** |
| --- | --- |
| **Simple LOOP** | Executes repeatedly until an **EXIT** statement is used. |
| **WHILE LOOP** | Executes as long as a **condition is true**. |
| **FOR LOOP** | Executes a fixed number of iterations. |

* **Examples:**

**Simple LOOP:**

DECLARE

counter NUMBER := 1;

BEGIN

LOOP

DBMS\_OUTPUT.PUT\_LINE('Counter: ' || counter);

counter := counter + 1;

EXIT WHEN counter > 5;

END LOOP;

END; Output: Counter 1 to 5

**FOR LOOP:**

BEGIN

FOR i IN 1..5 LOOP

DBMS\_OUTPUT.PUT\_LINE('Iteration: ' || i);

END LOOP;

END;

Output: Iteration 1 to 5

**WHILE LOOP:**

DECLARE

counter NUMBER := 1;

BEGIN

WHILE counter <= 5 LOOP

DBMS\_OUTPUT.PUT\_LINE('Counter: ' || counter);

counter := counter + 1;

END LOOP;

END; Output: Counter 1 to 5

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

**Role of Control Structures in Complex Queries**

PL/SQL control structures—like **IF-THEN-ELSE**, **LOOP**, **WHILE**, and **FOR**—allow developers to implement **logic and decision-making** within database operations. They make it possible to write queries and operations that go **beyond standard SQL** capabilities.

**Key Benefits**

| **Control Structure Feature** | **How It Helps in Complex Queries** |
| --- | --- |
| **Conditional Execution (IF-THEN-ELSE)** | Executes different SQL statements based on runtime conditions. Example: give bonuses to employees only if salary > threshold. |
| **Iteration (LOOP, FOR, WHILE)** | Processes multiple rows or performs repetitive tasks without manually repeating SQL statements. Example: apply updates to all employees in a department. |
| **Dynamic Decision Making** | Allows selecting which SQL statements to execute based on intermediate results. Example: update different tables depending on status of a transaction. |
| **Error Handling Integration** | Control structures combined with EXCEPTION blocks can handle errors gracefully, allowing complex operations to continue or rollback safely. |
| **Modularity and Readability** | Breaks complex operations into smaller logical steps, making queries easier to maintain and debug. |
| **Automation** | Replaces multiple separate SQL statements with a single block, reducing network overhead and improving efficiency. |

**Example Use Case**

DECLARE

emp\_salary NUMBER;

BEGIN

FOR emp IN (SELECT EmployeeID, Salary FROM Employees) LOOP

IF emp.Salary > 50000 THEN

UPDATE Employees

SET Bonus = Salary \* 0.10

WHERE EmployeeID = emp.EmployeeID;

ELSE

INSERT INTO BonusLog(EmployeeID, Reason)

VALUES(emp.EmployeeID, 'No bonus');

END IF;

END LOOP;

END;

**Explanation:**

* Iterates through all employees using a **FOR LOOP**.
* Uses **IF-THEN** to decide whether to update the bonus or log a reason.
* Automates a complex operation that would be cumbersome using plain SQL.

✅ **In short:**

* **Control structures** in PL/SQL allow conditional logic, looping, and error handling.
* They **enhance SQL** by enabling automation, decision-making, and iterative processing, making it possible to handle **complex business rules** directly in the database.

**18. SQL Cursors**

**Theory Questions:**

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

* **Definition:**
  + A **cursor** in PL/SQL is a **pointer that allows traversal over the rows of a query result** one row at a time.
  + Cursors are used when **more than one row** is returned by a query, and you want to **process each row individually**.

**Types of Cursors**

| **Cursor Type** | **Definition** | **Key Features / Usage** |
| --- | --- | --- |
| **Implicit Cursor** | Automatically created by Oracle when a **DML statement** (INSERT, UPDATE, DELETE, SELECT INTO) returns a single row. | - Simple and automatic.- Cannot fetch multiple rows iteratively.- Example: SELECT INTO for a single row. |
| **Explicit Cursor** | Defined explicitly by the programmer to handle **queries that return multiple rows**. | - Requires declaration, opening, fetching, and closing.- Allows row-by-row processing.- Example: iterating over all employees in a table. |

**Example**

**Implicit Cursor Example:**

DECLARE

emp\_name VARCHAR2(50);

BEGIN

SELECT Name INTO emp\_name

FROM Employees

WHERE EmployeeID = 101;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || emp\_name);

END;

* Oracle automatically creates an implicit cursor.
* Only works if **one row is returned**.

**Explicit Cursor Example:**

DECLARE

CURSOR emp\_cursor IS

SELECT EmployeeID, Salary FROM Employees;

emp\_record emp\_cursor%ROWTYPE;

BEGIN

OPEN emp\_cursor;

LOOP

FETCH emp\_cursor INTO emp\_record;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('EmployeeID: ' || emp\_record.EmployeeID || ', Salary: ' || emp\_record.Salary);

END LOOP;

CLOSE emp\_cursor;

END;

* Cursor is explicitly declared, opened, and fetched.
* Processes **multiple rows** one at a time.

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

| **Scenario** | **Use Explicit Cursor?** | **Reason** |
| --- | --- | --- |
| Single-row query | ❌ Usually not needed | Implicit cursor is simpler and automatic. |
| Multi-row query | ✅ Required | Implicit cursor cannot handle multiple rows iteratively. |
| Row-by-row processing | ✅ Required | Explicit cursor allows fetching each row and performing logic per row. |
| Complex business logic | ✅ Helpful | Can include loops, conditions, and computations per row. |
| Dynamic control & exception handling | ✅ Helpful | Explicit cursors can be combined with control structures for custom operations. |

**Key Notes:**

* Implicit cursors are **easy to use** for simple queries returning one row.
* Explicit cursors provide **fine-grained control** for multi-row queries, loops, and complex operations.

✅ **In short:**

* **Cursors** let you process query results **row by row**.
* **Implicit cursors** → automatic, single-row processing.
* **Explicit cursors** → manual, multi-row processing with full control.

**19. Rollback and Commit Savepoint**

**Theory Questions:**

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

* **Definition of SAVEPOINT:**
  + A **SAVEPOINT** is a **marker within a transaction** that allows partial rollback.
  + It enables a transaction to **rollback to a specific point** without undoing the entire transaction.
* **Purpose:**
  + Provides **fine-grained control** over transactions.
  + Useful when executing multiple operations and only some need to be undone due to errors.

**Interaction with ROLLBACK and COMMIT**

| **Command** | **Effect on SAVEPOINT** |
| --- | --- |
| **ROLLBACK TO savepoint\_name** | Reverts the transaction to the state at the savepoint. Changes after the savepoint are undone. |
| **ROLLBACK (without savepoint)** | Reverts the **entire transaction**; all changes are undone, savepoints are lost. |
| **COMMIT** | Makes all changes **permanent** and removes all savepoints. |

**Example: Using SAVEPOINT**

BEGIN TRANSACTION;

INSERT INTO Employees(EmployeeID, Name) VALUES(101, 'Rahul');

SAVEPOINT BeforeBonus;

UPDATE Employees SET Bonus = 1000 WHERE EmployeeID = 101;

-- Error occurs

ROLLBACK TO BeforeBonus;

COMMIT;

**Explanation:**

1. Insert a new employee → changes are pending.
2. Set SAVEPOINT BeforeBonus.
3. Update bonus → changes are after savepoint.
4. Rollback to BeforeBonus → undo bonus update but keep employee insertion.
5. Commit → makes insertion permanent.

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

| **Scenario** | **Usefulness of SAVEPOINT** |
| --- | --- |
| **Complex Transactions** | When a transaction involves multiple steps; allows partial rollback without undoing all changes. |
| **Error Handling** | If an error occurs after a certain point, rollback only to that savepoint instead of the start. |
| **Batch Processing** | Useful in loops or batch inserts/updates; can rollback specific failed iterations. |
| **Testing and Debugging** | Helps developers test transactions step-by-step and undo intermediate steps. |
| **Nested Transactions Simulation** | In databases without true nested transactions, savepoints can **simulate nested transaction behavior**. |

✅ **In short:**

* **SAVEPOINT** → marks an intermediate point in a transaction.
* **ROLLBACK TO savepoint** → undoes changes after the savepoint, keeping previous changes intact.
* **COMMIT** → saves all changes permanently and clears savepoints.
* **Usefulness:** Provides **control, flexibility, and safety** in complex database operations.