Introduction To DBMS

1. Introduction to SQL

Theory Question:

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

Ans:

- **SQL allows you to:**

1. Create databases and tables.
2. Insert new records (data).
3. Read/Retrieve data using queries.
4. Update existing data.
5. Delete data.
6. Control access and manage database security.

- **Why SQL is Essential in Database Management**

1. Data Management – Helps in storing, organizing, and managing large volumes of data efficiently.
2. Easy Data Retrieval – SQL queries allow you to fetch exactly the data you need quickly.
3. Data Analysis – Useful for analyzing and reporting from business databases.
4. Standard Language – Almost all relational database systems (MySQL, Oracle, PostgreSQL, SQL Server) understand SQL.
5. Data Security – Provides permissions and roles to protect sensitive data.

2.Explain the difference between DBMS and RDBMS.

Ans:

1. DBMS (Database Management System)

It is software that stores and manages data**.**
Data is stored in files (can be hierarchical or navigational).
Relationships between data are not strongly maintained**.**

Examples:

Microsoft Access
File System (basic form)

Example:
Imagine a notebook where you write down student details. If you need to find a student's marks, you may have to search page by page.

2. RDBMS (Relational Database Management System)

It is an advanced type of DBMS where data is stored in tables
(rows and columns).
Relationships between tables are maintained using primary keys and foreign keys.
Supports SQL for managing data.

Examples:

MySQL
Oracle Database
PostgreSQL
SQL Server

Example:
Think of a well-organized Excel sheet where one table has student details and another has marks. You can link them using the student ID to find any record easily.

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

Ans:

Role of SQL in Managing Relational Databases

Relational Databases (RDBMS) store data in tables (rows and columns).
SQL is the language used to interact with these databases.

Main Roles of SQL

1. **Data Definition (DDL)**
   SQL helps to create, alter, and delete tables.
   Example:
   CREATE TABLE Students (id INT, name VARCHAR(50));

2. **Data Manipulation (DML)**
   Used to insert, update, delete, and retrieve data.
   Example:
   INSERT INTO Students VALUES (1, 'ved');

3. **Data Retrieval (DQL)**
   SQL makes it easy to query (search) data with conditions.
   Example:
   SELECT name FROM Students WHERE id = 1;

4. **Data Control (DCL)**
   Controls who can access or modify data (security).
   Example:
   GRANT SELECT ON Students TO user1

2. SQL Syntax

Theory Questions:

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

Ans:

The basic components of SQL syntax are:

1. **Keywords** – Reserved words that have special meaning in SQL (e.g., SELECT, INSERT, UPDATE, DELETE, FROM, WHERE).
2. **Identifiers** – Names of databases, tables, columns, or aliases (e.g., customers, order_id).
3. **Clauses** – Parts of an SQL statement that perform a specific function (e.g., SELECT clause, WHERE clause, ORDER BY clause).
4. **Expressions** – Combinations of columns, values, and functions used to produce a value (e.g., price * quantity).
5. **Operators** – Symbols or words used to compare or combine values (e.g., =, >, <, AND, OR, LIKE).
6. **Functions** – Built-in SQL methods for calculations or text manipulation (e.g., COUNT(), SUM(), UPPER()).
7. **Data Types** – Define the kind of data stored in a column (e.g., INT, VARCHAR, DATE).

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

Ans :

The general structure of an SQL SELECT statement is:

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

Explanation:

- **SELECT** → Specifies the columns to retrieve.
- **FROM** → Specifies the table to retrieve data from.

- **WHERE** → Filters rows based on a condition. *(optional)*
- **GROUP BY** → Groups rows that have the same values in specified columns. *(optional)*
- **HAVING** → Filters groups based on a condition. *(optional)*
- **ORDER BY** → Sorts the result set in ascending (ASC) or descending (DESC) order. *(optional)*

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

   In SQL, clauses are the building blocks of a statement — each clause performs a specific task in retrieving, filtering, grouping, or sorting data.

   Role of Clauses in SQL Statements

   1. **SELECT Clause**
      - Purpose: Defines which columns or calculated values to display.
      - **Example:**

        ```
        SELECT name, age
        FROM students;
        ```
   2. **FROM Clause**
      - Purpose: Specifies the table(s) from which to retrieve data.
      - **Example:**

        ```
        SELECT *
        FROM employees;
        ```
   3. **WHERE Clause**
      - Purpose: Filters rows based on a condition before grouping or sorting.
      - **Example:**

        ```
        SELECT *
        FROM employees
        WHERE salary > 50000;
        ```
   4. **GROUP BY Clause**
      - Purpose: Groups rows that have the same values in specified columns (used with aggregate functions).
      - **Example:**

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

5. **HAVING Clause**

- Purpose: Filters groups after aggregation (works like WHERE but for grouped data).
- **Example:**

```
SELECT department, COUNT(*)
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;
```

6. **ORDER BY Clause**

- Purpose: Sorts the result set in ascending (ASC) or descending (DESC) order.
- **Example:**

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

3. SQL Constraints

Theory Questions:

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

Types of Constraints in SQL

Constraint	**Description**	**Example**
NOT NULL	Ensures a column cannot have NULL values (empty values).	name VARCHAR(50) NOT NULL
UNIQUE	Ensures all values in a column are different.	email VARCHAR(100) UNIQUE
PRIMARY KEY	Uniquely identifies each record in a table. It is NOT NULL + UNIQUE.	id INT PRIMARY KEY
FOREIGN KEY	Creates a link between two tables. Ensures the value exists in another table's primary key.	FOREIGN KEY (dept_id) REFERENCES departments(id)
CHECK	Ensures that a column's values meet a condition.	age INT CHECK (age >= 18)
DEFAULT	Assigns a default value if no value is provided.	status VARCHAR(10) DEFAULT 'active'

Example :

```
CREATE TABLE Students (
   student_id INT PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   email VARCHAR(100) UNIQUE,
   age INT CHECK (age >= 18),
   course_id INT,
   status VARCHAR(10) DEFAULT 'active',
   FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

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

Ans:

Here's the difference between PRIMARY KEY and FOREIGN KEY in SQL in simple terms:

Feature	**PRIMARY KEY**	**FOREIGN KEY**
Purpose	Uniquely identifies each row in a table.	Creates a link between two tables.
Uniqueness	Must have unique values for every row.	Can have duplicate values (if many rows link to the same parent).
NULL values	Cannot be NULL.	Can be NULL (unless specified as NOT NULL).
Number per table	Only one primary key per table (can be single or multiple columns).	Can have multiple foreign keys in a table.
Relation	Works within the same table.	Refers to the PRIMARY KEY (or UNIQUE key) of another table.
Example	student_id INT PRIMARY KEY	course_id INT FOREIGN KEY REFERENCES Courses(course_id)

Example with both keys

```
CREATE TABLE Courses (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(50)
);

CREATE TABLE Students (
   student_id INT PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   course_id INT,
   FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);
```

- **Courses**: course_id is **PRIMARY KEY** (unique for each course).
- **Students**: course_id is **FOREIGN KEY** (points to Courses table).

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

Role of NOT NULL and UNIQUE constraints in SQL

1. NOT NULL Constraint

- Purpose: Ensures that a column must always have a value (cannot be left empty).
- Why use it: Prevents missing/unknown data in important fields.

Example:

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

Here, both student_id and name must have a value when inserting a record.

2. UNIQUE Constraint

- Purpose: Ensures that all values in a column are different.
- Why use it: Avoids duplicate entries for certain data like email or username.

Example:

```
CREATE TABLE Students (
    email VARCHAR(100) UNIQUE
);
```

Here, no two students can have the same email.

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

Theory Questions:

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

Ans:

Definition:

DDL is a part of SQL used to define, create, modify, and delete the structure of database objects such as tables, views, indexes, and schemas.

It works on the structure of the database, not the actual data.

Common DDL Commands:

Command	**Purpose**
CREATE	Creates new database objects (e.g., tables, views).
ALTER	Modifies the structure of existing objects (e.g., adding/removing columns).
DROP	Deletes database objects permanently.
TRUNCATE	Removes all data from a table but keeps its structure.
RENAME	Changes the name of a database object.

Example:

```
CREATE TABLE Students (
   student_id INT PRIMARY KEY,
   name VARCHAR(50),
   age INT
);
```

Here, CREATE TABLE is a DDL command.

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

Ans:

CREATE Command in SQL

Definition:

The CREATE command in SQL is a DDL (Data Definition Language) command used to create new database objects such as tables, views, indexes, or databases.
It defines the structure of the object.

Syntax (for creating a table):

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

Where:

- **table_name** → Name of the table.
- **column1, column2** → Names of columns.
- **datatype** → Type of data the column will store (INT, VARCHAR, DATE, etc.).
- **[constraint]** → Optional rules like PRIMARY KEY, NOT NULL, UNIQUE.

Example:

```
CREATE TABLE Students (
   student_id INT PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   age INT CHECK (age >= 18),
   email VARCHAR(100) UNIQUE
);
```

- **This creates a Students table with:**
  - **student_id** → Primary Key
  - **name** → Must have a value (NOT NULL)
  - **age** → Must be 18 or above
  - **email** → Must be unique

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

Purpose of Specifying Data Types and Constraints during Table Creation

1. **Data Types**

- Purpose: Define what kind of data can be stored in each column.
- Why important:
  - Saves storage space.
  - Improves performance.
  - Prevents invalid data (e.g., storing text in a number field).
- **Example:**

age INT, → Only numbers allowed
name VARCHAR(50) →Text up to 50 characters

2. **Constraints**

- Purpose: Enforce rules on data to maintain accuracy, validity, and consistency.
- **Why important:**
  - Prevents incorrect or duplicate data.
  - Maintains relationships between tables.
  - Ensures important fields are not empty.
- **Example:**

```
email VARCHAR(100) UNIQUE NOT NULL,  -- No duplicates, no empty values
age INT CHECK (age >= 18)            -- Age must be 18 or above
```

In short:

- **Data types** → Control the format and size of stored data.
- **Constraints** → Control the rules and validity of stored data.

5. ALTER Command

Theory Questions:

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

Ans:

Definition: The ALTER command is a DDL (Data Definition Language) command used to modify the structure of an existing database object (like a table) without deleting it.

Uses of ALTER command:

1. Add a new column to a table.
2. Modify the data type or size of an existing column.
3. Rename a column or table.
4. Drop (delete) a column from a table.
5. Add or remove constraints (e.g., PRIMARY KEY, UNIQUE).

Examples:

1. Add a new column

```
ALTER TABLE Students
ADD phone VARCHAR(15);
```

2. Modify a column's data type

```
ALTER TABLE Students
MODIFY name VARCHAR(100);
```

3. Drop (remove) a column

```
ALTER TABLE Students
DROP COLUMN phone;
```

4. Rename a table

```
ALTER TABLE Students
RENAME TO Learners;
```

In short:

- The ALTER command lets you change the design of a table without losing existing data.
- Do you want me to also give you a table of all ALTER operations for quick revision?

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

Ans:

ALTER TABLE is used to change a table's structure after it has been created.

1. Add a Column:

Syntax:

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

Example:

```
ALTER TABLE Students
ADD phone VARCHAR(15);
```

→ **Adds a new column phone**.

2. Modify a Column:

Syntax:

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

Example:

```
ALTER TABLE Students
MODIFY name VARCHAR(100);
```

→**Changes name column size to 100 characters.**

3. Drop a Column:

Syntax:

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

Example:

```
ALTER TABLE Students
DROP COLUMN phone;
```

→**Deletes the phone column.**

6. DROP Command

Theory Questions:

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

Ans:

Definition:

The DROP command is a DDL (Data Definition Language) command used to permanently delete a database object (like a table, database, view, or index) along with all its data and structure.

Function / Purpose:

- Removes the object from the database completely.
- Deletes both the structure and the stored data.
- Cannot be undone once executed (unless you have a backup).

Syntax:

DROP object_type object_name;

Examples:

1. Drop a table:

DROP TABLE Students;

→Deletes the Students table and all its data.

2. Drop a database:

DROP DATABASE SchoolDB;

→Deletes the entire SchoolDB database.

3. Drop a view:

DROP VIEW StudentView;

→Deletes a view named StudentView.

- **Important:**
  - Use DROP carefully because it cannot be rolled back.
  - If you just want to remove data but keep the table structure, use TRUNCATE or DELETE instead.

2. What are the implications of dropping a table from a database?
Ans:
Implications of Dropping a Table in SQL
When you run:
DROP TABLE table_name;
it has the following effects:

1. Permanent Deletion
- The entire table is removed from the database.
- Both the structure and all the data inside the table are lost.
- Cannot be undone (unless you have a backup).

2. Loss of Constraints and Relationships
- All constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, etc.) on that table are deleted.
- If other tables have FOREIGN KEYS referencing this table, the DROP command may fail unless you remove those references first.

3. Impact on Dependent Objects
- Views, stored procedures, or queries using this table will stop working (errors will occur).
- Any indexes on the table will also be deleted automatically.

4. Cannot Roll Back in Many Cases
- In most SQL systems, DROP TABLE is auto-committed, meaning you can't roll it back after execution.

- **In short:**
  Dropping a table = table gone forever (data, structure, constraints, and dependencies are lost). Always take a backup before using it.

7.Data Manipulation Language (DML)

Theory Questions:

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

Ans:

Here’s a simple explanation of the INSERT, UPDATE, and DELETE

commands in SQL:

1. INSERT Command

Purpose:

Adds new data (a row/record) into a table.

Syntax:

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

Example:

```
INSERT INTO Students (student_id, name, age)
VALUES (1, 'Rahul', 20);
```

2. UPDATE Command

Purpose:

Changes or updates existing data in a table.

Syntax:

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

Example:

```
UPDATE Students
SET age = 21
WHERE student_id = 1;
```

→**Always use WHERE to avoid updating all rows by mistake.**

3. DELETE Command

Purpose:

Removes one or more rows from a table.

Syntax:

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

Example:

```
DELETE FROM Students
WHERE student_id = 1;
```

→ **Without WHERE, it will delete all rows.**

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

Ans:

The WHERE clause is extremely important in UPDATE and DELETE commands because it controls which rows in the table are affected.

Why it's important:

1. Targets specific rows

It lets you apply the change only to rows that meet certain conditions.
Without it, every row in the table may be updated or deleted.

UPDATE:

UPDATE Example (with WHERE):

```
UPDATE Students
SET age = 21
WHERE student_id = 1;
```

→Only the student with student_id = 1 gets updated.

UPDATE Example (without WHERE):

```
UPDATE Students
SET age = 21;
```

→All students will have their age set to 21 – likely a mistake.

DELETE:

DELETE Example (with WHERE):

```
DELETE FROM Students
WHERE age < 18;
```

→ Deletes only students under age 18.

DELETE Example (without WHERE):

```
DELETE FROM Students;
```

Deletes ALL rows from the Students table.

In short:

The WHERE clause is like a filter.

Without it, you might accidentally change or delete the entire table.

8. Data Query Language (DQL)

Theory Questions:

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

Ans:

What is it?

The SELECT statement is the most commonly used SQL command.
It is used to query (fetch) data from one or more tables in a database.

Basic Syntax:

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

How it works:

- **SELECT** → Tells SQL what columns you want.
- **FROM** → Tells SQL which table to get the data from.
- **WHERE** → (Optional) Filters the results based on a condition.

Examples:

1. Select all columns:

```
SELECT * FROM Students;
```

→ Fetches all rows and columns from the Students table.

2. Select specific columns:

```
SELECT name, age FROM Students;
```

→Fetches only the name and age of all students.

3. Select with condition (WHERE):

```
SELECT name FROM Students
WHERE age > 18;
```

→Fetches names of students who are older than 18.

4. Select with sorting:

```
SELECT name, age FROM Students
ORDER BY age DESC;
```

→Fetches names and ages of students, sorted by age (highest first).

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

Ans:

ORDER BY and WHERE Clauses in SQL

1.WHERE Clause

Purpose:

The WHERE clause is used to filter rows. It selects only those rows that match a given condition.

Syntax:

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

Example:

```
SELECT name, age
FROM Students
WHERE age >= 18;
```

→This query shows the names and ages of students who are 18 or older.

2.ORDER BY Clause

Purpose:

The ORDER BY clause is used to sort the result in ascending (ASC) or descending (DESC) order.

Syntax:

```
SELECT column1, column2
FROM table_name
ORDER BY column1 ASC|DESC;
```

Example:

```
SELECT name, age
FROM Students
ORDER BY age DESC;
```

→This query shows all students sorted by age from highest to lowest.

3.Using WHERE and ORDER BY Together

Example:

```
SELECT name, age
FROM Students
WHERE age >= 18
ORDER BY name ASC;
```

→This query selects students who are 18 or older and sorts them by name in alphabetical order.

9.Data Control Language (DCL)

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

Ans:

1. GRANT Command

- Used to give permission to a user.
- Lets the user do actions like SELECT, INSERT, UPDATE, DELETE on a table.

Syntax:

```
GRANT permission_name
ON table_name
TO user_name;
```

Example:

```
GRANT SELECT, INSERT
ON Students
TO user1;
```

→This allows user1 to view and add data to the Students table.

2. REVOKE Command

- Used to take back permission from a user.
- Stops the user from doing certain actions on a table.

Syntax:

```
REVOKE permission_name
ON table_name
FROM user_name;
```

Example:

```
REVOKE INSERT
ON Students
FROM user1;
```

→This stops user1 from inserting data into the Students table.

2.How do you manage privileges using these commands?

Ans:

Managing Privileges in SQL

You can control who can do what in the database using:

- **GRANT** → Give permission
- **REVOKE** → Remove permission

1. GRANT – To Give Permission

- Use GRANT when you want to allow a user to do something with a table (like see data, add data, change data).

Example:

```
GRANT SELECT, INSERT
ON Students
TO user1;
```

→This means user1 can see and add data in the Students table.

2. REVOKE – To Remove Permission

- Use REVOKE when you want to take back permission from a user.

Example:

```
REVOKE INSERT
ON Students
FROM user1;
```

→Now user1 cannot add data in the Students table anymore.

Why use GRANT and REVOKE?

- To control who can access or change data.
- To keep your database safe.
- To give different users different levels of access.

10. Transaction Control Language (TCL)

Theory Questions:

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

In SQL, COMMIT and ROLLBACK are transaction control commands used to manage changes made by SQL statements. Their main purpose is to maintain data integrity and allow controlled execution of a set of operations.

- COMMIT

- Purpose: To save all the changes made during the current transaction to the database permanently.
- Once a COMMIT is issued, the changes cannot be undone.
- It ends the current transaction.

Example:

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;
```

→This transfers ₹100 from account 1 to account 2 and saves it permanently.

- ROLLBACK

- Purpose: To undo all changes made in the current transaction.
- It is used when an error occurs or when you decide not to apply the changes.
- It restores the database to the state it was in before the transaction began.

Example:

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
ROLLBACK;
```

→This cancels the transfer — no changes will be saved.

2.Explain how transactions are managed in SQL databases.

Ans:

What is a Transaction?

A transaction is a set of SQL commands that work together like one unit.

It means:

Either all the steps inside the transaction should run successfully,
or none of them should happen.

Example: Transferring money

1. ₹100 is subtracted from Account A
2. ₹100 is added to Account B
   Both steps must happen together. If one fails, both should be cancelled.

How Transactions Work in SQL

1. **Start the transaction**
   → It begins automatically or by using:

```
BEGIN TRANSACTION;
```

1. **Run your SQL commands**

   Example:

```
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
```

2. **End the transaction using COMMIT or ROLLBACK**

   1. **Use COMMIT;**
      If everything is successful → save the changes permanently.
   2. **Use ROLLBACK;**
      If any problem occurs → cancel all the changes.

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:

Types of JOINs:

1. INNER JOIN:
Returns only matching rows from both tables.

2. LEFT JOIN (Left Outer Join):
Returns all rows from the left table, and matching rows from the right.
No match → NULL.

3. RIGHT JOIN (Right Outer Join):
Returns all rows from the right table, and matching rows from the left.
No match → NULL.

4. FULL OUTER JOIN:
Returns all rows from both tables.
No match → NULL on missing side.

Summary Table:

JOIN Type	**Returns**
INNER JOIN	Only matching rows
LEFT JOIN	All left + matching right rows
RIGHT JOIN	All right + matching left rows
FULL OUTER JOIN	All rows from both (matched + unmatched)

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

Ans:

JOIN is used to combine data from two or more tables based on a common column (like an ID).

We use JOIN when the data we need is stored in different tables, but we want to see it together.

Example:

Students Table

student_id	**name**
1	Rahul
2	Priya

Marks Table

student_id	**marks**
1	85
2	90

We want to get the student’s name and marks in one result.

How to Use JOIN:

```
SELECT Students.name, Marks.marks
FROM Students
JOIN Marks
ON Students.student_id = Marks.student_id;
```

This SQL query:

- Combines data from both tables
- Matches rows where student_id is the same
- Shows name and marks together

12. SQL Group By

Theory Questions:

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

Ans:

GROUP BY is used to group same type of data together.

It is used with aggregate functions like:

SUM() → total

COUNT() → counting rows

AVG() → average

MAX() → biggest value

MIN() → smallest value

Suppose you want to know:

→ Total marks of each student

→ Number of products sold in each city

Then you use GROUP BY.

Example Table: Marks

student_name	subject	marks
Rahul	Math	80
Rahul	Science	70
Priya	Math	90
Priya	Science	85

SQL Query:

```
SELECT student_name, SUM(marks)
FROM Marks
GROUP BY student_name;
```

It gives:

student_name	total_marks
Rahul	150
Priya	175

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

Ans:

Point	**GROUP BY**	**ORDER BY**
Purpose	Used to group rows with the same values	Used to sort rows in ascending or descending order
Function Usage	Used mostly with aggregate functions like SUM(), COUNT(), AVG()	Does not require aggregate functions
Result	Returns one row for each group	Returns all rows, just sorted
Changes in Data?	Yes, it summarizes data	No, it only changes the order of rows
Grouping	Groups data based on column(s)	Sorts data based on column(s)
Default Sorting	No default sorting	Default is ascending (ASC) order
Syntax	GROUP BY column_name	`ORDER BY column_name [ASC
Used With	Reports, summaries, totals (e.g., total sales per city)	Viewing data in a specific order (e.g., names A–Z)
Example	SELECT city, COUNT(*) FROM Customers GROUP BY city;	SELECT * FROM Customers ORDER BY name ASC;

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:
A Stored Procedure is a pre-written set of SQL statements that are saved in the database. You can call (run) the stored procedure whenever you want, instead of writing the same SQL code again and again.
Example of a Stored Procedure:
```
CREATE PROCEDURE GetAllStudents()
AS
BEGIN
  SELECT * FROM Students;
END;
```

To run this procedure:
```
EXEC GetAllStudents;
```

How it Differs from a Standard SQL Query

Point	**Stored Procedure**	**Standard SQL Query**
Definition	A saved set of SQL commands	A single SQL command written and run as needed
Reusability	Can be used again and again	Must be written again each time
Stored in DB?	Yes, stored in the database	No, just written and executed directly
Can take input?	Yes, it can take parameters	Usually works with fixed data
Performance	Faster for repeated tasks (precompiled)	Slower if repeated multiple times

Point	Stored Procedure	Standard SQL Query
Use Case	Used for complex tasks, automation, and reuse	Used for one-time or simple queries

2.Explain the advantages of using stored procedures.

Ans:

Advantages of Using Stored Procedures in SQL

Advantage	Explanation
Reusability	You can write the code once and use it again and again.
Faster Execution	Stored procedures are precompiled, so they run faster than normal SQL queries.
Better Security	Users can be given permission to run the procedure without giving access to the tables.
Less Code Repetition	No need to write the same SQL again and again.
Easier Maintenance	If changes are needed, update the procedure in one place only.
Can Accept Parameters	You can pass values (like IDs, dates, etc.) into stored procedures for flexible results.
Supports Logic	You can use IF, WHILE, loops inside stored procedures — just like in programming.
Improves Productivity	Developers save time by using ready-made procedures for repeated tasks.

14. SQL View

Theory Questions:

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

Ans:

A View is a virtual table in SQL.
It is created using a SELECT query and does not store data itself.
Instead, it shows data from one or more real tables.

Syntax to Create a View:

```
CREATE VIEW view_name AS
SELECT column1, column2
FROM table_name
WHERE condition;
```

Example:

If we have a table Employees, we can create a view to show only employees from the "HR" department:

```
CREATE VIEW HR_Employees AS
SELECT name, department
FROM Employees
WHERE department = 'HR';
```

Now we can use:

```
SELECT * FROM HR_Employees;
```

This will show only HR employees.

Difference Between View and Table

Point	View	Table
Definition	A virtual table created from a SELECT query	A real table that stores data
Stores Data?	No, it does not store data	Yes, it stores data physically
Based On	Based on one or more tables	Independent, base structure

Point	View	Table
Usage	Used to simplify complex queries	Used to store actual data
Updatable	Some views can be updated, some cannot	Tables can always be updated
Space Required	No extra space needed	Takes space in the database
Security	Can hide sensitive columns from users	Shows all data unless restricted

2.Explain the advantages of using views in SQL databases

Ans:

Advantages of Using Views in SQL

1. **Simplifies Complex Queries**
   Views help simplify long or complex SQL queries by saving them once and using them multiple times.
2. **Increases Security**
   Views can hide sensitive data by showing only selected columns to users.
3. **No Extra Storage Needed**
   Views do not store data themselves, so they save storage space in the database.
4. **Easy to Reuse**
   Once created, a view can be used in many queries without writing the same SQL again.
5. **Auto-updated Data**
   When the original table changes, the view automatically shows the updated data.
6. **Improves Data Control**
   By giving access to views instead of full tables, you can control what data users can see.
7. **Joins Made Simple**
   Views can join data from multiple tables and show it as one virtual table.
8. **Better Organization**
   Frequently used queries can be stored in views to keep the code clean and easy to manage.

15. SQL Triggers

Theory Questions:

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

Ans:

A trigger is a special type of program in SQL that automatically runs (fires) when a certain action happens on a table.

It is used to perform tasks automatically like checking data, updating other tables, or preventing wrong changes.

A trigger works when events like INSERT, UPDATE, or DELETE occur.

Types of Triggers in SQL

1. **BEFORE Trigger**
   - Runs before the action (INSERT/UPDATE/DELETE) happens.
   - Used to check or change data before it is saved.
2. **AFTER Trigger**
   - Runs after the action has happened.
   - Used to log changes, update other tables, or perform audits.
3. **INSTEAD OF Trigger**
   - Used instead of the action, usually on views.
   - Helpful when you want to customize what happens instead of a regular insert/update/delete.

When Triggers Are Used

- To automatically update another table when data changes
- To validate data before saving it
- To log changes in a separate table (audit trail)
- To prevent invalid actions (like deleting important data)
- To enforce business rules without needing application code

Example

```
CREATE TRIGGER check_salary
BEFORE INSERT ON Employees
FOR EACH ROW
BEGIN
  IF NEW.salary < 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Salary cannot be negative';
  END IF;
END;
```

→This trigger stops anyone from inserting a negative salary.

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

Trigger Type	When It Runs	Use Case (Why Used)	Example Task
INSERT Trigger	Runs when a new row is added to a table	To check or modify data before adding	Stop inserting invalid salary values
UPDATE Trigger	Runs when a row is updated	To track changes or validate updates	Log old and new values in a log table
DELETE Trigger	Runs when a row is deleted	To prevent important data from being deleted, or log it	Save deleted data in backup table

Short Description of Each:

1. **INSERT Trigger**
   - Runs automatically when new data is added
   - Example: Check if age is more than 18 before adding a student
2. **UPDATE Trigger**
   - Runs when existing data is changed
   - Example: Log the old salary before updating it to new salary
3. **DELETE Trigger**
   - Runs when data is removed from the table
   - Example: Copy deleted row to archive table before deleting

16. Introduction to PL/SQL

Theory Questions:

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

Ans:

PL/SQL stands for Procedural Language for SQL.

It is Oracle's extension of SQL that adds programming features like:

- Variables
- Conditions (IF statements)
- Loops (FOR, WHILE)
- Functions and Procedures
- Error Handling (EXCEPTION)

How PL/SQL Extends SQL's Capabilities

1. **Adds Programming Logic to SQL**
   – SQL can only run one command at a time. PL/SQL allows multiple commands to run with logic (like IF, LOOP, etc.).
2. **Supports Variables and Constants**
   – You can create and use variables to store data and process it.
3. **Allows Use of Loops and Conditions**
   – You can write conditional logic and loops in PL/SQL, which is not possible in normal SQL.
4. **Can Create Procedures and Functions**
   – PL/SQL lets you create reusable blocks of code called procedures and functions.
5. **Error Handling**
   – PL/SQL provides exception handling to manage errors smoothly.
6. **Better Performance**
   – PL/SQL blocks are sent to the database as a single unit, which can reduce time and increase performance.

Example Of PL/SQL Block

```
DECLARE
  total NUMBER := 0;
BEGIN
  SELECT salary INTO total FROM employees WHERE id = 101;
  DBMS_OUTPUT.PUT_LINE('Salary is: ' || total);
END;
```

→This block uses variables, a SELECT query, and output — something not possible with plain SQL alone.

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

Benefits of Using PL/SQL

1. **Combines SQL with Programming Logic**
   PL/SQL allows you to use SQL with programming features like IF, LOOP, and CASE, which plain SQL doesn’t support.

2. **Supports Code Reuse**
   You can write procedures, functions, and packages once and reuse them many times.

3. **Improves Performance**
   PL/SQL code runs inside the database, reducing the number of calls between application and database — making it faster.

4. **Better Error Handling**
   PL/SQL has a built-in exception handling system to catch and handle errors easily.

5. **Saves Time with Blocks**
   Multiple SQL statements can be grouped in a single PL/SQL block and executed together, which saves time.

6. **Easier Maintenance**
   Since logic is stored in procedures or packages, changes can be made easily without touching the full application code.

7. **Increases Security**
   You can restrict user access to only run certain PL/SQL procedures without giving direct access to tables.

8. **Helps in Business Logic Implementation**
   PL/SQL is great for writing complex business rules directly in the database.

17. PL/SQL Control Structures.

Theory Questions:

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

Ans:

Control structures are used to control the flow of a program.
They help you make decisions and repeat actions in PL/SQL.
There are 3 main types of control structures:

1. **Conditional Control** → like IF-THEN, IF-THEN-ELSE, CASE
2. **Looping Control** → like LOOP, WHILE, FOR
3. **Sequential Control** → normal step-by-step execution

1. IF-THEN Control Structure

It is used to check a condition and run code only if the condition is true.

Syntax:

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

Example:

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

You can also use:

- IF...THEN...ELSE
- IF...THEN...ELSIF...ELSE for more conditions.

2. LOOP Control Structure

It is used to repeat a block of code again and again.
Types of Loops in PL/SQL:

- LOOP...EXIT WHEN
- WHILE LOOP
- FOR LOOP

Example of Basic LOOP:

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

→This will print numbers 1 to 5.

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

Ans:

Control structures in PL/SQL allow you to:

1. Add Decision-Making to SQL

- With IF-THEN, you can write logic to check conditions and run specific code.
- Example: Update bonus only if salary is above a limit.

2. Repeat Tasks with Loops

- Loops like FOR, WHILE, or LOOP...EXIT let you run SQL statements multiple times.
- Example: Insert 100 rows in a table using a loop.

3. Handle Different Cases

- Using IF...ELSIF or CASE, you can run different SQL queries based on different values.
- Example: Give a grade (A, B, C) based on marks using IF...ELSE.

4. Improve Code Structure

- Control structures help you write clean and organized logic inside your PL/SQL blocks.
- Instead of writing many separate SQL statements, you can combine them with logic in one block.

5. Make Queries Smart and Efficient

- You can avoid running unnecessary queries by checking conditions first.
- Example: Only run an UPDATE if the data really needs to change.

18. SQL Cursors

Theory Questions:

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

Ans:

What is a Cursor in PL/SQL:-

A cursor is a pointer that helps you fetch and process data row by row from a query result. When a query returns multiple rows, a cursor is used to go through each row one by one.

PL/SQL has two types of cursors:

1. Implicit Cursor

- Created automatically by PL/SQL when you run a SQL statement (like SELECT, INSERT, UPDATE, DELETE).
- You don't need to declare it.
- Used when a query returns only one row.

Example:

```
BEGIN
  UPDATE employees SET salary = salary + 1000 WHERE dept = 'Sales';
  IF SQL%ROWCOUNT > 0 THEN
    DBMS_OUTPUT.PUT_LINE('Rows updated: ' || SQL%ROWCOUNT);
  END IF;
END;
```

2. Explicit Cursor

- Created manually by the programmer to handle queries that return more than one row.
- You need to:
  - Declare it
  - Open it
  - Fetch from it
  - Close it

Example:

```
DECLARE
  CURSOR emp_cursor IS SELECT name FROM employees;
  emp_name employees.name%TYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_name;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(emp_name);
  END LOOP;
  CLOSE emp_cursor;
END;
```

Difference Between Implicit and Explicit Cursors

Feature	**Implicit Cursor**	**Explicit Cursor**
Created by	PL/SQL automatically	Programmer manually
When used	For single-row queries	For multiple-row queries
Declaration needed	No	Yes
Control	Limited	Full control over row processing
Example use	UPDATE, DELETE, INSERT	SELECT that returns many rows

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

Ans:

I use an explicit cursor when:

1. The Query Returns More Than One Row

- Implicit cursors handle only one row.
- Use an explicit cursor when you need to process each row one by one.

Example:

Display the names of all employees in a department.

2. You Need Full Control Over Row Processing

- With explicit cursors, you can:
    - Open the cursor
    - Fetch each row
    - Exit when done
    - Close it
- This gives you step-by-step control.

3. You Want to Use Looping with Rows

- When you want to loop through all rows in a query result, use an explicit cursor.
- Great for printing, calculations, or condition checks for each row.

4. You Want to Use Cursor Attributes

- Explicit cursors give you useful attributes like:
    - %FOUND, %NOTFOUND, %ROWCOUNT, %ISOPEN

5. Complex Business Logic Per Row

- When each row needs special handling or logic, an explicit cursor is better.

Summary Table

Use Case	**Use Explicit Cursor?**
Query returns many rows	Yes
Need full control with loop/fetch	Yes
One-time update/insert/delete	No (Use Implicit)
Need row-by-row calculations	Yes
Just checking if rows exist	No (Use Implicit)

19.Rollback and Commit Savepoint

Theory Questions:

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

Ans:

A SAVEPOINT is a marker (or checkpoint) that you set inside a transaction.
It lets you partially roll back the transaction to a specific point without undoing everything.

Why Use SAVEPOINT?

- You are doing many steps in one transaction.
- You want to go back to a certain step if something goes wrong — but not cancel the whole thing.
- It gives you better control over your transaction.

How ROLLBACK Works with SAVEPOINT

- ROLLBACK TO SAVEPOINT name;
  → Cancels all changes after that savepoint, but keeps the changes before it.
- ROLLBACK;
  → Cancels everything since the transaction began, including all savepoints.

How COMMIT Works with SAVEPOINT

- COMMIT; saves all changes in the transaction.
- After a COMMIT, all savepoints are removed — you cannot roll back to them anymore.

Example

```
BEGIN;

SAVEPOINT sp1;
-- Step 1: Insert record A

SAVEPOINT sp2;
-- Step 2: Insert record B

ROLLBACK TO sp2;
-- → Undo Step 2 only, Step 1 is still there

COMMIT;
 -- → Finalize everything that's left
```

Summary Table

Command	What It Does
SAVEPOINT name	Set a checkpoint in the transaction
ROLLBACK TO name	Undo changes after the savepoint
ROLLBACK	Undo all changes in the transaction
COMMIT	Save all changes and remove savepoints

2. When is it useful to use savepoints in a database transaction?
Ans:
1. When You Want Partial Rollback
If your transaction has many steps, and only one step fails, you can rollback only that part using a savepoint instead of canceling the whole transaction.

2. When Working with Complex Transactions
For example, inserting into multiple tables. If one insert fails, you can go back to the last savepoint and try again without losing earlier successful inserts.

3. When You Want Better Error Handling
Savepoints help you handle errors smoothly. You can test parts of a transaction and decide whether to continue or rollback to a safe point.

4. When You Need to Test or Validate Data
If you want to check some conditions after a few steps, and if those checks fail, you can rollback to the savepoint instead of starting over.

4. **When Performance Matters**
   Without savepoints, rolling back a full transaction can be costly. Savepoints help save time by keeping successful parts intact.

Example Situation
You are transferring money between two bank accounts:
1. Deduct from Account A – success
2. Add to Account B – fails (account not found)
3. Rollback only to step 1 using a savepoint
4. Fix issue or try a different account
5. Commit the correct steps

Without savepoints, step 1 would also be undone.