Module 4 – Introduction to DBMS

1. Introduction to SQL

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

What is SQL?

SQL (Structured Query Language) is a standard programming language used to communicate with and manage relational databases.

It is used to:

- Create database structures (tables, views, indexes)
- Insert, retrieve, update, and delete data
- Control access and security of the database

Why is it essential in database management?

- **SQL (Structured Query Language)** is a standard language for managing relational databases. It is essential because it allows creating, modifying, and querying databases efficiently, ensures data integrity, provides security, and is widely used in all database-driven applications.

2. Explain the difference between DBMS and RDBMS.

Feature	DBMS (Database Management System)	RDBMS (Relational Database Management System)
Data Storage	Stores data as files	Stores data in tables (rows & columns)
Relationships	Does not support relationships	Supports relationships using primary key & foreign key
Normalization	No normalization	Supports normalization to reduce redundancy
Data Integrity	Limited	Enforces data integrity with constraints
Examples	dBase, File System	MySQL, Oracle, PostgreSQL, SQL Server

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

1.DDL – Data Definition Language

- Creates and changes the structure of the database (tables, columns, etc.)

Main commands:

- CREATE – Make a new table/database
- ALTER – Change a table (add/remove column)
- DROP – Delete a table/database
- TRUNCATE – Remove all data but keep table structure

2.DML – Data Manipulation Language

- Works with the data inside tables (add, change, remove data)

Main commands:

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

3.DQL – Data Query Language

- Retrieves data from the database

Main command:

- SELECT – Get data based on a condition

4.DCL – Data Control Language

- Controls who can access or change data

Main commands:

- Commit – The COMMIT command in SQL is used to save all changes made in the current transaction permanently to the database.
- Rollback – The ROLLBACK command in SQL is used to undo changes made in the current transaction before they are committed.

4. What are the key features of SQL?

Here's the key features of SQL in very simple words:

1. Store and get data – You can save and read data easily.
2. Standard language – Works in almost all databases.
3. Many operations – Create tables, add data, update, delete, search.
4. Powerful search – Can filter, sort, and combine data from many tables.
5. Data safety – Rules to keep data correct (like Primary Key, Foreign Key).
6. Security – Can give or remove permission for users.
7. Transactions – Can save changes or undo mistakes (COMMIT, ROLLBACK).

2. SQL Syntax

1. What are the basic components of SQL syntax?

Basic components of SQL syntax:

1. Keywords – Special reserved words (like SELECT, INSERT, WHERE) used to tell the database what to do.
2. Identifiers – Names for tables, columns, databases (like students, name, marks).
3. Clauses – Parts of a query that perform a specific task (like WHERE, ORDER BY, GROUP BY).
4. Expressions – Formulas or conditions that give a value (like marks > 50).
5. Predicates – Conditions that return true/false (like WHERE age > 18).
6. Statements – Complete commands (like SELECT * FROM students;).
7. Semicolon (;) – Ends a statement.

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

General structure of an SQL SELECT

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

Explanation of parts:

SELECT – Columns you want to display.

FROM – Table from which data is taken.

WHERE – Filter rows based on a condition.

GROUP BY – Group rows that have the same values.

HAVING – Filter groups (used after GROUP BY).

ORDER BY – Sort the result (ascending/descending).

3. Explain the role of clauses in SQL statements.

Role of clauses in SQL statements:

- Clauses are parts of SQL statements that tell the database how to work with the data.
- They help in filtering, grouping, and sorting data.

Examples:

- WHERE clause → Chooses rows based on a condition.
- GROUP BY clause → Groups rows with the same value.
- HAVING clause → Applies condition to groups.
- ORDER BY clause → Sorts the result.

3. SQL Constraints

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

What are Constraints in SQL?

- Constraints are rules in SQL that control what type of data can be stored in a table.
- They help keep the data correct, valid, and safe.

Types of Constraints (with simple meaning):

1. PRIMARY KEY
    - Uniquely identifies each row in a table.
    - Cannot be NULL and cannot have duplicate values.
    - Example: student_id in a students table.
2. FOREIGN KEY
    - Connects two tables.
    - Ensures the value exists in another table's primary key.
    - Example: course_id in enrollments table refers to course_id in courses table.
3. UNIQUE
    - Ensures all values in a column are different.
    - Can have only one NULL value.
4. NOT NULL
    - Ensures a column cannot have NULL (empty) values.
5. CHECK
    - Ensures a column's values meet a condition.
    - Example: CHECK (age >= 18)
6. DEFAULT
    - Sets a default value for a column if no value is given.

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

PRIMARY KEY	FOREIGN KEY
Uniquely identifies each row in its own table.	Links one table to another table.
Cannot have NULL values.	Can have NULL values (if allowed).
Only one primary key per table.	Can have many foreign keys in a table.
Values must be unique in the column.	Values can be repeated if they exist in the referenced primary key.
Example: student_id in Students table.	Example: student_id in Enrollments table refers to Students.student_id.

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

1. NOT NULL Constraint

- Definition:
  Ensures that a column must always have a value. It cannot be left blank when inserting or updating a record.
- Key Points:
  - Applied at column level.
  - Prevents storing NULL values in that column.
  - Often used for important fields like name, date of birth, etc.

Example:

```
CREATE TABLE Students (
   student_id INT PRIMARY KEY,
   name VARCHAR(50) NOT NULL,  -- Name cannot be empty
   age INT
);
```

2. UNIQUE Constraint

- Definition:
  Ensures that all values in a column are different from each other.
- Key Points:
  - Allows only one occurrence of each value in that column.
  - NULL is allowed (but usually only one NULL).
  - A table can have multiple UNIQUE constraints on different columns.
  - Often used for fields like email, username, or phone number.

Example:

```
CREATE TABLE Users (
   user_id INT PRIMARY KEY,
   email VARCHAR(100) UNIQUE,  -- No two users can have the same email
   phone VARCHAR(15) UNIQUE    -- No two users can have the same phone
);
```

- NOT NULL → Prevents missing data.
- UNIQUE → Prevents duplicate data.

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

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

SQL Data Definition Language (DDL)

- DDL is the part of SQL used to create, change, and delete the structure of database objects (like tables, views, indexes).
- It works with the structure of the database, not the data inside.
- Changes made by DDL commands are permanent (auto COMMIT).

Main DDL Commands:

1. **CREATE** – Make a new table, database, view, etc.
2. **ALTER** – Change the structure (add/remove columns, modify datatype).
3. **DROP** – Delete the table or database completely.
4. **TRUNCATE** – Remove all rows from a table (but keep structure).
5. **RENAME** – Change the name of a table or column.

2. Explain the CREATE command and its syntax.

Create command

- The CREATE command is used to make new database objects like a database, table, or view.
- Most common use: Create a table with columns and their data types.

Syntax for CREATE TABLE

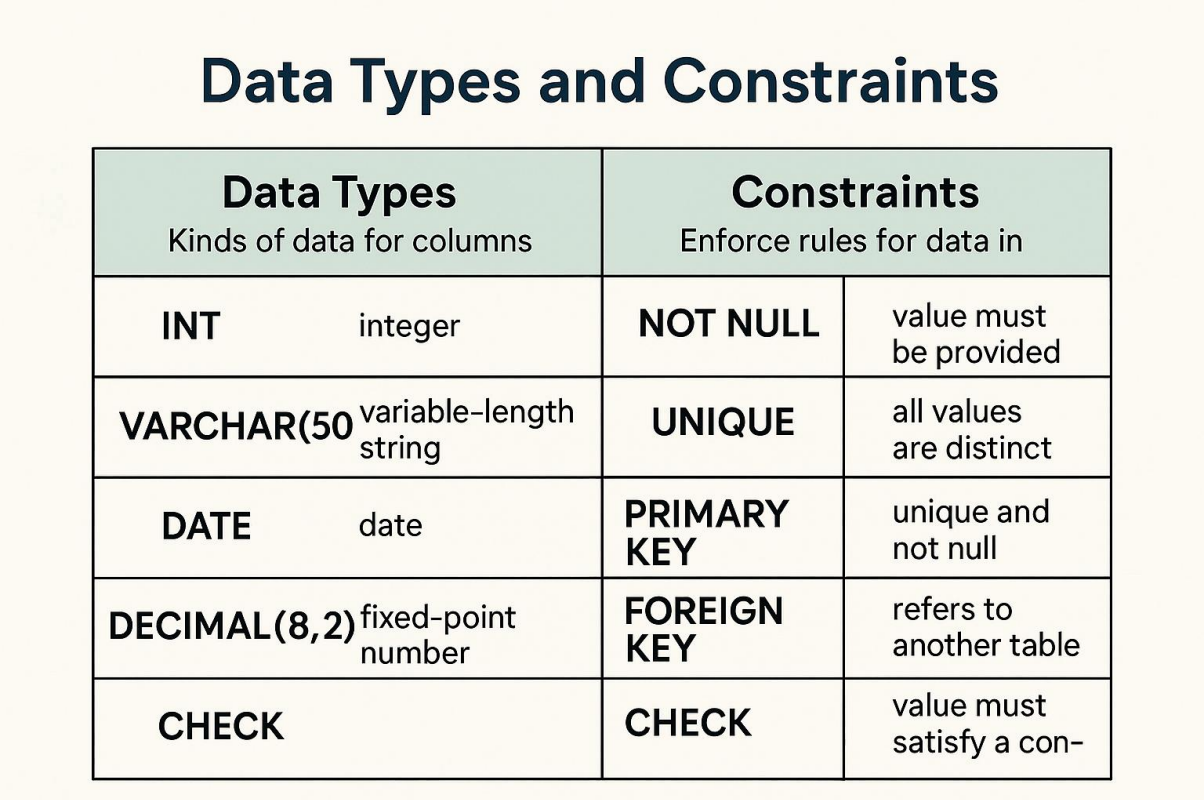
```
CREATE TABLE table_name (
   column1 datatype constraint,
   column2 datatype constraint,
   ...
);
```

- **table_name** → Name of the table.
- **column** → Name of column.
- **datatype** → Type of data (INT, VARCHAR, DATE, etc.).
- **constraint** → 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 >= 5),
   email VARCHAR(100) UNIQUE
);
```

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



5. ALTER Command

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

ALTER Command in SQL

- The ALTER command in SQL is used to change the structure of an existing table without deleting or recreating it.

- It allows modifications such as adding, deleting, or changing columns, and adding or removing constraints.

Main Uses of ALTER Command

1. Adding a Column
    - A new column can be added to an existing table.
    - Example: Adding a phone column to the customers table.

2. Modifying a Column
    - Change the data type, size, or constraints of an existing column.
    - Example: Changing a column from VARCHAR(20) to VARCHAR(50).

3. Renaming a Column or Table
    - Change the name of a column or the table itself.
    - Example: Renaming emp_name to employee_name.

4. Dropping a Column
    - Remove an existing column from a table.

5. Adding or Dropping Constraints
    - Add new constraints like UNIQUE, PRIMARY KEY, FOREIGN KEY, or remove them if needed.

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

1. Adding a Column

- Used to insert a new column into an existing table.
- Syntax:

```
ALTER TABLE table_name
ADD column_name data_type(size);
```

- Example:

```
ALTER TABLE students ADD age INT;
```

2. Modifying a Column

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

```
ALTER TABLE table_name
MODIFY column_name new_data_type(size);
```

- Example:

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

3. Dropping a Column

- Used to remove a column permanently from the table.
- Syntax:

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

- Example:

```
ALTER TABLE students DROP COLUMN age;
```

6. DROP Command

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

DROP Command in SQL – Function

The DROP command is used to delete an entire database object (like a table, view, or database) permanently from the system.

- When a table is dropped:
    - Structure is removed (all columns, constraints, indexes).
    - All data inside the table is deleted permanently.
    - Cannot be recovered unless there is a backup.

Key Points

- DROP TABLE → Deletes a table completely.
- DROP DATABASE → Deletes the whole database.
- It is a DDL (Data Definition Language) command.
- Once executed, it cannot be rolled back (unless supported by the system with transactions).

Syntax

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE students;
```

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

1. Loss of Table Structure

- The table definition (columns, data types, constraints) is permanently removed.
- The table no longer exists in the database schema.

2. Permanent Data Deletion

- All rows (data) stored in the table are deleted permanently.
- Recovery is only possible through backups.

3. Removal of Constraints and Relationships

- All constraints (PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK) are deleted.
- Other tables referencing it via FOREIGN KEY may be affected (error or orphaned records).

4. Loss of Indexes and Triggers

- Any indexes or triggers associated with the table are also deleted automatically.

5. No Rollback

- Being a DDL (Data Definition Language) command, DROP cannot be rolled back once committed.

7. Data Manipulation Language (DML)

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

1. INSERT Command

- Purpose:
  Used to add new records (rows) into an existing table.
- Function:
  - Can insert values into all columns or specific columns.
  - Ensures data matches the data type and constraints of each column.
- Key Points:
  - If column list is not specified, values must be given in the same order as table definition.
  - Fails if constraints (like NOT NULL, UNIQUE) are violated.
- Example:

  INSERT INTO employees (emp_id, name, salary) VALUES (101, 'Rahul', 50000)

2. UPDATE Command

- Purpose:
  Used to modify existing records in a table.
- Function:
  - Can update one column or multiple columns at once.
  - Usually combined with WHERE clause to prevent updating all rows.
- Key Points:
  - Without WHERE, all rows will be updated.
  - Must follow column data type rules.
- Example:

  UPDATE employees SET salary = 55000 WHERE emp_id = 101;

3. DELETE Command

- Purpose:
  Used to remove records from a table based on a condition.
- Function:
  - Deletes only the data, not the table structure.
  - Can delete all rows if WHERE is not used.
- Key Points:
  - DELETE is slower than TRUNCATE because it logs each deleted row.
  - Often used with conditions to prevent accidental full deletion.
- Example:

```
DELETE FROM employees WHERE emp_id = 101;
```

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

- **Importance of WHERE Clause in UPDATE and DELETE**

The WHERE clause is essential in UPDATE and DELETE commands because it controls which rows are affected**.**

- With WHERE → Only rows that meet the condition are updated or deleted.
- Without WHERE → All rows in the table will be updated or deleted.

It is important because it:

1. Prevents accidental changes to all data.
2. Targets only specific records.
3. Protects important data from loss**.**

- **Example:**

```
UPDATE students SET marks = 90 WHERE id = 5;
        -- Only student with ID 5 is updated

DELETE FROM students WHERE id = 5;
        -- Only student with ID 5 is deleted
```

8. Data Query Language (DQL)

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

- The SELECT statement is the most important SQL command.
- It is used to retrieve data from one or more tables in a database.
- The data returned is stored in a result set (like a virtual table).

Syntax

```
SELECT column1, column2, ...
FROM table_name
[WHERE condition]
[ORDER BY column]
[GROUP BY column]
[LIMIT number];
```

Example Table

Table: employees

id	name	department	salary
1	Riya	HR	35000
2	Aarav	IT	50000
3	Priya	IT	45000

Example Queries

- Select all data

```
SELECT * FROM employees;
```

- Select specific columns

  SELECT name, salary FROM employees;

- Select with condition

  SELECT name FROM employees WHERE department = 'IT';

- Select sorted results

  SELECT name, salary FROM employees ORDER BY salary DESC;

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

1. WHERE Clause

Purpose:

- Filters rows from a table based on a condition.
- Only the rows that satisfy the condition are shown in the result.

Syntax:

```
SELECT column1, column2

FROM table_name

WHERE condition;
```

Example:

If we have a table student:

id	name	marks
1	Riya	85
2	Aarav	70
3	Priya	92

Query to get students with marks above 80:

SELECT name, marks

FROM students

WHERE marks > 80;

Output:

name	marks
Riya	85
Priya	92

2. ORDER BY Clause

Purpose:

- Sorts the results of a query in ascending (ASC) or descending (DESC) order.
- Default is ascending.

Syntax:

```
SELECT column1, column2

FROM table_name

ORDER BY column1 ASC|DESC;
```

Example:

Sorting students by marks (highest first):

```
SELECT name, marks FROM students ORDER BY marks DESC;
```

Output:

name	marks
Priya	92
Riya	85
Aarav	70

9. Data Control Language (DCL)

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

1. Purpose of GRANT in SQL

- GRANT is used to give permissions to a user or role on a database object (like a table, view, or database).
- These permissions allow the user to perform actions like SELECT, INSERT, UPDATE, DELETE etc.

Syntax:

```
GRANT privilege_name
ON object_name
TO user_name;
```

Example:
Give SELECT and INSERT permission on students table to user Aarav:

```
GRANT SELECT, INSERT
ON students
TO Aarav;
```

2. Purpose of REVOKE in SQL

- REVOKE is used to remove permissions that were previously given to a user or role.
- It takes back the rights given by GRANT.

Syntax:

```
REVOKE privilege_name
ON object_name
FROM user_name;
```

Example:
Remove INSERT permission from user Aarav on students table:

REVOKE INSERT

ON students

FROM Aarav;

2. How do you manage privileges using these commands?

What are Privileges?

- Privileges are permissions that define what a user can or cannot do in a database.
- Two main types:
    - **System Privileges** → Affect the whole database (e.g., create tables, create users).
    - **Object Privileges** → Affect specific objects (e.g., SELECT, INSERT on a table).

Managing Privileges with GRANT

- The GRANT command assigns privileges to users or roles.
- You can:
    - Grant single privileges (e.g., SELECT)
    - Grant multiple privileges (e.g., SELECT, INSERT, UPDATE)
    - Grant ALL PRIVILEGES (everything)

Important Points about GRANT:

- You can grant privileges to multiple users at once.
- You can grant privileges with GRANT OPTION → allows that user to give the same privilege to others.

Example:

```
GRANT SELECT, INSERT
ON students
TO User1, User2
WITH GRANT OPTION;
```

Managing Privileges with REVOKE

- The REVOKE command **removes privileges** from users or roles.
- You specify exactly which permissions to remove.
- If WITH GRANT OPTION was used earlier, REVOKE also removes the ability to pass privileges to others.

Example:

```
REVOKE INSERT
ON students
FROM User1;
```

10. Transaction Control Language (TCL)

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

1. Purpose of COMMIT

- COMMIT is used to save all changes permanently in the database.
- After COMMIT, changes cannot be undone.

Key Points:

- Ends the current transaction.
- Makes all changes (INSERT, UPDATE, DELETE) permanent.
- Automatically releases locks on the database.

Example:

```
UPDATE students
SET marks = marks + 5
WHERE class = '10';
COMMIT;
```

2. Purpose of ROLLBACK

- ROLLBACK is used to undo all changes made in the current transaction before COMMIT.
- Returns the database to its previous state.

Key Points:

- Cancels all changes in the current transaction.
- Works only if COMMIT has not been executed yet.

Example:

```
UPDATE students
SET marks = marks + 5
WHERE class = '10';
ROLLBACK;
```

2. Explain how transactions are managed in SQL databases.

What is a Transaction?

- A transaction is a sequence of one or more SQL operations (INSERT, UPDATE, DELETE) that are executed as a single unit of work.
- A transaction must be completed fully or not at all (all-or-nothing principle).

how transactions are managed

1. Start Transaction
    - Happens automatically when you run SQL statements, or explicitly using

      ```
      START TRANSACTION;
      ```
2. Execute SQL Statements
    - Perform operations like INSERT, UPDATE, DELETE.
    - Changes are temporary until committed.
3. Validation
    - If everything works without error:

      ```
      COMMIT;  -- Saves changes permanently
      ```
    - If there's an error or you change your mind:

      ```
      ROLLBACK;  -- Cancels changes
      ```
4. End Transaction
    - After COMMIT or ROLLBACK, the transaction ends and locks are released.

11. SQL Joins

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

JOIN in SQL

- A JOIN is used in SQL to combine data from two or more tables based on a related column, usually a key (like teacher_id in teachers and students tables).
- The purpose of JOIN is to retrieve meaningful combined data without storing duplicate information in a single table.

Types of JOINs

1. INNER JOIN

- Returns only the rows that have matching values in both tables.
- If there is no match between the two tables, that row will not appear in the result.
- Example in words: Only teachers who have students will be listed, and only students who have assigned teachers will be shown.

2. LEFT JOIN (LEFT OUTER JOIN)

- Returns all rows from the left table and the matching rows from the right table.
- If there is no match in the right table, NULL values are shown for the missing columns.
- Example in words: All teachers are listed, even those who have no students (their student details will be blank).

3. RIGHT JOIN (RIGHT OUTER JOIN)

- Returns all rows from the right table and the matching rows from the left table.
- If there is no match in the left table, NULL values are shown for the missing columns.
- Example in words: All students are listed, even those who do not have any assigned teacher (their teacher details will be blank).

4. FULL OUTER JOIN

- Returns all rows from both tables.
- If there is a match, data is combined; if there is no match, NULL values fill the missing side.
- Example in words: All teachers and all students are listed, whether or not they are linked to each other.

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

How Joins Are Used to Combine Data from Multiple Tables

- In relational databases, data is usually stored in separate tables to avoid duplication and maintain normalization.
- JOINs are used to bring related data together into a single combined result.
- A JOIN works by matching rows from two or more tables based on a common column (usually a primary key in one table and a foreign key in another).

Practical Example

Teachers Table

teacher_id	teacher_name
1	Ramesh
2	Priya
3	Amit

Students Table

student_id	student_name	teacher_id
101	Rahul	1
102	Neha	2
103	Arjun	1
104	Meera	3

JOIN Result

teacher_name	student_name
Ramesh	Rahul
Ramesh	Arjun
Priya	Neha
Amit	Meera

12. SQL Group By

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

- The GROUP BY clause is used to arrange rows into groups based on the values of one or more columns.

- It is generally used with aggregate functions like COUNT(), SUM(), AVG(), MAX(), MIN() to perform calculations for each group separately.

Aggregate Functions

- Examples:
    - COUNT() → Counts number of rows
    - SUM() → Adds numeric values
    - AVG() → Finds average value
    - MAX() → Finds maximum value
    - MIN() → Finds minimum value

Imagine a students table:

Class	Student	Marks
10A	Rahul	80
10A	Neha	90
10B	Arjun	85
10B	Priya	95

If we use GROUP BY Class with AVG(Marks), the result will be:

Class	Avg_Marks
10A	85
10B	90

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

GROUP BY (Definition & Purpose)

- Purpose: Groups rows that have the same values in specified columns.
- Use: Often used with aggregate functions (SUM(), COUNT(), AVG(), etc.) to perform calculations for each group.
- Effect: Reduces multiple rows into summary rows for each group.
- Example in words: “Show the total sales for each product category.”

ORDER BY (Definition & Purpose)

- Purpose: Arranges the rows of the result set in ascending (ASC) or descending (DESC) order based on one or more columns.
- Use: Does not group data, only changes the display order.
- Effect: The number of rows remains the same; only the sequence changes.
- Example in words: “Show all products sorted by price from high to low.”

13. SQL Stored Procedure

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

A stored procedure is a precompiled set of SQL statements (queries, conditions, loops, etc.) stored in the database.

- It can perform complex tasks like inserting, updating, deleting, or selecting data.
- It is saved in the database and can be executed multiple times by calling its name.

Example of creating a stored procedure:

```
DELIMITER //

CREATE PROCEDURE GetAllEmployees()

BEGIN

   SELECT * FROM employees;

END //

DELIMITER ;
```

How It Differs from a Standard SQL Query

Aspect	Stored Procedure	Standard SQL Query
Definition	A set of SQL statements stored in the database, executed as a program.	A single SQL command (like SELECT, INSERT, etc.) executed immediately.
Reusability	Can be reused multiple times without rewriting the logic.	Needs to be written/executed each time.
Performance	Precompiled and optimized by the database → runs faster.	Compiled and executed each time → slower for repeated use.
Complexity	Can include loops, conditions, variables, error handling.	Limited to single SQL commands without procedural logic.
Security	Access can be controlled via permissions; sensitive logic hidden from users.	Anyone with query access can run the SQL directly.

2. Explain the advantages of using stored procedures.

Advantages of Stored Procedures (Simple Points)

1. Fast – Runs faster because it is precompiled.
2. Reusable – Write once, use many times.
3. Secure – Can hide tables and allow only procedure execution.
4. Easy to Maintain – Change logic in one place; all programs use updated version.
5. Less Network Traffic – Only call the procedure instead of sending many queries.
6. Handles Complex Tasks – Can use loops, conditions, and error handling.

14. SQL View

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

view

- A view is a virtual table based on the result of a SQL query.
- It does not store data physically; it just shows data from one or more tables.
- It's like a saved query that you can use as if it were a table.

Difference Between View and Table

Aspect	View	Table
Storage	Does not store data (only stores query).	Stores actual data.
Creation	Created using CREATE VIEW.	Created using CREATE TABLE.
Data Source	Data comes from one or more tables.	Data is physically inside the table.
Updatability	Some views are read-only.	Tables can always be updated (unless restricted).
Purpose	Used to simplify queries, provide security, or show specific data.	Used to store and manage data.

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

Advantages of Views

1. **Simplifies Queries**
    - Complex joins or filters can be saved in a view.
    - You just SELECT * FROM view_name; instead of writing the whole query again.

2. **Improves Security**
    - You can show only selected columns/rows from a table.
    - Sensitive data (like salaries, passwords) can be hidden.

3. **Reusability**
    - Once created, a view can be used many times without rewriting the logic.

4. **Easy Maintenance**
    - If the query changes, update the view once — all programs using it get the updated result.

5. **Consistency**
    - Ensures everyone sees the same calculated or filtered data.

6. **Data Independence**
    - If table structure changes, you can adjust the view without affecting queries that use it.

15. SQL Triggers

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

What is a Trigger?

- A trigger is a special program in the database that runs automatically when a specific event (like INSERT, UPDATE, or DELETE) happens on a table.

- It is used for automatic actions like logging changes, validating data, or updating related tables.

Types of Triggers

1. Based on Time

- BEFORE Trigger → Runs before data changes. (Use: Check or fix data before saving)
- AFTER Trigger → Runs after data changes. (Use: Log changes, update other tables)

2. Based on Action

- INSERT Trigger → Runs when new data is added.
- UPDATE Trigger → Runs when data is changed.
- DELETE Trigger → Runs when data is removed.

When Are Triggers Used?

- Audit Logging – Track changes to important tables.
- Enforcing Business Rules – Example: Prevent deleting active customers.
- Automatic Updates – Example: Update total sales after a purchase.
- Data Validation – Example: Ensure phone numbers have correct format.
- Cascading Changes – Example: Delete related records automatically.

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

Trigger Type	When It Runs	What It Can Do	Example
INSERT Trigger	Runs after or before a new row is added.	Check or modify new data, log the insertion.	When a new employee is added, log it in an audit table.
UPDATE Trigger	Runs after or before a row is changed.	Track old and new values, validate updates.	When salary changes, store old salary in history table.
DELETE Trigger	Runs after or before a row is deleted.	Stop deletion, save deleted data somewhere.	When a customer is deleted, store details in backup table.

16. Introduction to PL/SQL

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

- SQL is used to talk to the database — you can insert, update, delete, or select data.
- PL/SQL is like a smarter version of SQL — it lets you write programs using SQL plus extra features.
- How PL/SQL Extends SQL

  SQL alone can:

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

  But SQL cannot:

  - Store variables or constants
  - Use loops and conditional logic
  - Handle exceptions gracefully

PL/SQL adds these capabilities:

- Variables & Constants: Store and manipulate intermediate values
- Control Structures: IF...ELSE, LOOP, WHILE, FOR
- Procedures & Functions: Reusable code blocks
- Exception Handling: Graceful error management (EXCEPTION...WHEN)
- Cursors: Handle query results row by row

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

Benefits of Using PL/SQL

1. Better Performance
   - PL/SQL sends an entire block of code to the database at once instead of sending individual SQL statements one by one.
   - This reduces network traffic and makes programs run faster.

2. Code Reusability
    - You can create procedures, functions, and packages that can be reused in different programs.
    - This saves time and effort.
3. Error Handling
    - PL/SQL has a built-in EXCEPTION section to handle errors gracefully.
    - This prevents the program from crashing and helps in debugging.
4. Security
    - You can store sensitive logic inside stored procedures or packages.
    - Users can be given permission to run the procedure without seeing the internal SQL code.
5. Block Structure
    - PL/SQL uses blocks (Declaration, Execution, Exception) which make the code organized, readable, and easy to maintain.
6. Integration with SQL
    - PL/SQL works tightly with SQL.
    - You can write SQL statements directly inside PL/SQL blocks along with variables, loops, and logic.
7. Portability
    - PL/SQL code can run on any Oracle database without changes.

17. PL/SQL Control Structures

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

Control Structures in PL/SQL

- Control structures are instructions that control the flow of execution in a PL/SQL program.
- They decide which statements run and how many times they run based on conditions or loops.

Types of Control Structures

1. Conditional Control → IF...THEN, IF...THEN...ELSE, IF...ELSIF...ELSE
2. Iterative Control (Loops) → LOOP, WHILE LOOP, FOR LOOP
3. Sequential Control → GOTO

1. IF–THEN Control Structure

- Used to execute statements only if a condition is true.
- Syntax:

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

- Example:

```
DECLARE
  v_salary NUMBER := 50000;
BEGIN
  IF v_salary > 40000 THEN
    DBMS_OUTPUT.PUT_LINE('High salary');
  END IF;
END;
```

2.LOOP Control Structure

- Used to repeat statements multiple times.
- A basic loop runs until explicitly exited using EXIT or EXIT WHEN.
- Syntax:

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

- Example:

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

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

1.Add Decision-Making (Conditional Logic)

- Using IF...THEN...ELSE, you can execute different SQL statements based on conditions.
- Example: Apply different discount rates for different customer types.

2.Automate Repetitive Tasks (Loops)

- Using LOOP, WHILE, or FOR, you can repeat actions automatically.
- Example: Update salaries for each employee in a department without writing many separate queries.

3.Combine Multiple SQL Operations in One Block

- A single PL/SQL block can include several SQL statements, decisions, and loops together.
- Example: Insert data, check conditions, update other tables, and handle errors — all in one program.

4.Handle Errors and Exceptions

- Control structures allow error handling so the program can continue or give a clear message when something goes wrong.
- Example: If inserting data fails due to duplicate entry, handle it gracefully.

18. SQL Cursors Theory

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

What is a Cursor in PL/SQL?

- A cursor is a pointer that stores the result of an SQL query executed in PL/SQL.
- It is used to retrieve rows one by one from the result set.
- Think of it like a pointer that moves through the rows of a table.

Types of Cursors

PL/SQL has two main types of cursors:

(1) Implicit cursor
(2) Explicit cursor

(1) Implicit Cursor

- Created automatically by Oracle when any SQL statement (SELECT INTO, INSERT, UPDATE, DELETE) is executed.
- Used when:
    - Query returns only one row.
    - You do not need to fetch rows one by one.
- Managed by Oracle: No need to declare, open, fetch, or close.

(2) Explicit Cursor

- Created manually by the programmer for queries that return multiple rows.
- Requires four steps:
    1. Declare the cursor.
    2. Open the cursor (execute query and store result).
    3. Fetch rows one by one.
    4. Close the cursor after use.

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

When to Use an Explicit Cursor Over an Implicit Cursor

You use an explicit cursor when:

1. Query Returns Multiple Rows
    - Implicit cursors handle only one row at a time.
    - If a query returns many rows, you need an explicit cursor to fetch them row by row.

2. Need Row-by-Row Processing
    - When each row needs to be processed individually (e.g., applying different logic for each record).

3. Need More Control Over Data Retrieval
    - Explicit cursors let you open, fetch, and close the cursor at your desired time.
    - You can pause between fetches, skip rows, or exit loops based on conditions.

4. Complex Business Logic
    - When business rules require checking conditions for each row or performing multiple actions on the result set.

19. Rollback and Commit Savepoint

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

SAVEPOINT in Transaction Management

- A SAVEPOINT is a marker inside a transaction.
- It lets you roll back part of a transaction to a specific point without undoing the entire transaction.
- You can set multiple savepoints in a single transaction.

Interaction with COMMIT and ROLLBACK

COMMIT

- Commits all changes made in the transaction since the last COMMIT or ROLLBACK.
- After COMMIT:
    - All savepoints are deleted.
    - Changes become permanent in the database.

ROLLBACK

- ROLLBACK TO SAVEPOINT → Undoes changes made after that savepoint but keeps changes before it.
- ROLLBACK (without savepoint) → Undoes the entire transaction to its initial state.

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

Situations Where Savepoints Are Useful

1. Complex Transactions
    - When a transaction has multiple steps (insert, update, delete) and you want the ability to undo only certain steps if needed.

2. Error Handling
    - If an error occurs at a certain stage, you can rollback to a savepoint instead of canceling the whole transaction.

3. Conditional Logic in Procedures
    - Inside procedures or PL/SQL blocks, savepoints allow rolling back part of the work depending on conditions.

4. Testing and Debugging
    - While testing a transaction, savepoints let you undo specific changes without restarting everything.

5. Long Transactions
    - In long processes (e.g., batch processing), savepoints help to commit or rollback in stages for better safety.