Module 4 – Introduction to DBMS

Introduction to SQL

Theory Questions:

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

->It is a structure query language.

->sql is a programming language. Used to interact with relational databases.
->it is used to perform CRUD operations.

- Create
- Read
- Update
- Delete

Why is SQL Important?

- Easily Fetch Data – Helps retrieve specific information from large databases.
- Modify Data – Allows adding, updating, or deleting records.
- Keeps Data Organized – Stores data in a structured way using tables.
- Ensures Security – Controls who can access or change the data.
- Works Everywhere – Used in many database systems like MySQL, SQL Server, and Oracle.

2. Explain the difference between DBMS and RDBMS.

DBMS	RDBMS
Data stored is in the file format	Data stored is in table format
Individual access of data elements	Multiple data elements are accessible together

No connection between data	Data in the form of a table are linked together
No support for distributed database	Support distributed database
Data stored is a small quantity	Data is stored in a large amount
DBMS supports a single user	RDBMS supports multiple users
The software and hardware requirements are low	The software and hardware requirements are higher
Example: XML, Microsoft Access.	Example: Oracle, SQL Server.

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

- SQL plays a crucial role in handling relational databases, which store data in tables with rows and columns. Here's how SQL helps:

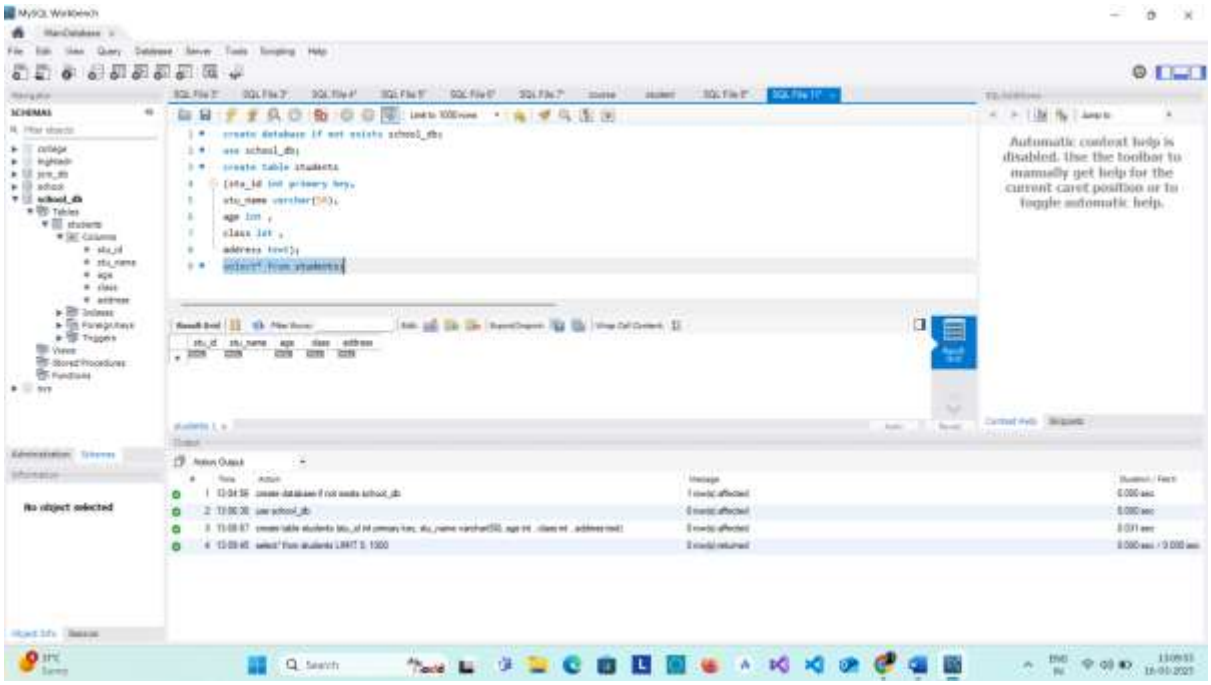
- Data Retrieval – Fetches specific data using queries (e.g., SELECT statement).
- Data Modification – Adds, updates, or deletes records (INSERT, UPDATE, DELETE).
- Data Organization – Defines database structure using tables and relationships (CREATE TABLE).
- Data Security – Controls user access and permissions (GRANT, REVOKE).
- Data Integrity – Ensures accuracy with constraints like PRIMARY KEY and FOREIGN KEY.
- Performance Optimization – Uses indexing and efficient queries to speed up data access.

4. What are the key features of SQL?
    - Key Features of SQL

- Data Querying – Retrieve data using SELECT statements.
- Data Manipulation – Insert, update, and delete records (INSERT, UPDATE, DELETE).
- Data Definition – Create and modify database structures (CREATE TABLE, ALTER TABLE).
- Data Control – Manage user permissions and access (GRANT, REVOKE).
- Data Integrity – Maintain accuracy with constraints like PRIMARY KEY, FOREIGN KEY, UNIQUE.
- Transaction Control – Ensure data consistency with COMMIT, ROLLBACK, SAVEPOINT.
- Scalability & Performance – Supports indexing and optimized queries for handling large datasets.
- Standardized Language – Works across different database systems like MySQL, PostgreSQL, SQL Server, and Oracle.

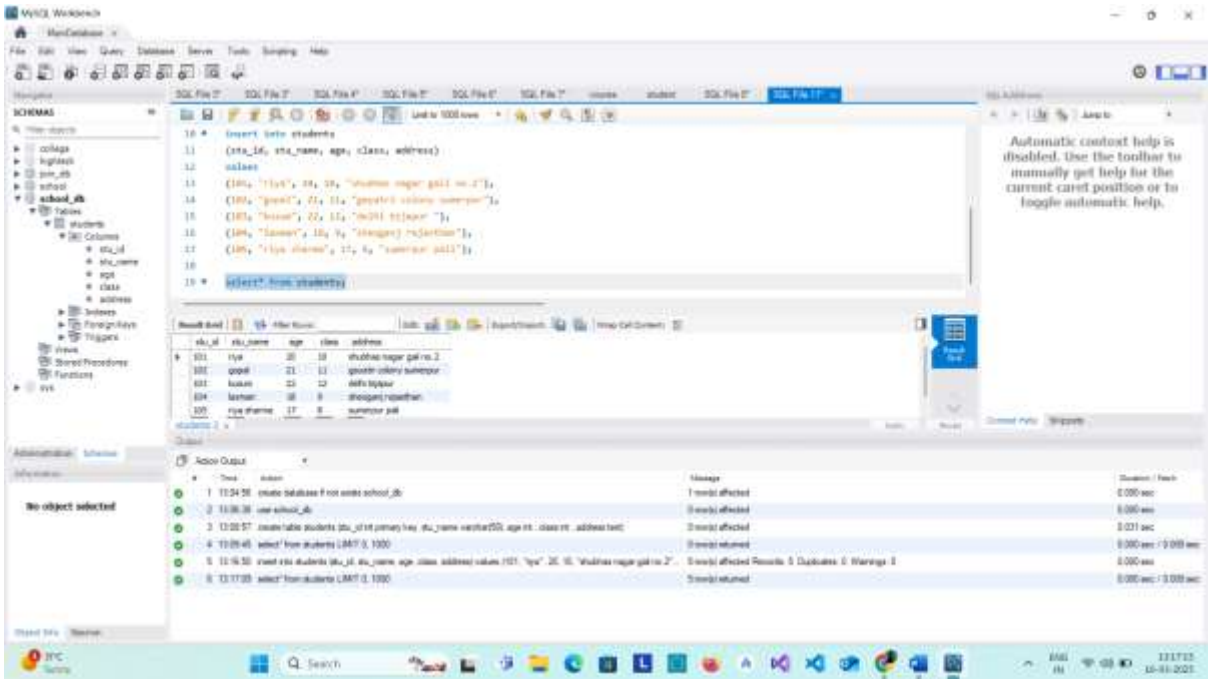
LAB EXERCISES:

• Lab 1: Create a new database named school_db and a table called students with the following columns: student_id, student_name, age, class, and address.



• Lab 2: Insert five records into the students table and retrieve all records using the SELECT

statement.



2. SQL Syntax

Theory Questions:

1. What are the basic components of SQL syntax?

1. SQL Statements

- Data Query Language (DQL) – Used to retrieve data
  1. `SELECT` – Retrieves data from tables
- Data Manipulation Language (DML) – Modifies existing data
  1. `INSERT` – Adds new records
  2. `UPDATE` – Modifies existing records
  3. `DELETE` – Removes records
- Data Definition Language (DDL) – Defines the database structure
  1. `CREATE TABLE` – Creates a new table
  2. `ALTER TABLE` – Modifies an existing table
  3. `DROP TABLE` – Deletes a table
- Data Control Language (DCL) – Manages user permissions
  1. `GRANT` – Gives access rights
  2. `REVOKE` – Removes access rights
- Transaction Control Language (TCL) – Manages transactions
  1. `COMMIT` – Saves changes
  2. `ROLLBACK` – Undoes changes
  3. `SAVEPOINT` – Creates checkpoints in transactions

2. SQL Clauses

- WHERE – Filters records based on a condition
- SELECT * FROM employees WHERE age > 30;
  - ORDER BY – Sorts results in ascending (ASC) or descending (DESC) order
  - SELECT * FROM employees ORDER BY salary DESC;

- GROUP BY – Groups rows based on column values
  - SELECT department, COUNT(*) FROM employees GROUP BY department;
- HAVING – Filters grouped records
  - SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000;

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

- Select*from students;

3.Explain the role of clauses in SQL statements.

Role of Clauses in SQL Statements

In SQL, clauses are used to refine queries and define specific conditions for filtering, grouping, and sorting data. They help retrieve, modify, and organize data efficiently. Clauses are typically used with SQL statements like SELECT, UPDATE, and DELETE.

Key SQL Clauses and Their Roles

1. WHERE Clause (Filters Data)

- Used to filter records based on a condition.
- Works with SELECT, UPDATE, and DELETE statements.

Example: Fetch employees older than 30 years.

SELECT * FROM employees WHERE age > 30;

Example: Delete employees in the "HR" department.

DELETE FROM employees WHERE department = 'HR';

2. ORDER BY Clause (Sorts Results)

- Used to sort query results in ascending (ASC) or descending (DESC) order.
- By default, it sorts in ascending order.

Example: Get employees sorted by salary (highest to lowest).

SELECT * FROM employees ORDER BY salary DESC;

Example: Sort employees alphabetically by name.

SELECT * FROM employees ORDER BY name ASC;

3. GROUP BY Clause (Groups Data)

- Groups rows with the same values in a specified column.
- Often used with aggregate functions like COUNT(), SUM(), AVG(), etc.

Example: Count employees in each department.

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

Example: Find the average salary in each department.

SELECT department, AVG(salary) FROM employees GROUP BY department;

4. HAVING Clause (Filters Grouped Data)

- Used with GROUP BY to filter grouped records.
- Similar to WHERE but works on aggregate functions.

Example: Get departments with more than 5 employees.

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

Example: Show departments where the average salary is above 50,000.

SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000;

5. LIMIT Clause (Restricts Number of Rows)

- Used to limit the number of rows returned by a query.
- Useful when dealing with large datasets.

Example: Get the top 5 highest-paid employees.

SELECT * FROM employees ORDER BY salary DESC LIMIT 5;

Example: Fetch only 3 employee records.

SELECT * FROM employees LIMIT 3;

6. JOIN Clause (Combines Data from Multiple Tables)

- Used to retrieve related data from two or more tables.
- Common types: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN.

Example: Fetch employee names along with their department names from two tables (employees and departments).

SELECT employees.name, departments.department_name

FROM employees

INNER JOIN departments ON employees.department_id = departments.id;

❖ SQL clauses play a vital role in refining and structuring queries. They help:

✓ Filter data (WHERE, HAVING)

✓ Sort data (ORDER BY)

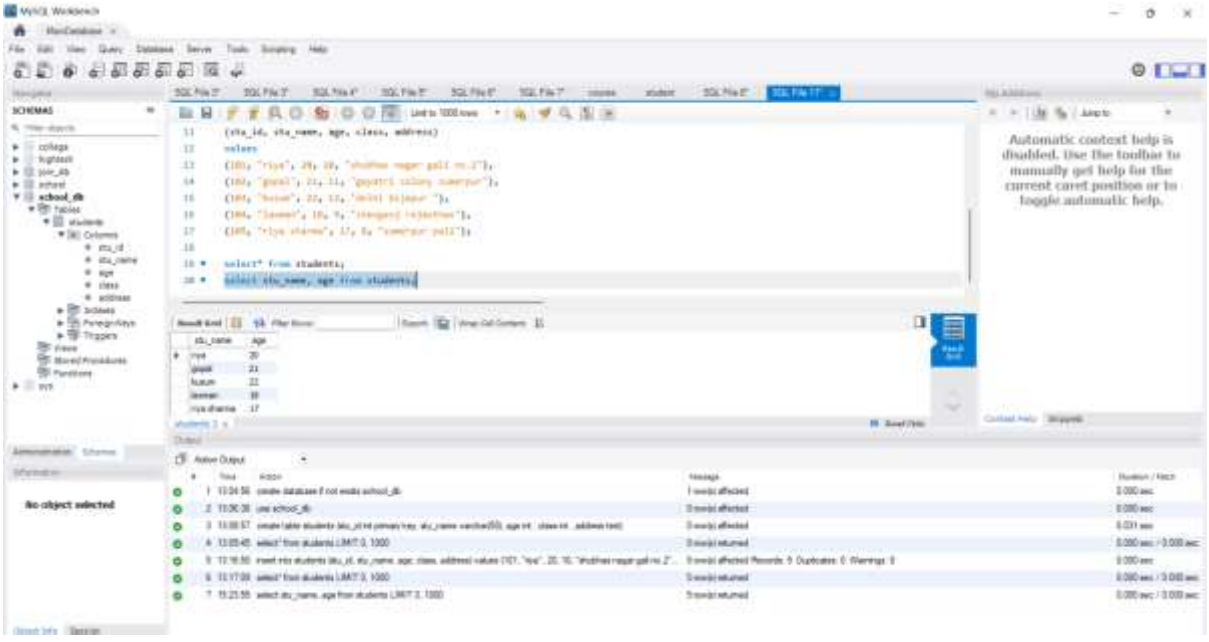
✓ Group data (GROUP BY)

✓ Limit results (LIMIT)

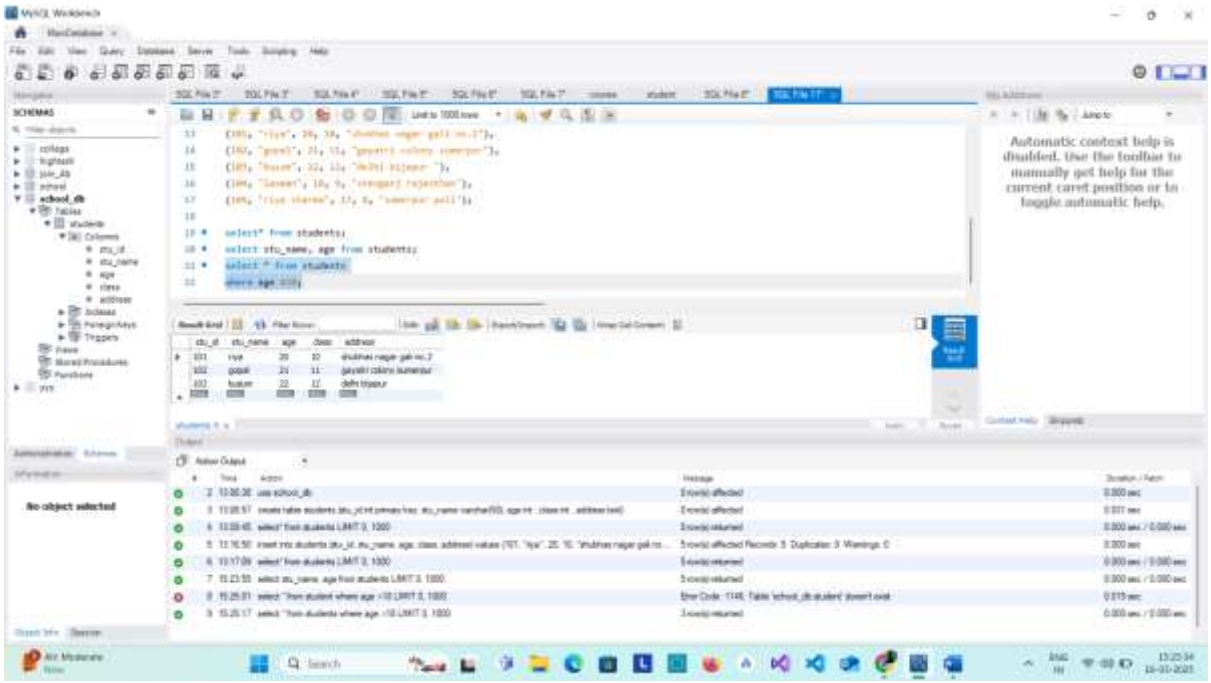
✓ Combine tables (JOIN)

LAB EXERCISES:

• Lab 1: Write SQL queries to retrieve specific columns (student_name and age) from the students table.



• Lab 2: Write SQL queries to retrieve all students whose age is greater



4. SQL Constraints

Theory Questions:

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

- Sql constraints are used to specify rules for data in a table.
- Constraints in SQL are rules applied to table columns to ensure data integrity, accuracy, and reliability. They restrict the type of data that can be inserted into a table, preventing invalid or inconsistent data.

Types of Constraints in SQL

1. PRIMARY KEY (Ensures Uniqueness & Non-null Values)

- Uniquely identifies each record in a table.
- Does not allow NULL and must be unique.

➢ Each table can have only one primary key

Eg. CREATE TABLE Employees (

emp_id INT PRIMARY KEY,

name VARCHAR(50),

department VARCHAR(50)

);

2. FOREIGN KEY (Maintains Relationships Between Tables)

- Establishes a relationship between two tables.
- Links a column in one table to the PRIMARY KEY of another table.
- Ensures referential integrity (data in one table must match data in another).

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

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

CREATE TABLE Departments (

dept_id INT PRIMARY KEY,

dept_name VARCHAR(50)

);

CREATE TABLE Employees (

```
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    dept_id INT,
    FOREIGN KEY (dept_id) REFERENCES Departments(dept_id)
);
```

3. UNIQUE (Ensures Values are Distinct in a Column)

- Ensures all values in a column are unique (but allows NULL).
- Unlike PRIMARY KEY, multiple UNIQUE constraints can exist in a table.

```
CREATE TABLE Users (
    user_id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE
);
```

4. NOT NULL (Prevents Empty Values in a Column)

- Ensures that a column cannot have NULL values.
- Guarantees that important fields always have a value.

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

5. CHECK (Enforces a Specific Condition on a Column)

- Ensures that values in a column meet a specified condition.

```
CREATE TABLE Employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT CHECK (age >= 18)
);
```

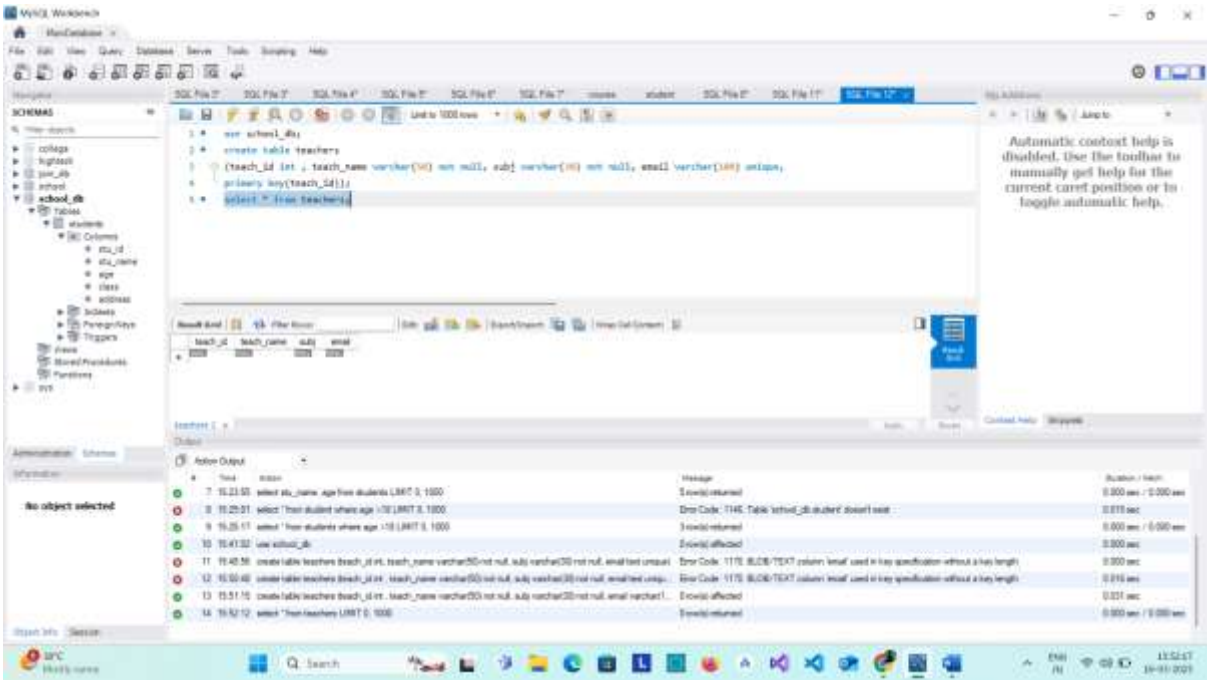
6. DEFAULT (Sets a Default Value for a Column if No Value is Provided)

- Assigns a default value when no value is inserted.

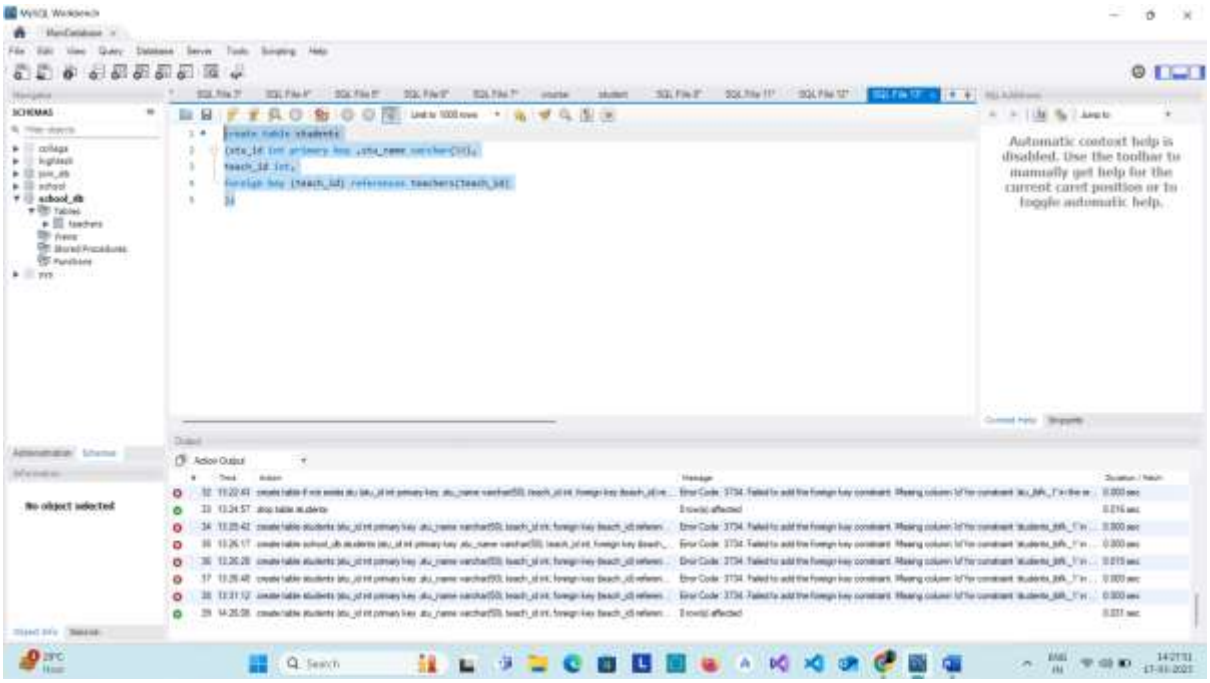
```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    order_date DATE DEFAULT CURRENT_DATE
);
```

LAB EXERCISES:

- Lab 1: Create a table teachers with the following columns: teacher_id (Primary Key), teacher_name (NOT NULL), subject (NOT NULL), and email (UNIQUE).



• Lab 2: Implement a FOREIGN KEY constraint to relate the teacher_id from the teachers table with the students table.



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

Theory Questions:

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

- DDL (Data Definition Language) is a subset of SQL that is used to define, modify, and manage the structure of database objects like tables, schemas, indexes, and views.
- DDL commands do not modify data but affect the database structure itself.
  DDL commands are auto-committed, meaning changes are permanent and cannot be rolled back.

2. Explain the CREATE command and its syntax.

CREATE (Creates a New Database Object)
Used to create databases, tables, indexes, views, etc.
Eg. CREATE TABLE employees (
  emp_id INT PRIMARY KEY,
  name VARCHAR(100),
  department VARCHAR(50),
  salary DECIMAL(10,2)
);

3. What is the purpose of specifying data types and constraints during table creation?
1. Purpose of Data Types
Data types define the kind of values a column can store, preventing incorrect data entry and optimizing storage.
Key Benefits of Data Types:

- Data Integrity – Ensures values match expected formats (e.g., age as an INT, not TEXT).
- Storage Efficiency – Saves space by allocating appropriate storage (e.g., VARCHAR(50) instead of TEXT).

Eg. CREATE TABLE students (
student_id INT,
name VARCHAR(100), -- Stores text (up to 100 characters)
birth_date DATE, -- Stores date values
gpa DECIMAL(3,2) -- Stores numbers like 3.75 (3 digits, 2 decimal places)
);

2. Purpose of Constraints

Constraints enforce rules on columns to maintain data consistency and accuracy.

Key Benefits of Constraints:

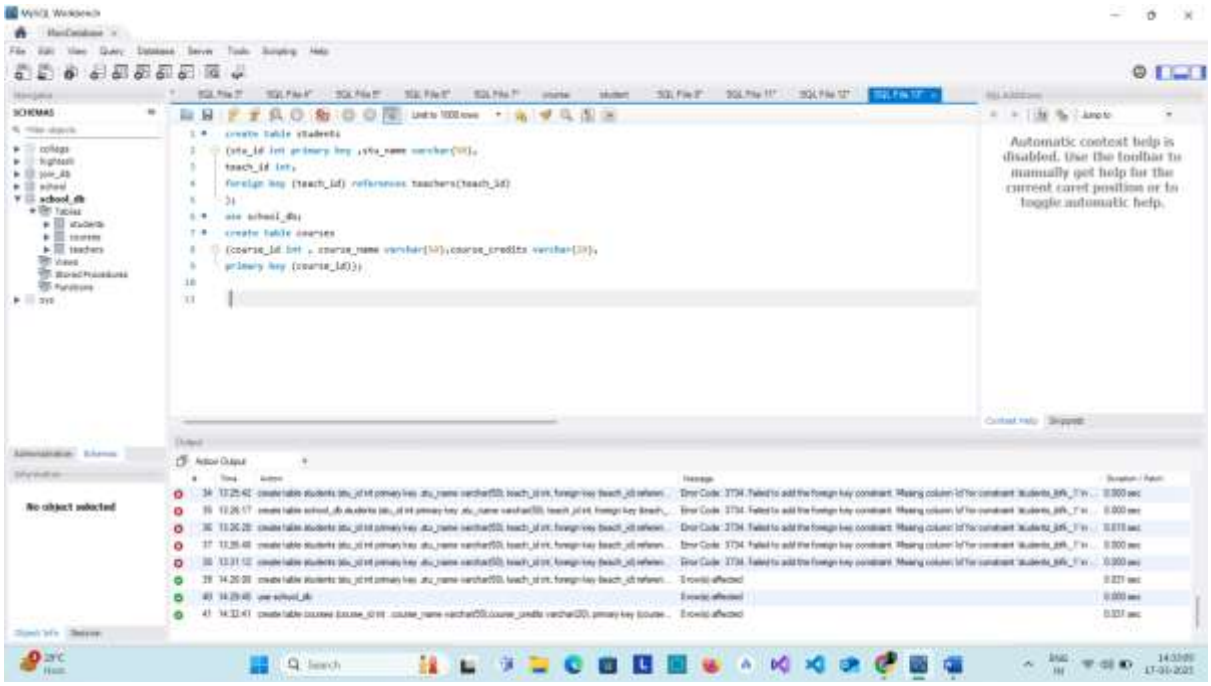
- Prevent Invalid Data Entries – Ensures required fields are not left empty (NOT NULL).
- Ensure Uniqueness – Prevents duplicate values (PRIMARY KEY, UNIQUE).
- Maintain Relationships – Links tables through foreign keys (FOREIGN KEY).
- Define Valid Ranges – Restricts values to specific conditions (CHECK).

Eg. CREATE TABLE students (
student_id INT PRIMARY KEY, -- Unique and required
name VARCHAR(100) NOT NULL, -- Name cannot be empty

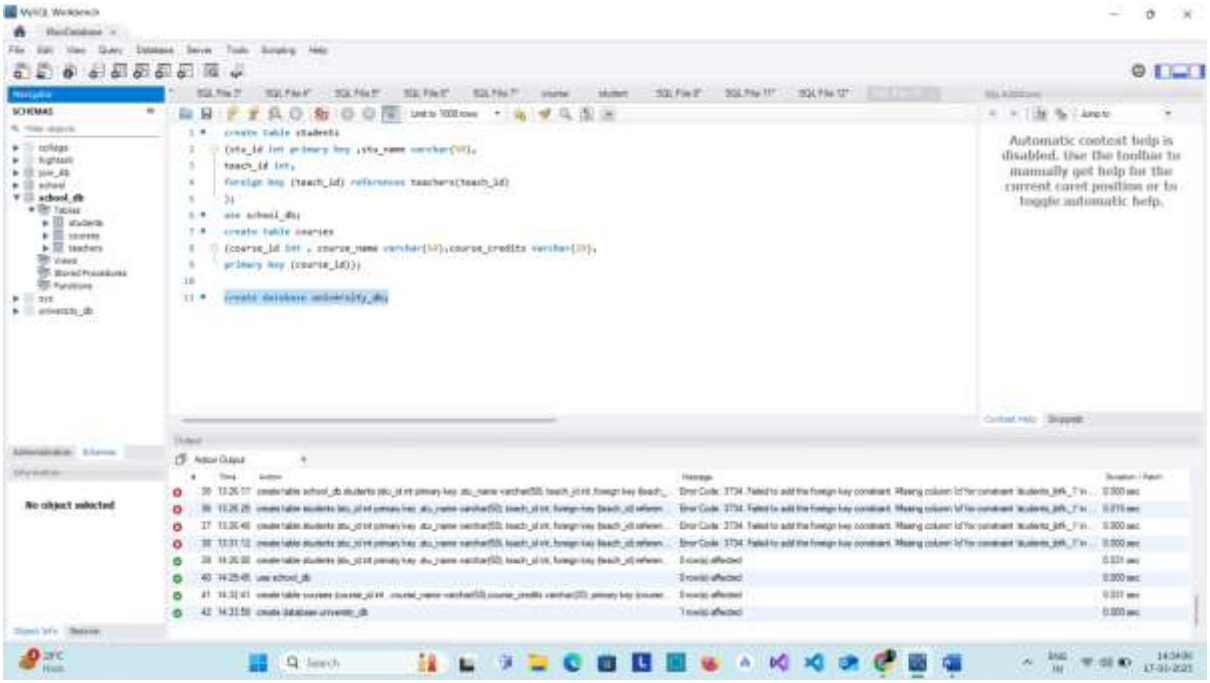
```
    age INT CHECK (age >= 18),      -- Students must be at
least 18
    email VARCHAR(100) UNIQUE,      -- No duplicate
emails allowed
    department_id INT,
    FOREIGN KEY (department_id) REFERENCES
departments(department_id)  -- Ensures valid
department reference
);
```

LAB EXERCISES:

• Lab 1: Create a table courses with columns: course_id, course_name, and course_credits. Set the course_id as the primary key.



• Lab 2: Use the CREATE command to create a database university_db



5. ALTER Command

Theory Questions:

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

   -> It is use to change the schema.

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

   - Add columns

     Alter table table_name

     Add column column_name datatype constraints;

   - Modify columns

     Alter table table_name

     Modify column_name datatype constraints;

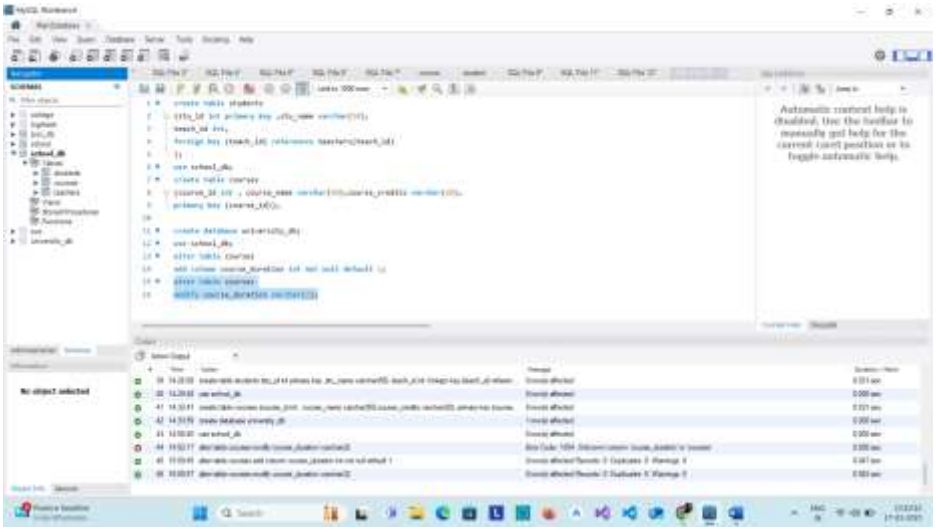
   - Drop columns

     Alter table table_name

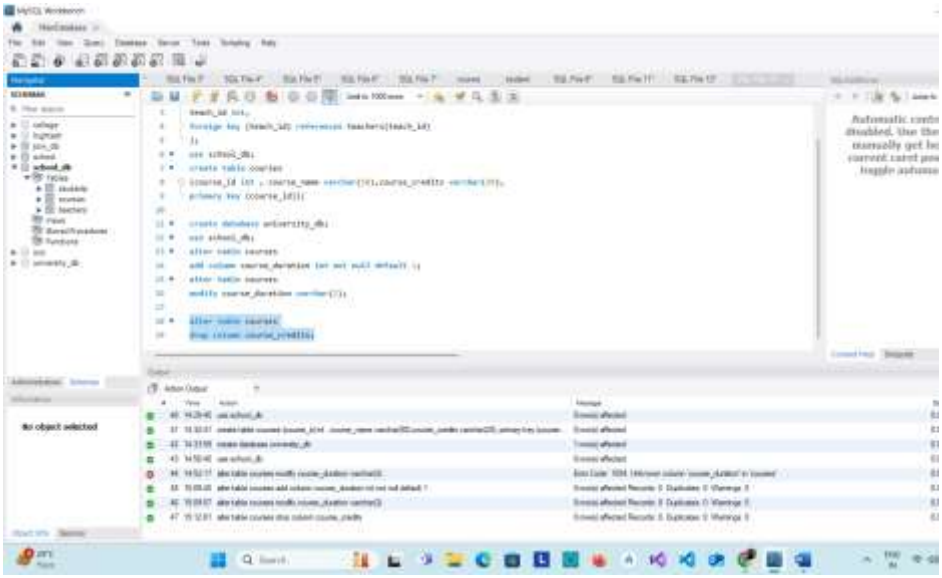
Drop column column_name;

LAB EXERCISES:

- Lab 1: Modify the courses table by adding a column course_duration using the ALTER command.



- Lab 2: Drop the course_credits column from the courses table.



6. DROP Command

Theory Questions:

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

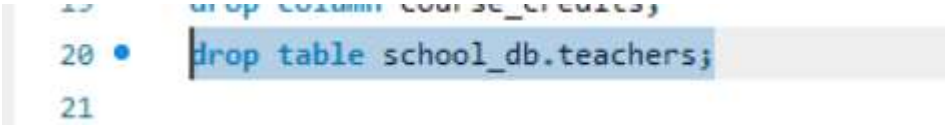
-> The DROP command in SQL is used to permanently delete database objects such as tables, databases, indexes, or views.

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

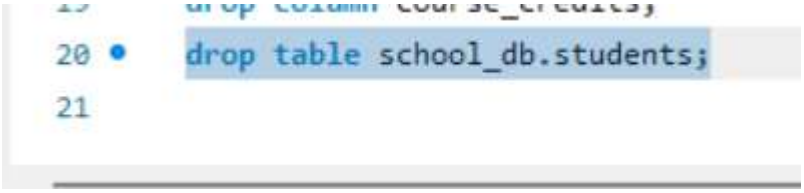
-> The DROP TABLE command permanently removes a table, including its structure and all stored data.

LAB EXERCISES:

- Lab 1: Drop the teachers table from the school_db database.



- Lab 2: Drop the students table from the school_db database and verify that the table has been removed.



7. Data Manipulation Language (DML)

Theory Questions:

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

->Insert : it is use to insert data in table.

->update: it is use to update existing row.

->delete: it is use to delete the existing rows.

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

-> The WHERE clause is crucial in UPDATE and DELETE operations because it specifies which records should be modified or removed. Without WHERE, the entire table could be affected, leading to data loss or unintended changes.

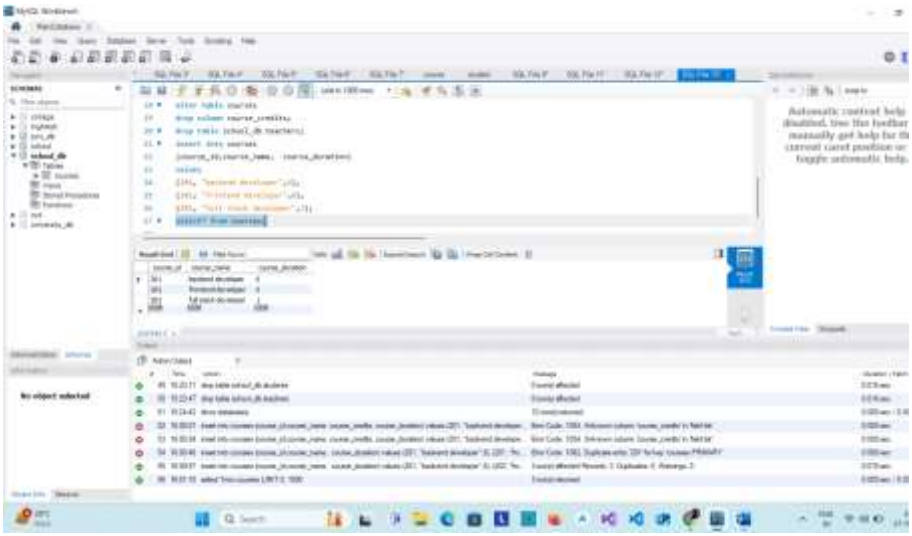
Prevents Updating or Deleting All Records

- If WHERE is not used, the operation applies to every row in the table.
- This can cause massive unintended changes or data loss.

Eg. UPDATE employees SET salary = 50000; -- Updates salary for ALL employees!
DELETE FROM employees; -- Deletes ALL records!

LAB EXERCISES:

• Lab 1: Insert three records into the courses table using the INSERT command.



• Lab 2: Update the course duration of a specific course using the UPDATE command.

```
update courses
set course_duration=2
where course_duration=6;
```

• Lab 3: Delete a course with a specific course_id from the courses table using the DELETE command.

```
delete from courses
where course_id=202;
select * from courses;
```

Result Grid | Filter Rows: | Edit:

course_id	course_name	course_duration
201	backend developer	6
203	full stack developer	1
NULL	NULL	NULL

8. Data Query Language (DQL)

Theory Questions:

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

   -> The SELECT statement is used in SQL to retrieve data from one or more tables in a database. It is the most commonly used SQL command and allows users to query and view data based on specific criteria.

   e.g SELECT * FROM employees;

   ->select all columns from data

   e.g. SELECT name, salary FROM employees;

   ->select specific column from table.

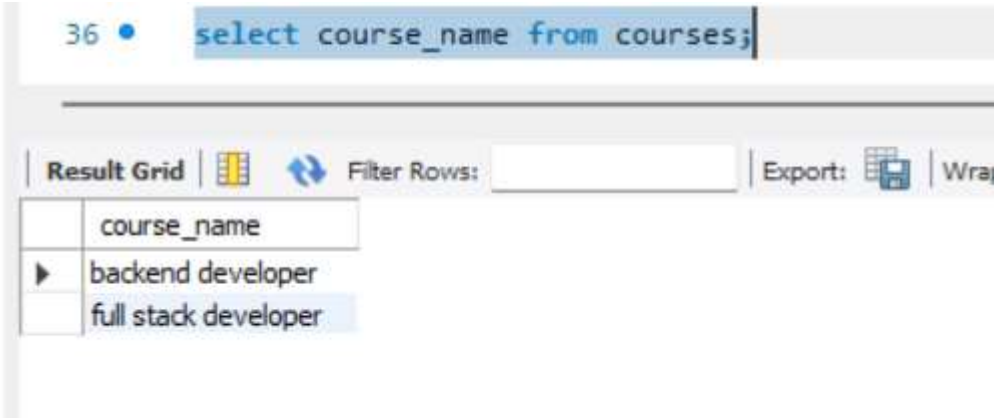
2. Explain the use of the ORDER BY and WHERE clauses in SQL queries.

->To define some conditions.

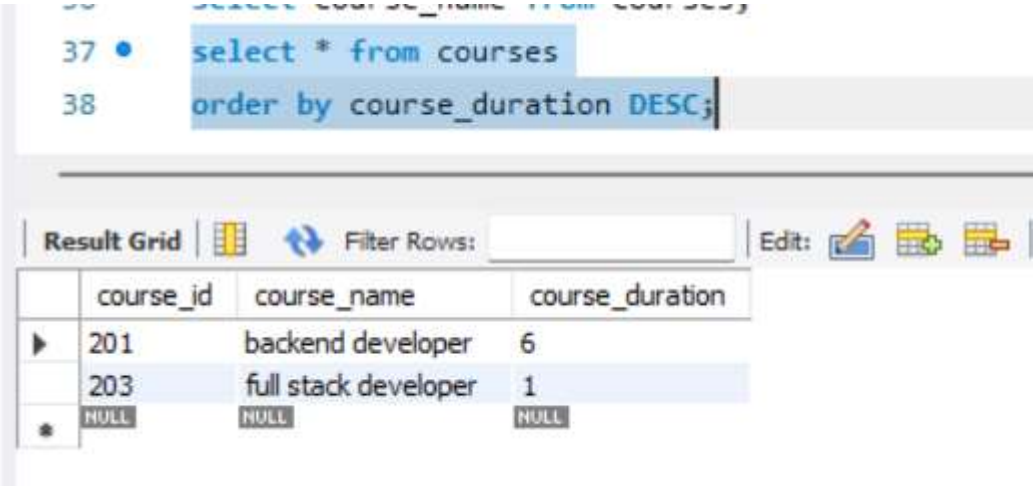
Eg. Select * from emp
Where marks>=90;

->To sort in ascending (ASC) or descending order (DESC).
Eg. Select* from student
Order by city ACS;
LAB EXERCISES:

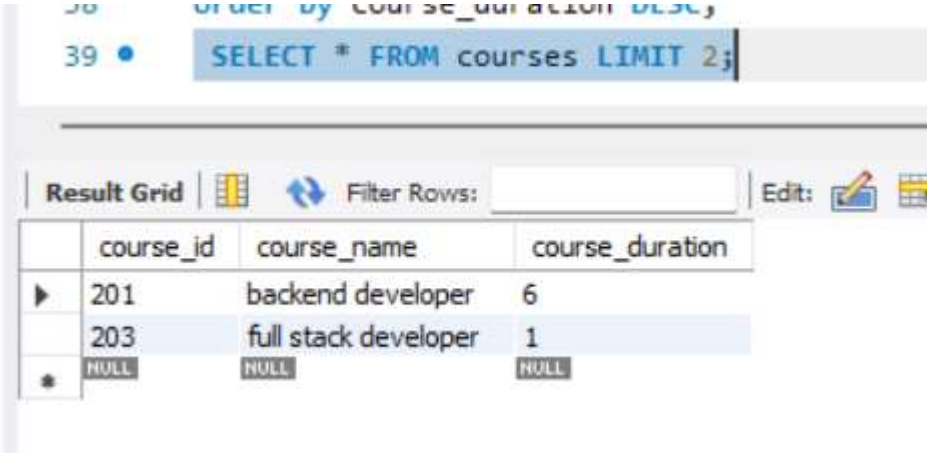
- Lab 1: Retrieve all courses from the courses table using the SELECT statement.



- Lab 2: Sort the courses based on course_duration in descending order using ORDER BY.



- Lab 3: Limit the results of the SELECT query to show only the top two courses using LIMIT



9. Data Control Language (DCL)

Theory Questions:

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

➔ Grant and Revoke are used to manage user privileges and control access to database objects like tables, view and procedures.

2. How do you manage privileges using these commands?

LAB EXERCISES:

- Lab 1: Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

Syntax: Grant privileges

ON Obejct_name

To user_name[with grant option];

Eg. Grant select privileges on the courses table to user1

Grant select

On courses

To user1;

- Lab 2: Revoke the INSERT permission from user1 and give it to user2.

Syntax: Revoke privileges

On object_name

From user_name;

Eg. Revoke Insert

On user2

From user1;

10. Transaction Control Language (TCL)

Theory Questions:

1. What is the purpose of the COMMIT and ROLLBACK commands in SQL?
   Commit Purpose: Saves all the changes made by the current transaction to the database permanently.
   Rollback Purpose: Undoes all changes made by the current transaction and restores the database to its previous state.

2. Explain how transactions are managed in SQL databases.
   -> A transaction in SQL is a sequence of one or more SQL statements that are executed as a single unit of work. A transaction ensures that all operations are completed

successfully, and if any part fails, the entire transaction is rolled back to maintain data integrity.

LAB EXERCISES:

- Lab 1: Insert a few rows into the courses table and use COMMIT to save the changes.

  Syntax: commit;

- Lab 2: Insert additional rows, then use ROLLBACK to undo the last insert operation.

  Syntax: rollback;

- Lab 3: Create a SAVEPOINT before updating the courses table, and use it to roll back specific changes.

```
-- Create the courses table
CREATE TABLE courses (
   course_id INT PRIMARY KEY,
   course_name VARCHAR(100),
   course_fee DECIMAL(10, 2)
);

-- Insert sample data
INSERT INTO courses VALUES
(1, 'Java', 5000),
(2, 'Python', 4500),
(3, 'SQL', 4000);

-- Begin transaction
BEGIN;
```

```
-- Create a SAVEPOINT before making changes
SAVEPOINT before_update;

-- Update course fees
UPDATE courses
SET course_fee = 6000
WHERE course_name = 'Java';

UPDATE courses
SET course_fee = 5000
WHERE course_name = 'Python';

-- Rollback to the savepoint to undo the second update only
ROLLBACK TO before_update;

-- Commit the changes made before the rollback
COMMIT;

-- View the final table data
SELECT * FROM courses;
```

10. SQL Joins

Theory Questions:

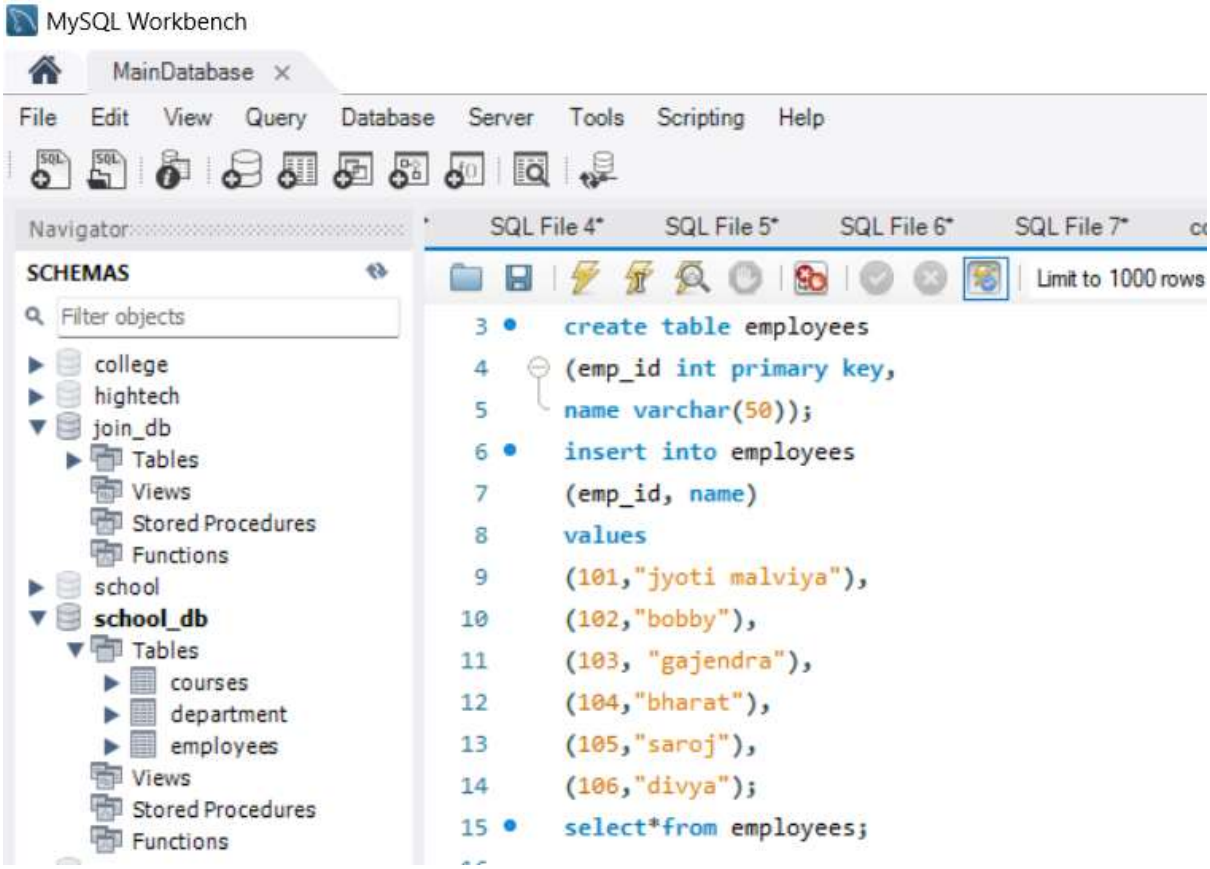
1. Explain the concept of JOIN in SQL. What is the difference between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL OUTER JOIN?

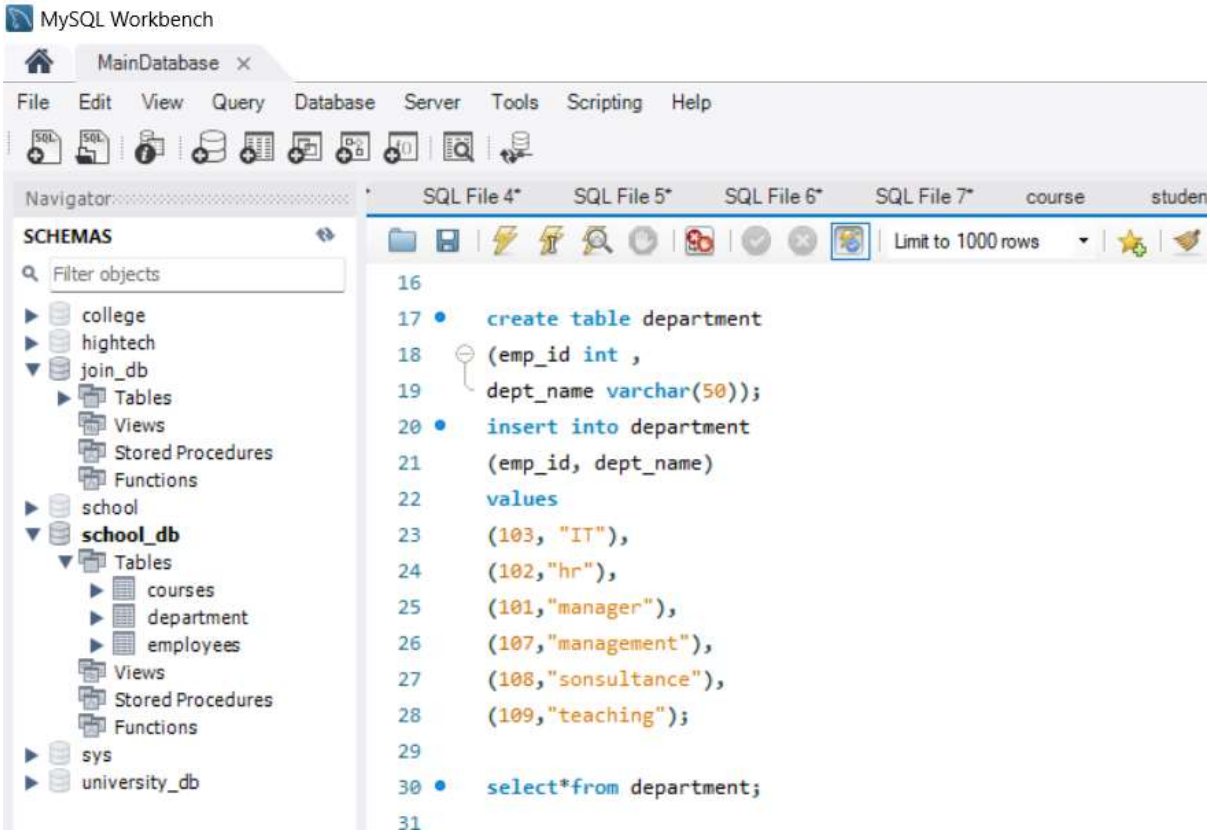
- ❖ ->Join is used to combine rows from two or more tables, based on a related column between them.
  - ➢ INNER JOIN-> returns records that have matching values in both table.
  - ➢ LEFT JOIN-> Returns all records from left table and the matched records from right table.
  - ➢ RIGHT JOIN-> Returns all records from right table and the matched records from left table.
  - ➢ FULL JOIN->Returns all records when there is a match in either left or right table.

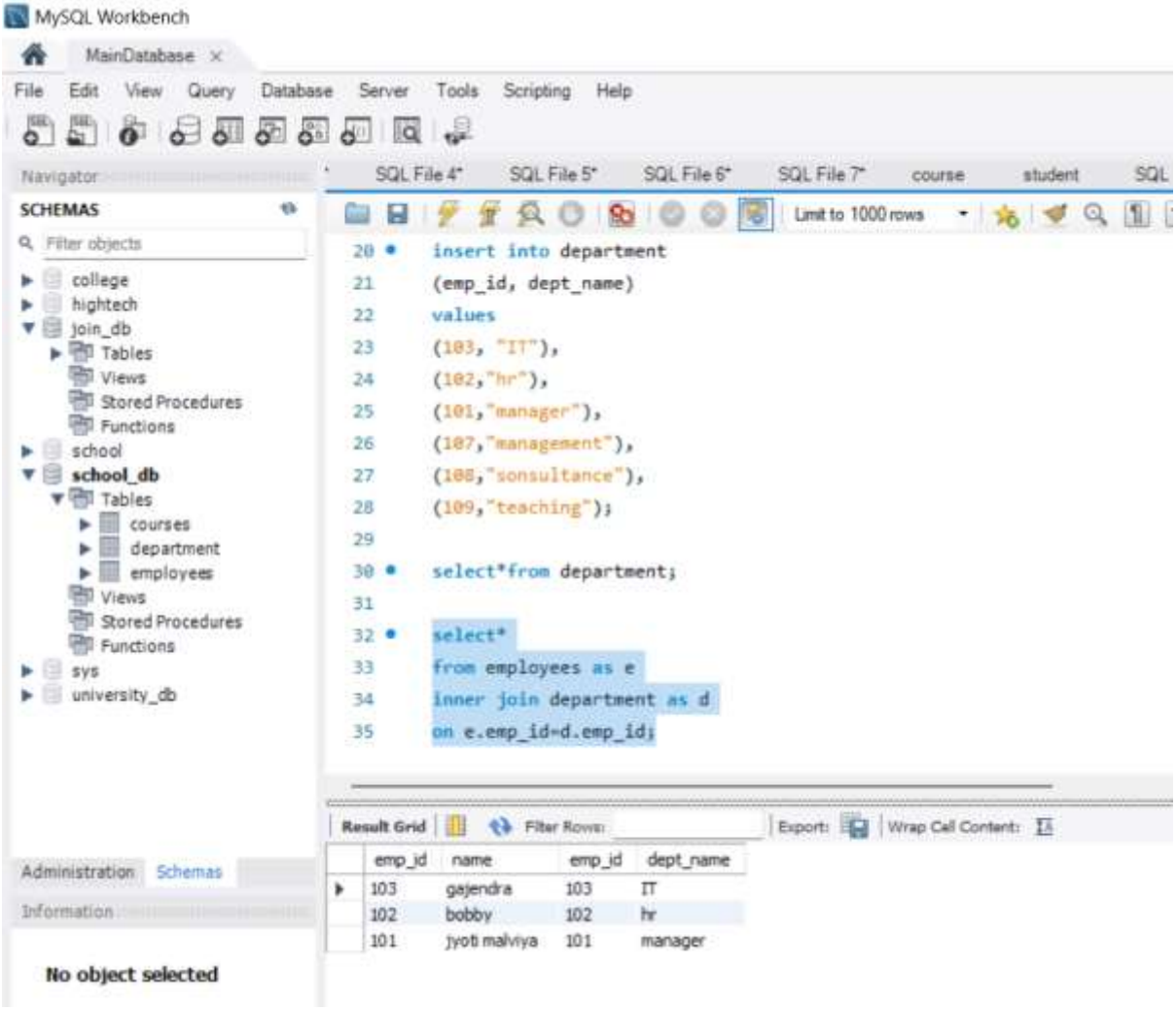
2. How are joins used to combine data from multiple tables?
   - ➢ Joins in SQL are used to combine data from multiple tables based on a related column between them.

LAB EXERCISES:

- Lab 1: Create two tables: departments and employees. Perform an INNER JOIN to display employees along with their respective departments.

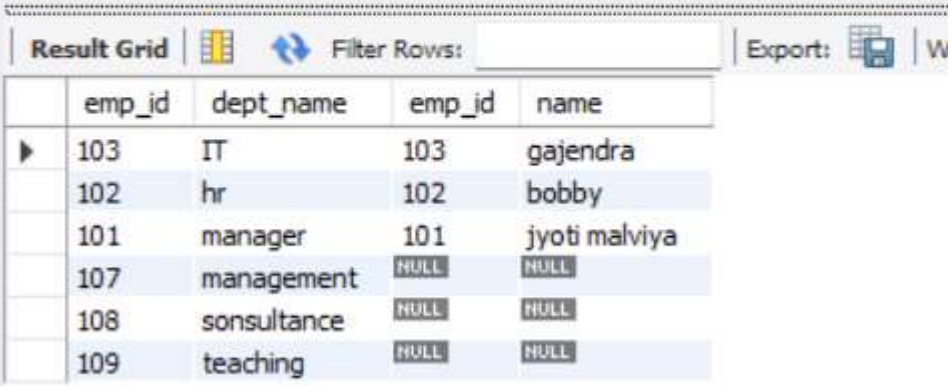

MySQL Workbench
MainDatabase
File Edit View Query Database Server Tools Scripting Help
Navigator:
SCHEMAS
Filter objects
college
hightech
join_db
Tables
Views
Stored Procedures
Functions
school
school_db
Tables
courses
department
employees
Views
Stored Procedures
Functions
SQL File 4* SQL File 5* SQL File 6* SQL File 7*
Limit to 1000 rows
3 create table employees
4 (emp_id int primary key,
5 name varchar(50));
6 insert into employees
7 (emp_id, name)
8 values
9 (101,"jyoti malviya"),
10 (102,"bobby"),
11 (103, "gajendra"),
12 (104,"bharat"),
13 (105,"saroj"),
14 (106,"divya");
15 select*from employees;


MySQL Workbench
MainDatabase
File Edit View Query Database Server Tools Scripting Help
Navigator:
SCHEMAS
Filter objects
college
hightech
join_db
Tables
Views
Stored Procedures
Functions
school
school_db
Tables
courses
department
employees
Views
Stored Procedures
Functions
sys
university_db
SQL File 4* SQL File 5* SQL File 6* SQL File 7* course studen
Limit to 1000 rows
16
17 create table department
18 (emp_id int ,
19 dept_name varchar(50));
20 insert into department
21 (emp_id, dept_name)
22 values
23 (103, "IT"),
24 (102,"hr"),
25 (101,"manager"),
26 (107,"management"),
27 (108,"sonsultance"),
28 (109,"teaching");
29
30 select*from department;
31



• Lab 2: Use a LEFT JOIN to show all departments, even those without employees.

```
31
32 •  select*
33     from department as d
34     left join employees as e
35     on e.emp_id=d.emp_id;
36
```



Result Grid | Filter Rows: | Export: | W

	emp_id	dept_name	emp_id	name
▶	103	IT	103	gajendra
	102	hr	102	bobby
	101	manager	101	jyoti malviya
	107	management	NULL	NULL
	108	sonsultance	NULL	NULL
	109	teaching	NULL	NULL

11. SQL Group By

Theory Questions:

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

   ->Groups row that have the same values into summary rows.

   ->It collects data from multiple records and groups the result by one or more column.

   Syntax::

   SELECT column_name
   ,AGGREGATE_FUNCTION(column_name)
   FROM table_name
   GROUP BY column_name;

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

Feature	GROUP BY	ORDER BY
Purpose	Groups rows based on column values	Sorts rows in ascending or descending order
Usage	Used with aggregate functions (SUM(), COUNT(), etc.)	Used to sort results after selection
Output	Returns grouped data, often summarized	Returns ordered rows as per specified column(s)
Aggregate	Mandatory for using aggregate functions	Not necessary to use with aggregate functions
Position	Comes before ORDER BY	Comes after GROUP BY
Functionality	Groups similar rows and performs aggregation	Arranges data in ascending (ASC) or descending (DESC) order

Example	GROUP BY column_name	`ORDER BY column_name ASC

LAB EXERCISES:

TOPS Technologies 2024

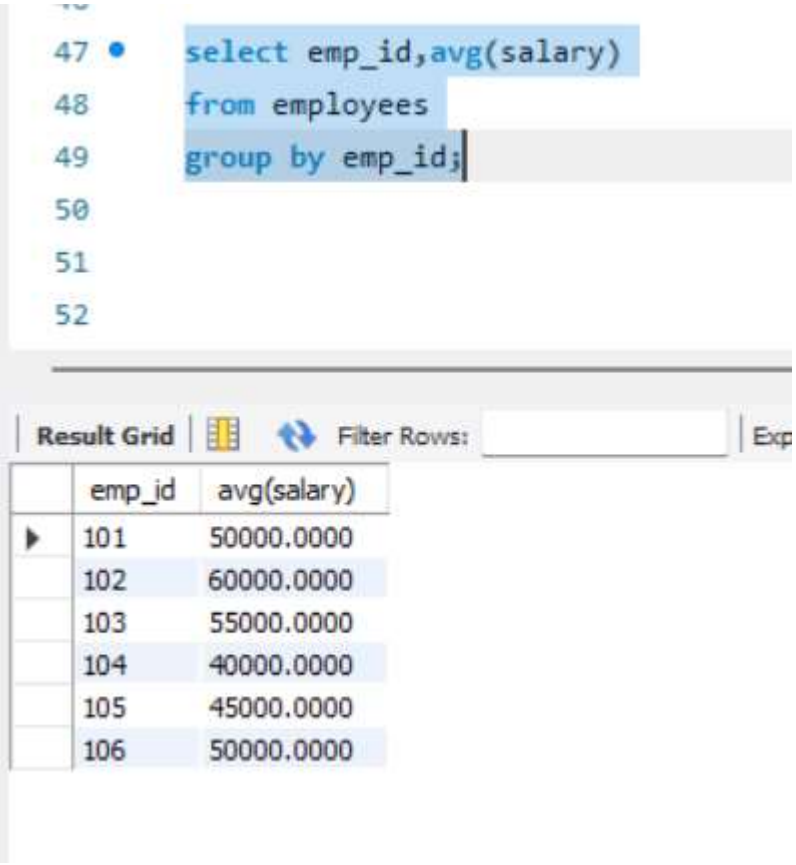
- Lab 1: Group employees by department and count the number of employees in each department using GROUP BY.

```
37 •  select dept_name,count(emp_id)
38    from department
39    group by dept_name;
40
```

Result Grid | Filter Rows: | Export: | Wra

dept_name	count(emp_id)
IT	1
hr	1
manager	1
management	1
sonsultance	1
teaching	1

- Lab 2: Use the AVG aggregate function to find the average salary of employees in each department.



13.SQL Stored Procedure :

THEORY EXERCISE:

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

  - Stored Procedure in SQL : A stored procedure in SQL is a precompiled collection of one or more SQL statements that can be executed as a single unit.

Feature	Stored Procedure	Standard SQL Query
Definition	A precompiled set of SQL statements	A single SQL statement that is

	stored in the database and executed as a unit.	executed once.
Execution	Stored procedures are executed by calling the procedure name.	Standard queries are executed directly when written.
Reusability	Can be reused multiple times without rewriting the SQL code.	Typically written and executed on the fly each time.
Security	Provides better security as you can restrict access to the procedure rather than the underlying data.	Direct access to the database objects can be required, which can be a security risk.

- <u>Explain the advantages of using stored procedures.</u>
  - Performance: Stored procedures improve performance by reducing network traffic and allowing for precompilation.
  - Reusability: You can reuse stored procedures across applications and queries, ensuring consistent logic execution.

- Security: They improve security by controlling access to data and reducing the risk of SQL injection.
- Maintainability: They allow for centralized logic that simplifies maintenance and updates.
- Error Handling: They provide advanced error handling and transaction control, ensuring data integrity.
- Reduced Client-Side Logic: They offload processing to the database server, simplifying application logic.

❖ Lab Exercise:

1) Write a stored procedure to retrieve all employees from the employees table based on department.

Ans:

```
DELIMITER $$

CREATE PROCEDURE find_emp(dep_id int)

BEGIN

SELECT employees.eid, employees.ename, employees.salary, departments.did FROM employees JOIN departments on employees.did = departments.did HAVING did = dep_id;

END;

CALL find_emp(1);
```

2) Write a stored procedure that accepts course_id as input and returns the course details.

Ans:

```
DELIMITER $$
CREATE PROCEDURE get_course(id int)
BEGIN
    SELECT * FROM courses WHERE course_id = id;
end;
CALL get_course(1);
```

course_id	course_name	course_duration
1	Python	4

1. SQL View :

THEORY EXERCISE:

- What is a view in SQL, and how is it different from a table?
  - What is a View in SQL? : A view in SQL is a virtual table that represents the result of a SELECT query. It is a stored query that can be treated like a table but does not physically store the data. Instead, it dynamically retrieves data from one or more tables whenever it is accessed.

Feature	Table	View
Definition	A table is a physical object that stores data in rows and columns.	A view is a virtual table that stores a SELECT query definition but not actual data.

Structure	Tables have a fixed structure, defined by columns with specific data types.	A view's structure is defined by the SELECT query; it can include joins, filters, and transformations.
Data Modification	Data in a table can be inserted, updated, or deleted directly.	Views are typically read-only, although some views (if updatable) allow modifications.
Indexes	Tables can have indexes to speed up data retrieval.	Views cannot have indexes; they rely on indexes in the underlying tables.

- Explain the advantages of using views in SQL databases.
  - Simplify Complex Queries: Abstract complex logic and reduce the need for users to write intricate SQL.
  - Data Abstraction and Security: Hide the underlying complexity and restrict access to sensitive data.
  - Reusability: Reuse common queries, ensuring consistency and reducing redundancy.

- Improved Security: Control access to specific data without granting full access to the underlying tables.
- Data Consistency: Provide a consistent and uniform data representation across applications.
- Better Organization: Logical separation and simplified maintenance of queries in the database.

LAB EXERCISES:

- Lab 1: Create a view to show all employees along with their department names.

Assumptions:

- employees table:
  - emp_id (Primary Key)
  - emp_name
  - dept_id (Foreign Key referencing departments.dept_id)
- departments table:
  - dept_id (Primary Key)
  - dept_name

Eg. CREATE VIEW employee_department AS
SELECT
  e.emp_id,
  e.emp_name,
  d.dept_name
FROM

```
    employees e
JOIN
    departments d ON e.dept_id = d.dept_id;
```

- Lab 2: Modify the view to exclude employees whose salaries are below $50,000.

Assumptions:

- employees table:
  - emp_id
  - emp_name
  - dept_id
  - salary
- departments table:
  - dept_id
  - dept_name

Steps:

1. Drop the Existing View

->DROP VIEW IF EXISTS employee_department;

2. Create the Modified View

```
->CREATE VIEW employee_department AS
SELECT
    e.emp_id,
    e.emp_name,
    d.dept_name,
    e.salary
```

```
FROM
   employees e
JOIN
   departments d ON e.dept_id = d.dept_id
WHERE
   e.salary >= 50000;
```

15. SQL Triggers

Theory Questions:

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

- What is a Trigger in SQL? : A trigger in SQL is a special kind of stored procedure that is automatically executed or fired by the database in response to a specific event or action on a particular table or view.
- Types of Triggers in SQL:
  - BEFORE Triggers.
  - AFTER Triggers.
  - INSTEAD OF Triggers.
- When Are Triggers Used?
  - Data Validation.
  - Enforcing Business Rules.
  - Maintaining Referential Integrity.
  - Auditing and Logging.

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

Type of Trigger	When It Is Fired	Common Use Cases	Key Points
INSERT Trigger	Fired after or before an INSERT operation is executed.	Data validation (before data is inserted)	Can be used for validation or modification of new data before insertion.
UPDATE Trigger	Fired after or before an UPDATE operation is executed.	Tracking changes (audit logs) - Preventing certain	Useful for tracking changes or enforcing business rules for updates.
DELETE Trigger	Fired after or before a DELETE operation is executed.	Cascading deletions in related tables	Useful for maintaining referential integrity or tracking deletions.

Lab Exercise:

1) Create a trigger to automatically log changes to the employees table when a new employee is added.

Ans:

```
CREATE TABLE employees_history(
   dep_id int,
   emp_id int,
```

```
    name text,
    salary int,
    time_changed timestamp,
    action_performed text
);
DELIMITER $$
CREATE TRIGGER insert_trigger AFTER INSERT ON
employees FOR EACH ROW
BEGIN
        INSERT INTO employees_history(dep_id, emp_id,
name, salary, action_performed) VALUES(new.did,
new.eid, new.ename, new.salary, 'Record Inserted');
END;
```

2) Create a trigger to update the last_modified timestamp whenever an employee record in updated.

Ans:

```
CREATE TABLE emp_update_history(
eidint,
enmae text,
salaryint,
last_modified timestamp,
didint
);
```

```
DELIMITER $$

CREATE TRIGGER update_trig AFTER UPDATE ON
employees FOR EACH ROW

BEGIN

    INSERT INTO emp_update_history(eid, ename,
salary, did) VALUES(new.eid, new.ename, new.salary,
new.did);

END;
```

16.Introduction to PL/SQL :

❖ THEORY EXERCISE:

- What is PL/SQL, and how does it extend SQL's capabilities?
  - What is PL/SQL? : PL/SQL (Procedural Language/SQL) is an extension of SQL developed by Oracle for managing and manipulating data in Oracle databases.
  - How PL/SQL Extends SQL's Capabilities :
    - ➢ Procedural Programming Constructs.
    - ➢ Exception Handling.
    - ➢ Modular Programming.
    - ➢ Caching and Performance Optimization.
    - ➢ Triggers.
    - ➢ Enhanced Security.

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

- Integration with SQL : Combines SQL with procedural programming for more powerful data handling.
- Improved Performance : Bulk operations, reduced round trips to the database, and better caching.
- Modularity and Reusability : Code can be organized into reusable procedures, functions, and packages.
- Exception Handling : Catch and handle errors gracefully, ensuring more robust applications.
- Security : Restrict access to sensitive data and logic through encapsulation.

❖ Lab Exercises:

1) Write a PL/SQL block to print the total number of employees from the employees table.

Ans:

```
DECLARE
        Employees_total NUMBER;
BEGIN
SELECT COUNT(*) INTO employees_total FROM
employees;
DBMS_OUTPUT.PUT_LINE('Total Number of Employees: '
|| employees_total);
END;
```

2) Create a PL/SQL block that calculates the total sales from an orders table.

Ans:

```
DECLARE
   total_sales NUMBER;
BEGIN
SELECT SUM(order_amount) INTO total_sales FROM
orders;
   IF v_total_sales IS NULL THEN
      v_total_sales := 0;
   END IF;
   DBMS_OUTPUT.PUT_LINE('Total Sales: ' ||
total_sales);
END;
```

17.PL/SQL Control Structures :

❖ THEORY EXERCISE:

- What are control structures in PL/SQL? Explain the IF-THEN and LOOP control structures.
  - Control Structures in PL/SQL : Control structures in PL/SQL allow you to dictate the flow of execution of your program based on certain conditions or to repeat certain actions.
  - IF-THEN Control Structure :
    - The IF-THEN structure is used for conditional branching in PL/SQL. It evaluates an expression (a condition) and, based on whether the condition is true or false, it either executes or skips a block of code
  - LOOP Control Structure :

- The LOOP structure in PL/SQL is used for executing a set of statements repeatedly until a specific condition is met. PL/SQL provides different types of loop constructs, such as the simple `LOOP`, `WHILE` loop, and `FOR` loop.

- How do control structures in PL/SQL help in writing complex queries?
  - Handling Conditional Logic (IF-THEN, IF-THEN-ELSE)
  - Looping through Data (LOOP, FOR LOOP, WHILE LOOP)
  - Complex Data Validations
  - Error Handling (EXCEPTION Block)
  - Complex Iterations and Nested Loops
  - Reducing the Number of SQL Queries

❖ Lab Exercises:

1) Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

Ans:

```
DECLARE
   v_employee_id   NUMBER := 101;
   v_department_id NUMBER;
BEGIN
   SELECT department_id INTO v_department_id
```

```
    FROM employees
    WHERE employee_id = v_employee_id;

    IF v_department_id = 10 THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' ||
v_employee_id || ' works in the HR department.');
    ELSIF v_department_id = 20 THEN
        DBMS_OUTPUT.PUT_LINE('Employee ' ||
v_employee_id || ' works in the Sales department.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Employee ' ||
v_employee_id || ' works in another department.');
    END IF;
END;
```

2) Use a FOR LOOP to iterate through employee records and display their names.

Ans:

```
DECLARE
    CURSOR emp_cursor IS
        SELECT employee_id, first_name, last_name FROM
employees;
BEGIN
    FOR emp_rec IN emp_cursor LOOP
```

```
        DBMS_OUTPUT.PUT_LINE('Employee ID: ' ||
emp_rec.employee_id ||
                        ', Name: ' || emp_rec.first_name || ' '
|| emp_rec.last_name);
    END LOOP;
END;
```

18.SQL Cursors :

❖ THEORY EXERCISE :

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

Feature	Implicit Cursor	Explicit Cursor
Definition	Automatically created by Oracle for single SQL statements.	Explicitly declared and managed by the developer.
Use Case	For simple SQL operations like `SELECT INTO`, `INSERT`, `UPDATE`, `DELETE`.	For complex SQL queries that return multiple rows.
Cursor Management	Managed automatically by Oracle; no need	Developer must open, fetch,

	to open, fetch, or close.	and close manually.
Performance	Generally faster for single-row operations.	More overhead, but essential for processing multiple rows with complex logic.

- When would you use an explicit cursor over an implicit one?
  - You would typically use an explicit cursor over an implicit cursor in the following situations:
    - ✓ When Processing Multiple Rows.
    - ✓ When You Need to Fetch Rows One by One.
    - ✓ When Handling Large Result Sets.
    - ✓ When You Need to Reuse the Cursor.
    - ✓ When You Need Full Control Over Cursor Behavior.

❖ Lab Exercise:

1) Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

Ans:

```
DECLARE
  CURSOR emp_cursor IS
    SELECT employee_id, first_name, last_name, department_id, salary
    FROM employees;
```

```
  emp_rec emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;

  LOOP
    FETCH emp_cursor INTO emp_rec;
    EXIT WHEN emp_cursor%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE('Employee ID: ' || emp_rec.employee_id ||
                ', Name: ' || emp_rec.first_name || ' ' || emp_rec.last_name ||
                ', Department ID: ' || emp_rec.department_id ||
                ', Salary: ' || emp_rec.salary);
  END LOOP;
  CLOSE emp_cursor;
END;
```

2) Create a cursor to retrieve all courses and display them one by one.

Ans:

```
DECLARE
  CURSOR course_cursor IS
    SELECT course_id, course_name, instructor FROM courses;

  course_rec course_cursor%ROWTYPE;
BEGIN
```

```
    OPEN course_cursor;

    LOOP
      FETCH course_cursor INTO course_rec;
      EXIT WHEN course_cursor%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE('Course ID: ' || course_rec.course_id ||
                           ', Course Name: ' || course_rec.course_name ||
                           ', Instructor: ' || course_rec.instructor);
    END LOOP;

    CLOSE course_cursor;
END;
```

19.Rollback and Commit Save point :

THEORY EXERCISE:

- Explain the concept of SAVEPOINT in transaction management. How do ROLLBACK and COMMIT interact with savepoints?
  - Concept of SAVEPOINT in Transaction Management : In SQL, a SAVEPOINT is a marker that allows you to define a point within a transaction to which you can later ROLLBACK if needed, without rolling back the entire transaction. It is useful for partial rollbacks where you want to undo certain changes in a transaction but not all of them.
  - How SAVEPOINT Works:

- You can set a SAVEPOINT at any point in a transaction.
- You can then execute further SQL commands or changes.
- If a problem arises or you decide to undo the work done after the savepoint, you can ROLLBACK to that specific savepoint, which will undo all changes made after it but keep the changes made before it.
- If everything is fine and you want to keep all changes, you can COMMIT the entire transaction, including the changes made before and after the savepoint.

- When is it useful to use savepoints in a database transaction?
  - Complex transactions involving multiple steps or operations that may fail at different points.
  - Error handling where you want to undo only specific parts of a transaction while keeping other successful operations intact.
  - Conditional rollbacks based on business logic or validation criteria during the transaction.
  - Long-running transactions with multiple operations that could benefit from partial rollback if something goes wrong.
  - Nested transactions, where you want to simulate the rollback of individual components within a transaction.

❖ Lab Exercises:

1) Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.

Ans:

INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)

VALUES (5001, 'John', 'Doe', 10, 50000);

SAVEPOINT before_insertion;

INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)

VALUES (5002, 'Jane', 'Smith', 20, 60000);

INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)

VALUES (5003, 'Alice', 'Johnson', 30, 55000);

ROLLBACK TO before_insertion;

COMMIT;

2) Commit part of a transaction after using a savepoint and then rollback the remaining changes.

Ans:

INSERT INTO employees (employee_id, first_name, last_name, department_id, salary)

VALUES (6001, 'Tom', 'Anderson', 10, 55000);

SAVEPOINT before_more_inserts;

```
INSERT INTO employees (employee_id, first_name, last_name,
department_id, salary)
VALUES (6002, 'Emma', 'Brown', 20, 60000);

INSERT INTO employees (employee_id, first_name, last_name,
department_id, salary)
VALUES (6003, 'Liam', 'Wilson', 30, 65000);

COMMIT;
ROLLBACK TO before_more_inserts;
```