Name : Dhruvit Savaliya

Module4

**Introduction to DBMS**

1. **Introduction to SQL**

* Theory Questions :

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

* Structured query language (SQL) is a programming language for storing and processing information in a relational database.
* A relational database stores information in tabular form.
* It is rows and columns representing different data attributes and the various relationships between the data values.
* It's used with well-known relational database management systems (RDBMS), including:

Oracle

Microsoft SQL Server

PostgreSQL

MySQL

MariaDB

SQLite

1. Explain the difference between DBMS and RDBMS.

|  |  |
| --- | --- |
| **DBMS** | **RDBMS** |
| DBMS stores data as file. | RDBMS stores data in tabular form. |
| Data elements need to access individually. | Multiple data elements can be accessed at the same time. |
| Normalization is not present. | Normalization is present. |
| It supports single user. | It supports multiple users. |
| Low software and hardware necessities. | Higher software and hardware necessities. |
| DBMS does not support distributed database. | RDBMS supports distributed database. |
| It deals with small quantity of data. | It deals with large amount of data. |

1. Describe the role of SQL inmanaging relational databases.

* Data definition : SQL allows a database administrator to define the organization and structure of stored data and the relationships amongst different stored data items.
* Data retrieval : SQL helps an application or user program to fetch stored data from a computer database and make use of it.
* Access control : SQL can also be deployed for restricting user permission for adding, retrieving or modifying stored data, thence protecting data from unauthorized access.
* Data sharing : SQL is used by concurrent users to coordinate data sharing to ensure the users don’t interfere with one another.
* Data integrity : SQL is also used in a database to define integrity constraints to prevent data corruption by system failure or inconsistent update.
* Data manipulation : SQL helps an application or user program to update the computer database by removing old data, modifying hitherto stored data, and adding new data.

1. What are the key features of SQL?

* Data Definition Language (DDL) : SQL provides a set of commands to define and modify the structure of a database, including creating tables, modifying table structure, and dropping tables.
* Data Manipulation Language (DML) : SQL provides a set of commands to manipulate data within a database, including adding, modifying, and deleting data.
* Query Language : SQL provides a rich set of commands for querying a database to retrieve data, including the ability to filter, sort, group, and join data from multiple tables.
* Transaction Control : SQL supports transaction processing, which allows users to group a set of database operations into a single transaction that can be rolled back in case of failure.
* Data Integrity : SQL includes features to enforce data integrity, such as the ability to specify constraints on the values that can be inserted or updated in a table, and to enforce referential integrity between tables.
* User Access Control : SQL provides mechanisms to control user access to a database, including the ability to grant and revoke privileges to perform certain operations on the database.
* Portability : SQL is a standardized language, meaning that SQL code written for one database management system can be used on another system with minimal modification.
* LAB EXERCISES :
* **Lab1 :** Create a new database named school\_db and a table called students with the following columns : student\_id, student\_name, age, class, andaddress.

CREATE DATABASE school\_db;

CREATE TABLE student (

student\_id int,

student\_name varchar(10),

age int,

class varchar(10),

address varchar(50)

);



* **Lab2 :** Insert five records in to the students table and retrieve all records using the SELECT statement.

INSERT INTO student VALUES (1,'dhruvit',20,'bca\_A','ahmedabad');

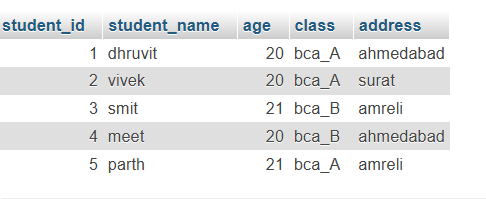
INSERT INTO student VALUES (2,'vivek',20,'bca\_A','surat');

INSERT INTO student VALUES (3,'smit',21,'bca\_B','amreli');

INSERT INTO student VALUES (4,'meet',20,'bca\_B','ahmedabad');

INSERT INTO student VALUES (5,'parth',21,'bca\_A','amreli');

SELECT \* FROM student;



1. **SQL Syntax**

* Theory Questions:

1. What are the basic components of SQL syntax?

* Keywords:

These are predefined words that perform specific functions in SQL, such as SELECT, INSERT, UPDATE, DELETE, FROM, WHERE, JOIN, and GROUP BY.

* Statements:

SQL is composed of various types of statements, each for different operations. Common statements include:

Data Query Language (DQL): SELECT statements retrieve data from the database.

Data Manipulation Language (DML): INSERT, UPDATE, and DELETE modify data.

Data Definition Language (DDL): CREATE, ALTER, and DROP manage database objects like tables.

Data Control Language (DCL): GRANT and REVOKE control permissions.

* Clauses:

Clauses are parts of a SQL statement that define conditions or organize data, such as WHERE, ORDER BY, GROUP BY, HAVING, and LIMIT.

* Functions:

SQL includes built-in functions to perform calculations, manipulate strings, and work with dates, such as COUNT, SUM, AVG, MIN, MAX, CONCAT, and NOW.

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

* SELECT column1, column2, ...

FROM table\_name

[WHERE condition]

[GROUP BY column1, column2, ...]

[HAVING condition]

[ORDER BY column1 [ASC | DESC], column2 [ASC | DESC], ...]

[LIMIT number];

1. Explain the role of clauses in SQL statements.

* SELECT Clause

Specifies the columns to be retrieved in the query.

Often includes column names or expressions and can use \* to retrieve all columns.

* FROM Clause

Indicates the table(s) from which to retrieve or manipulate data.

Essential for all data queries, as it points to the data source.

* WHERE Clause

Filters rows based on specific conditions.

Only rows that satisfy the condition(s) in WHERE are included in the result set.

* GROUP BY Clause

Groups rows that have the same values in specified columns.

Commonly used with aggregate functions (e.g., SUM, COUNT) to perform calculations on groups of rows.

* HAVING Clause

Filters groups created by GROUP BY, based on aggregate conditions.

Similar to WHERE, but operates on grouped data rather than individual rows.

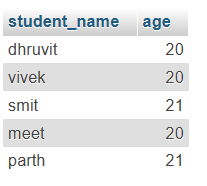
* ORDER BY Clause

Specifies the sorting order of the result set.

Columns are sorted in ascending order by default, but DESC can be used for descending order.

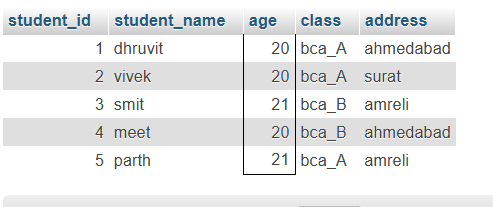
* LAB EXERCISES:
* **Lab1 :** Write SQL queries to retrieve specific columns (student\_name and age) from the students table.

SELECT student\_name,age from student;



* **Lab2 :** Write SQL queries to retrieve all students whose age is greaterthan 10.

SELECT \* FROM student WHERE age>10;



**3. SQL Constraints**

* Theory Questions:

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

* NOT NULL

Ensures that a column cannot have a NULL value.

CREATE TABLE Employees (

ID INT NOT NULL,

Name VARCHAR(50) NOT NULL

);

* UNIQUE

Ensures all values in a column are distinct.

CREATE TABLE Employees (

Email VARCHAR(100) UNIQUE

);

* PRIMARY KEY

A combination of NOT NULL and UNIQUE. It uniquely identifies each row in a table.

CREATE TABLE Employees (

ID INT PRIMARY KEY,

Name VARCHAR(50)

);

* FOREIGN KEY

Establishes a relationship between two tables by linking a column in one table to the PRIMARY KEY of another table.

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

EmployeeID INT,

FOREIGN KEY (EmployeeID) REFERENCES Employees(ID)

);

* CHECK

Ensures that all values in a column satisfy a specific condition.

CREATE TABLE Employees (

ID INT,

Age INT CHECK (Age >= 18)

);

* DEFAULT

Assigns a default value to a column when no value is provided.

CREATE TABLE Employees (

ID INT,

IsActive BOOLEAN DEFAULT TRUE

);

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

* PRIMARY KEY

Uniquely identifies each record (row) in a table.

Ensures that the column(s) designated as the primary key is both unique and not NULL.

A table can have only one primary key.

The primary key column(s) must contain unique values.

Used within the same table to enforce the uniqueness of each row.

Often serves as the unique identifier for linking to foreign keys in other tables.

* FOREIGN KEY

Creates a link between two tables by referencing the PRIMARY KEY (or a UNIQUE key) in another table.

Enforces referential integrity by ensuring that a value in the foreign key column exists in the referenced primary key column.

A table can have multiple foreign keys.

A foreign key can accept duplicate values.

Used to establish relationships between tables.

Ensures data consistency across tables.

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

* NOT NULL Constraint

Ensures that a column cannot contain NULL values.

Applied at the column level.

Enforces mandatory data entry for a specific column.

Used for columns where data is always required.

**Role**:

**Guarantees that critical fields contain meaningful data.**

**Prevents accidental omission of important information.**

* UNIQUE Constraint

Ensures that all values in a column (or combination of columns) are distinct.

Can be applied to a single column or a group of columns.

Allows only one NULL value per column

Multiple UNIQUE constraints can be defined on different columns of a table.

**Role**:

**Prevents duplication of data in a column or set of columns.**

**Ensures data integrity and consistency.**

* LAB EXERCISES :
* **Lab1 :** Create a table teachers with the following columns : teacher\_id (PrimaryKey) , teacher\_name (NOTNULL) , subject (NOTNULL) , and email(UNIUE).

CREATE TABLE teachers (

teacher\_id int PRIMARY KEY,

teacher\_name varchar(10) NOT NULL,

subject varchar(20) NOT NULL,

email varchar(20) UNIQUE

);



* **Lab2 :** Implement a FOREIGN KEY constraint to relate the teacher\_id from the teachers table with the students table.

CREATE TABLE student (

stud\_id int PRIMARY KEY,

teacher\_id int,

stud\_name varchar(10),

FOREIGN KEY (teacher\_id) REFERENCES teachers(teacher\_id)

);



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

* Theory Questions :

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

* CREATE

Used to create new database objects like tables, indexes, views, and databases.

* ALTER

Used to modify an existing database object, such as adding a column to a table or changing data types.

* DROP

Deletes an entire database object, like a table or a view, along with all of its data.

* TRUNCATE

Empties a table of its data without deleting the table structure. Unlike DELETE, it does not log individual row deletions and is faster for large tables.

1. Explain the CREATE command and its syntax.

* The CREATE command in SQL is used to create new database objects, such as tables, views, indexes, databases, or functions.
* The syntax for the CREATE command varies depending on the type of database object being created.
* 1. Creating a Database

CREATE DATABASE database\_name;

* 2. Creating a Table

CREATE TABLE table\_name (

column1 datatype constraints,

column2 datatype constraints,

...

);

* 3. Creating a View

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

* 4. Creating an Index

CREATE INDEX index\_name ON table\_name (column\_name);

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

* 1. Data Types

Data types define the kind of data that can be stored in each column of a table, such as integers, text, dates, or decimal values.

Data Integrity: Ensures that only appropriate data is stored in each column.

Memory Optimization: Helps the database allocate storage efficiently by setting aside only the required amount of memory for each column.

Performance Optimization: Databases can retrieve and process data more quickly when the data types are well-defined, as the system knows the expected format and size of each value, allowing for faster

indexing and querying. Data Consistency: Provides consistency across similar data fields, making data easier to manage, especially when dealing with complex queries, joins, or aggregations.

* 2. Constraints

Constraints enforce specific rules on the data within a column or across columns, ensuring accuracy, consistency, and reliability.

PRIMARY KEY: Uniquely identifies each row in a table.

FOREIGN KEY: Enforces a relationship between two tables by ensuring that values in one column correspond to values in another table, maintaining referential integrity.

NOT NULL: Ensures a column cannot contain NULL values, meaning every row must have a value in this column, which is important for required fields like IDs or names.

UNIQUE: Ensures all values in a column are distinct, useful for fields like email addresses or usernames where duplicate entries would cause issues.

CHECK: Defines a specific condition that values in the column must meet, allowing complex validation.

DEFAULT: Specifies a default value for a column if no value is provided, ensuring consistency in cases where specific values are not specified.

* LAB EXERCISES :
* **Lab1 :** Create a table courses with columns : course\_id, course\_name, and

course\_credits. Set the course\_id as the primarykey.

CREATE TABLE courses (

courses\_id int PRIMARY KEY,

courses\_name varchar(20),

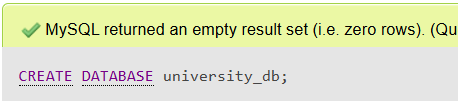
courses\_credits int

);



* **Lab2 :** Use the CREATE command to create database university\_db.

CREATE DATABASE university\_db;



**5. ALTER Command**

* Theory Questions :

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

* The ALTER command in SQL is used to modify the structure of an existing database table.
* It allows you to add, delete, or modify columns, as well as change other table properties like constraints or the table name.
* Add a New Column : Adds a new column to an existing table.
* Modify an Existing Column : Changes the data type, size, or attributes of an existing column.
* Rename a Column : supported in some RDBMS like MySQL, SQL Server, etc.
* Delete a Column : Removes an existing column from a table.
* Add Constraints : Adds constraints like UNIQUE, NOT NULL, FOREIGN KEY, etc., to a table.
* Drop Constraints : Removes an existing constraint from a table.
* Rename a Table : Changes the name of a table.

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

* 1. Add a Column

You can add one or more columns to an existing table using the ADD clause.

Syntax:

ALTER TABLE table\_name

ADD column\_name data\_type [constraint];

* 2. Modify a Column

You can change the data type, size, or constraints of an existing column using the MODIFY clause.

Syntax:

ALTER TABLE table\_name

MODIFY column\_name new\_data\_type [new\_constraint];

* 3. Drop a Column

You can remove a column from a table using the DROP COLUMN clause.

Syntax:

ALTER TABLE table\_name

DROP COLUMN column\_name;

* LAB EXERCISES :
* **Lab1 :** Modify the courses table by adding a column course\_duration using the ALTER command.

ALTER TABLE courses ADD course\_duration varchar(10);



* **Lab2 :** Drop the course\_credits column from the courses table.

ALTER TABLE courses DROP COLUMN courses\_credits;



**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 remove database objects such as tables, databases, columns, constraints, or indexes.
* nce executed, the object and its associated data are irretrievably deleted.
* Delete a Table : Removes an entire table and all the data stored in it.
* Delete a Database : Removes an entire database, including all its tables, views, and other objects.
* Delete a Column : Removes a column from a table.
* Delete an Index : Removes an index from a table.
* Delete a Constraint : Removes a specific constraint from a table, such as FOREIGN KEY or UNIQUE.

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

* Permanent Data Loss

All data stored in the table is permanently deleted.

This operation is irreversible unless you have a backup.

* Schema Deletion

The structure of the table, including its columns, data types, constraints (e.g., PRIMARY KEY, FOREIGN KEY), and indexes, is completely removed.

* Impact on Dependent Objects

If the table is referenced by foreign keys in other tables, the drop may fail unless those dependencies are removed or the foreign keys are explicitly dropped.

* Indexes and Constraints Removal

Any associated indexes, triggers, and constraints are automatically deleted along with the table.

* Cascade Effects

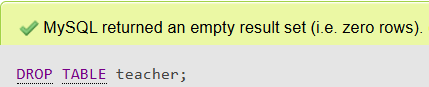
If cascading delete options (ON DELETE CASCADE) are configured, dropping a table could trigger additional deletions in related tables, though this depends on the DBMS and configuration.

* No Undo Without Backup

Unlike DELETE, the DROP command does not generate a log of the data or changes for recovery.

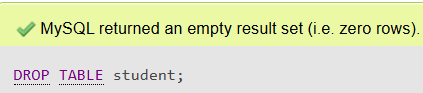
* LAB EXERCISES :
* **Lab1 :** Drop the teachers table from the school\_db database.

DROP TABLE teacher;



* **Lab2 :** Drop the students table from the school\_db database and verify that the table has been removed.

DROP TABLE student;



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

* Theory Questions :

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

* 1. INSERT Command

The INSERT command is used to add new records (rows) into a table in a database.

Syntax:

INSERT INTO table\_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);

* 2. UPDATE Command

The UPDATE command modifies existing records in a table.

Syntax:

UPDATE table\_name

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

WHERE condition;

* 3. DELETE Command

The DELETE command removes records from a table.

Syntax:

DELETE FROM table\_name

WHERE condition;

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

* Prevent Accidental Changes

UPDATE: Without the WHERE clause, all rows in the table will be updated with the same values.

DELETE: Without the WHERE clause, all rows will be deleted, resulting in data loss.

* Apply Changes to Specific Rows

The WHERE clause allows you to target specific rows that meet certain criteria.

This ensures that only the intended rows are affected.

* Preserve Data Integrity

By targeting specific rows, the WHERE clause helps maintain the integrity of the remaining data in the table.

* Avoid Unrecoverable Data Loss

In DELETE operations, missing the WHERE clause can lead to irretrievable data loss unless you have a backup or are within a transactional context that allows rollback.

* LAB EXERCISES :
* **Lab1 :** Insert three records in to the courses table using the INSERT command.

INSERT INTO courses VALUES(1,"BCA","3 Year"),(2,"MCA","2 Year"),(3,"BBA","3 Year");



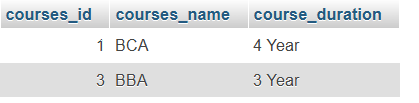
* **Lab2 :** Update the course duration of a specific course using the UPDATE command .

UPDATE courses set course\_duration='4 Year' WHERE courses\_id=1;



* **Lab3 :** Delete a course with a specific course\_id from the courses table using the DELETE command.

DELETE FROM courses WHERE courses\_id=2;



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

* Theory Questions :

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

* 1. Retrieve All Columns

SELECT \*

FROM Employees;

This retrieves all columns and rows from the Employees table.

* 2. Retrieve Specific Columns

SELECT FirstName, LastName

FROM Employees;

This retrieves only the FirstName and LastName columns.

* 3. Filter Data with WHERE

SELECT FirstName, LastName

FROM Employees

WHERE Department = 'Sales';

This retrieves employees whose department is "Sales."

* 4. Sort Results with ORDER BY

SELECT FirstName, LastName

FROM Employees

ORDER BY LastName ASC;

This sorts employees by their last name in ascending order.

* 5. Aggregate Data with GROUP BY and HAVING

SELECT Department, COUNT(\*) AS EmployeeCount

FROM Employees

GROUP BY Department

HAVING COUNT(\*) > 5;

This retrieves departments with more than 5 employees.

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

* WHERE Clause

The WHERE clause is used to filter rows in a table based on specific conditions.

It ensures that only rows meeting the specified criteria are included in the result set.

Filters data based on logical conditions.

Syntax:

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

* ORDER BY Clause

The ORDER BY clause is used to sort the rows in the result set based on one or more columns.

default, sorting is ascending (ASC), but it can also be descending (DESC).

Sorts results based on one or more columns.

Can use both ASC (ascending) and DESC (descending) keywords.

Syntax:

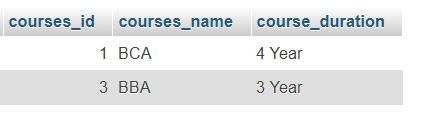
SELECT column1, column2, ...

FROM table\_name

ORDER BY column1 ASC|DESC, column2 ASC|DESC;

* LAB EXERCISES :
* **Lab1 :** Retrieve all courses from the courses table using the SELECT statement.

SELECT \* from courses;



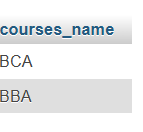
* **Lab2 :** Sort the courses based on course\_duration in descending order using ORDER BY .

SELECT course\_duration FROM courses ORDER BY course\_duration DESC;



* **Lab3 :** Limit the results of the SELECT query to show only the top two courses using LIMIT.

SELECT courses\_name FROM courses LIMIT 10;



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

* Theory Questions :

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

* GRANT

The GRANT command is used to provide specific permissions or privileges to users or roles on database objects.

It allows users to perform specific actions, such as reading, writing, or modifying data.

Syntax:

GRANT privilege(s) ON object TO user [WITH GRANT OPTION];

privilege(s): The type of permission (e.g., SELECT, INSERT, UPDATE, DELETE).

object: The database object (e.g., table name).

user: The user or role receiving the privilege.

WITH GRANT OPTION: (Optional) Allows the user to grant the same privileges to others.

* REVOKE

The REVOKE command is used to remove previously granted permissions from users or roles.

It ensures that a user or role can no longer perform specific actions on a database object.

Syntax:

REVOKE privilege(s) ON object FROM user;

privilege(s): The type of permission to remove.

object: The database object.

user: The user or role whose permission is being removed.

1. How do you manage privileges using these commands?

* 1. Granting Privileges

To allow users to perform specific actions on database objects, use the GRANT command.

Steps:

Identify the user or role: Decide who needs the permissions.

Determine the privileges: Specify the operations

Assign privileges: Use the GRANT command.

* 2. Revoking Privileges

To remove access or limit actions, use the REVOKE command.

Steps:

Identify the user or role: Decide whose permissions need to be removed.

Determine the privileges: Specify the exact operations to revoke.

Revoke privileges: Use the REVOKE command.

* 3. Managing Permissions Using Roles

Using roles simplifies privilege management by grouping users under a single entity.

Steps:

Create a role: CREATE ROLE SalesTeam;

Grant privileges to the role: GRANT SELECT, INSERT ON Orders TO SalesTeam;

Assign users to the role: GRANT SalesTeam TO User1, User2;

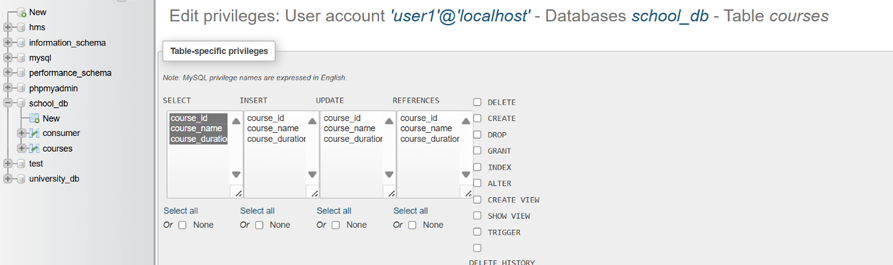
Revoke privileges from the role: REVOKE SELECT ON Orders FROM SalesTeam;

* LAB EXERCISES :
* **Lab1 :** Create two new users user1 and user2 and grant user1 permission to SELECT from the courses table.

CREATE USER user1 IDENTIFIED BY 'password1';

CREATE USER user2 IDENTIFIED BY 'password2';

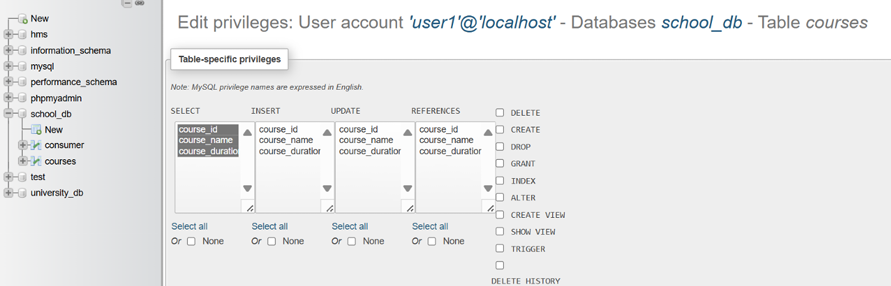
GRANT SELECT ON courses TO user1;



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

REVOKE INSERT ON courses FROM 'user1'@'localhost';

GRANT INSERT ON courses TO 'user2'@'localhost';



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

* Theory Questions :

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

* COMMIT

Saves all the changes made during the current transaction permanently to the database.

Once a COMMIT is executed, the changes cannot be undone.

Ensures that the database is updated and reflects all the modifications made during the transaction.

* ROLLBACK

Reverts all changes made during the current transaction, restoring the database to its state before the transaction began.

It is used to undo changes if an error or issue occurs during the transaction.

1. Explain how transactions are managed in SQL databases.

* Begin Transaction:

A transaction begins explicitly using BEGIN TRANSACTION (or implicitly, depending on the DBMS).

* Execute SQL Statements:

Perform a series of SQL operations (e.g., INSERT, UPDATE, DELETE).

* Commit the Transaction:

If all operations are successful, the transaction is finalized using COMMIT, making changes permanent.

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

INSERT INTO courses VALUES(2,'MCA','4 Year');

COMMIT;



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

INSERT INTO courses VALUES(4,'MBA','2 Year');

ROLLBACK;



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

START TRANSACTION;

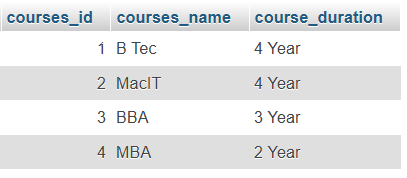
SAVEPOINT b\_update;

UPDATE courses SET courses\_name = 'B Tec' WHERE courses\_id = 1;

UPDATE courses SET courses\_name = 'MacIT' WHERE courses\_id = 2;

ROLLBACK TO b\_update;

COMMIT;



**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?

* INNER JOIN

Returns only the rows where there is a match between the columns in both tables.

Rows without matches in either table are excluded.

Syntax:

SELECT columns

FROM table1

INNER JOIN table2

ON table1.column = table2.column;

* LEFT JOIN (or LEFT OUTER JOIN)

Returns all rows from the left table and the matching rows from the right table.

If there is no match, NULL values are returned for columns from the right table.

Syntax:

SELECT columns

FROM table1

LEFT JOIN table2

ON table1.column = table2.column;

* RIGHT JOIN (or RIGHT OUTER JOIN)

Returns all rows from the right table and the matching rows from the left table.

If there is no match, NULL values are returned for columns from the left table.

Syntax:

SELECT columns

FROM table1

RIGHT JOIN table2

ON table1.column = table2.column;

* FULL OUTER JOIN

Combines the results of both LEFT JOIN and RIGHT JOIN.

Returns all rows when there is a match in either table, filling unmatched columns with NULLs.

Syntax:

SELECT columns

FROM table1

FULL OUTER JOIN table2

ON table1.column = table2.column;

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

* Identify the Relationship Between Tables:

In a relational database, tables are often linked by keys:

A primary key uniquely identifies a row in a table.

A foreign key in one table refers to the primary key in another table.

Example: A students table with course\_id as a foreign key to the courses table.

* Write the Join Condition:

Use the ON clause to specify the column(s) to match.

Example: students.course\_id = courses.course\_id.

* Choose the Type of Join:

Depending on the data you want to include use the appropriate type of JOIN: INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN.

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

CREATE TABLE departments

(

department\_id INT PRIMARY KEY,

department\_name VARCHAR(50) NOT NULL

);

CREATE TABLE employees

(

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(50) NOT NULL,

department\_id INT,

FOREIGN KEY (department\_id) REFERENCES

departments(department\_id)

);

SELECT employees.employee\_id,employees.employee\_name,departments.department\_name FROM employees INNER JOIN departments ON

employees.department\_id=departments.department\_id;



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

SELECT departments.department\_id,

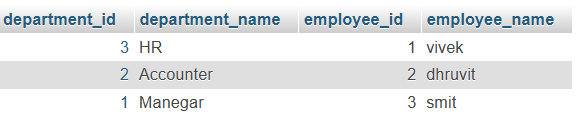
departments.department\_name,

employees.employee\_id, employees.employee\_name

FROM departments

LEFT JOIN employees ON departments.department\_id =

employees.department\_id;



**12. SQL Group By**

* Theory Questions :

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

* The GROUP BY clause in SQL is used to group rows that have the same values in specified columns into summary rows, such as calculating the total, average, count, or other aggregate functions for each group.
* It is often used in combination with aggregate functions like SUM(), COUNT(), AVG(), MIN(), and MAX() to perform calculations on grouped data.
* Syntax

SELECT column1, column2, aggregate\_function(column3)

FROM table\_name

GROUP BY column1, column2;

* The GROUP BY clause divides the rows of a table into groups based on the values in one or more columns.
* Aggregate functions compute a value for each group.
* Only columns listed in the GROUP BY clause or used in aggregate functions can be included in the SELECT list.

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

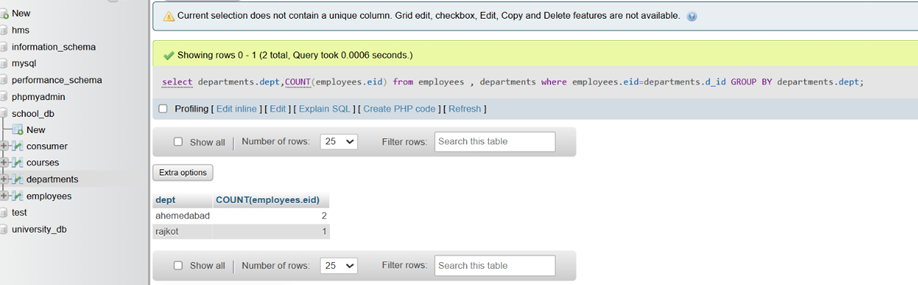
|  |  |
| --- | --- |
| GROUP BY | ORDER BY |
| Groups rows into smaller sets. | No grouping; works on individual rows or final grouped rows. |
| Does not sort the rows. | Always sorts the rows in a specified order. |
| Required when using aggregate functions on grouped data. | Optional; used for sorting the results. |
| Affects the logical grouping of rows. | Affects only the display order of rows. |

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

select departments.dept,COUNT(employees.eid) from

employees , departments where employees.eid=depart

ments.d\_id GROUP BY departments.dept;

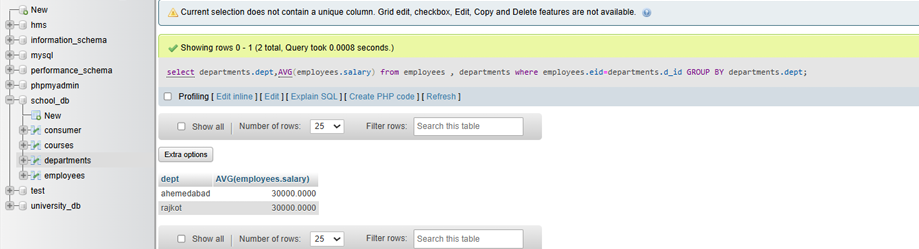


* **Lab 2:** Use the AVG aggregate function to find the average salary of employees in each department.

select departments.dept,AVG(employees.salary) from e

mployees , departments where employees.eid=depart

ments.d\_id GROUP BY departments.dept;



**13. SQL Stored Procedure**

* Theory Questions :

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

* A stored procedure in SQL is a precompiled set of SQL statements and optional control-flow logic stored in the database.
* **Reusable**: Once defined, they can be called multiple times from various applications or scripts.
* **Precompiled**: Stored procedures are compiled and optimized by the database engine, leading to faster execution compared to ad-hoc SQL queries.
* **Parameterization**: They can accept input parameters and return output values, allowing dynamic execution.
* **Security**: Permissions can be controlled at the stored procedure level, adding an additional layer of access control.
* Difference Between Stored Procedures and Standard SQL Queries

|  |  |
| --- | --- |
| Stored Procedure | Standard SQL Query |
| A predefined set of SQL statements stored in the database. | A single SQL statement or a set of statements written ad hoc. |
| Precompiled and optimized by the database engine. | Compiled and executed on the fly. |
| Reusable with consistent logic. | Requires rewriting or copying for reuse. |
| Supports input and output parameters for dynamic execution. | Parameters are manually added in each instance. |
| Faster due to precompilation and reduced parsing overhead. | Slower as it requires compilation each time it is executed. |

1. Explain the advantages of using stored procedures.

* Performance Optimization

Stored procedures are compiled and optimized by the database engine when created, leading to faster execution compared to ad-hoc queries, which are compiled each time they run.

* Code Reusability

Stored procedures allow you to encapsulate business logic in the database, making it reusable across multiple applications or modules.

* Security

Permissions can be granted at the procedure level, allowing users to execute stored procedures without needing direct access to the underlying tables or data.

* Simplified Maintenance

Modifying the logic in a stored procedure automatically applies the changes wherever the procedure is used, ensuring consistency.

* Scalability

Stored procedures allow you to encapsulate complex logic within the database, enabling the database server to handle computation-intensive operations, which can improve application scalability.

* Error Handling

Stored procedures can include robust error-handling mechanisms, such as TRY...CATCH blocks, to manage and log errors more effectively.

* LAB EXERCISES :
* **Lab 1:** Write a stored procedure to retrieve all employees from the employees table based on department.

DELIMITER //

CREATE

get\_employees\_by\_department(IN VARCHAR(100))

PROCEDURE dept\_name

BEGIN

DROP TEMPORARY TABLE IF EXISTS temp\_employees;

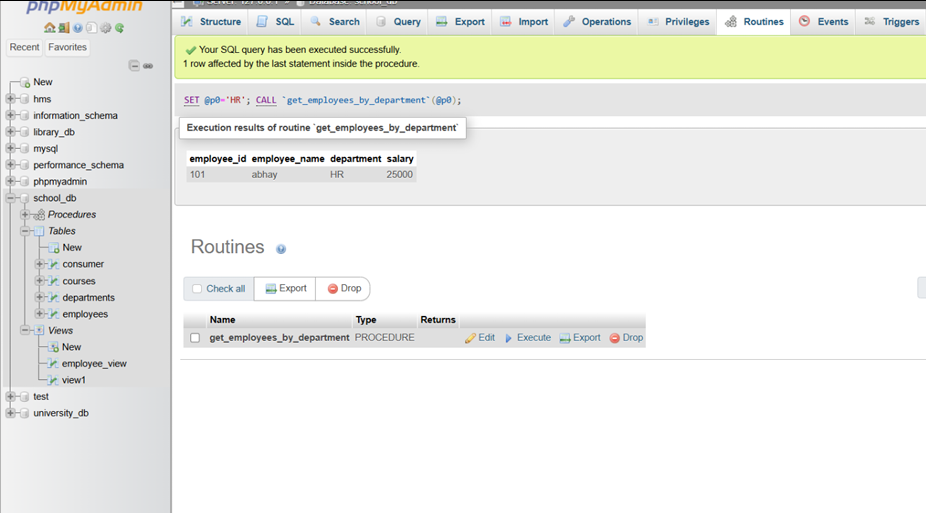
CREATE TEMPORARY TABLE temp\_employees AS SELECT employee\_id, employee\_name, department, salary FROM employees

WHERE department = dept\_name;

SELECT \* FROM temp\_employees;

END //

DELIMITER ;



* **Lab 2:** Write a stored procedure that accepts course\_id as input and returns the course details.

DELIMITER //

CREATE PROCEDURE get\_course\_details(IN coursee\_id INT)

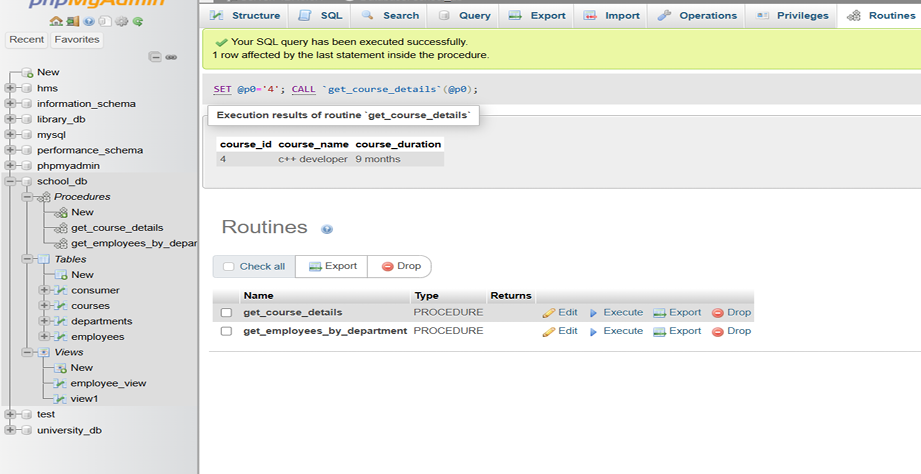
BEGIN

SELECT course\_id, course\_name, course\_duration FROM courses

WHERE course\_id = coursee\_id;

END //

DELIMITER ;



**14. SQL View**

* Theory Questions :

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

* **Virtual Table**: A view is a logical representation of data from one or more tables.
* **Dynamic Data**: The data in a view changes dynamically based on the underlying tables.
* **Simplifies Complex Queries**: By encapsulating a complex query within a view, you can reuse it as a simple query.
* **Security**: Views can restrict access to specific columns or rows, providing a layer of abstraction and security.

|  |  |
| --- | --- |
| View | Table |
| A virtual table based on a SQL query. | A physical structure storing data in rows and columns. |
| Does not store data physically; only stores the query definition. | Physically stores data in the database. |
| Slightly slower as it retrieves data dynamically. | Faster since data is stored physically. |
| Can sometimes allow updates, but with restrictions. | Fully updatable. |

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

* Simplification of Complex Queries

A view can simplify repetitive or complex SQL queries by encapsulating them into a reusable structure.

* Security and Access Control

Views allow you to restrict user access to specific rows or columns while hiding sensitive data in the underlying tables.

* Consistent Interface

If the schema of underlying tables changes, the view definition can remain consistent, ensuring applications depending on the view are unaffected.

* Reusability and Modularity

By defining a view, you centralize the query logic, reducing redundancy and ensuring consistency across different parts of an application.

* Performance Optimization

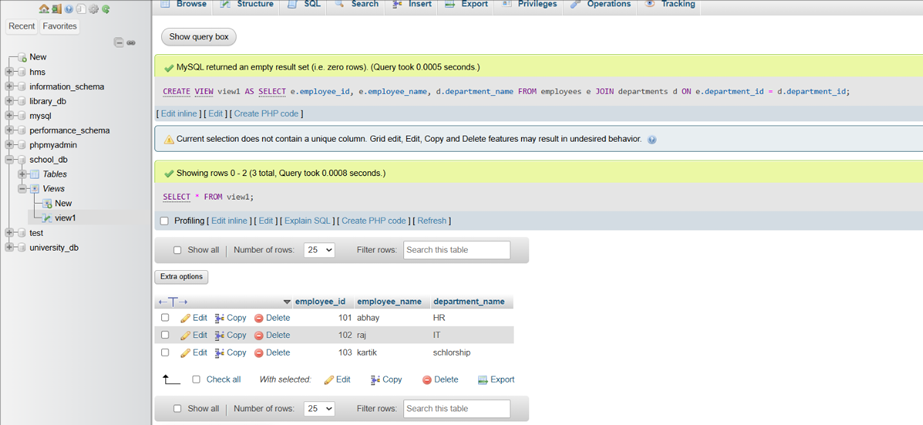
For materialized views (in databases that support them), results can be stored physically, speeding up access for frequently used queries.

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

CREATE VIEW view1 AS SELECT e.employee\_id, e.emplo

yee\_name, d.department\_name FROM employees e

JOIN departments d ON e.department\_id = d.department\_id;



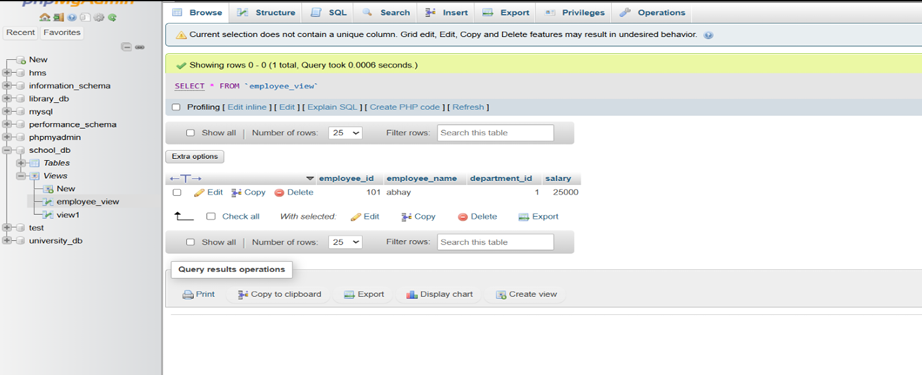
* **Lab 2:** Modify the view to exclude employees whose salaries are below $50,000.

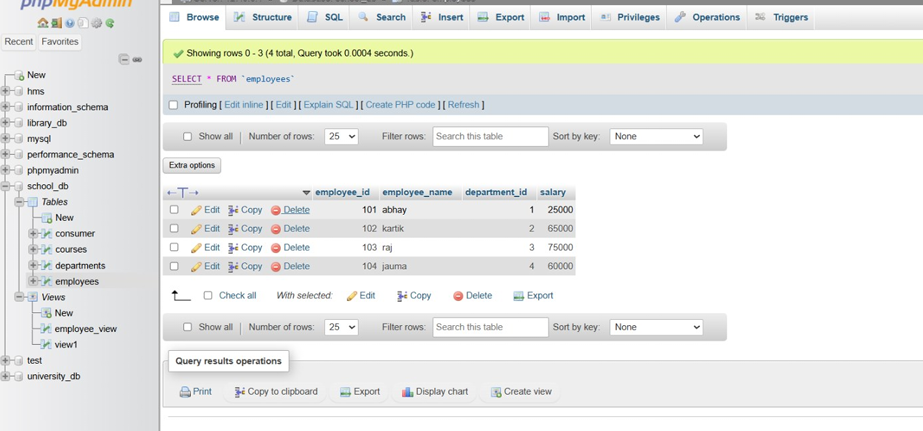
CREATE OR REPLACE VIEW employee\_view AS

SELECT employee\_id, employee\_name, salary

FROM employees

WHERE salary <= 50000;





**15. SQL Triggers**

* Theory Questions :

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

* A trigger in SQL is a special type of stored procedure that automatically executes in response to certain events on a specified table or view.
* Triggers are used to enforce business rules, maintain data integrity, log activities, or automatically perform actions when data changes.
* Types of Triggers

Triggers are generally categorized based on:

Execution Timing

Event Type

* Based on Execution Timing

Executes after the triggering event (e.g., INSERT, UPDATE, DELETE) is completed. Most commonly used to enforce business rules or log actions.

Overrides the triggering event, allowing custom logic to execute instead. Often used for views or complex operations where direct execution is not feasible.

* Based on Event Type

Executes when a new row is inserted into the table. Used for validating input or maintaining audit logs.

Executes when existing data is updated. Often used for enforcing business rules or tracking changes.

Executes when rows are deleted from the table. Commonly used for archiving or preventing accidental deletions.

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

|  |  |  |
| --- | --- | --- |
| INSERT Trigger | UPDATE Trigger | DELETE Trigger |
| INSERT operations. | UPDATE operations. | DELETE operations. |
| INSERTED (contains new rows). | INSERTED (new values), DELETED (old values). | DELETED (contains deleted rows). |
| Add related data, validate inserts, audit. | Track changes, validate updates, enforce rules. | Archive data, prevent deletions, audit. |
| Only affects new rows being added. | Affects rows being modified. | Affects rows being removed. |
| Logging, derived table population. | Auditing, enforcing business rules. | Archiving, preventing data loss. |

* LAB EXERCISES :
* **Lab 1:** Create a trigger to automatically log changes to the employees table when a new employee is added.

DELIMITER //

CREATE TRIGGER before\_employee\_update

BEFORE UPDATE ON employees

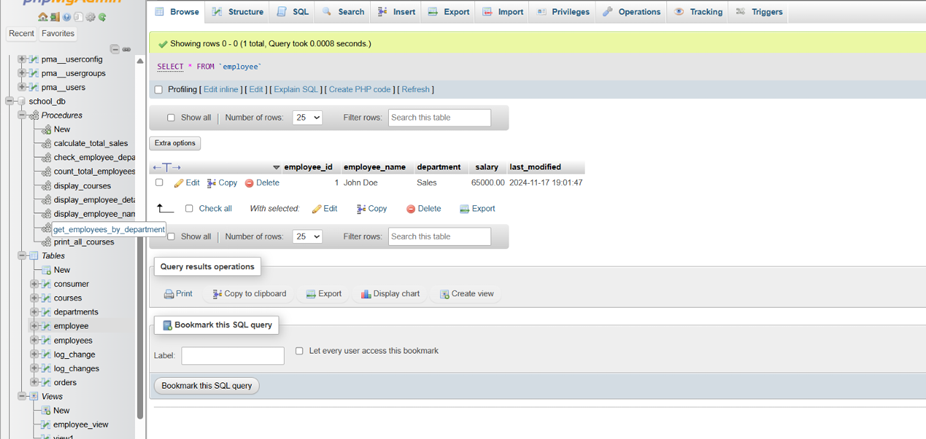
FOR EACH ROW

BEGIN

SET NEW.last\_modified = CURRENT\_TIMESTAMP;

END //

DELIMITER ;



* **Lab 2:** Create a trigger to update the last\_modified timestamp whenever an employee record is updated.

ALTER TABLE employees

ADD COLUMN last\_modified TIMESTAMP DEFAULT

CURRENT\_TIMESTAMP

ON

CURRENT\_TIMESTAMP;

DELIMITER //

CREATE TRIGGER update\_last\_modified

BEFORE UPDATE ON employees

FOR EACH ROW

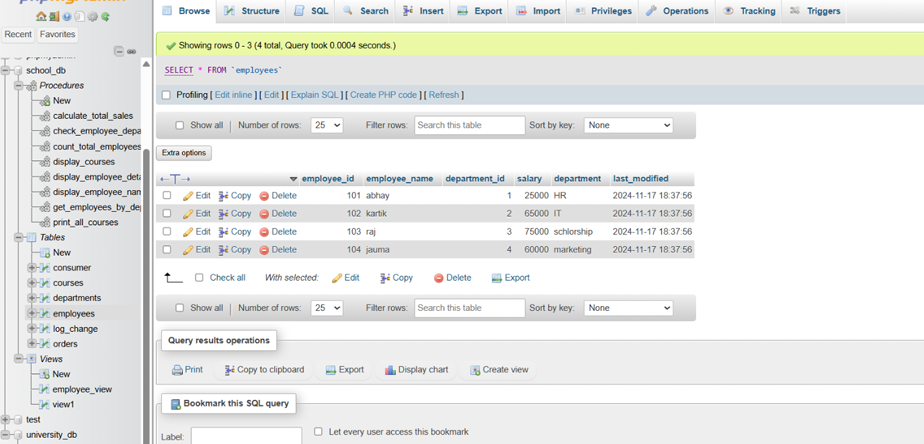
BEGIN

UPDATE

SET NEW.last\_modified = CURRENT\_TIMESTAMP;

END //

DELIMITER ;



**16. Introduction to PL/SQL**

* Theory Questions :

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

* PL/SQL (Procedural Language/SQL) is Oracle Corporation's procedural extension to SQL, designed to enhance the functionality and power of SQL by adding procedural constructs.
* It allows for writing structured and complex logic within the database, making it a powerful tool for developers working with Oracle databases.
* **Procedural Constructs**: Includes loops, conditional statements (IF, CASE), and exception handling.
* **Encapsulation**: Supports modular programming with stored procedures, functions, packages, and triggers.
* **Variables and Constants**: Enables the declaration and use of variables, constants, and complex data types.
* **Integration with SQL**: Seamlessly integrates SQL for querying and manipulating data.
* **Error Handling**: Provides robust exception handling mechanisms.
* Syntax of PL/SQL Block

DECLARE

-- Variable declarations

BEGIN

-- Executable statements

EXCEPTION

-- Error handling

END;

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

* 1. Tight Integration with SQL

PL/SQL allows embedding SQL statements within procedural logic, making it easy to query, manipulate, and manage data directly.

* 2. Procedural Capabilities

Includes loops (FOR, WHILE), conditionals (IF, CASE), and other procedural constructs.

* 3. Modular Programming

Allows encapsulation of logic into reusable stored procedures, functions, and packages.

* 4. Performance Optimization

PL/SQL executes blocks of code on the database server, reducing the need for multiple round trips between the application and database.

* 5. Robust Error Handling

PL/SQL provides structured mechanisms to handle runtime errors using EXCEPTION blocks.

* LAB EXERCISES :
* **Lab 1:** Write a PL/SQL block to print the total number of employees from the employees table.

DELIMITER //

CREATE PROCEDURE count\_total\_employees()

BEGIN

DECLARE v\_total\_employees INT;

SELECT

COUNT(employee\_id)

v\_total\_employees FROM employees;

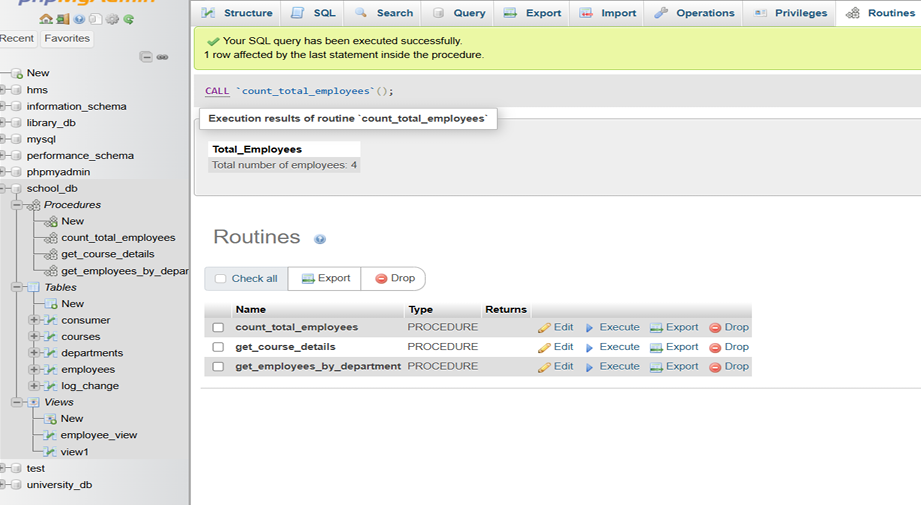
INTO

SELECT CONCAT('Total number of employees: ',

v\_total\_employees) AS Total\_Employees;

END //

DELIMITER ;



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

DELIMITER //

CREATE PROCEDURE calculate\_total\_sales()

BEGIN

DECLARE v\_total\_sales DECIMAL(10, 2);

SELECT SUM(order\_amount) INTO v\_total\_sales

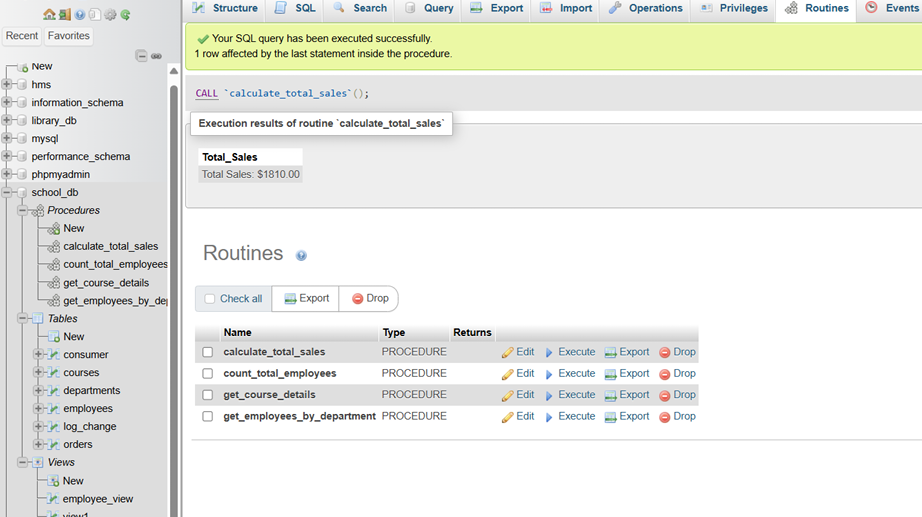
FROM orders;

SELECT CONCAT('Total Sales: $', v\_total\_sales) AS

Total\_Sales;

END //

DELIMITER ;



**17. PL/SQL Control Structures**

* Theory Questions :

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

* Conditional Statements: Execute code based on conditions.

IF-THEN

IF-THEN-ELSE

IF-THEN-ELSIF

CASE

* Iterative Statements: Repeatedly execute a block of code.

LOOP

WHILE LOOP

FOR LOOP

* Sequential Control: Unconditionally control execution flow using:

GOTO

* 1. IF-THEN Control Structure

The IF-THEN statement executes a block of code if a specified condition evaluates to TRUE.

Syntax

IF condition THEN

-- Code to execute if condition is TRUE

END IF;

* 2. LOOP Control Structures

LOOP statements allow the repeated execution of a block of code. PL/SQL provides different types of loops:

Syntax

LOOP

-- Code to execute

EXIT WHEN condition;

END LOOP;

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

* 1. Conditional Logic

IF-THEN-ELSE: Allows decision-making based on specific conditions, enabling dynamic execution of different SQL statements.

* 2. Iteration with Loops

**FOR Loop:** Automates repetitive operations, such as processing multiple rows or executing a query multiple times.

**WHILE Loop:** Executes a block of code as long as a condition remains true, useful for handling dynamic query execution or conditions.

* 3. Handling Exceptions

PL/SQL allows you to gracefully handle errors like division by zero, constraint violations, or missing data. This ensures that your queries run reliably even in complex scenarios.

* 4. Combining SQL and Procedural Logic

PL/SQL enables embedding procedural logic in SQL operations.

* 5. Dynamic SQL Execution

Control structures in PL/SQL allow you to build and execute SQL statements dynamically using EXECUTE IMMEDIATE or DBMS\_SQL. This is essential for queries that need to adapt to runtime conditions.

* LAB EXERCISES :
* **Lab 1:** Write a PL/SQL block using an IF-THEN condition to check the department of an employee.

DELIMITER //

CREATE PROCEDURE check\_employee\_department(IN

employee\_id INT)

BEGIN

DECLARE v\_department VARCHAR(100);

DECLARE done INT DEFAULT FALSE;

DECLARE cur CURSOR FOR

SELECT department FROM employees WHERE

employee\_id = employee\_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET

done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_department;

IF done THEN

LEAVE read\_loop;

END IF;

IF v\_department = 'Sales' THEN

SELECT 'The employee works in the Sales

department.' AS message;

ELSEIF v\_department = 'HR' THEN

SELECT 'The employee works in the HR

department.' AS message;

ELSE

SELECT 'The employee works in another

department.' AS message;

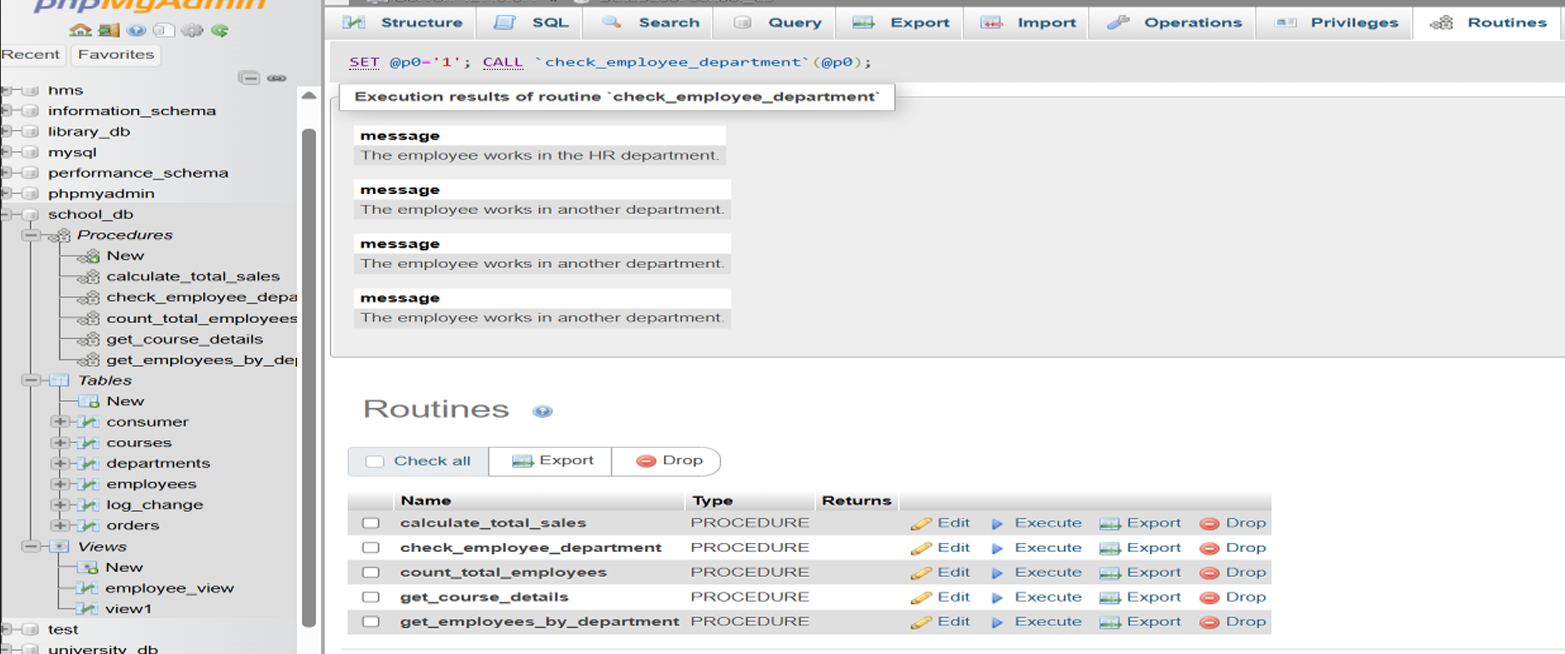
END IF;

END LOOP;

CLOSE cur;

END //

DELIMITER ;



* **Lab 2:** Use a FOR LOOP to iterate through employee records and display their names.

DELIMITER //

CREATE PROCEDURE display\_employee\_names()

BEGIN

DECLARE v\_employee\_name VARCHAR(100);

DECLARE done INT DEFAULT 0;

DECLARE emp\_cursor CURSOR FOR

SELECT employee\_name FROM employees;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET

done = 1;

OPEN emp\_cursor;

read\_loop: LOOP

FETCH emp\_cursor INTO v\_employee\_name;

IF done THEN

LEAVE read\_loop;

END IF;

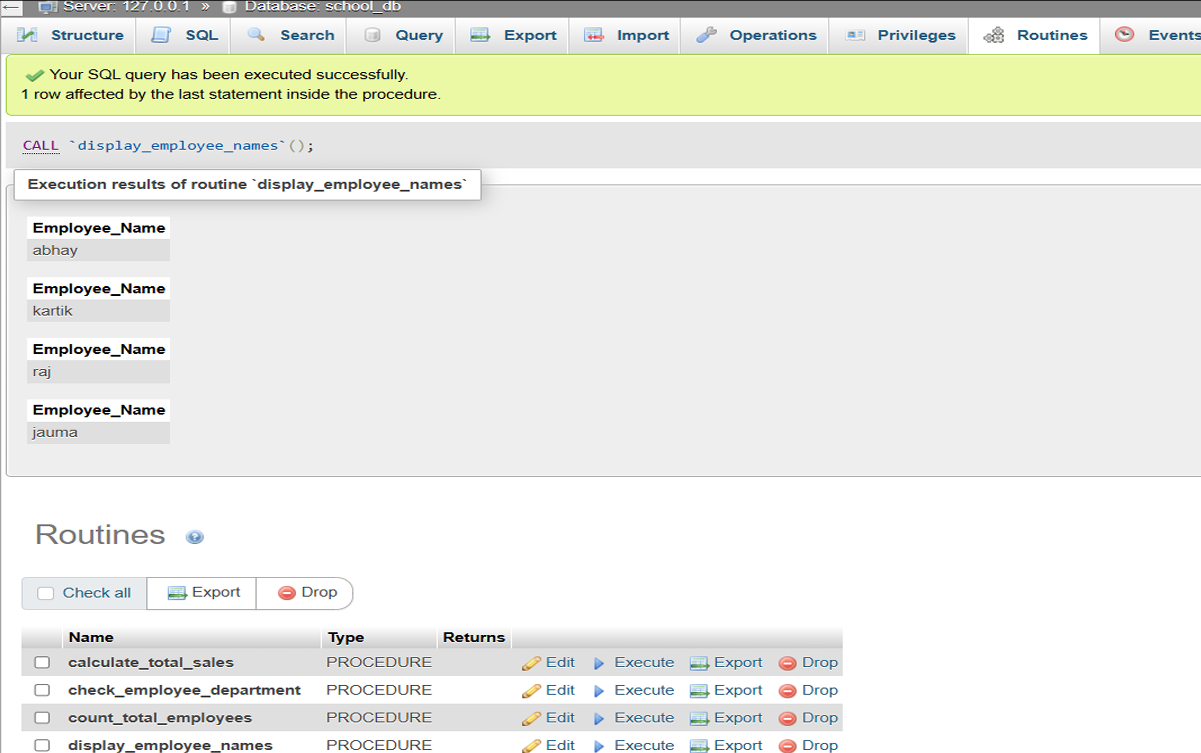
SELECT v\_employee\_name AS Employee\_Name;

END LOOP;

CLOSE emp\_cursor;

END //

DELIMITER ;



**18. SQL Cursors**

* Theory Questions :

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

* In PL/SQL, a cursor is a pointer to the context area where the results of a SQL query are processed and stored.
* It allows you to retrieve and manipulate query results row by row.

|  |  |
| --- | --- |
| Implicit Cursor | Explicit Cursor |
| Automatically created by Oracle. | Must be explicitly declared by the programmer. |
| Oracle manages the lifecycle (open, fetch, close). | Programmer has full control (open, fetch, close). |
| Single-row queries or DML statements. | Multi-row queries or when row-by-row processing is needed. |
| Automatically fetches a single row or processes DML. | Must fetch rows manually, one at a time or in bulk. |

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

* When Processing Multiple Rows

Implicit cursors can handle only single-row queries, but if your query returns multiple rows, an explicit cursor allows you to process each row individually using a FETCH loop.

* For Row-by-Row Processing

When you need to perform specific operations for each row in a result set, explicit cursors let you process the data one row at a time.

* When Fetching Data Dynamically

If you need to dynamically control the fetch operation explicit cursors give you this control.

* When Using Cursor Attributes for Advanced Logic

Explicit cursors allow you to use attributes like %ROWCOUNT, %FOUND, %NOTFOUND, and %ISOPEN to track and manage the state of the cursor.

* When a Query is Repeatedly Used

If you need to execute the same query multiple times in a procedure or a loop, an explicit cursor can be declared once and reused to avoid redundant code.

* LAB EXERCISES :
* **Lab 1:** Write a PL/SQL block using an explicit cursor to retrieve and display employee details.

DELIMITER //

CREATE PROCEDURE display\_employee\_details()

BEGIN

DECLARE v\_employee\_id INT;

DECLARE v\_employee\_name VARCHAR(100);

DECLARE v\_department VARCHAR(100);

DECLARE v\_salary DECIMAL(10, 2);

DECLARE done INT DEFAULT FALSE;

DECLARE emp\_cursor CURSOR FOR

SELECT

employee\_id, employee\_name,

department, salary FROM employees;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET

done = TRUE;

OPEN emp\_cursor;

read\_loop: LOOP

FETCH

emp\_cursor

INTO

v\_employee\_id,

v\_employee\_name, v\_department, v\_salary;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT CONCAT('Employee ID: ', v\_employee\_id, ',

Name: ', v\_employee\_name, ', Department: ',

v\_department, Salary:',

v\_salary)AS',

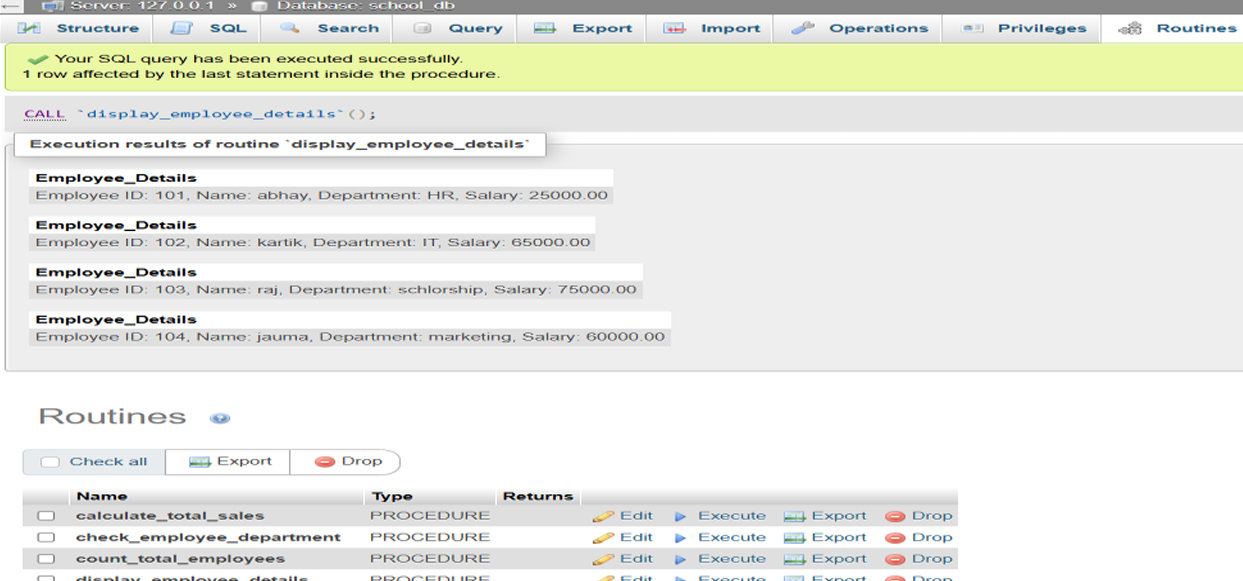
Employee\_Details;

END LOOP;

CLOSE emp\_cursor;

END //

DELIMITER ;



* **Lab 2:** Create a cursor to retrieve all courses and display them one by one.

DELIMITER //

CREATE PROCEDURE print\_all\_courses()

BEGIN

DECLARE v\_course\_id INT;

DECLARE v\_course\_name VARCHAR(100);

DECLARE v\_duration INT;

DECLARE done INT DEFAULT FALSE;

DECLARE course\_cursor CURSOR FOR

SELECT course\_id, course\_name, course\_duration

FROM courses;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET

done = TRUE;

OPEN course\_cursor;

read\_loop: LOOP

FETCH course\_cursor INTO v\_course\_id,

v\_course\_name,v\_duration;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT CONCAT('Course ID: ', v\_course\_id, ',

Name: ', v\_course\_name, ', Duration: ', v\_duration) AS

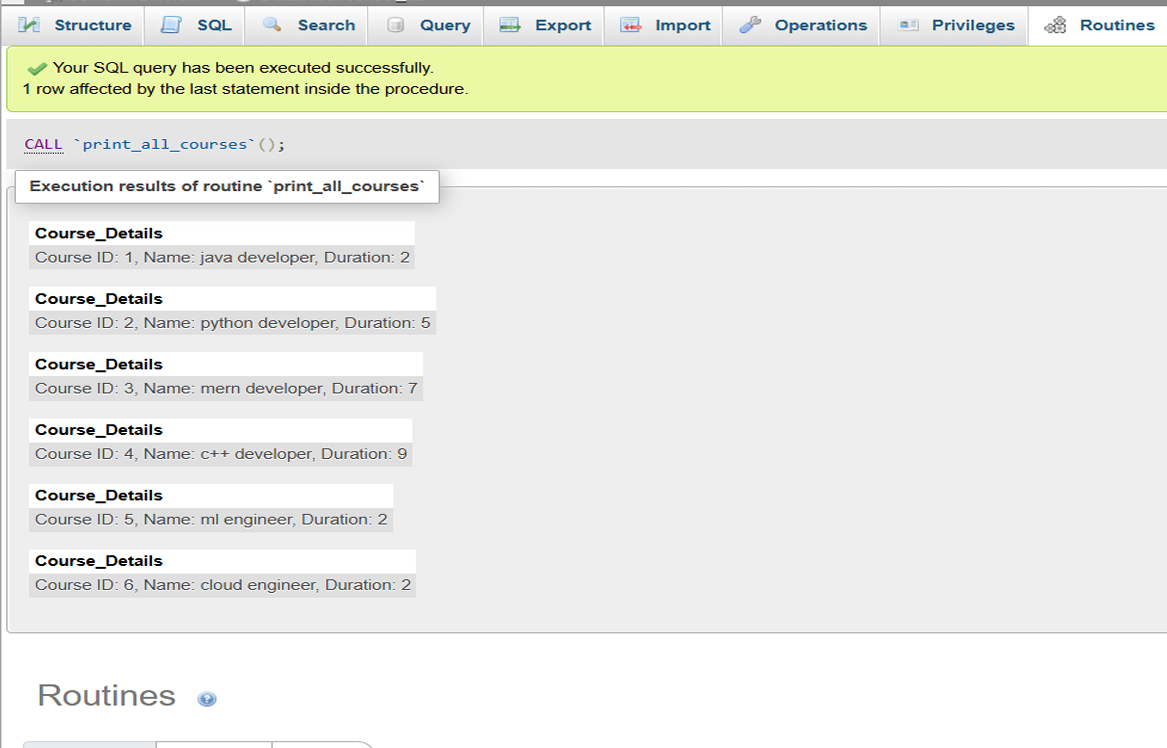
Course\_Details;

END LOOP;

CLOSE course\_cursor;

END //

DELIMITER ;



**19. Rollback and Commit Savepoint**

* Theory Questions :

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

* A SAVEPOINT is a feature in transaction management that allows you to set a marker or checkpoint within a transaction.
* It enables you to partially roll back a transaction to a specific point, instead of rolling back the entire transaction.
* ROLLBACK with SAVEPOINT

If a transaction encounters an error or a condition that requires undoing changes, you can use ROLLBACK TO SAVEPOINT to revert the transaction to the state at the savepoint.

Changes made after the savepoint are undone, but the changes made before the savepoint remain intact.

* COMMIT after SAVEPOINT

When you issue a COMMIT, all changes made in the transaction (including those after a savepoint) are finalized and made permanent in the database.

Savepoints are ignored after the commit, as the transaction ends.

* Multiple Savepoints

You can define multiple savepoints within a transaction and selectively roll back to any one of them.

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

* Savepoints are particularly useful in database transactions for scenarios requiring fine-grained control over which parts of a transaction to commit or roll back.
* Partial Rollback for Error Handling

Inserting multiple rows into different tables, and encountering a constraint violation or other error partway through.

* Complex Business Logic

An e-commerce transaction involving order placement, inventory update, and payment processing.

* Long-Running Transactions

A batch update process involving hundreds of rows.

* Conditional Logic in Transactions

Performing a series of updates, and based on subsequent checks, deciding whether to proceed, roll back partially, or retry specific steps.

* Multi-Step Processes

Data migration, where several tables are updated.

* Development and Debugging

During the development of database scripts or applications, savepoints are useful for testing transaction behavior.

* LAB EXERCISES :
* **Lab 1:** Perform a transaction where you create a savepoint, insert records, then rollback to the savepoint.

START TRANSACTION;

INSERT INTO employees (employee\_id,employee\_name, department\_id) VALUES (1, 'Alice', 1);

INSERT INTO employees(employee\_id,employee\_name, department\_id) VALUES (2, 'Bob', 2);

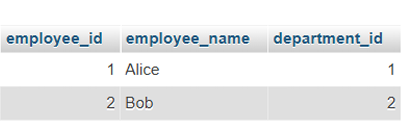
SAVEPOINT before\_addition;

INSERT INTO employees(employee\_id,employee\_name, department\_id) VALUES (3, 'Charlie',1);

INSERT INTO employees(employee\_id,employee\_name, department\_id) VALUES (4, 'David', 3);

ROLLBACK TO SAVEPOINT before\_addition;

COMMIT;



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

START TRANSACTION;

INSERT INTO employees (employee\_id,employee\_name, department\_id) VALUES (1, 'Abhay',5);

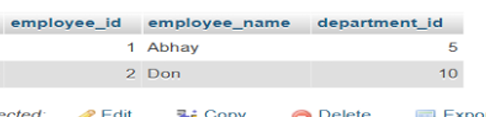
INSERT INTO employees(employee\_id,employee\_name, department\_id) VALUES (2, 'Don', 10);

SAVEPOINT savepoint1;

INSERT INTO employees(employee\_id,employee\_name, department\_id) VALUES (3, 'Che', 15);

ROLLBACK TO savepoint1;

COMMIT;



**EXTRA LAB PRACTISE FOR DATABASE CONCEPTS :**

1. **Introduction to SQL**

* **Lab 3:** Create a database called library\_db and a table books with columns: book\_id, title, author, publisher, year\_of\_publication, and price. Insert five records into the table.

CREATE DATABASE library\_db;

CREATE TABLE books(

book\_id int,

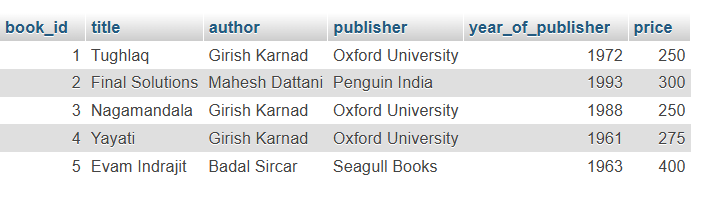
title varchar(20),

author varchar(20),

publisher varchar(20),

year\_of\_publisher int,

price int );



* **Lab 4:** Create a table members in library\_db with columns: member\_id, member\_name, date\_of\_membership, and email. Insert five records into this table.

CREATE TABLE members(

member\_id int,

member\_name varchar(20),

date\_of\_membership date,

email varchar(30)

);

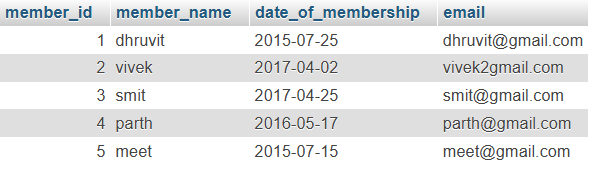
INSERT INTO members VALUES (1,'dhruvit','2015-7','dhruvit@gmail.com'),

(2,'vivek','2017-4-2','vivek2gmail.com'),

(3,'smit','2017-4-25','smit@gmail.com'),

(4,'parth','2016-5-17','parth@gmail.com'),

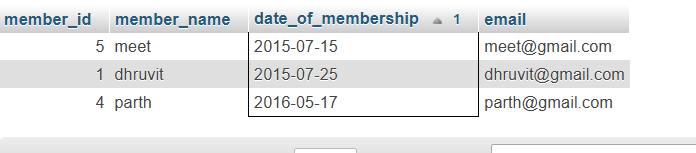
(5,'meet','2015-7-15','meet@gmail.com');



1. **SQL Syntax**

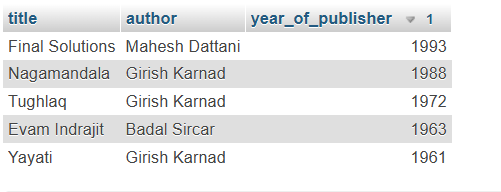
* **Lab 3:** Retrieve all members who joined the library before 2022. Use appropriate SQL syntax with WHERE and ORDER BY.

SELECT \* FROM members WHERE date\_of\_membership<'2017-01-01' ORDER BY date\_of\_membership;



* **Lab 4:** Write SQL queries to display the titles of books published by a specific author. Sort the results by year\_of\_publication in descending order.

SELECT title,author,year\_of\_publisher from books ORDER BY year\_of\_publisher DESC;



1. **SQL Constraints**

* **Lab 3:** Add a CHECK constraint to ensure that the price of books in the books table is greater than 0.

CREATE TABLE book (

book\_id int PRIMARY KEY,

book\_name varchar(10),

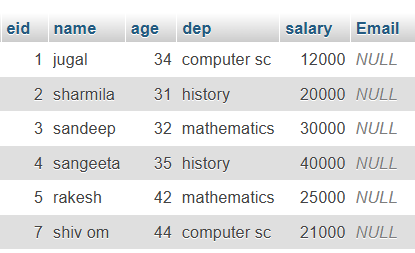
price int CHECK (price>0)

);



* **Lab 4:** Modify the members table to add a UNIQUE constraint on the email column, ensuring that each member has a unique email address.

ALTER TABLE emp ADD email varchar(20) UNIQUE;



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

* **Lab 3:** Create a table authors with the following columns: author\_id, first\_name, last\_name, and country. Set author\_id as the primary key.

CREATE TABLE authors (

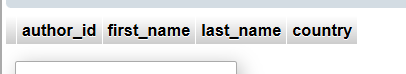
author\_id int PRIMARY KEY,

first\_name varchar(10),

last\_name varchar(10),

country varchar(10)

);



* **Lab 4:** Create a table publishers with columns: publisher\_id, publisher\_name, contact\_number, and address. Set publisher\_id as the primary key and contact\_number as unique.

CREATE TABLE publishers (

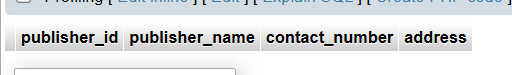
publisher\_id int PRIMARY KEY,

publisher\_name varchar(20),

contact\_number int UNIQUE,

address varchar(30)

);



1. **ALTER Command**

* **Lab 3:** Add a new column genre to the books table.Update the genre for all existing records.

ALTER TABLE books ADD genre varchar(10);



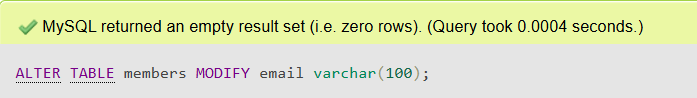
UPDATE books set genre = "History" WHERE book\_id IN(1,3,5);

UPDATE books set genre = "Science Fiction" WHERE book\_id IN(2,4);



* **Lab 4:** Modify the members table to increase the length of the email column to 100 characters.

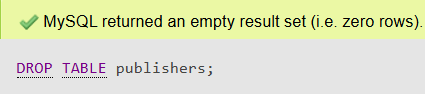
ALTER TABLE members MODIFY email varchar(100);



1. **DROP Command**

* **Lab 3:** Drop the publishers table from the database after verifying its structure.

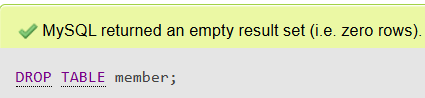
DROP TABLE publishers;



* **Lab 4:** Create a backup of the members table and then drop the original members table.

CREATE TABLE member AS SELECT \* FROM members;

DROP TABLE member;

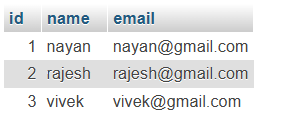
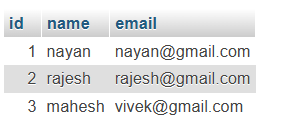


1. **Data ManipulationLanguage(DML)**

* **Lab 4:** Insert three new authors into the authors table, then update the last name of one of the authors.

INSERT INTO author VALUES(1,'nayan','nayan@gmail.com'),(2,'rajesh','rajesh@gmail.com'),(3,'vivek','vivek@gmail.com');

UPDATE author set name='mahesh' WHERE id=3;

* **Lab 5:** Delete a book from the books table where the price is higher than $100.

DELETE FROM books WHERE price>280;



1. **UPDATE Command**

* **Lab 3:** Update the year\_of\_publication of a book with a specific book\_id.

UPDATE books SET year\_of\_publisher=2000 WHERE book\_id=1;



* **Lab 4:** Increase the price of all books published before 2015 by 10%.

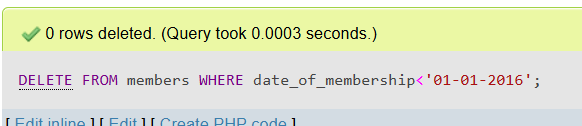
UPDATE books SET price = price\*10 WHERE year\_of\_publisher>1961;



1. **DELETE Command**

* **Lab 3:** Remove all members who joined before 2020 from the members table.

DELETE FROM members WHERE date\_of\_membership<'01-01-2016';



* **Lab 4:** Delete all books that have a NULL value in the author column.

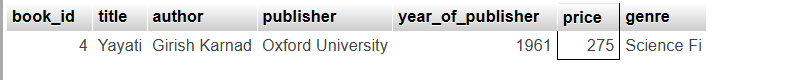
DELETE FROM books WHERE author='null';



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

* **Lab 4:** Write a query to retrieve all books with price between $50 and $100.

SELECT \* FROM books WHERE price BETWEEN 200 and 300;



* **Lab 5:** Retrieve the list of books sorted by author in ascending order and limit the results to the top 3 entries.

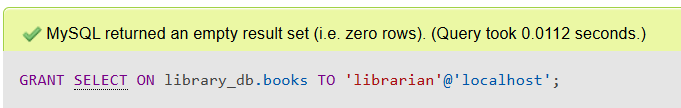
SELECT \* FROM books ORDER BY author ASC LIMIT 3;



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

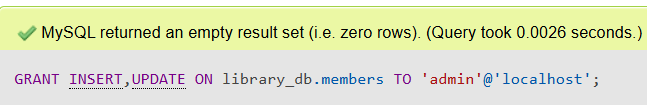
* **Lab 3:** Grant SELECT permission to a user named librarian on the books table.

GRANT SELECT ON library\_db.books TO 'librarian'@'localhost';



* **Lab 4:** Grant INSERT and UPDATE permissions to the user admin on the members table.

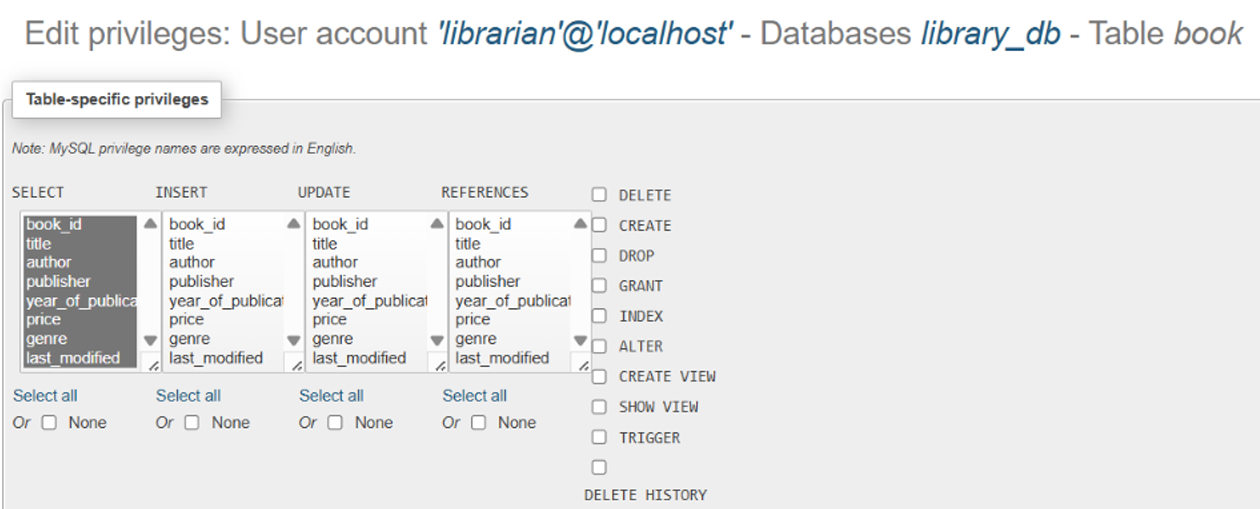
GRANT INSERT,UPDATE ON library\_db.members TO 'admin'@'localhost';



1. **REVOKE Command**

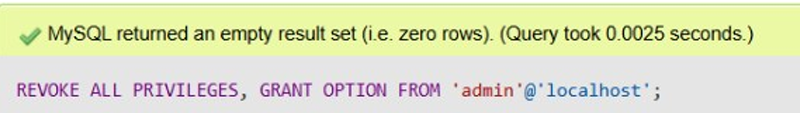
* **Lab 3:** Revoke the INSERT privilege from the user librarian on the books table.

REVOKE INSERT ON library\_db.book FROM 'librarian'@' localhost';



* **Lab 4:** Revoke all permissions from user admin on the members table.

REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'admin'@'localhost';



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

* **Lab 3:** Use COMMIT after inserting multiple records into the books table, then make another insertion and perform a ROLLBACK.

START TRANSACTION;

INSERT INTO book VALUES(4,"HarHarMAHADEV","yellow","ula",2024,4450,20);

COMMIT;

START TRANSACTION;

INSERT INTO book VALUES(5,"hajir moj","shivalay","lolaji",2018,980,80);

ROLLBACK;



* **Lab 4:** Set a SAVEPOINT before making updates to the members table, perform some updates, and then roll back to the SAVEPOINT.

START TRANSACTION;

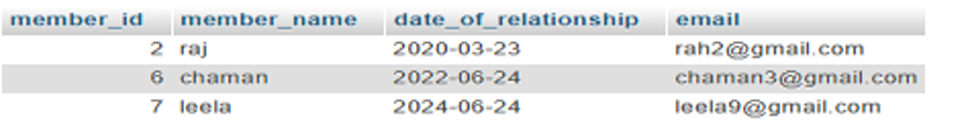
SAVEPOINT before\_update;

UPDATE members\_backup SET email = 'yela1@gmail.com' WHERE member\_id = 6;

UPDATE members\_backup SET email = 'loal2@gmail.com' WHERE member\_id = 7;

ROLLBACK TO SAVEPOINT before\_update;

COMMIT;



1. **SQL Joins**

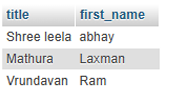
* **Lab 3:** Perform an INNER JOIN between books and authors tables to display the title of books and their respective authors' names.

SELECT book.title,authors.first\_name

FROM book

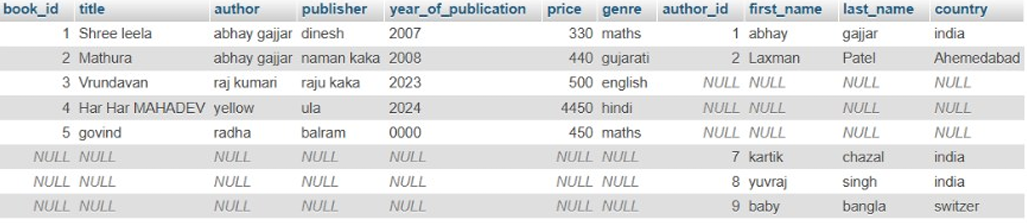
INNER JOIN authors

WHERE book.book\_id=authors.author\_id;



* **Lab 4:** Use a FULL OUTER JOIN to retrieve all records from the books and authors tables, including those with no matching entries in the other table.

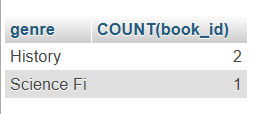
SELECT \* FROM book LEFT JOIN authors ON book.book \_id = authors.author\_id UNION SELECT \* FROM book RI GHT JOIN authors ON book.book\_id = authors.author\_id;



1. **SQL Group By**

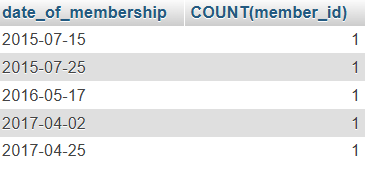
* **Lab 3:** Group books by genre and display the total number of books in each genre.

SELECT genre , COUNT(book\_id) FROM books GROUP BY genre;



* **Lab 4:** Group members by the year they joined and find the number of members who joined each year.

SELECT date\_of\_membership,COUNT(member\_id) FROM members GROUP BY date\_of\_membership;



1. **SQL Stored Procedure**

* **Lab 3:** Write a stored procedure to retrieve all books by a particular author.

DELIMITER //

CREATE PROCEDURE get\_books\_by\_author(IN author\_name VARCHAR(100))

BEGIN

SELECT \* FROM book WHERE author = author\_name;

END //

DELIMITER ;



* **Lab 4:** Write a stored procedure that takes book\_id as an argument and returns the price of the book.

DELIMITER //

CREATE PROCEDURE get\_book\_price(IN p\_book\_id INT,

OUT p\_price DECIMAL(10, 2))

BEGIN

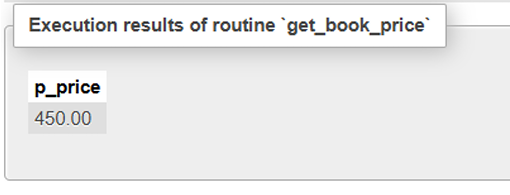
SELECT price INTO p\_price

FROM book

WHERE book\_id = p\_book\_id;

END //

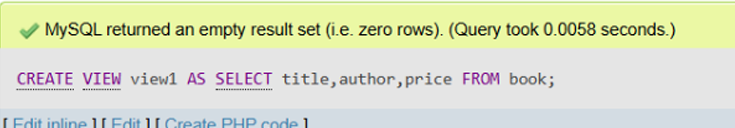
DELIMITER ;



1. **SQL View**

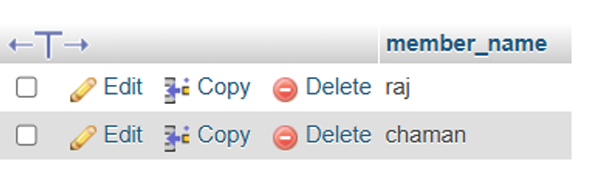
* **Lab 3:** Create a view to show only the title, author, and price of books from the books table.

CREATE VIEW view1 AS SELECT title,author,price FROM book;



* **Lab 4:** Create a view to display members who joined before 2020.

CREATE VIEW view2 AS SELECT member\_name FROM members\_backup WHERE date\_of\_relationship <= 2022;



1. **SQL Trigger**

* **Lab 3:** Create a trigger to automatically update the last\_modified timestamp of the books table whenever a record is updated.

ALTER TABLE book

ADD COLUMN last\_modified TIMESTAMP DEFAULTCURRENT\_TIMESTAMP ON UPDATE

CURRENT\_TIMESTAMP;

DELIMITER //

CREATE TRIGGER before\_books\_update

BEFORE UPDATE ON book

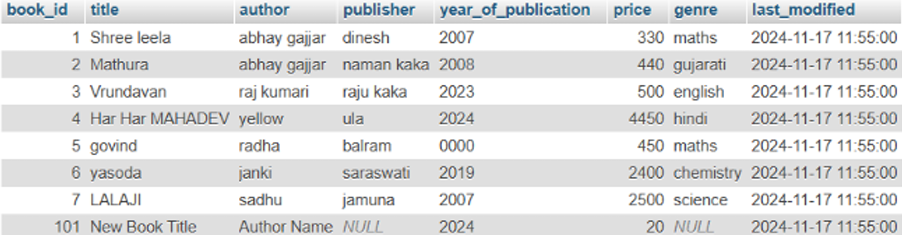
FOR EACH ROW

BEGIN

SET NEW.last\_modified = CURRENT\_TIMESTAMP;

END //

DELIMITER ;



* **Lab 4:** Create a trigger that inserts a log entry into a log\_changes table whenever a DELETE operation is performed on the books table.

CREATE TABLE IF NOT EXISTS log\_changes (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

book\_id INT, itle VARCHAR(100),

author VARCHAR(100),

deleted\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP);

DELIMITER //

CREATE TRIGGER after\_books\_delete

AFTER DELETE ON books FOR EACH ROW

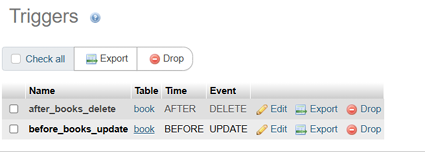
BEGIN

DEFAULT INSERT INTO log\_changes (book\_id, title, author,deleted\_at)

VALUES (OLD.book\_id, OLD.title,OLD.author, CURRENT\_TIMESTAMP);

END //

DELIMITER ;



1. **Introduction to PL/SQL**

* **Lab 3:** Write a PL/SQL block to insert a new book into the books table and display a confirmation message.

BEGIN

DECLARE v\_book\_id INT DEFAULT 101;

DECLARE v\_title VARCHAR(100) DEFAULT 'New BookTitle';

DECLARE v\_author VARCHAR(100) DEFAULT 'AuthorName';

DECLARE v\_price DECIMAL(10, 2) DEFAULT 19.99;

DECLARE v\_year\_of\_publication year DEFAULT 2024;

INSERT INTO book (book\_id, title, author, price,year\_of\_publication)

VALUES (v\_book\_id, v\_title, v\_author, v\_price,v\_year\_of\_publication);

SELECT CONCAT('New book "', v\_title, '" by ', v\_author,

' has been successfully added.') AS message;

END



* **Lab 4:** Write a PL/SQL block to display the total number of books in the books table.

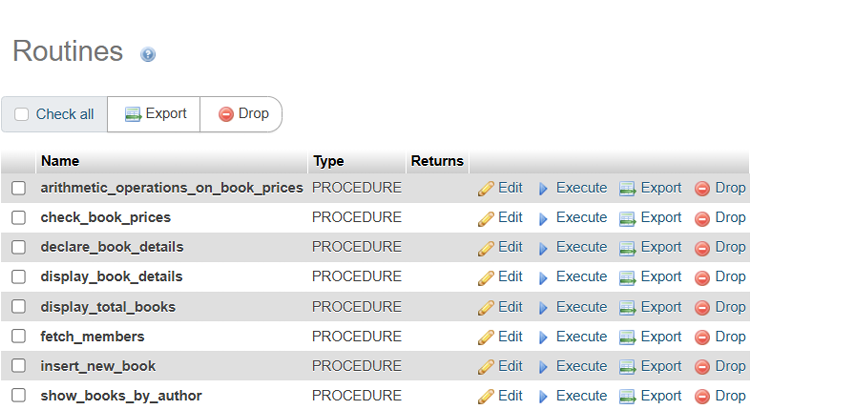
BEGIN

DECLARE v\_total\_books INT;

SELECT COUNT(book\_id) INTO v\_total\_books FROM book;

SELECT CONCAT('Total number of books: ',v\_total\_books) AS Total\_Books;

END



1. **PL/SQL Syntax**

* **Lab 3:** Write a PL/SQL block to declare variables for book\_id and price, assign values, and display the results.

DELIMITER //

CREATE PROCEDURE declare\_book\_details()

BEGIN

DECLARE v\_book\_id INT DEFAULT 101;

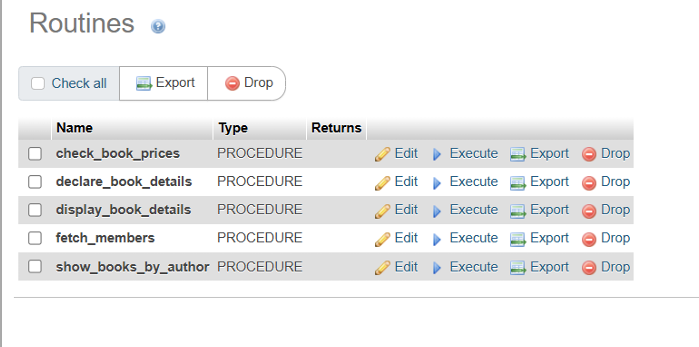
DECLARE v\_price DECIMAL(10, 2) DEFAULT 29.99;

SELECT CONCAT('Book ID: ', v\_book\_id) AS Book\_ID,

CONCAT('Price: $', v\_price) AS Price;

END //

DELIMITER ;



* **Lab 4:** Write a PL/SQL block using constants and perform arithmetic operations on book prices.

DELIMITER //

CREATE PROCEDURE arithmetic\_operations\_on\_book\_prices()

BEGIN

DECLARE c\_price1 DECIMAL(10, 2) DEFAULT 39.99;

DECLARE c\_price2 DECIMAL(10, 2) DEFAULT 49.99;

DECLARE c\_discount DECIMAL(5, 2) DEFAULT 0.10;

DECLARE v\_total\_price DECIMAL(10, 2);

DECLARE v\_discounted\_price1 DECIMAL(10, 2);

DECLARE v\_discounted\_price2 DECIMAL(10, 2);

SET v\_total\_price = c\_price1 + c\_price2;

SET v\_discounted\_price1 = c\_price1 - (c\_price1 \* c\_discount);

SET v\_discounted\_price2 = c\_price2 - (c\_price2 \* c\_discount);

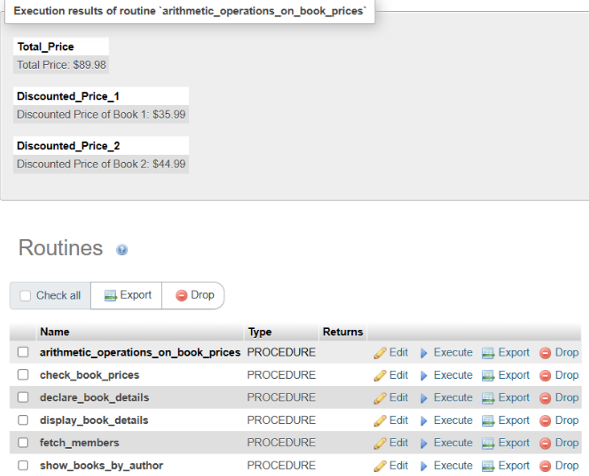
SELECT CONCAT('Total Price: $', v\_total\_price) AS Total\_Price;

SELECT CONCAT('Discounted Price of Book 1: $', v\_discounted\_price1) AS Discounted\_Price\_1;

SELECT CONCAT('Discounted Price of Book 2: $', v\_discounted\_price2) AS Discounted\_Price\_2;

END //

DELIMITER ;



1. **PL/SQL Control Structures**

* **Lab 3:** Write a PL/SQL block using IF-THEN-ELSE to check if a book's price is above $100 and print a message accordingly.

DELIMITER //

CREATE PROCEDURE check\_book\_prices()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE v\_title VARCHAR(100);

DECLARE v\_price DECIMAL(10, 2);

DECLARE book\_cursor CURSOR FOR SELECT title, price FROM book;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN book\_cursor;

read\_loop: LOOP

FETCH book\_cursor INTO v\_title, v\_price;

IF done THEN

LEAVE read\_loop;

END IF;

IF v\_price > 350 THEN

SELECT CONCAT('The price of "', v\_title, '" is

above 350.') AS message;

ELSE

SELECT CONCAT('The price of "', v\_title, '" is 350

or less.') AS message;

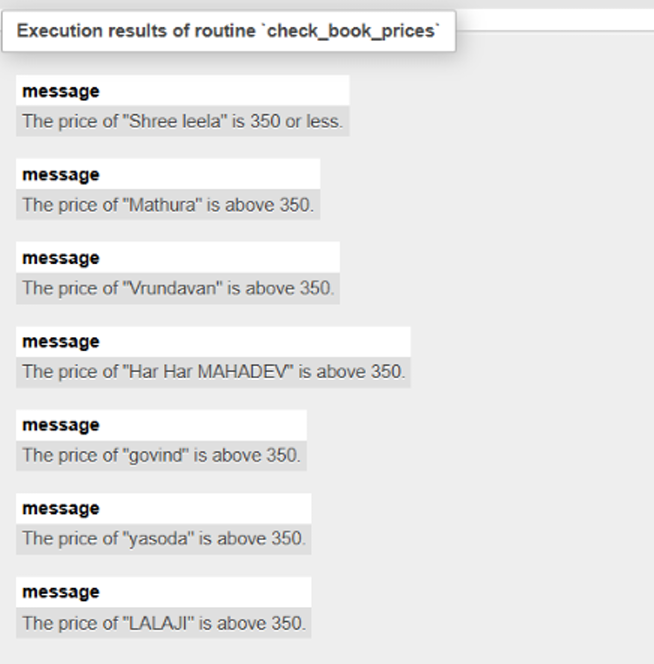
END IF;

END LOOP;

CLOSE book\_cursor;

END //

DELIMITER ;



* **Lab 4:** Use a FOR LOOP in PL/SQL to display the details of all books one by one.

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE v\_title VARCHAR(100);

DECLARE v\_author VARCHAR(100);

DECLARE v\_publisher VARCHAR(100);

DECLARE v\_year\_of\_publication INT;

DECLARE v\_price DECIMAL(10, 2);

DECLARE v\_genre VARCHAR(100);

DECLARE book\_cursor CURSOR FOR

SELECT title, author, publisher, year\_of\_publication,

price, genre FROM book;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN book\_cursor;

read\_loop: LOOP

FETCH book\_cursor INTO v\_title, v\_author,

v\_publisher, v\_year\_of\_publication, v\_price, v\_genre;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT CONCAT('Title: ', v\_title, ', Author: ',

v\_author, ', Publisher: ', v\_publisher, ', Year: ',

v\_year\_of\_publication, ', Price: $', v\_price, ', Genre: ',

v\_genre) AS book\_details;

END LOOP;

CLOSE book\_cursor;

END



1. **SQL Cursors**

* **Lab 3:** Write a PL/SQL block using an explicit cursor to fetch and display all records from the members table.

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE member\_id INT;

DECLARE member\_name VARCHAR(100);

DECLARE email VARCHAR(100);

DECLARE date\_of\_relationship DATE;

DECLARE member\_cursor CURSOR FOR

SELECT member\_id, member\_name, email,

date\_of\_relationship FROM members\_backup;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN member\_cursor;

read\_loop: LOOP

FETCH member\_cursor INTO member\_id,

member\_name, email, date\_of\_relationship;

IF done THEN

LEAVE read\_loop;

END IF;

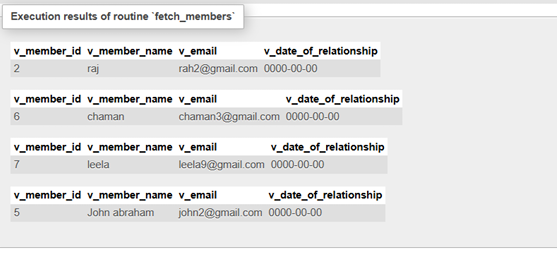
SELECT member\_id, member\_name, email,

date\_of\_relationship;

END LOOP;

CLOSE member\_cursor;

END



* **Lab 4:** Create a cursor to retrieve books by a particular author and display their titles.

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE book\_title VARCHAR(100);

DECLARE author\_name VARCHAR(100);

DECLARE book\_cursor CURSOR FOR

SELECT title, author FROM book WHERE author = 'abhay gajjar';

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN book\_cursor;

read\_loop: LOOP

FETCH book\_cursor INTO book\_title,

author\_name;

IF done THEN

LEAVE read\_loop;

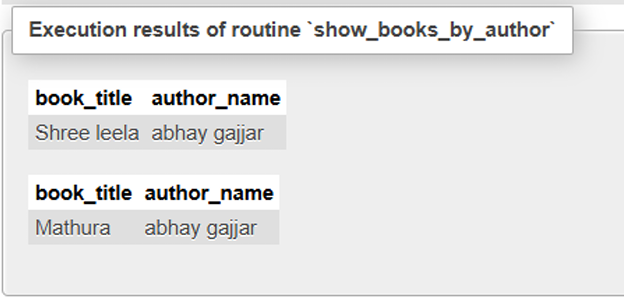
END IF;

SELECT book\_title, author\_name;

END LOOP;

CLOSE book\_cursor;

END



1. **Rollback and Commit Savepoint**

* **Lab 3:** Perform a transaction that includes inserting a new member, setting a SAVEPOINT, and rolling back to the savepoint after making updates.

START TRANSACTION;

INSERT INTO members\_backup (member\_id,

member\_name, email, date\_of\_relationship) VALUES

(5, 'John abraham', 'john2@gmail.com', 2025);

SAVEPOINT before\_update;

UPDATE members\_backup

SET email ='grant8@gmail.com'

WHERE member\_id = 1;

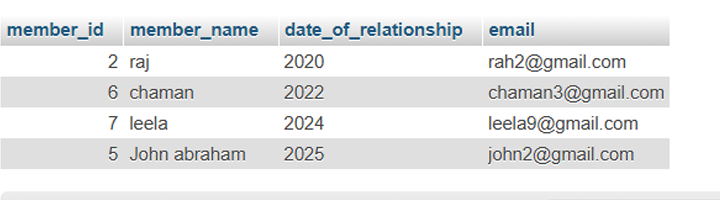
UPDATE members\_backup

SET email = 'ola9@gmail.com'

WHERE member\_id = 2;

ROLLBACK TO SAVEPOINT before\_update;

COMMIT;



* **Lab 4:** Use COMMIT after successfully inserting multiple books into the books table, then use ROLLBACK to undo a set of changes made after a savepoint.

START TRANSACTION;

INSERT INTO book

VALUES(6,"yasoda","janki","saraswati",2019,2400,60),(

7,"LALAJI","sadhu","jamuna",2007,2500,70);

COMMIT;

SAVEPOINT BEFORE\_update;

UPDATE book

SET price=300

WHERE book\_id=1;

UPDATE book

SET price=500

WHERE book\_id=2;

ROLLBACK TO SAVEPOINT BEFORE\_update;

COMMIT;

