Module 4

Introduction to DBMS

1. Introduction to SQL

- <u>Theory Questions :</u>

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

2. 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.

3. 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.

4. 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.

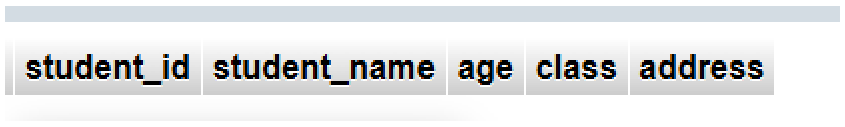
- User Access Control : SQL provides mechanisms to control user access to a database, including the ability to grant and revoke privileges to perform certain operations on the database.

- 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.

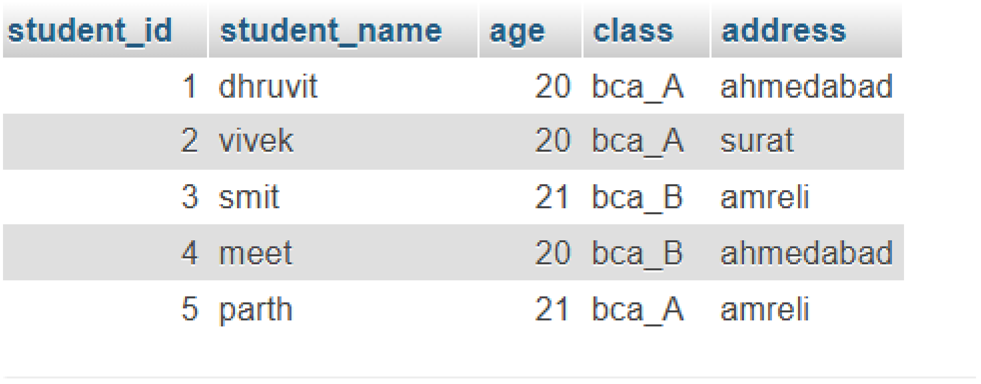
```
CREATE DATABASE school_db;

CREATE TABLE student (
   student_id int,
   student_name varchar(10),
   age int,
   class varchar(10),
   address varchar(50)
);
```



student_id	student_name	age	class	address

- **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;
```



student_id	student_name	age	class	address
1	dhruvit	20	bca_A	ahmedabad
2	vivek	20	bca_A	surat
3	smit	21	bca_B	amreli
4	meet	20	bca_B	ahmedabad
5	parth	21	bca_A	amreli

2. SQL Syntax

- <u>Theory Questions:</u>

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.

2. 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];

3. 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.

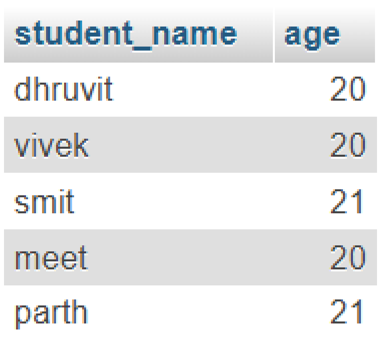
- ORDER BY Clause

  Specifies the sorting order of the result set.
  Columns are sorted in ascending order by default, but DESC can be used for descending order.

- <u>LAB EXERCISES:</u>

➢ **Lab1 :** Write SQL queries to retrieve specific columns (student_name and age) from the students table.

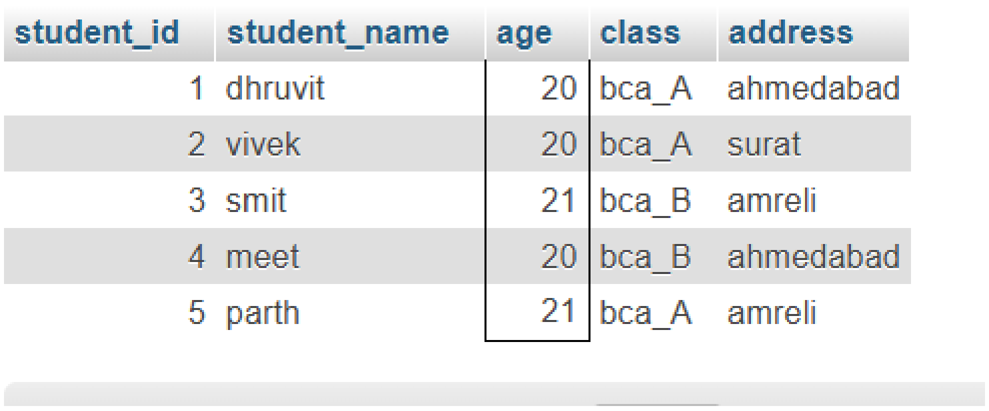
SELECT student_name,age from student;



student_name	age
dhruvit	20
vivek	20
smit	21
meet	20
parth	21

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

SELECT * FROM student WHERE age>10;



student_id	student_name	age	class	address
1	dhruvit	20	bca_A	ahmedabad
2	vivek	20	bca_A	surat
3	smit	21	bca_B	amreli
4	meet	20	bca_B	ahmedabad
5	parth	21	bca_A	amreli

3. SQL Constraints

- <u>Theory Questions:</u>

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
);
```

2. 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.

3. 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:

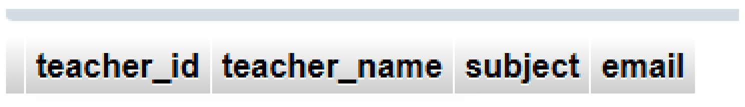
Prevents duplication of data in a column or set of columns.

Ensures data integrity and consistency.

- <u>LAB EXERCISES :</u>

➢ **Lab1 :** Create a table teachers with the following columns : teacher_id (PrimaryKey) , teacher_name (NOTNULL) , subject (NOTNULL) , and email(UNIUE).

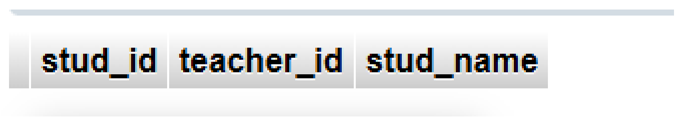
```
CREATE TABLE teachers (
   teacher_id int PRIMARY KEY,
   teacher_name varchar(10) NOT NULL,
   subject varchar(20) NOT NULL,
   email varchar(20) UNIQUE
);
```



teacher_id	teacher_name	subject	email

➢ **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)
);
```



stud_id	teacher_id	stud_name

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

- <u>Theory Questions :</u>

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.

2. 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);
```

3. 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.

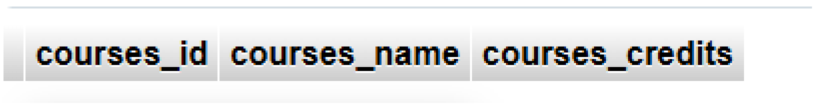
CHECK: Defines a specific condition that values in the column must meet, allowing complex validation.

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.

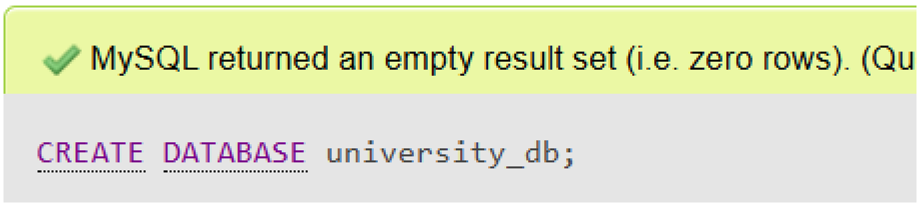
```
CREATE TABLE courses (
   courses_id int PRIMARY KEY,
   courses_name varchar(20),
   courses_credits int
 );
```



courses_id	courses_name	courses_credits

➢ **Lab2 :** Use the CREATE command to create database university_db.

```
CREATE DATABASE university_db;
```



```
✓ MySQL returned an empty result set (i.e. zero rows). (Qu

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.

2. 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

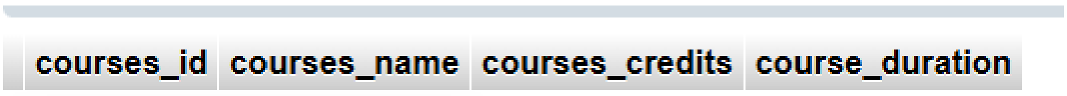
  You can remove a column from a table using the DROP COLUMN clause.

  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.

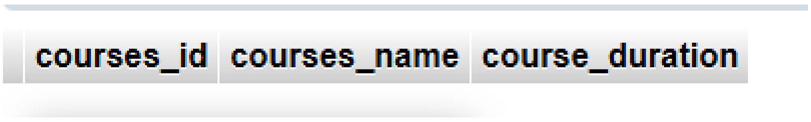
  ```
  ALTER TABLE courses ADD course_duration varchar(10);
  ```



courses_id	courses_name	courses_credits	course_duration

➢ **Lab2 :** Drop the course_credits column from the courses table.

```
ALTER TABLE courses DROP COLUMN courses_credits;
```



courses_id	courses_name	course_duration

6. DROP Command :

- <u>Theory Questions :</u>

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.

2. 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.

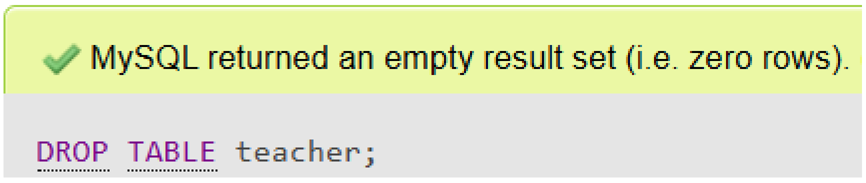
- No Undo Without Backup

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

- <u>LAB EXERCISES :</u>

- **Lab1 :** Drop the teachers table from the school_db database.

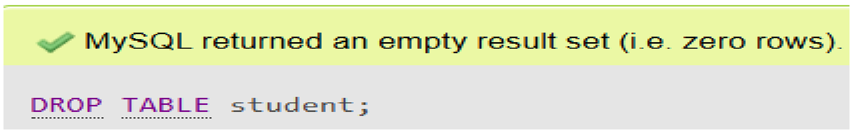
  DROP TABLE teacher;



```
✓ MySQL returned an empty result set (i.e. zero rows).
DROP TABLE teacher;
```

- **Lab2 :** Drop the students table from the school_db database and verify that the table has been removed.

  DROP TABLE student;



```
✓ MySQL returned an empty result set (i.e. zero rows).
DROP TABLE student;
```

7. Data Manipulation Language (DML)

- <u>Theory Questions :</u>

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;
  ```

2. 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

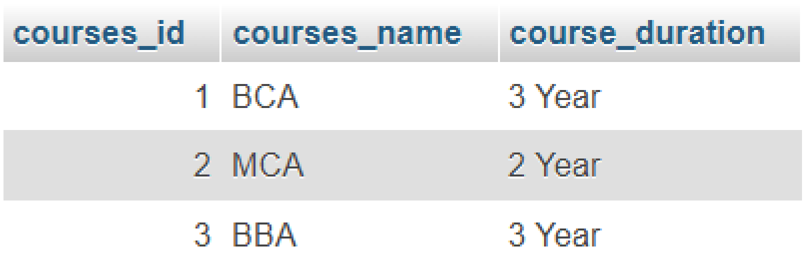
By targeting specific rows, the WHERE clause helps maintain the integrity of the remaining data in the table.

- 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.

- <u>LAB EXERCISES :</u>

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

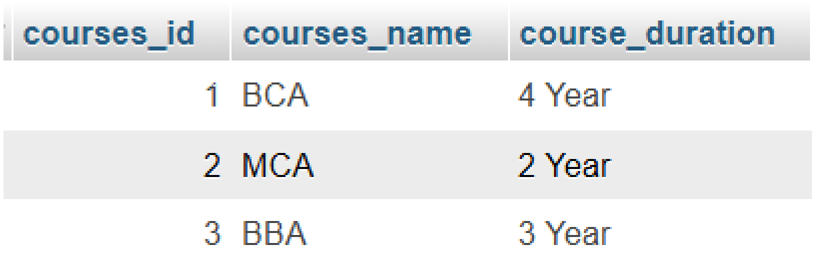
  INSERT INTO courses VALUES(1,"BCA","3 Year"),(2,"MCA","2 Year"),(3,"BBA","3 Year");



courses_id	courses_name	course_duration
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 .

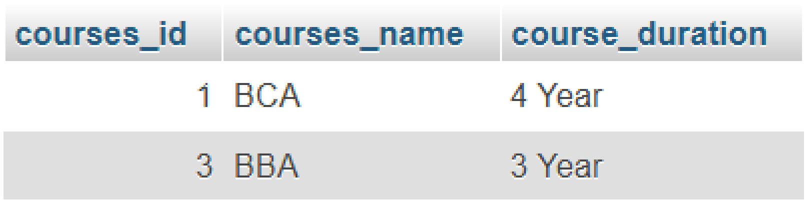
  UPDATE courses set course_duration='4 Year' WHERE courses_id=1;



courses_id	courses_name	course_duration
1	BCA	4 Year
2	MCA	2 Year
3	BBA	3 Year

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

  DELETE FROM courses WHERE courses_id=2;



courses_id	courses_name	course_duration
1	BCA	4 Year
3	BBA	3 Year

8. Data Query Language (DQL)

- <u>Theory Questions :</u>

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.

2. 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.

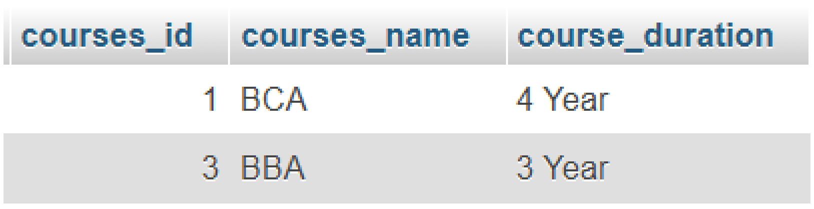
  Syntax:

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

- <u>LAB EXERCISES :</u>

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

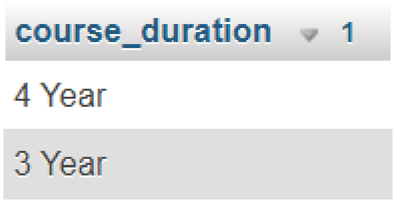
```
SELECT * from courses;
```



courses_id	courses_name	course_duration
1	BCA	4 Year
3	BBA	3 Year

➢ **Lab2 :** Sort the courses based on course_duration in descending order using ORDER BY .

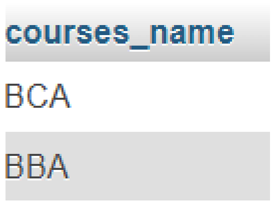
SELECT course_duration FROM courses ORDER BY course_duration DESC;



course_duration ▾ 1
4 Year
3 Year

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

SELECT courses_name FROM courses LIMIT 10;



courses_name
BCA
BBA

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.

2. 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;

- <u>LAB EXERCISES :</u>

➢ **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;

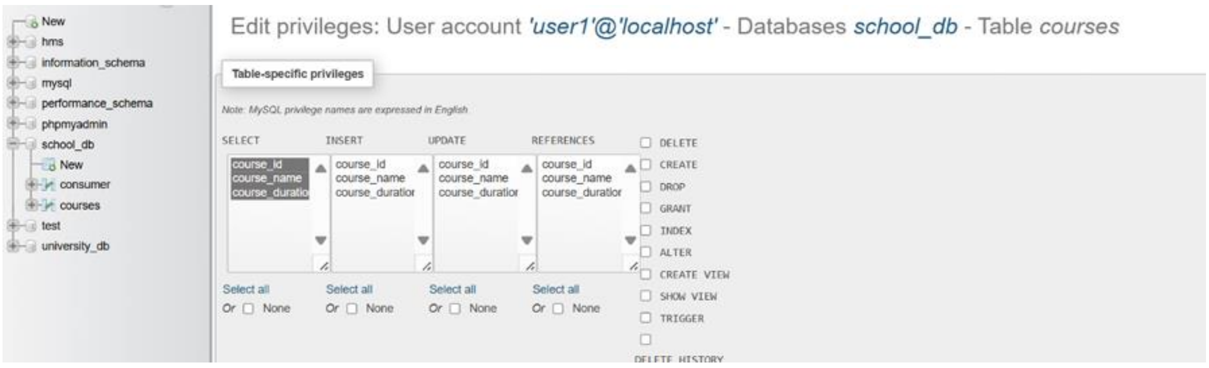

Edit privileges: User account *'user1'@'localhost'* - Databases *school_db* - Table *courses*

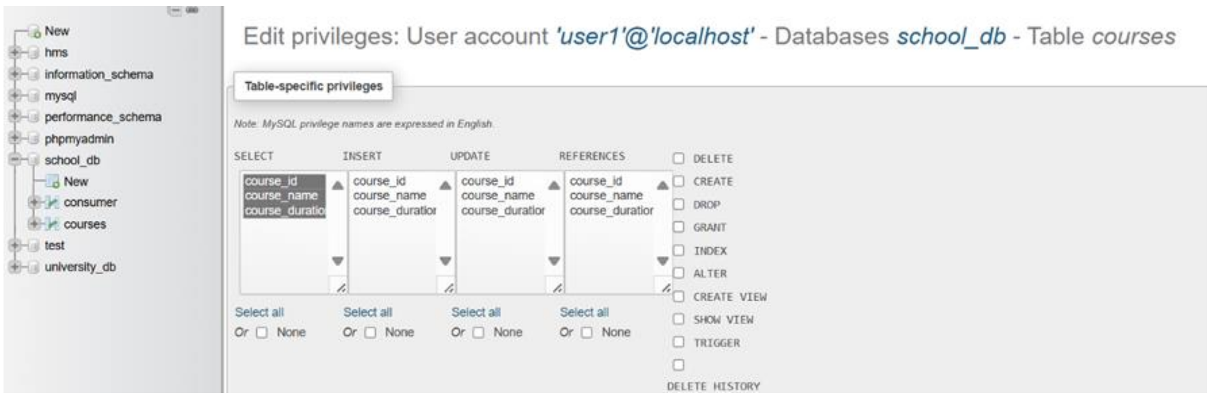
Table-specific privileges

Note: MySQL privilege names are expressed in English

SELECT: course_id, course_name, course_duratio (selected)
INSERT: course_id, course_name, course_duratior
UPDATE: course_id, course_name, course_duratior
REFERENCES: course_id, course_name, course_duratior

☐ DELETE
☐ CREATE
☐ DROP
☐ GRANT
☐ INDEX
☐ ALTER
☐ CREATE VIEW
☐ SHOW VIEW
☐ TRIGGER
☐ DELETE HISTORY

➢ **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';



Edit privileges: User account *'user1'@'localhost'* - Databases *school_db* - Table *courses*

Table-specific privileges

Note: MySQL privilege names are expressed in English.

SELECT	INSERT	UPDATE	REFERENCES
course_id	course_id	course_id	course_id
course_name	course_name	course_name	course_name
course_duratio	course_duratior	course_duratior	course_duratior
Select all	Select all	Select all	Select all
Or ☐ None	Or ☐ None	Or ☐ None	Or ☐ None

☐ DELETE
☐ CREATE
☐ DROP
☐ GRANT
☐ INDEX
☐ ALTER
☐ CREATE VIEW
☐ SHOW VIEW
☐ TRIGGER
☐ DELETE HISTORY

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.

2. 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).

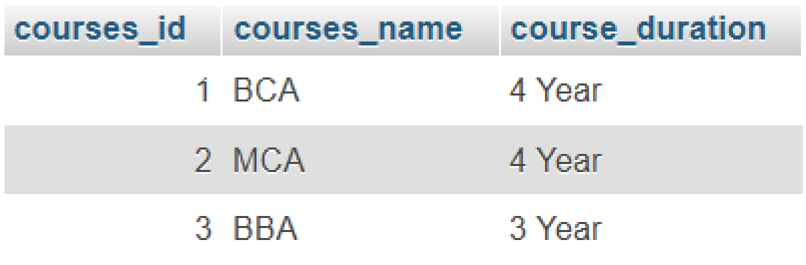
➢ Commit the Transaction:

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

- <u>LAB EXERCISES :</u>

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

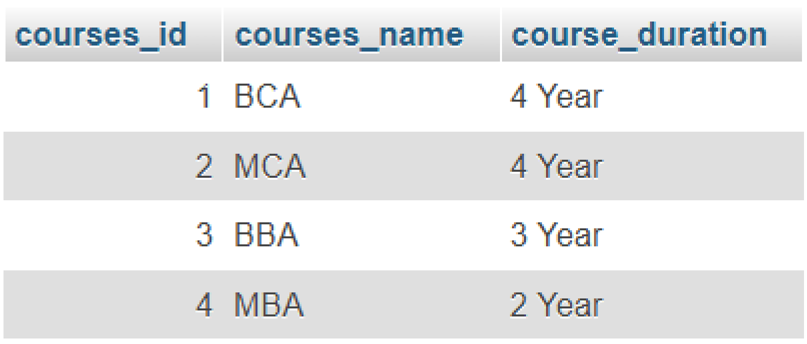
```
INSERT INTO courses VALUES(2,'MCA','4 Year');
COMMIT;
```



courses_id	courses_name	course_duration
1	BCA	4 Year
2	MCA	4 Year
3	BBA	3 Year

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

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



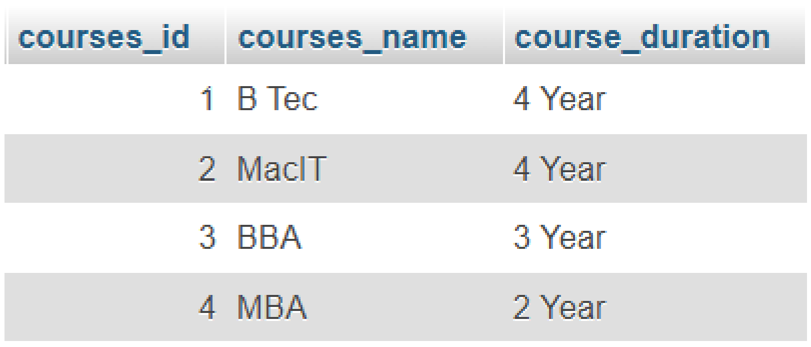
courses_id	courses_name	course_duration
1	BCA	4 Year
2	MCA	4 Year
3	BBA	3 Year
4	MBA	2 Year

➢ **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;



courses_id	courses_name	course_duration
1	B Tec	4 Year
2	MacIT	4 Year
3	BBA	3 Year
4	MBA	2 Year

11. SQL Joins

- <u>Theory Questions :</u>

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;
  ```

2. 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.

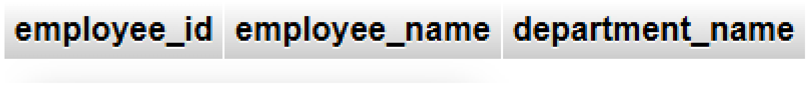
- Choose the Type of Join:

  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.

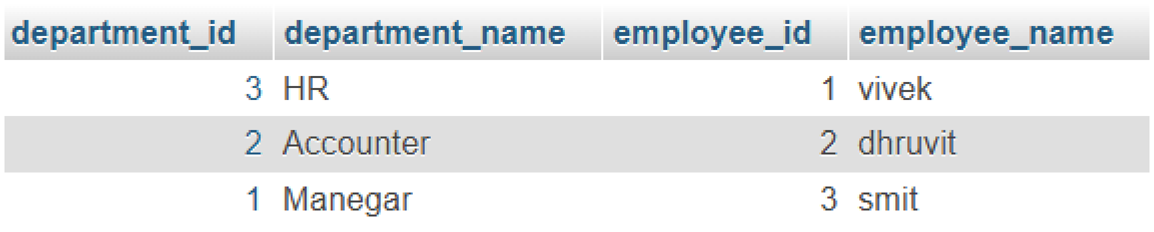
```
CREATE TABLE departments
(
department_id INT PRIMARY KEY,
department_name VARCHAR(50) NOT NULL
);

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.departmen
t_name FROM employees INNER JOIN departments ON
employees.department_id=departments.department_id;
```



employee_id	employee_name	department_name

- **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;
```



department_id	department_name	employee_id	employee_name
3	HR	1	vivek
2	Accounter	2	dhruvit
1	Manegar	3	smit

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.

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

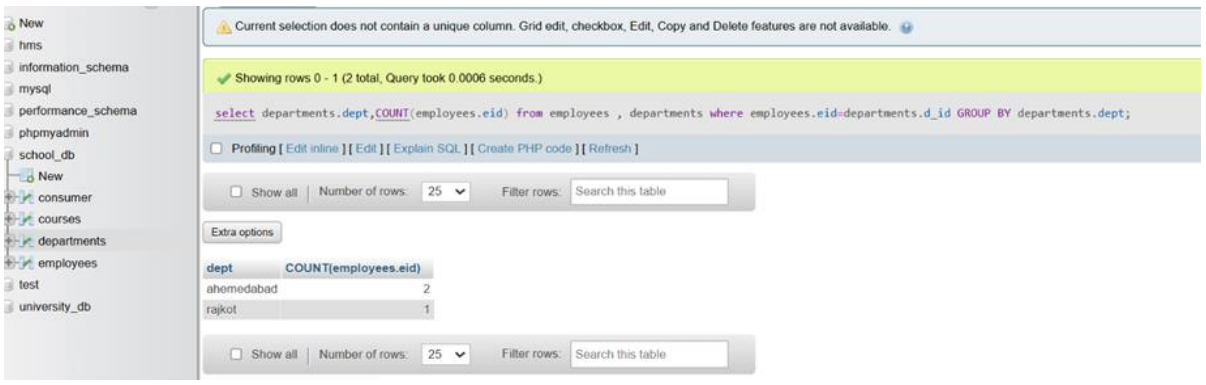
GROUP BY	ORDER BY
Groups rows into smaller sets.	No grouping; works on individual rows or final grouped rows.
Does not sort the rows.	Always sorts the rows in a specified order.
Required when using aggregate functions on grouped data.	Optional; used for sorting the results.

Affects the logical grouping of rows.	Affects only the display order of rows.

- <u>LAB EXERCISES :</u>

➢ **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;



Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

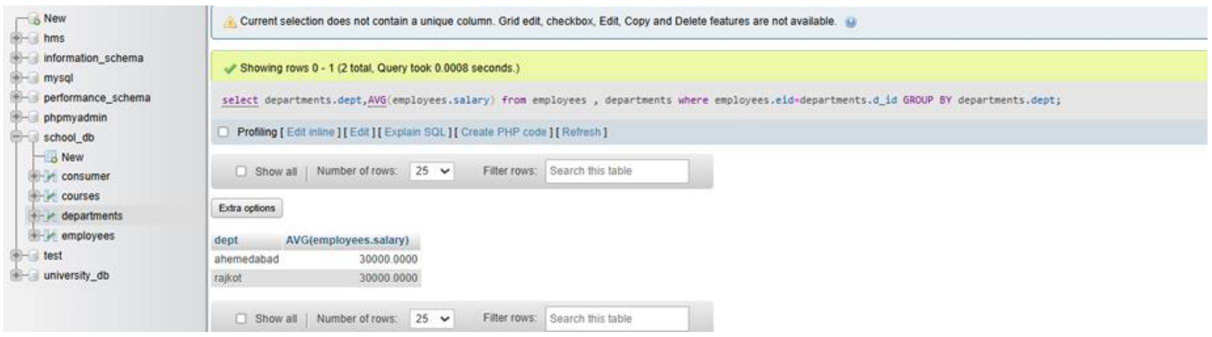
Showing rows 0 - 1 (2 total, Query took 0.0006 seconds.)

```
select departments.dept,COUNT(employees.eid) from employees , departments where employees.eid=departments.d_id GROUP BY departments.dept;
```

dept	COUNT(employees.eid)
ahemedabad	2
rajkot	1

➢ **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;



Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

Showing rows 0 - 1 (2 total, Query took 0.0008 seconds.)

```
select departments.dept,AVG(employees.salary) from employees , departments where employees.eid=departments.d_id GROUP BY departments.dept;
```

dept	AVG(employees.salary)
ahemedabad	30000.0000
rajkot	30000.0000

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.

2. 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

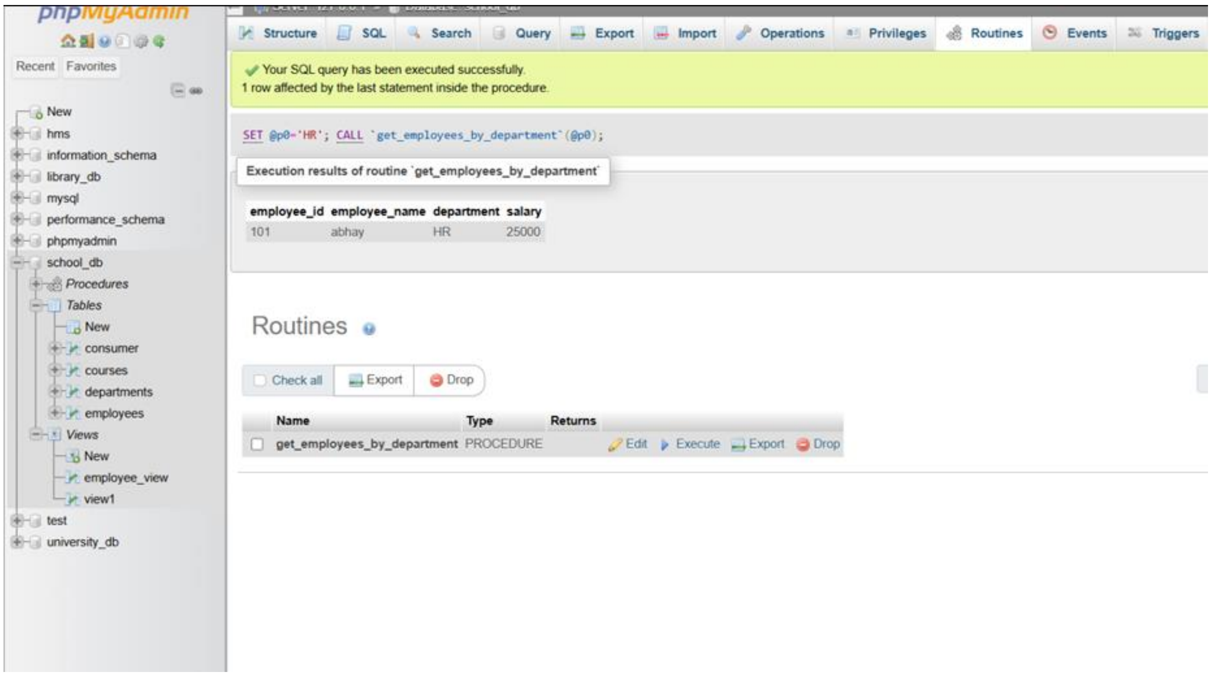
  Stored procedures can include robust error-handling mechanisms, such as TRY...CATCH blocks, to manage and log errors more effectively.

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 ;



Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

```
SET @p0='HR'; CALL `get_employees_by_department`(@p0);
```

Execution results of routine `get_employees_by_department`

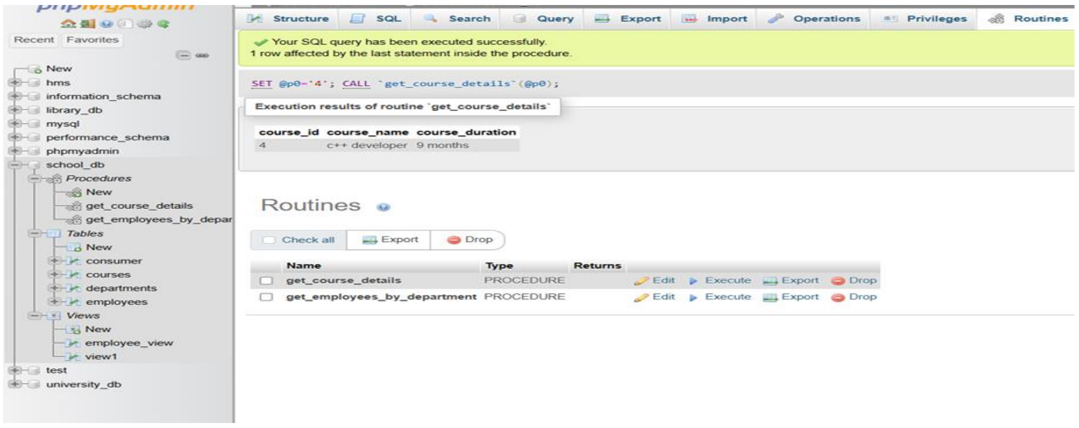
employee_id	employee_name	department	salary
101	abhay	HR	25000

Routines

Name	Type	Returns
get_employees_by_department	PROCEDURE	

- **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 ;
```



Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

```
SET @p0='4'; CALL `get_course_details`(@p0);
```

Execution results of routine `get_course_details`

course_id	course_name	course_duration
4	c++ developer	9 months

Routines

Name	Type	Returns
get_course_details	PROCEDURE	
get_employees_by_department	PROCEDURE	

14. SQL View

- <u>Theory Questions :</u>

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.

2. 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.

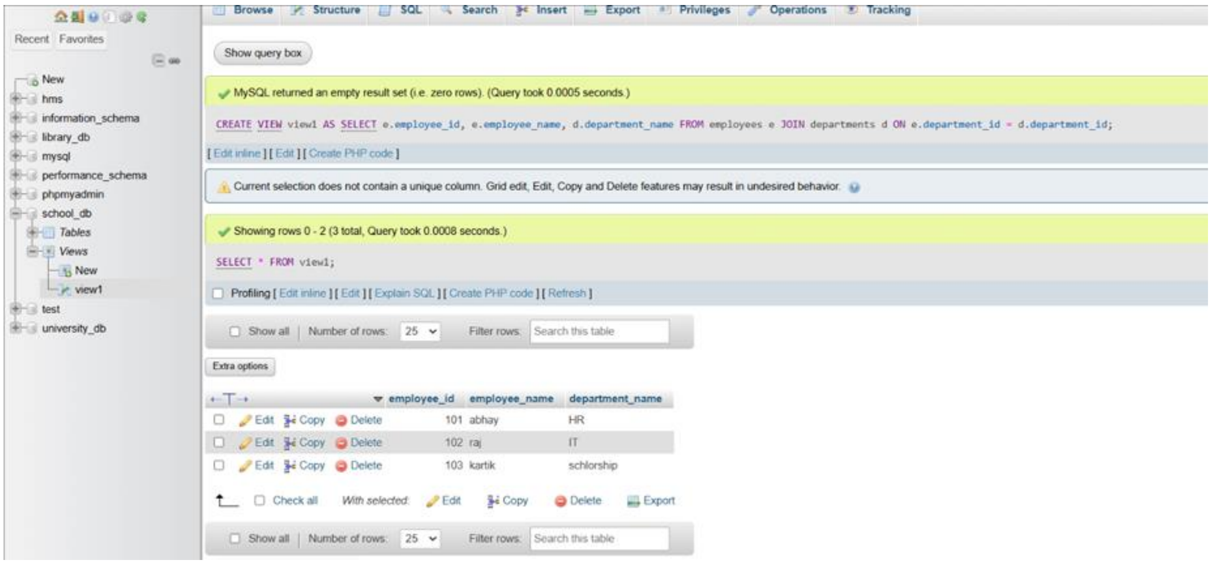
- Performance Optimization

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

- <u>LAB EXERCISES :</u>

➢ **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;



✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0005 seconds.)

```
CREATE VIEW view1 AS SELECT e.employee_id, e.employee_name, d.department_name FROM employees e JOIN departments d ON e.department_id = d.department_id;
```

Current selection does not contain a unique column. Grid edit, Edit, Copy and Delete features may result in undesired behavior.

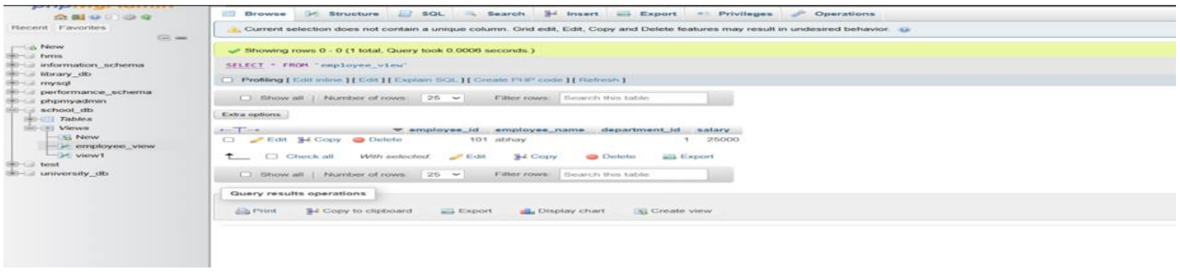
✓ Showing rows 0 - 2 (3 total, Query took 0.0008 seconds.)

```
SELECT * FROM view1;
```

employee_id	employee_name	department_name
101	abhay	HR
102	raj	IT
103	kartik	schlorship

➢ **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;

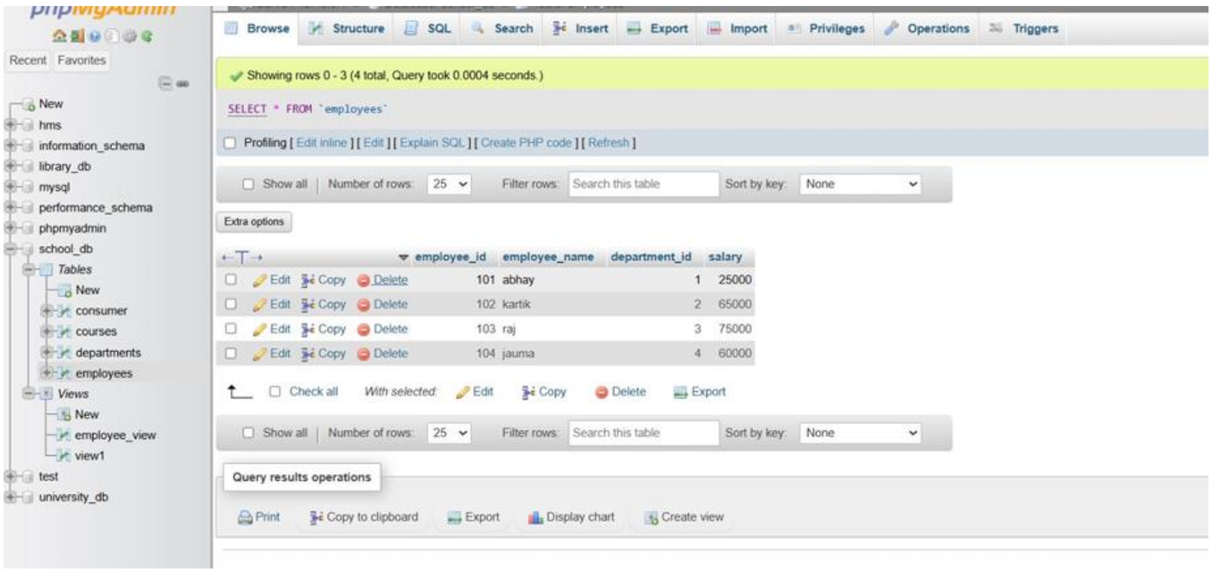


Current selection does not contain a unique column. Grid edit, Edit, Copy and Delete features may result in undesired behavior.

✓ Showing rows 0 - 0 (1 total, Query took 0.0006 seconds.)

```
SELECT * FROM `employee_view`
```

employee_id	employee_name	department_id	salary
101	abhay	1	25000



Showing rows 0 - 3 (4 total, Query took 0.0004 seconds.)

```
SELECT * FROM `employees`
```

employee_id	employee_name	department_id	salary
101	abhay	1	25000
102	kartik	2	65000
103	raj	3	75000
104	jauma	4	60000

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.

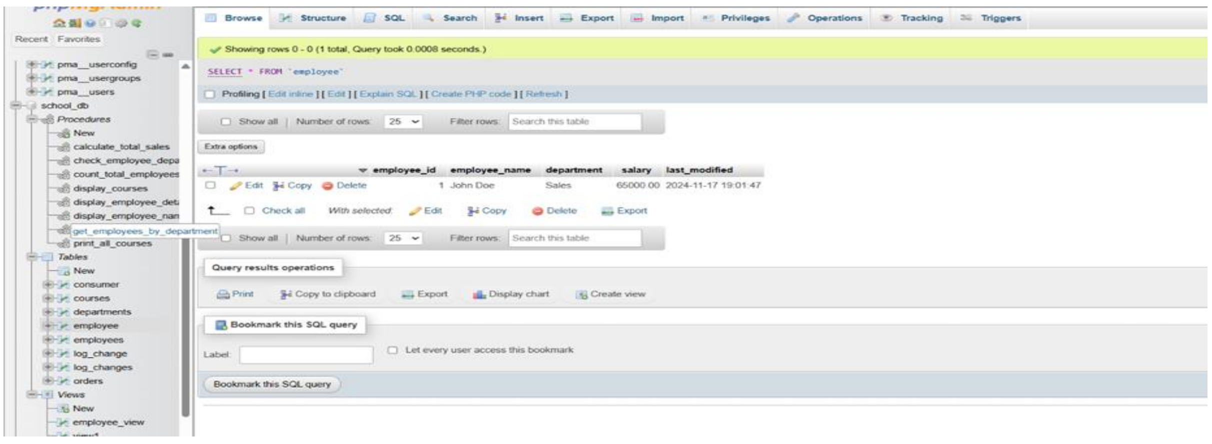
2. Explain the difference between INSERT, UPDATE, and DELETE triggers.

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.

- <u>LAB EXERCISES :</u>

➢ **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 ;
```



Showing rows 0 - 0 (1 total, Query took 0.0008 seconds.)

```
SELECT * FROM `employee`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 Filter rows: Search this table

Extra options

	employee_id	employee_name	department	salary	last_modified
Edit Copy Delete	1	John Doe	Sales	65000.00	2024-11-17 19:01:47

Check all With selected: Edit Copy Delete Export

Show all | Number of rows: 25 Filter rows: Search this table

Query results operations

Print Copy to clipboard Export Display chart Create view

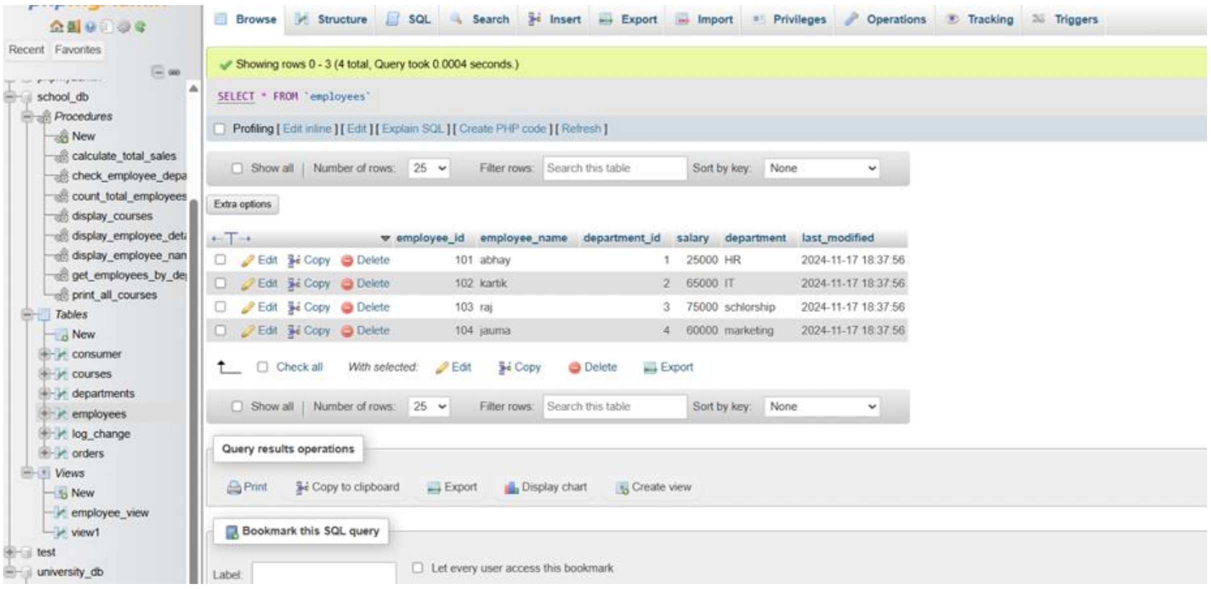
Bookmark this SQL query

Label: Let every user access this bookmark

Bookmark this SQL query

➢ **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 ;
```



✓ Showing rows 0 - 3 (4 total, Query took 0.0004 seconds.)

```
SELECT * FROM `employees`
```

employee_id	employee_name	department_id	salary	department	last_modified
101	abhay	1	25000	HR	2024-11-17 18:37:56
102	kartik	2	65000	IT	2024-11-17 18:37:56
103	raj	3	75000	schlorship	2024-11-17 18:37:56
104	jauma	4	60000	marketing	2024-11-17 18:37:56

16. Introduction to PL/SQL

• <u>Theory Questions :</u>

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;
```

2. 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.

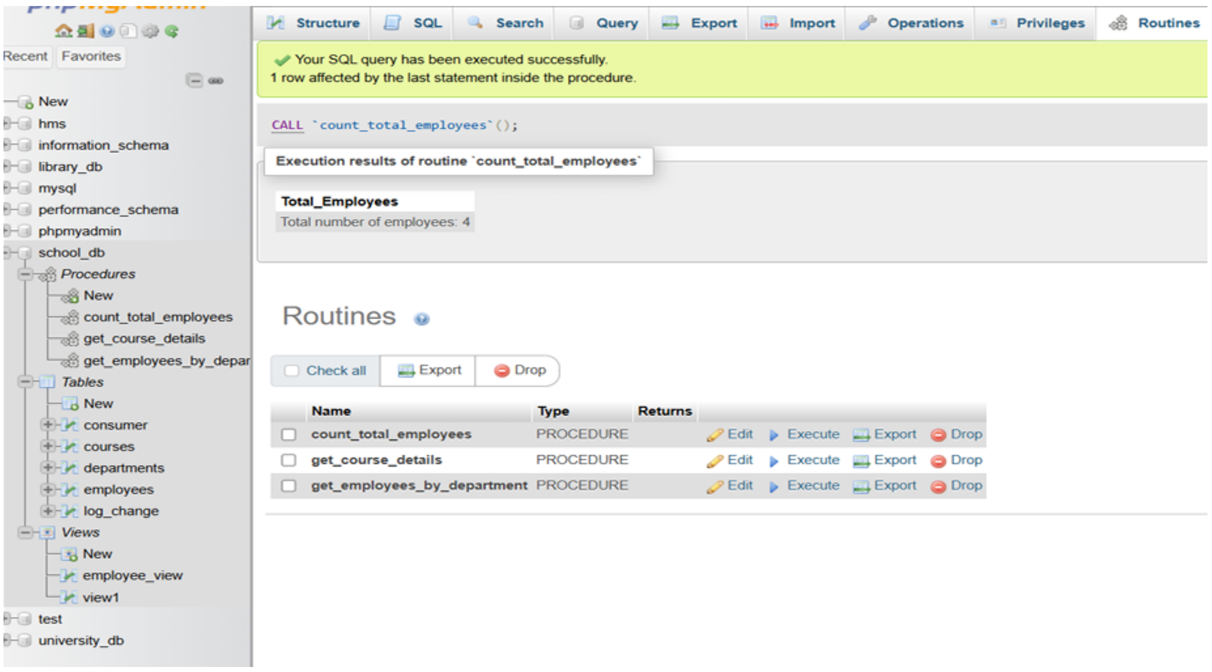
- 5. Robust Error Handling

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

- <u>LAB EXERCISES :</u>

➢ **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 ;
```



✓ Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

```
CALL `count_total_employees`();
```

Execution results of routine `count_total_employees`

Total_Employees
Total number of employees: 4

Routines

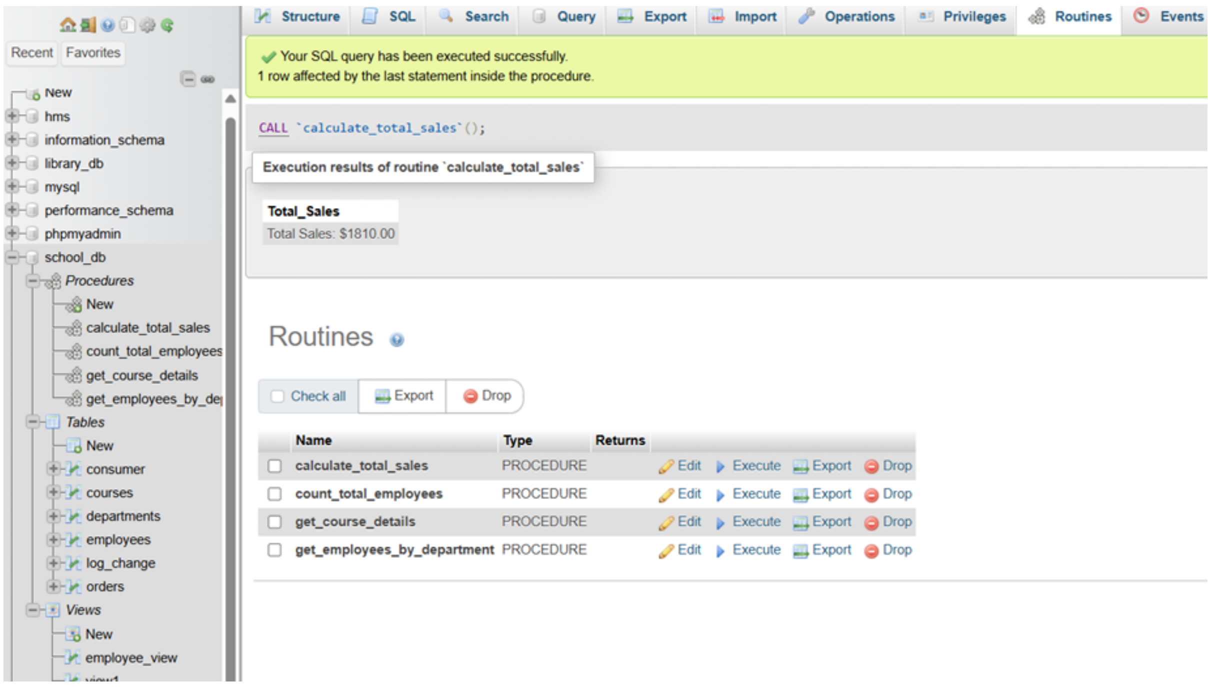
Check all | Export | Drop

	Name	Type	Returns				
☐	count_total_employees	PROCEDURE		Edit	Execute	Export	Drop
☐	get_course_details	PROCEDURE		Edit	Execute	Export	Drop
☐	get_employees_by_department	PROCEDURE		Edit	Execute	Export	Drop

➢ **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 ;
```



Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

```
CALL `calculate_total_sales`();
```

Execution results of routine `calculate_total_sales`

Total_Sales
Total Sales: $1810.00

Routines

Check all | Export | Drop

	Name	Type	Returns	
☐	calculate_total_sales	PROCEDURE		Edit Execute Export Drop
☐	count_total_employees	PROCEDURE		Edit Execute Export Drop
☐	get_course_details	PROCEDURE		Edit Execute Export Drop
☐	get_employees_by_department	PROCEDURE		Edit Execute Export Drop

17. PL/SQL Control Structures

- <u>Theory Questions :</u>

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;
  ```

2. 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.

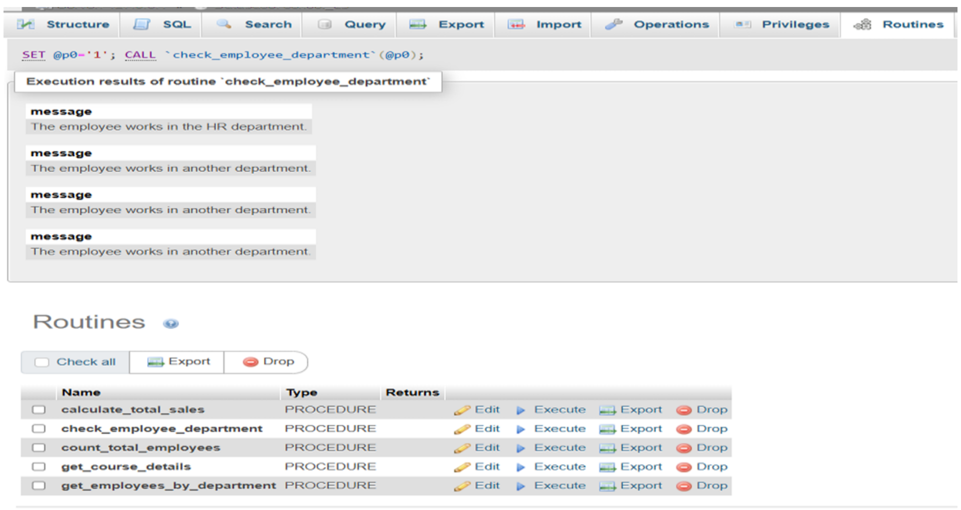
- 4. Combining SQL and Procedural Logic
  PL/SQL enables embedding procedural logic in SQL operations.

- 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.

- <u>LAB EXERCISES :</u>

➢ **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 ;
```



```
SET @p0='1'; CALL `check_employee_department`(@p0);
```

Execution results of routine `check_employee_department`

```
message
The employee works in the HR department.

message
The employee works in another department.

message
The employee works in another department.

message
The employee works in another department.
```

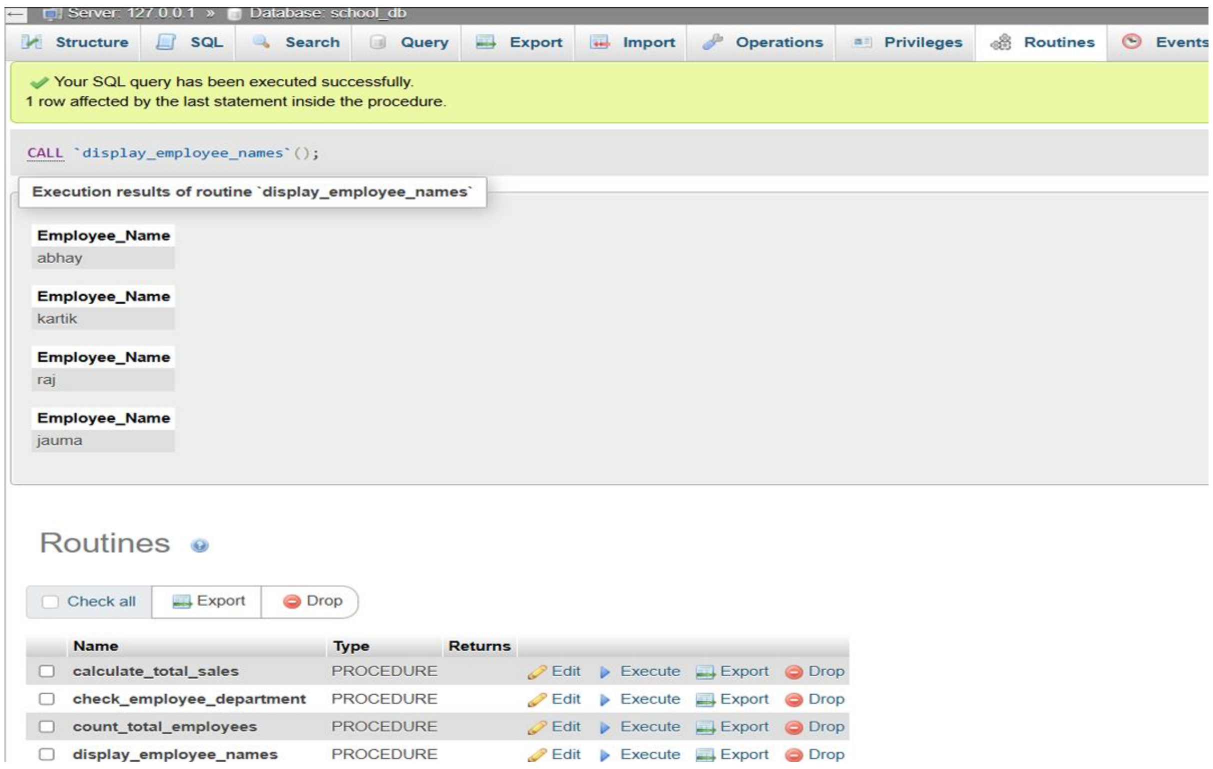
Routines

	Name	Type	Returns				
☐	calculate_total_sales	PROCEDURE		Edit	Execute	Export	Drop
☐	check_employee_department	PROCEDURE		Edit	Execute	Export	Drop
☐	count_total_employees	PROCEDURE		Edit	Execute	Export	Drop
☐	get_course_details	PROCEDURE		Edit	Execute	Export	Drop
☐	get_employees_by_department	PROCEDURE		Edit	Execute	Export	Drop

- **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 ;
```



Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

```
CALL `display_employee_names`();
```

Execution results of routine `display_employee_names`

Employee_Name
abhay

Employee_Name
kartik

Employee_Name
raj

Employee_Name
jauma

Routines

	Name	Type	Returns				
☐	calculate_total_sales	PROCEDURE		Edit	Execute	Export	Drop
☐	check_employee_department	PROCEDURE		Edit	Execute	Export	Drop
☐	count_total_employees	PROCEDURE		Edit	Execute	Export	Drop
☐	display_employee_names	PROCEDURE		Edit	Execute	Export	Drop

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.

2. 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.

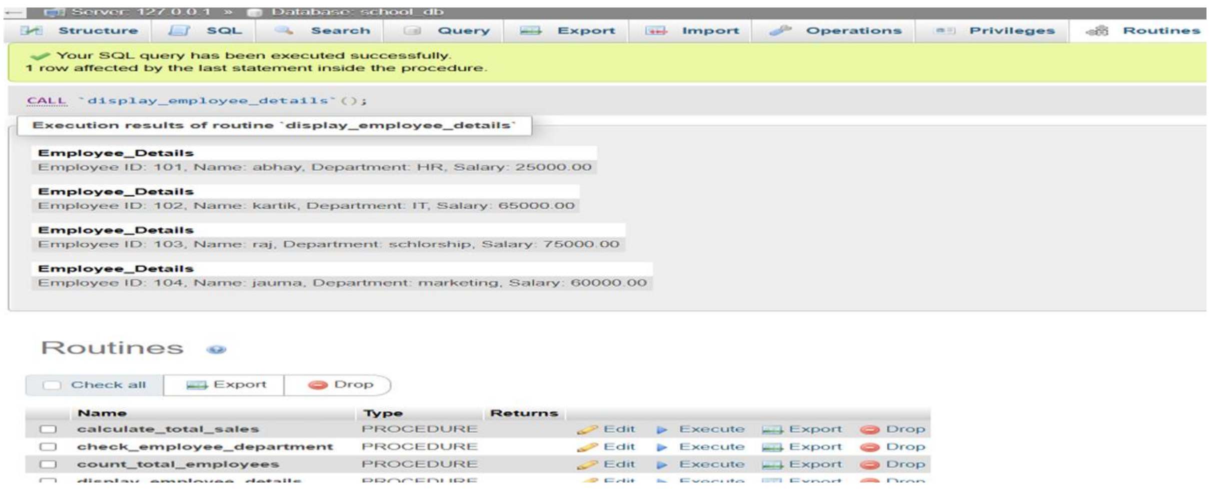
- When a Query is Repeatedly Used

  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 ;
```



```
Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

CALL `display_employee_details`();

Execution results of routine `display_employee_details`

Employee_Details
Employee ID: 101, Name: abhay, Department: HR, Salary: 25000.00

Employee_Details
Employee ID: 102, Name: kartik, Department: IT, Salary: 65000.00

Employee_Details
Employee ID: 103, Name: raj, Department: schlorship, Salary: 75000.00

Employee_Details
Employee ID: 104, Name: jauma, Department: marketing, Salary: 60000.00
```

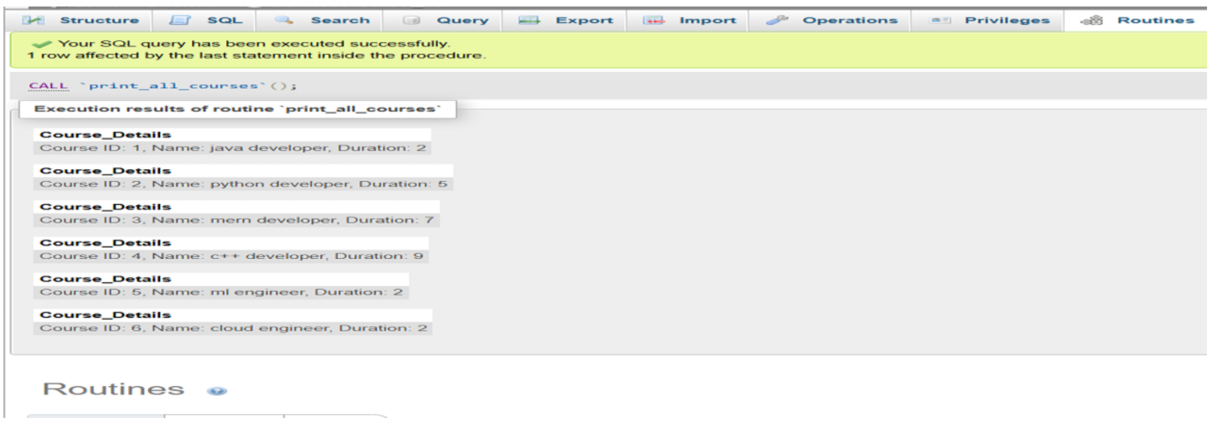
Routines

Check all | Export | Drop

	Name	Type	Returns				
☐	calculate_total_sales	PROCEDURE		Edit	Execute	Export	Drop
☐	check_employee_department	PROCEDURE		Edit	Execute	Export	Drop
☐	count_total_employees	PROCEDURE		Edit	Execute	Export	Drop
☐	display_employee_details	PROCEDURE		Edit	Execute	Export	Drop

➢ **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 ;
```



```
Your SQL query has been executed successfully.
1 row affected by the last statement inside the procedure.

CALL `print_all_courses`();

Execution results of routine `print_all_courses`

Course_Details
Course ID: 1, Name: java developer, Duration: 2
Course_Details
Course ID: 2, Name: python developer, Duration: 5
Course_Details
Course ID: 3, Name: mern developer, Duration: 7
Course_Details
Course ID: 4, Name: c++ developer, Duration: 9
Course_Details
Course ID: 5, Name: ml engineer, Duration: 2
Course_Details
Course ID: 6, Name: cloud engineer, Duration: 2
```

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.

2. 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.

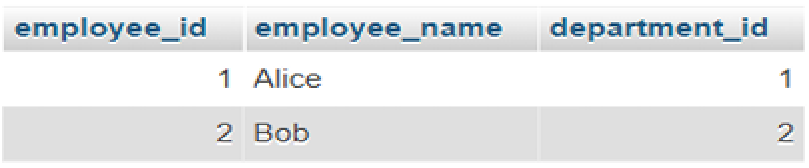
- Multi-Step Processes
  Data migration, where several tables are updated.

- 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.

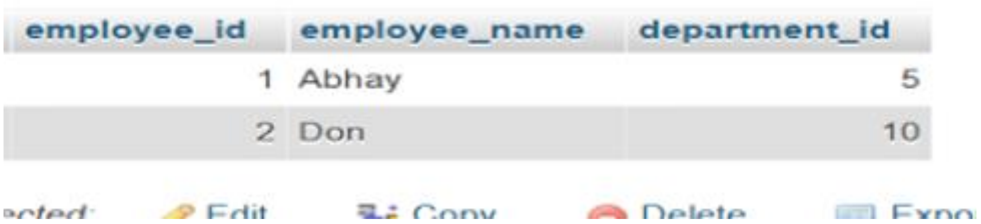
```
START TRANSACTION;

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;
```



employee_id	employee_name	department_id
1	Alice	1
2	Bob	2

- **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;
```



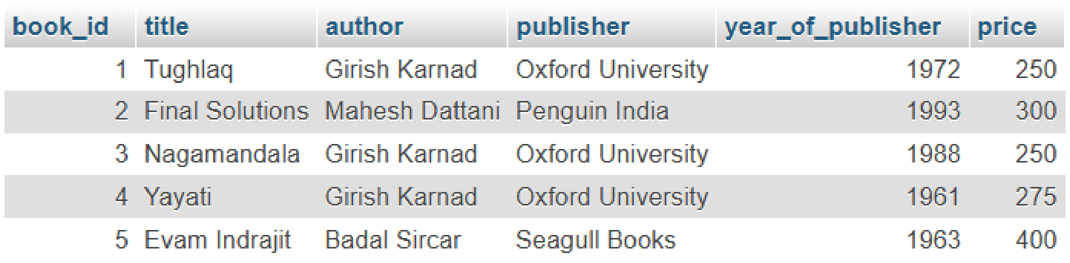
employee_id	employee_name	department_id
1	Abhay	5
2	Don	10

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.

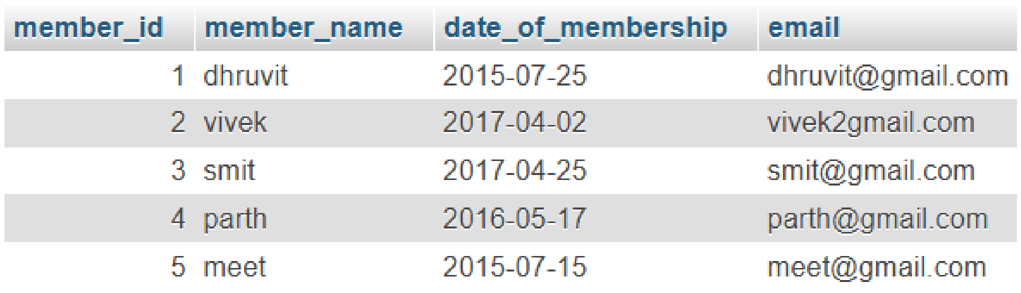
```
CREATE DATABASE library_db;
CREATE TABLE books(
book_id int,
title varchar(20),
author varchar(20),
publisher varchar(20),
year_of_publisher int,
price int   );
```



book_id	title	author	publisher	year_of_publisher	price
1	Tughlaq	Girish Karnad	Oxford University	1972	250
2	Final Solutions	Mahesh Dattani	Penguin India	1993	300
3	Nagamandala	Girish Karnad	Oxford University	1988	250
4	Yayati	Girish Karnad	Oxford University	1961	275
5	Evam Indrajit	Badal Sircar	Seagull Books	1963	400

- **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');
```

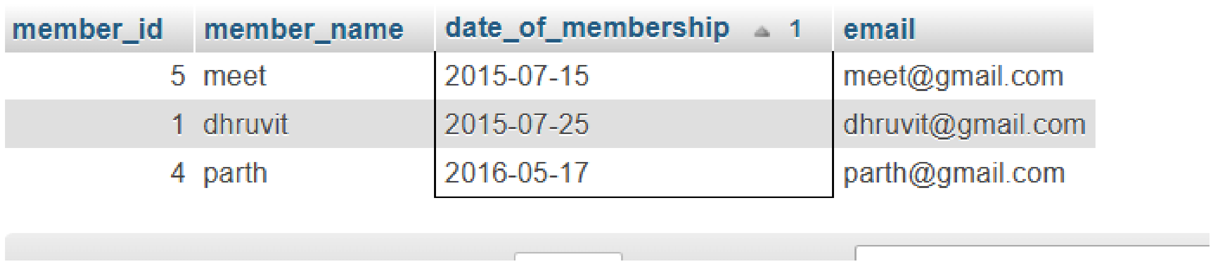


member_id	member_name	date_of_membership	email
1	dhruvit	2015-07-25	dhruvit@gmail.com
2	vivek	2017-04-02	vivek2gmail.com
3	smit	2017-04-25	smit@gmail.com
4	parth	2016-05-17	parth@gmail.com
5	meet	2015-07-15	meet@gmail.com

2. SQL Syntax

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

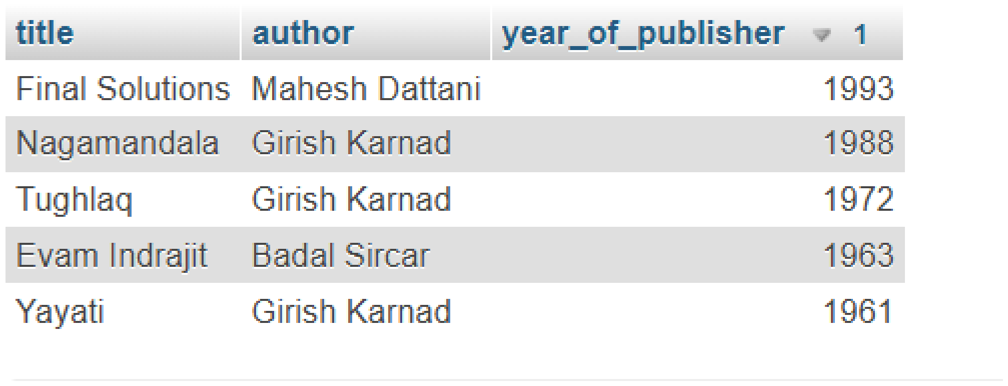
  SELECT * FROM members WHERE date_of_membership<'2017-01-01' ORDER BY date_of_membership;



member_id	member_name	date_of_membership ▲ 1	email
5	meet	2015-07-15	meet@gmail.com
1	dhruvit	2015-07-25	dhruvit@gmail.com
4	parth	2016-05-17	parth@gmail.com

- **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;

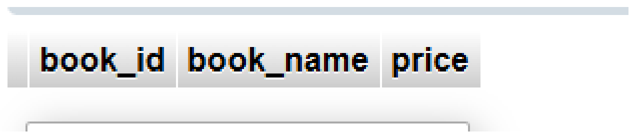


title	author	year_of_publisher ▼ 1
Final Solutions	Mahesh Dattani	1993
Nagamandala	Girish Karnad	1988
Tughlaq	Girish Karnad	1972
Evam Indrajit	Badal Sircar	1963
Yayati	Girish Karnad	1961

3. SQL Constraints

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

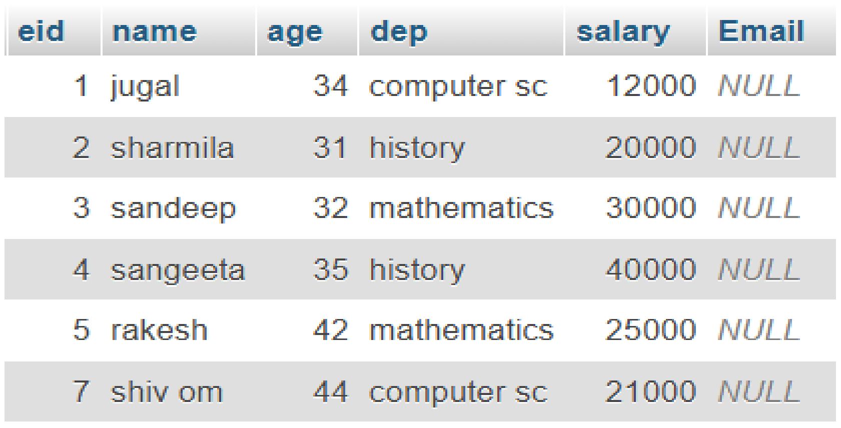
```
CREATE TABLE book (
book_id int PRIMARY KEY,
book_name varchar(10),
price int CHECK (price>0)
);
```



book_id	book_name	price

- **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;
```

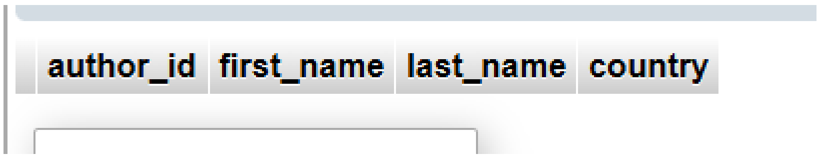


eid	name	age	dep	salary	Email
1	jugal	34	computer sc	12000	NULL
2	sharmila	31	history	20000	NULL
3	sandeep	32	mathematics	30000	NULL
4	sangeeta	35	history	40000	NULL
5	rakesh	42	mathematics	25000	NULL
7	shiv om	44	computer sc	21000	NULL

4. 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.

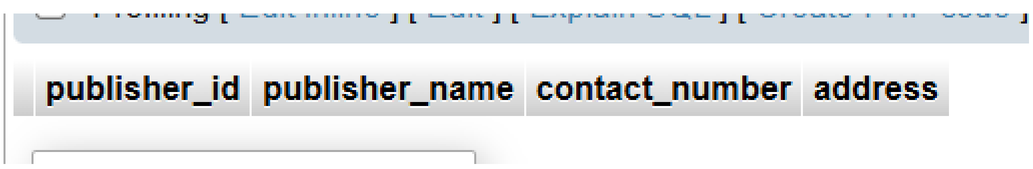
```
CREATE TABLE authors (
author_id int PRIMARY KEY,
first_name varchar(10),
last_name varchar(10),
country varchar(10)
);
```



author_id	first_name	last_name	country

- **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)
);
```

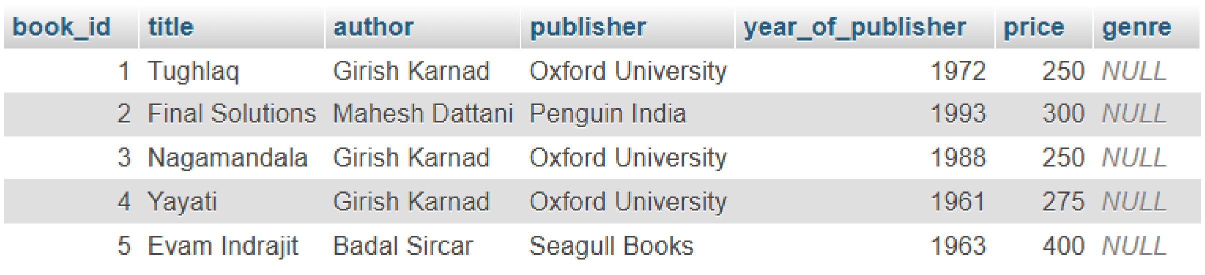


publisher_id	publisher_name	contact_number	address

5. ALTER Command

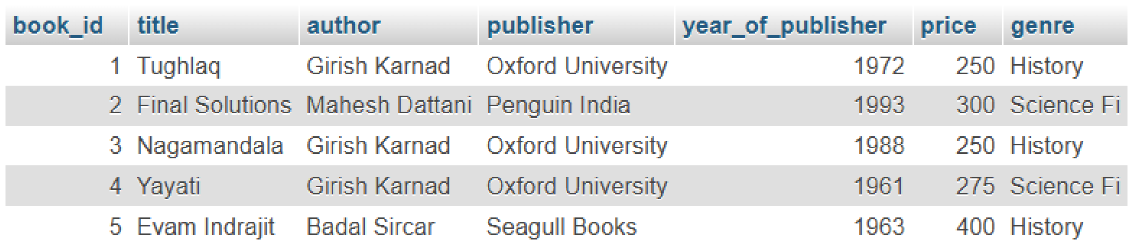
- **Lab 3:** Add a new column genre to the books table.Update the genre for all existing records.

```
ALTER TABLE books ADD genre varchar(10);
```



book_id	title	author	publisher	year_of_publisher	price	genre
1	Tughlaq	Girish Karnad	Oxford University	1972	250	NULL
2	Final Solutions	Mahesh Dattani	Penguin India	1993	300	NULL
3	Nagamandala	Girish Karnad	Oxford University	1988	250	NULL
4	Yayati	Girish Karnad	Oxford University	1961	275	NULL
5	Evam Indrajit	Badal Sircar	Seagull Books	1963	400	NULL

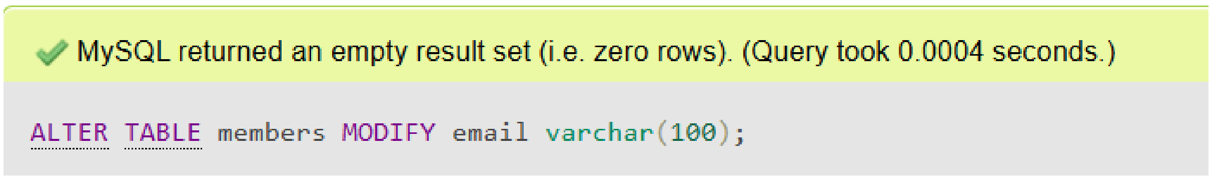
```
UPDATE books set genre = "History" WHERE book_id IN(1,3,5);
UPDATE books set genre = "Science Fiction" WHERE book_id IN(2,4);
```



book_id	title	author	publisher	year_of_publisher	price	genre
1	Tughlaq	Girish Karnad	Oxford University	1972	250	History
2	Final Solutions	Mahesh Dattani	Penguin India	1993	300	Science Fi
3	Nagamandala	Girish Karnad	Oxford University	1988	250	History
4	Yayati	Girish Karnad	Oxford University	1961	275	Science Fi
5	Evam Indrajit	Badal Sircar	Seagull Books	1963	400	History

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

  ALTER TABLE members MODIFY email varchar(100);



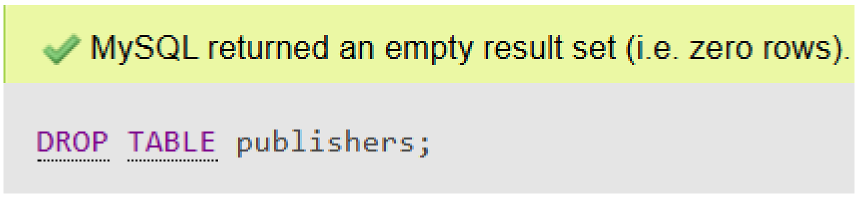
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

```
ALTER TABLE members MODIFY email varchar(100);
```

6. DROP Command

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

  DROP TABLE publishers;

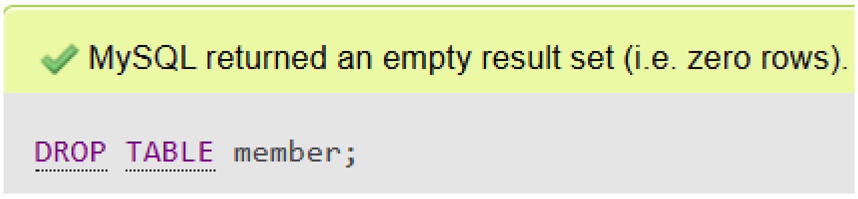


✓ MySQL returned an empty result set (i.e. zero rows).

```
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;



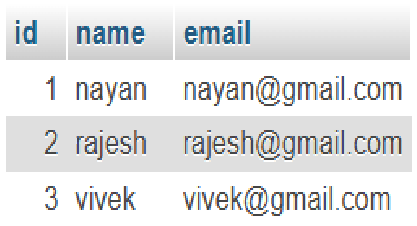
✓ MySQL returned an empty result set (i.e. zero rows).

```
DROP TABLE member;
```

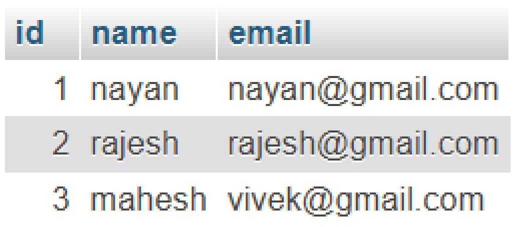
7. Data ManipulationLanguage(DML)

- **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;



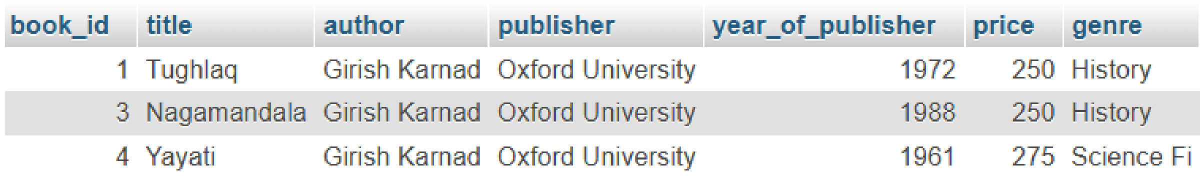
id	name	email
1	nayan	nayan@gmail.com
2	rajesh	rajesh@gmail.com
3	vivek	vivek@gmail.com



id	name	email
1	nayan	nayan@gmail.com
2	rajesh	rajesh@gmail.com
3	mahesh	vivek@gmail.com

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

  DELETE FROM books WHERE price>280;

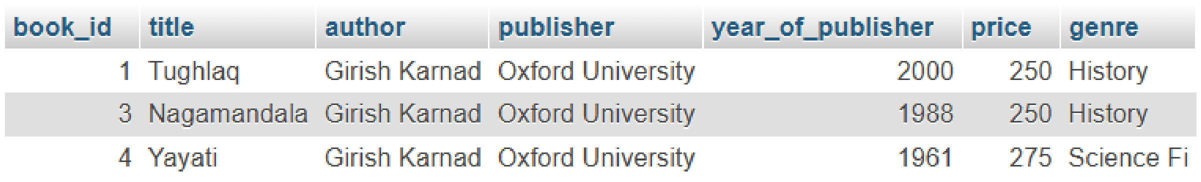


book_id	title	author	publisher	year_of_publisher	price	genre
1	Tughlaq	Girish Karnad	Oxford University	1972	250	History
3	Nagamandala	Girish Karnad	Oxford University	1988	250	History
4	Yayati	Girish Karnad	Oxford University	1961	275	Science Fi

8. UPDATE Command

- **Lab 3:** Update the year_of_publication of a book with a specific book_id.

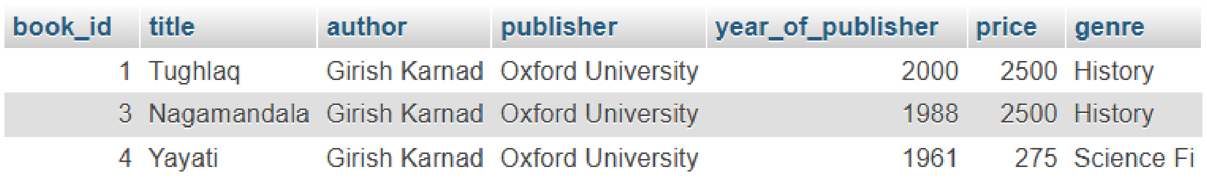
  UPDATE books SET year_of_publisher=2000 WHERE book_id=1;



book_id	title	author	publisher	year_of_publisher	price	genre
1	Tughlaq	Girish Karnad	Oxford University	2000	250	History
3	Nagamandala	Girish Karnad	Oxford University	1988	250	History
4	Yayati	Girish Karnad	Oxford University	1961	275	Science Fi

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

  UPDATE books SET price = price*10 WHERE year_of_publisher>1961;

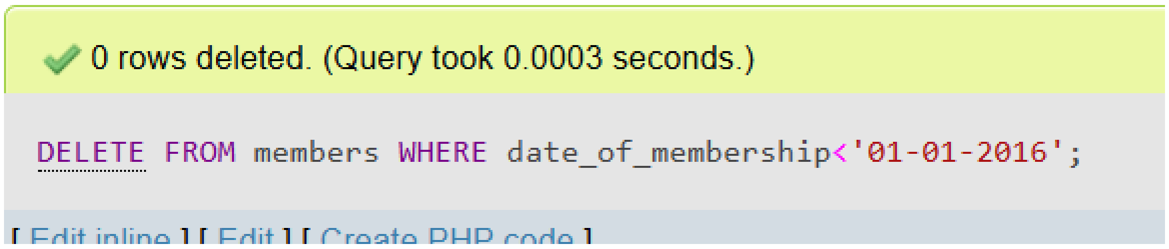


book_id	title	author	publisher	year_of_publisher	price	genre
1	Tughlaq	Girish Karnad	Oxford University	2000	2500	History
3	Nagamandala	Girish Karnad	Oxford University	1988	2500	History
4	Yayati	Girish Karnad	Oxford University	1961	275	Science Fi

9. DELETE Command

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

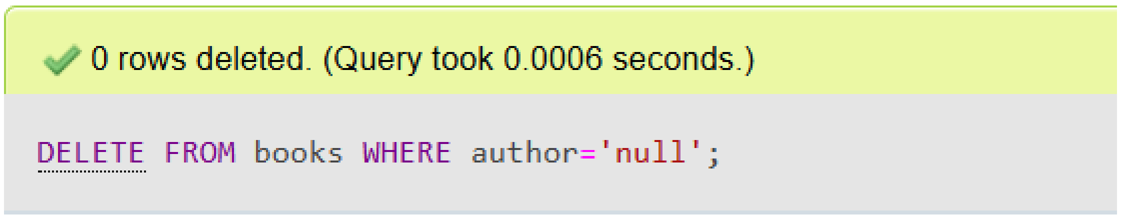
  DELETE FROM members WHERE date_of_membership<'01-01-2016';



```
0 rows deleted. (Query took 0.0003 seconds.)

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';

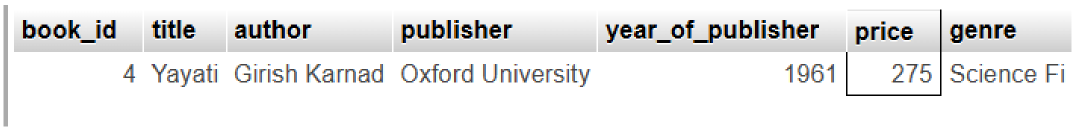


```
0 rows deleted. (Query took 0.0006 seconds.)

DELETE FROM books WHERE author='null';
```

10. Data Query Language (DQL)

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

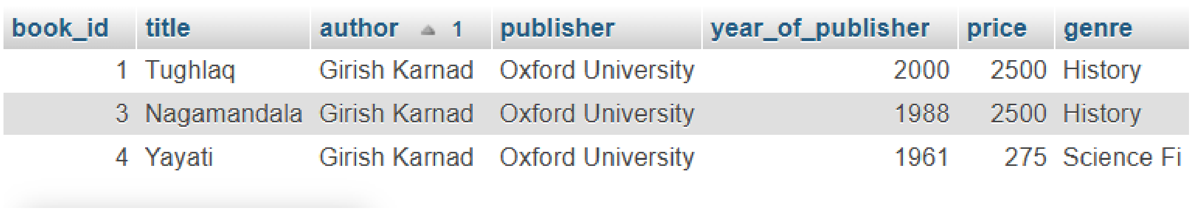
  SELECT * FROM books WHERE price BETWEEN 200 and 300;



book_id	title	author	publisher	year_of_publisher	price	genre
4	Yayati	Girish Karnad	Oxford University	1961	275	Science Fi

- **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;

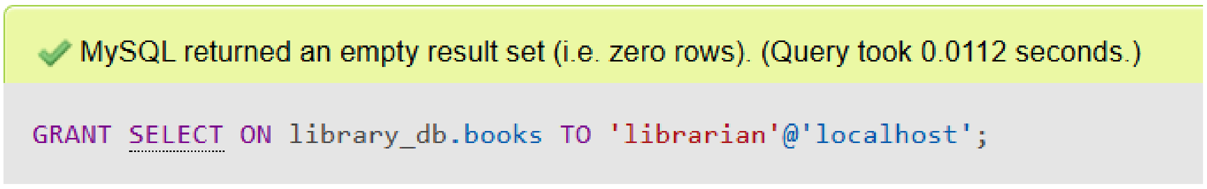


book_id	title	author ▲ 1	publisher	year_of_publisher	price	genre
1	Tughlaq	Girish Karnad	Oxford University	2000	2500	History
3	Nagamandala	Girish Karnad	Oxford University	1988	2500	History
4	Yayati	Girish Karnad	Oxford University	1961	275	Science Fi

11. Data Control Language (DCL)

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

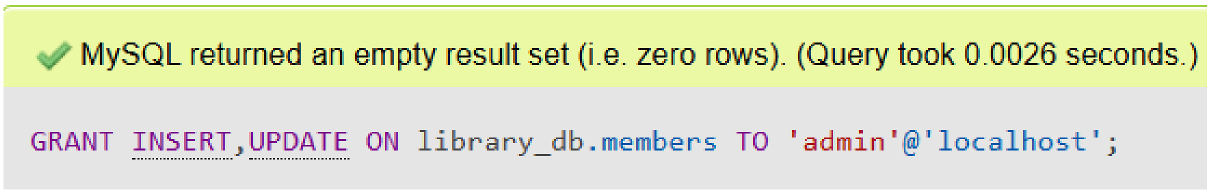
  GRANT SELECT ON library_db.books TO 'librarian'@'localhost';



```
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0112 seconds.)

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';



```
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0026 seconds.)

GRANT INSERT,UPDATE ON library_db.members TO 'admin'@'localhost';
```

12. 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';

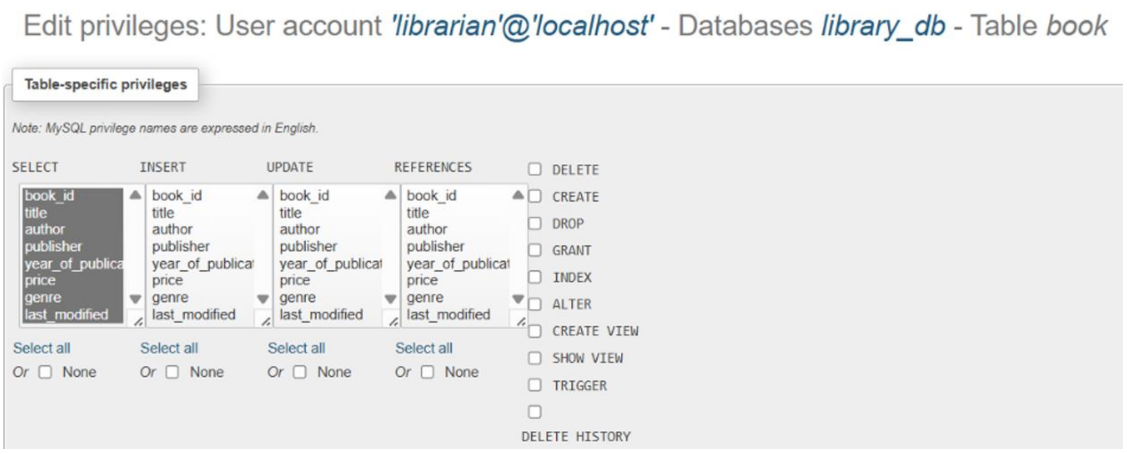

Edit privileges: User account *'librarian'@'localhost'* - Databases *library_db* - Table *book*

Table-specific privileges

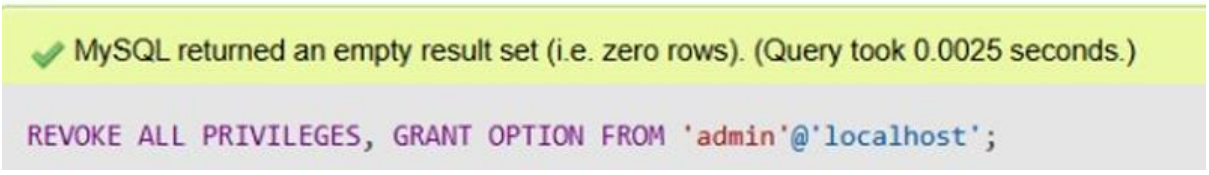
Note: MySQL privilege names are expressed in English.

SELECT	INSERT	UPDATE	REFERENCES
book_id	book_id	book_id	book_id
title	title	title	title
author	author	author	author
publisher	publisher	publisher	publisher
year_of_publica	year_of_publicat	year_of_publicat	year_of_publicat
price	price	price	price
genre	genre	genre	genre
last_modified	last_modified	last_modified	last_modified
Select all	Select all	Select all	Select all
Or ☐ None	Or ☐ None	Or ☐ None	Or ☐ None

☐ DELETE
☐ CREATE
☐ DROP
☐ GRANT
☐ INDEX
☐ ALTER
☐ CREATE VIEW
☐ SHOW VIEW
☐ TRIGGER
☐ DELETE HISTORY

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

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



```
✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0025 seconds.)

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

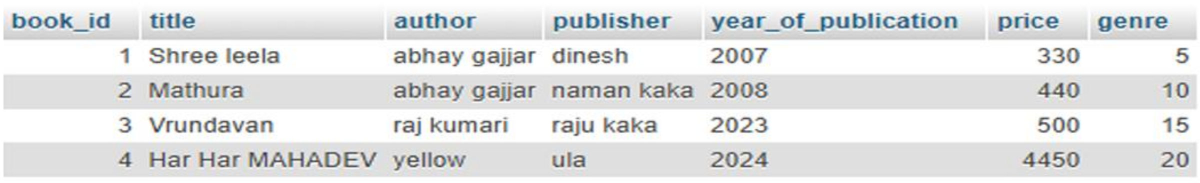
13. Transaction Control Language (TCL)

- **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;

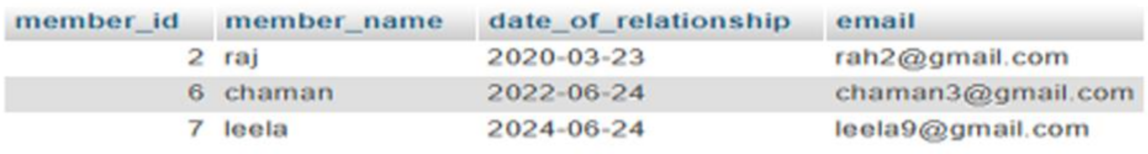
  START TRANSACTION;
  INSERT INTO book VALUES(5,"hajir moj","shivalay","lolaji",2018,980,80);
  ROLLBACK;



book_id	title	author	publisher	year_of_publication	price	genre
1	Shree leela	abhay gajjar	dinesh	2007	330	5
2	Mathura	abhay gajjar	naman kaka	2008	440	10
3	Vrundavan	raj kumari	raju kaka	2023	500	15
4	Har Har MAHADEV	yellow	ula	2024	4450	20

- **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;
```

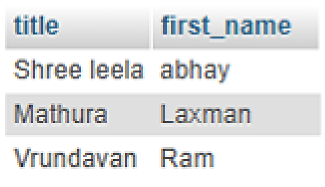


member_id	member_name	date_of_relationship	email
2	raj	2020-03-23	rah2@gmail.com
6	chaman	2022-06-24	chaman3@gmail.com
7	leela	2024-06-24	leela9@gmail.com

14. SQL Joins

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

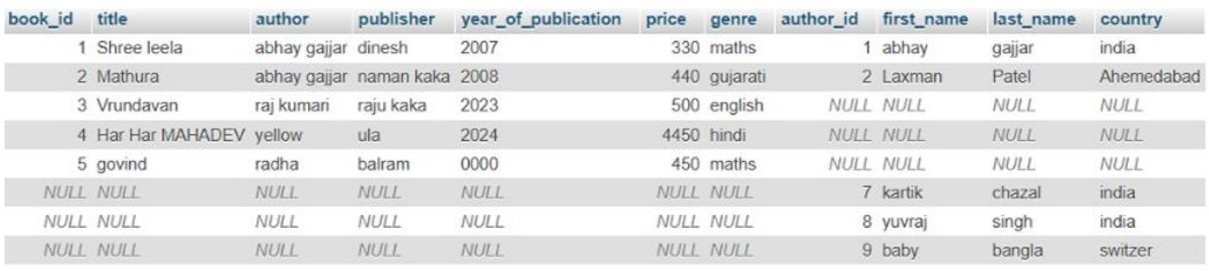
```
SELECT book.title,authors.first_name
FROM book
INNER JOIN authors
WHERE book.book_id=authors.author_id;
```



title	first_name
Shree leela	abhay
Mathura	Laxman
Vrundavan	Ram

- **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;
```

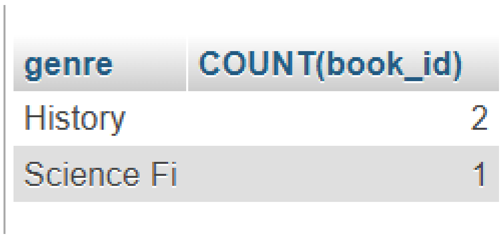


book_id	title	author	publisher	year_of_publication	price	genre	author_id	first_name	last_name	country
1	Shree leela	abhay gajjar	dinesh	2007	330	maths	1	abhay	gajjar	india
2	Mathura	abhay gajjar	naman kaka	2008	440	gujarati	2	Laxman	Patel	Ahemedabad
3	Vrundavan	raj kumari	raju kaka	2023	500	english	NULL	NULL	NULL	NULL
4	Har Har MAHADEV	yellow	ula	2024	4450	hindi	NULL	NULL	NULL	NULL
5	govind	radha	balram	0000	450	maths	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	7	kartik	chazal	india
NULL	NULL	NULL	NULL	NULL	NULL	NULL	8	yuvraj	singh	india
NULL	NULL	NULL	NULL	NULL	NULL	NULL	9	baby	bangla	switzer

15. SQL Group By

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

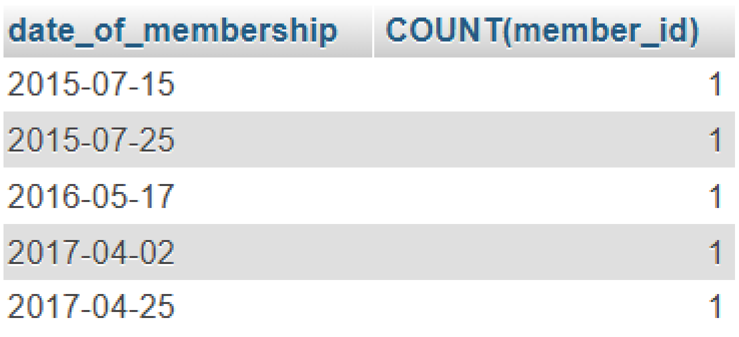
SELECT genre , COUNT(book_id) FROM books GROUP BY genre;



genre	COUNT(book_id)
History	2
Science Fi	1

➢ **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;

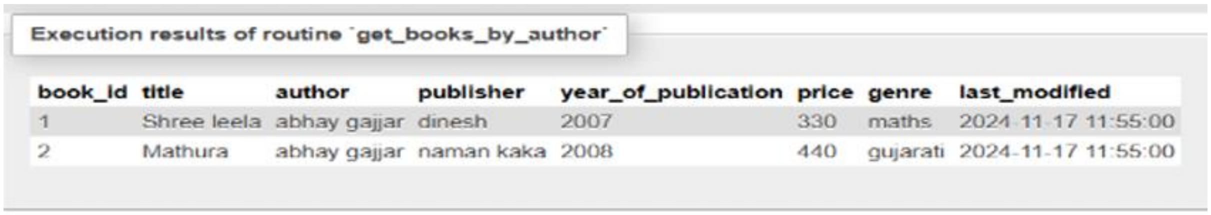


date_of_membership	COUNT(member_id)
2015-07-15	1
2015-07-25	1
2016-05-17	1
2017-04-02	1
2017-04-25	1

16. 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 ;
```

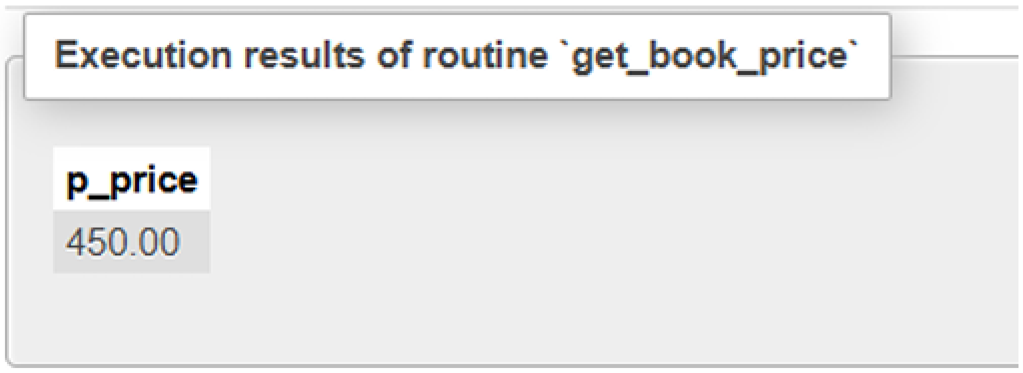


Execution results of routine `get_books_by_author`

book_id	title	author	publisher	year_of_publication	price	genre	last_modified
1	Shree leela	abhay gajjar	dinesh	2007	330	maths	2024-11-17 11:55:00
2	Mathura	abhay gajjar	naman kaka	2008	440	gujarati	2024-11-17 11:55:00

- **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 ;
```



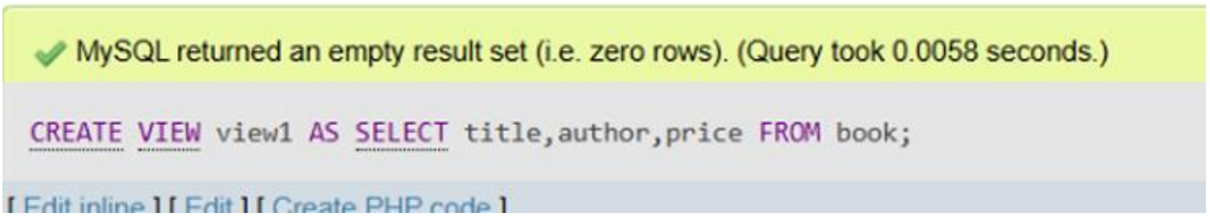
Execution results of routine `get_book_price`

p_price
450.00

17. SQL View

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

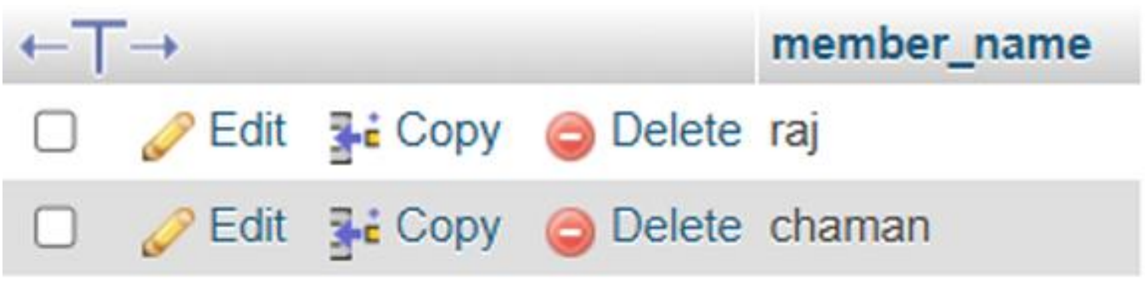
  CREATE VIEW view1 AS SELECT title,author,price FROM book;



```
✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0058 seconds.)

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

[ Edit inline ] [ Edit ] [ Create PHP code ]
```

- **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;



←T→	member_name
☐ Edit Copy Delete	raj
☐ Edit Copy Delete	chaman

18. SQL Trigger

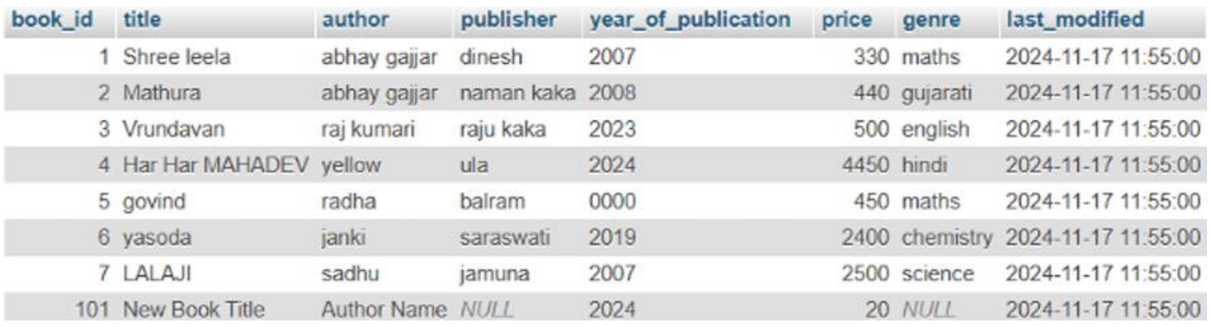
- **Lab 3:** Create a trigger to automatically update the last_modified timestamp of the books table whenever a record is updated.

  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

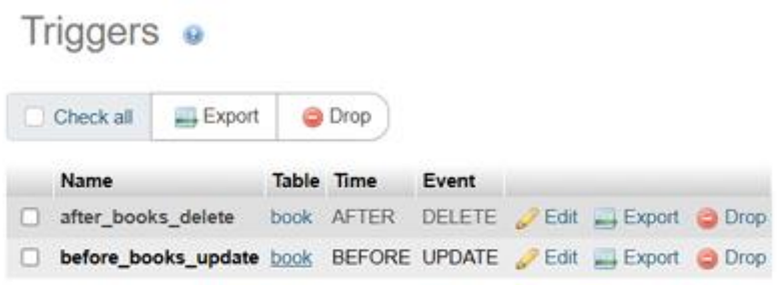
```
SET NEW.last_modified = CURRENT_TIMESTAMP;
END //
DELIMITER ;
```



book_id	title	author	publisher	year_of_publication	price	genre	last_modified
1	Shree leela	abhay gajjar	dinesh	2007	330	maths	2024-11-17 11:55:00
2	Mathura	abhay gajjar	naman kaka	2008	440	gujarati	2024-11-17 11:55:00
3	Vrundavan	raj kumari	raju kaka	2023	500	english	2024-11-17 11:55:00
4	Har Har MAHADEV	yellow	ula	2024	4450	hindi	2024-11-17 11:55:00
5	govind	radha	balram	0000	450	maths	2024-11-17 11:55:00
6	yasoda	janki	saraswati	2019	2400	chemistry	2024-11-17 11:55:00
7	LALAJI	sadhu	jamuna	2007	2500	science	2024-11-17 11:55:00
101	New Book Title	Author Name	NULL	2024	20	NULL	2024-11-17 11:55:00

- **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 ;
```

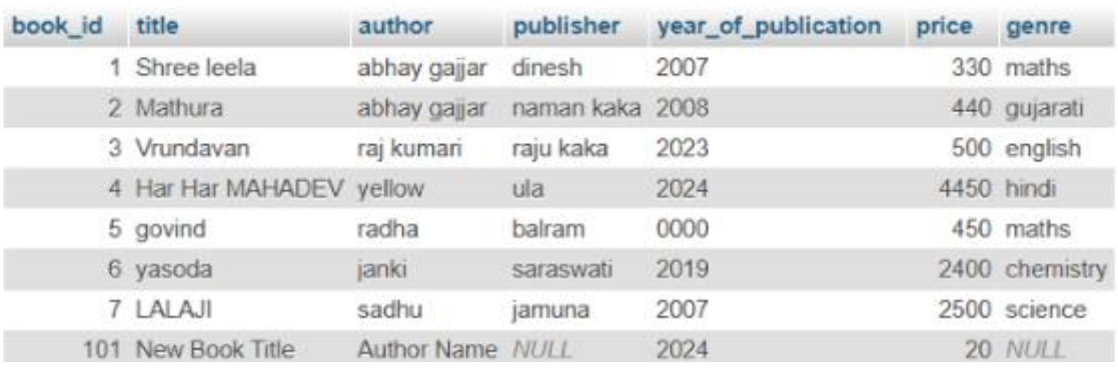


	Name	Table	Time	Event			
☐	after_books_delete	book	AFTER	DELETE	Edit	Export	Drop
☐	before_books_update	book	BEFORE	UPDATE	Edit	Export	Drop

19. 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.

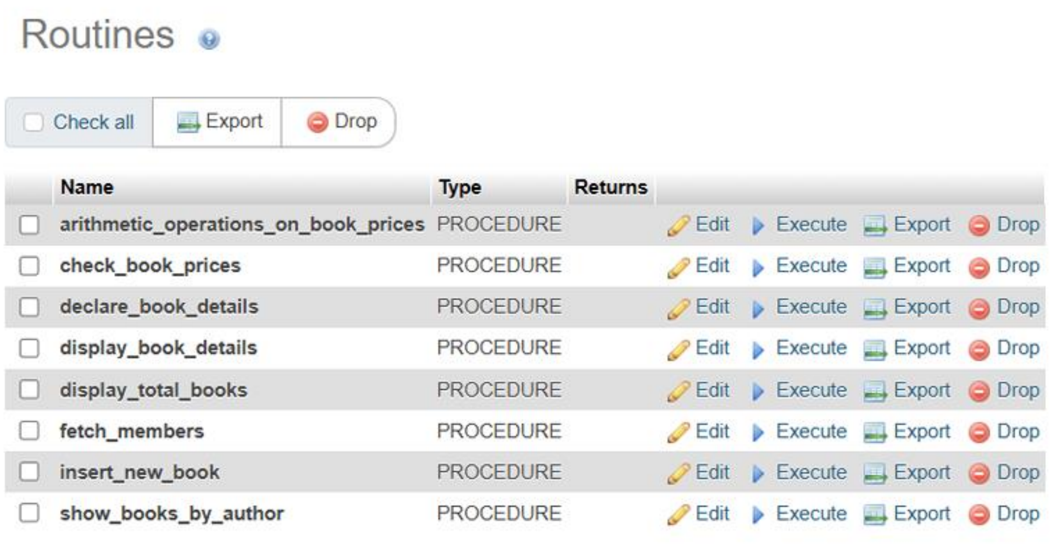
```
BEGIN
DECLARE v_book_id INT DEFAULT 101;
DECLARE v_title VARCHAR(100) DEFAULT 'New BookTitle';

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
```



book_id	title	author	publisher	year_of_publication	price	genre
1	Shree leela	abhay gajjar	dinesh	2007	330	maths
2	Mathura	abhay gajjar	naman kaka	2008	440	gujarati
3	Vrundavan	raj kumari	raju kaka	2023	500	english
4	Har Har MAHADEV	yellow	ula	2024	4450	hindi
5	govind	radha	balram	0000	450	maths
6	yasoda	janki	saraswati	2019	2400	chemistry
7	LALAJI	sadhu	jamuna	2007	2500	science
101	New Book Title	Author Name	NULL	2024	20	NULL

- **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
```

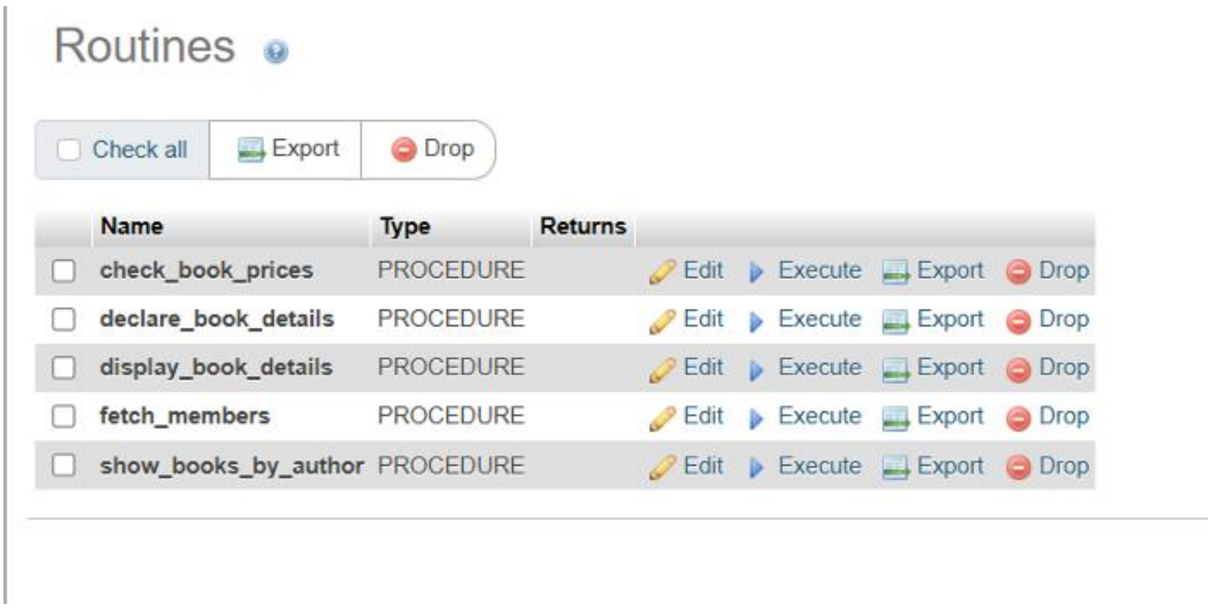

Routines

	Name	Type	Returns				
☐	arithmetic_operations_on_book_prices	PROCEDURE		Edit	Execute	Export	Drop
☐	check_book_prices	PROCEDURE		Edit	Execute	Export	Drop
☐	declare_book_details	PROCEDURE		Edit	Execute	Export	Drop
☐	display_book_details	PROCEDURE		Edit	Execute	Export	Drop
☐	display_total_books	PROCEDURE		Edit	Execute	Export	Drop
☐	fetch_members	PROCEDURE		Edit	Execute	Export	Drop
☐	insert_new_book	PROCEDURE		Edit	Execute	Export	Drop
☐	show_books_by_author	PROCEDURE		Edit	Execute	Export	Drop

20. PL/SQL Syntax

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

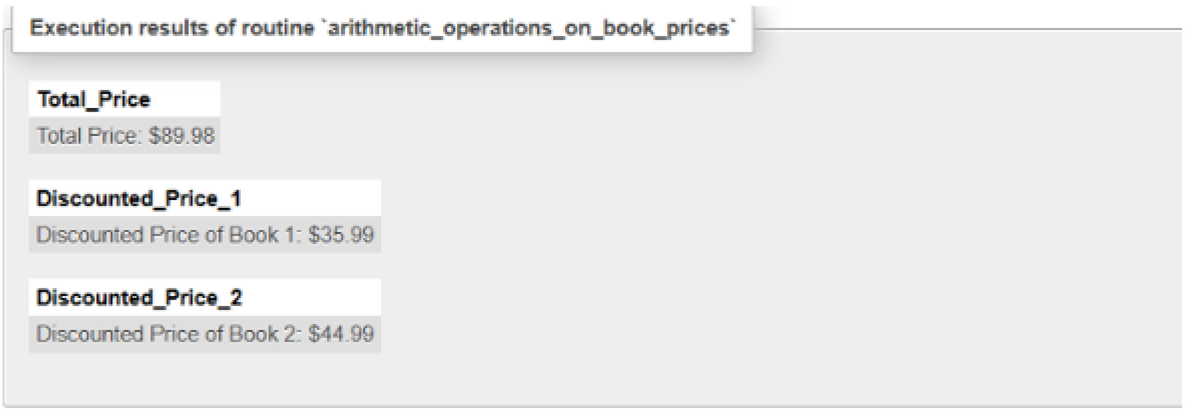
```
DELIMITER //

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 ;
```


Routines

	Name	Type	Returns	
☐	check_book_prices	PROCEDURE		Edit Execute Export Drop
☐	declare_book_details	PROCEDURE		Edit Execute Export Drop
☐	display_book_details	PROCEDURE		Edit Execute Export Drop
☐	fetch_members	PROCEDURE		Edit Execute Export Drop
☐	show_books_by_author	PROCEDURE		Edit Execute Export Drop

➢ **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 ;
```

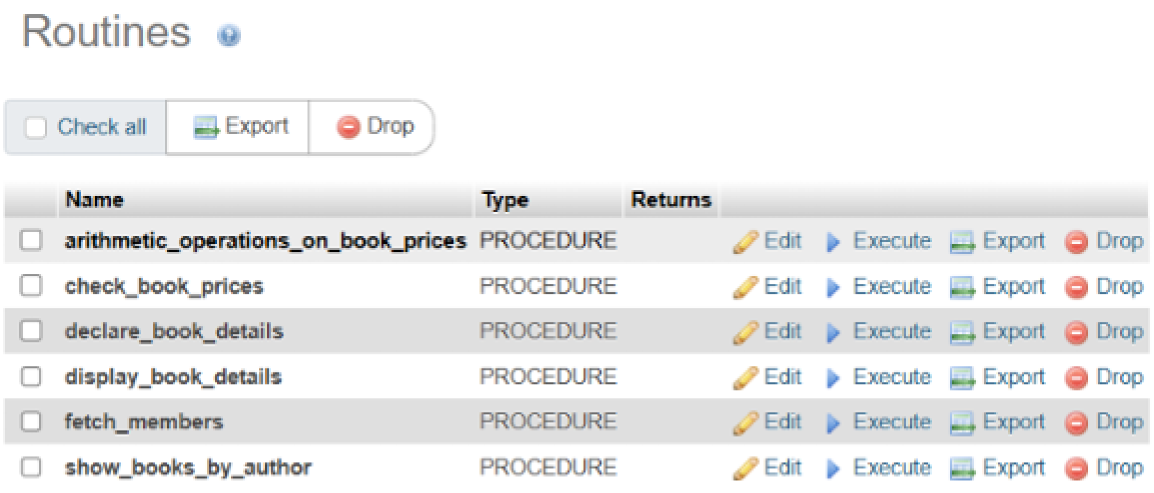


Execution results of routine `arithmetic_operations_on_book_prices`

Total_Price
Total Price: $89.98

Discounted_Price_1
Discounted Price of Book 1: $35.99

Discounted_Price_2
Discounted Price of Book 2: $44.99



Routines

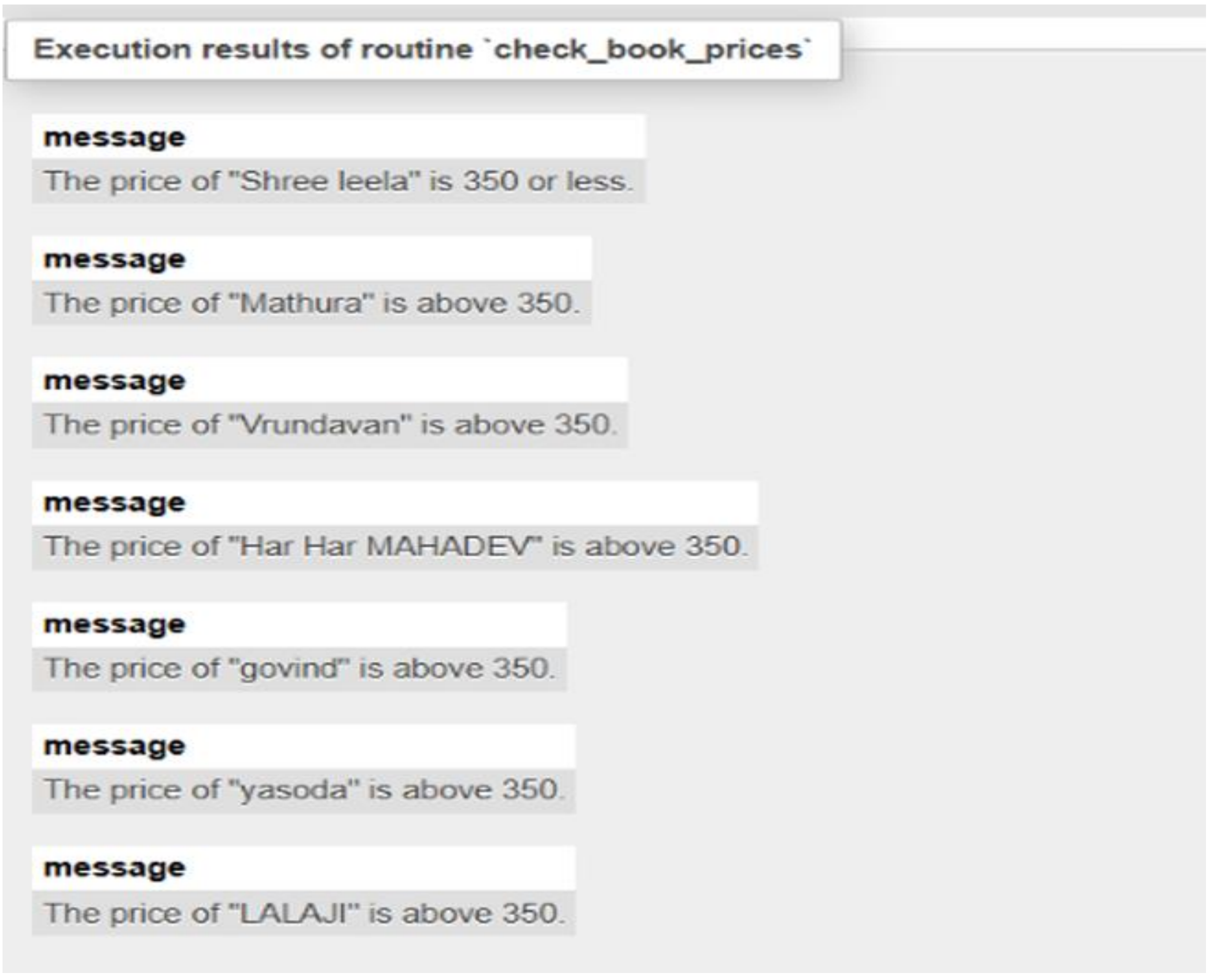
	Name	Type	Returns				
☐	arithmetic_operations_on_book_prices	PROCEDURE		Edit	Execute	Export	Drop
☐	check_book_prices	PROCEDURE		Edit	Execute	Export	Drop
☐	declare_book_details	PROCEDURE		Edit	Execute	Export	Drop
☐	display_book_details	PROCEDURE		Edit	Execute	Export	Drop
☐	fetch_members	PROCEDURE		Edit	Execute	Export	Drop
☐	show_books_by_author	PROCEDURE		Edit	Execute	Export	Drop

21. 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;
```

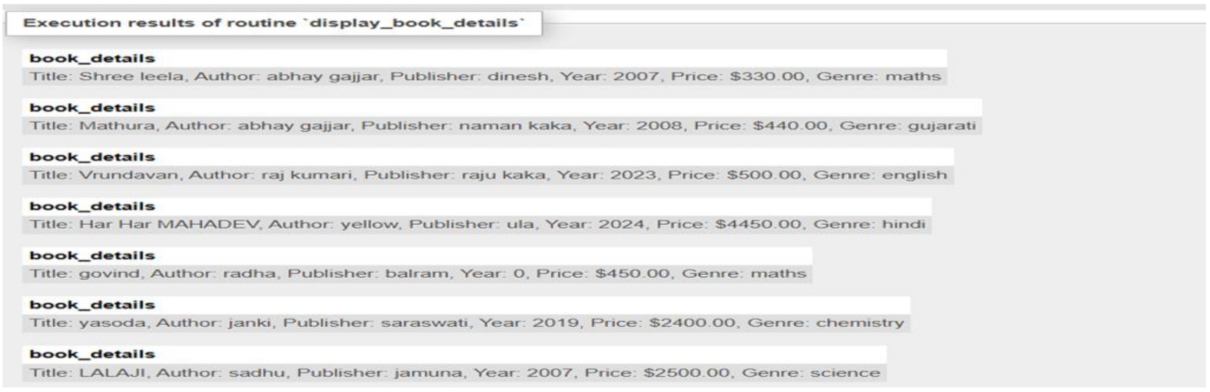
```
IF done THEN
LEAVE read_loop;
END IF;

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 ;
```



```
Execution results of routine `check_book_prices`

message
The price of "Shree leela" is 350 or less.

message
The price of "Mathura" is above 350.

message
The price of "Vrundavan" is above 350.

message
The price of "Har Har MAHADEV" is above 350.

message
The price of "govind" is above 350.

message
The price of "yasoda" is above 350.

message
The price of "LALAJI" is above 350.
```

- **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
```

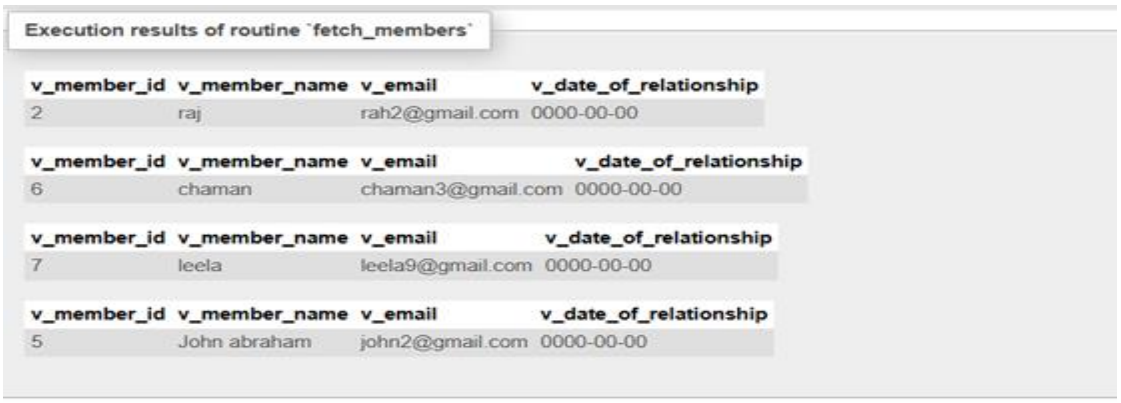


```
book_details
Title: Shree leela, Author: abhay gajjar, Publisher: dinesh, Year: 2007, Price: $330.00, Genre: maths
book_details
Title: Mathura, Author: abhay gajjar, Publisher: naman kaka, Year: 2008, Price: $440.00, Genre: gujarati
book_details
Title: Vrundavan, Author: raj kumari, Publisher: raju kaka, Year: 2023, Price: $500.00, Genre: english
book_details
Title: Har Har MAHADEV, Author: yellow, Publisher: ula, Year: 2024, Price: $4450.00, Genre: hindi
book_details
Title: govind, Author: radha, Publisher: balram, Year: 0, Price: $450.00, Genre: maths
book_details
Title: yasoda, Author: janki, Publisher: saraswati, Year: 2019, Price: $2400.00, Genre: chemistry
book_details
Title: LALAJI, Author: sadhu, Publisher: jamuna, Year: 2007, Price: $2500.00, Genre: science
```

22. 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
```

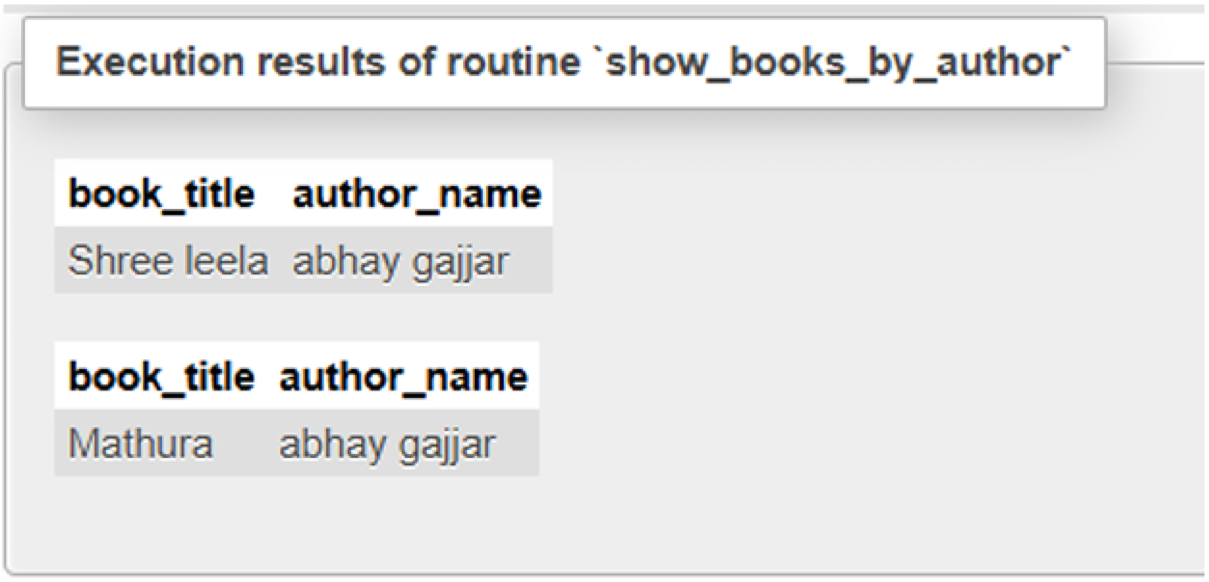


Execution results of routine `fetch_members`

```
v_member_id  v_member_name  v_email            v_date_of_relationship
2            raj            rah2@gmail.com     0000-00-00

v_member_id  v_member_name  v_email            v_date_of_relationship
6            chaman         chaman3@gmail.com  0000-00-00

v_member_id  v_member_name  v_email            v_date_of_relationship
7            leela          leela9@gmail.com   0000-00-00

v_member_id  v_member_name  v_email            v_date_of_relationship
5            John abraham   john2@gmail.com    0000-00-00
```

- **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
```



Execution results of routine `show_books_by_author`

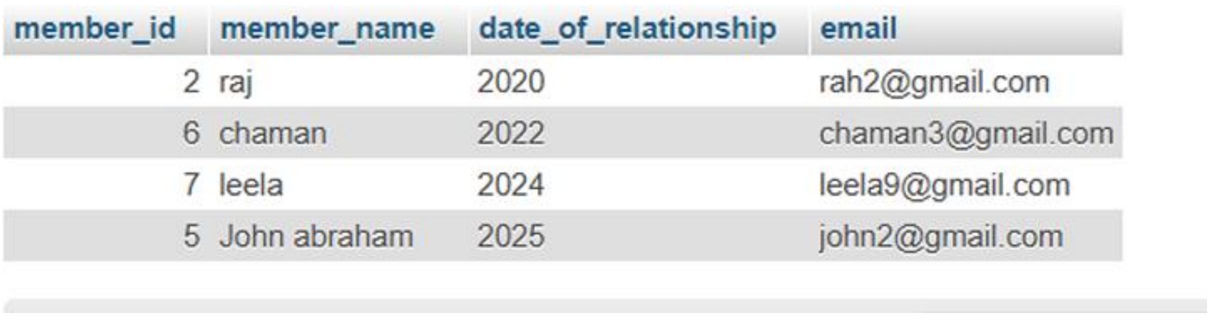
book_title	author_name
Shree leela	abhay gajjar

book_title	author_name
Mathura	abhay gajjar

23. 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;
```

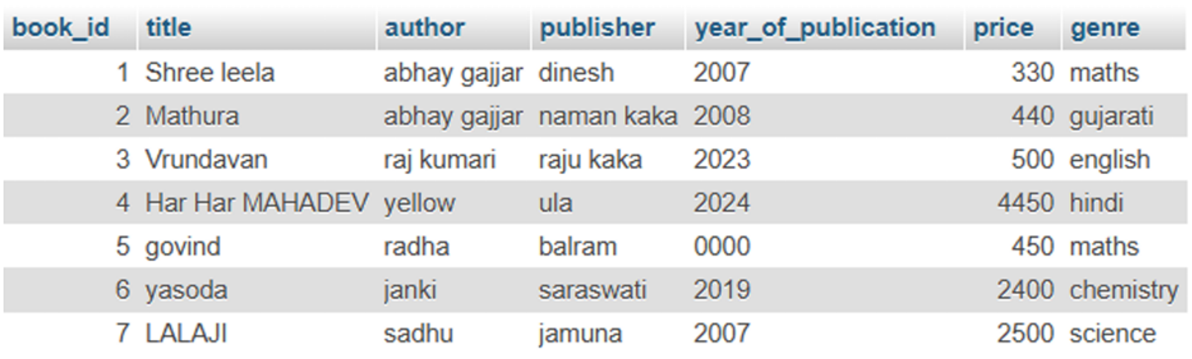


member_id	member_name	date_of_relationship	email
2	raj	2020	rah2@gmail.com
6	chaman	2022	chaman3@gmail.com
7	leela	2024	leela9@gmail.com
5	John abraham	2025	john2@gmail.com

- **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;
```



book_id	title	author	publisher	year_of_publication	price	genre
1	Shree leela	abhay gajjar	dinesh	2007	330	maths
2	Mathura	abhay gajjar	naman kaka	2008	440	gujarati
3	Vrundavan	raj kumari	raju kaka	2023	500	english
4	Har Har MAHADEV	yellow	ula	2024	4450	hindi
5	govind	radha	balram	0000	450	maths
6	yasoda	janki	saraswati	2019	2400	chemistry
7	LALAJI	sadhu	jamuna	2007	2500	science