1. DDL (Data Definition Language) and DML (Data Manipulation Language) are two distinct subsets of SQL (Structured Query Language) that serve different purposes in managing databases.

DDL	DML
It deals with how data is stored in the database and aids in defining the structure or schema of a database.	It enables us to manipulate the data that is kept in the database, including retrieve, update, and delete.
Data Definition Language is the full name of this language	Data Manipulation Language is the full name of this language.
There is no additional categorisation for the DDL instructions	The DML commands are divided into declarative and procedural (non-procedural) DMLs.
The commonly used commands under DDL language are: • CREATE • DROP • ALTER • TRUNCATE • RENAME	The commonly used commands under DML language are: • INSERT • UPDATE • DELETE • SELECT
DDL commands automatically commit changes made to the database, making them permanent.	Database modifications are not irreversible since DML commands are not automatically committed.
The entire database or table is impacted by the DDL command	The DML commands will affect the single or multiple records based on the specified condition.
A WHERE clause is not used in DDL instructions since record filtration is not feasible in this situation	DML statements allow the use of a WHERE clause while changing data in a database.

2. The PRIMARY KEY and FOREIGN KEY constraints are fundamental in relational database design, serving distinct but related purposes for maintaining data integrity and defining relationships.

   1. PRIMARY KEY

   A primary key, also called a primary keyword, is a column in a relational database table that's distinctive for each record. It's a unique identifier,

such as a driver's license number, telephone number with area code or vehicle identification number (VIN)

Unique Identification: Ensures that no two rows have the same primary key value.

Data Integrity: Guarantees that the column(s) chosen for the primary key:

Cannot contain duplicate values (UNIQUE constraint).

Cannot contain NULL values (NOT NULL constraint).

Foundation for Relationships: It is the column that other tables reference using a foreign key.

Example:

```
CREATE TABLE Customers (
   CustomerID INT NOT NULL,
   FirstName VARCHAR(50),
   LastName VARCHAR(50),
   CONSTRAINT PK_Customer PRIMARY KEY (CustomerID) );
```

In this example, CustomerID is the primary key, meaning every customer must have a unique, non-null CustomerID.

2. FOREIGN KEY

A **FOREIGN KEY** is a column or a set of columns in one table that references the primary key (or a unique key) in another table (or sometimes the same table).

Use:

• **Relationship Establishment:** Creates a link between two tables, essential for relating data (e.g., linking an order to a customer).

• **Referential Integrity:** Enforces that values in the foreign key column must already exist in the primary key column of the referenced ("parent") table. This prevents "orphaned" records (e.g., an order being placed for a customer who doesn't exist).

Example (Creating an Orders table that links to Customers):

```
CREATE TABLE Orders (
   OrderID INT NOT NULL,
   OrderDate DATE,
   CustomerID INT,
```

```
    CONSTRAINT PK_Order PRIMARY KEY (OrderID),
    CONSTRAINT FK_CustomerOrder FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID)
);
```

In this example, CustomerID in the Orders table is the foreign key. It must contain a value that already exists in the CustomerID (primary key) column of the Customers table, or it can be NULL (depending on the column definition). This constraint ensures that every order is associated with a valid, existing customer.

3. SQL is understanding the **DELETE**, **DROP**, and **TRUNCATE** commands is important for efficient **data management**. While these commands are all used to **remove data**, they differ significantly in **functionality**, **usage**, and **performance**. Knowing when and how to use each command can improve the **efficiency** and **integrity** of our **database**.

DELETE	DROP	TRUNCATE
Deletes specific rows based on condition	Deletes the entire table or database	Deletes all rows but retains table structure
DELETE FROM table_name WHERE condition;	DROP TABLE table_name;	TRUNCATE TABLE table_name;
Can be Rollback	Cannot be Rollback	Cannot be Rollback
Removes selected rows	Removes table and data completely	Removes all rows
Slower, as each row is processed individually	Instant removal, affecting schema	Faster than DELETE but slower than DROP
Fires triggers	Does not fire triggers	Does not fire triggers
It is faster than DELETE but slower than TRUNCATE as it firstly deletes the rows and then the table from the database.	It is slower than the DROP and TRUNCATE commands as it deletes one row at a time based on the specified conditions.	It is faster than both the DELETE and DROP commands as it deletes all the records at a time without any condition.

4. The LIKE operator in SQL is used in a WHERE clause to search for a specified pattern in a column. It enables pattern matching within text data, allowing for more flexible filtering than exact value comparisons.

Code

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;
```

Wildcard Characters:

The LIKE operator uses wildcard characters to define the search pattern:

% (Percent Sign):
Represents zero, one, or multiple characters.
'a%' finds values that start with "a".
'%a' finds values that end with "a".
'%or%' finds values that contain "or" anywhere.

_ (Underscore):

Represents a single character.
'h_t' finds values like "hot", "hat", "hit", etc.
'K___%' finds values starting with "K" and having at least 4 characters.

Examples: Finding names starting with 'S'.

Code

```
SELECT CustomerName
 FROM Customers
 WHERE CustomerName LIKE 'S%';
```

Finding names containing 'an'.

Code

```
SELECT ProductName
  FROM Products
  WHERE ProductName LIKE '%an%';
```

Finding names with 'o' as the second letter:

Code

```
SELECT EmployeeName
FROM Employees
WHERE EmployeeName LIKE '_o%';
```

Case Sensitivity:

The case sensitivity of the LIKE operator can vary depending on the specific SQL database system (e.g., MySQL, SQL Server, PostgreSQL). In many systems, LIKE is case-insensitive by default, but this can often be configured or overridden using functions like LOWER() or UPPER().

BETWEEN operator:

The BETWEEN operator in SQL is used in the WHERE clause to select values that fall within a specified, inclusive range. It can be used with numbers, text, or dates to simplify range-based filtering, making queries more readable than using multiple AND conditions.

BETWEEN What it does: BETWEEN tests whether a value lies within a specified inclusive range.

Inclusivity: Both the lower and upper bounds are included in the result.Data types: Works with numbers, dates, and text (based on the database's collation/ordering and the data type).

Typical usage: in the WHERE clause to filter rows whose column value falls within a range

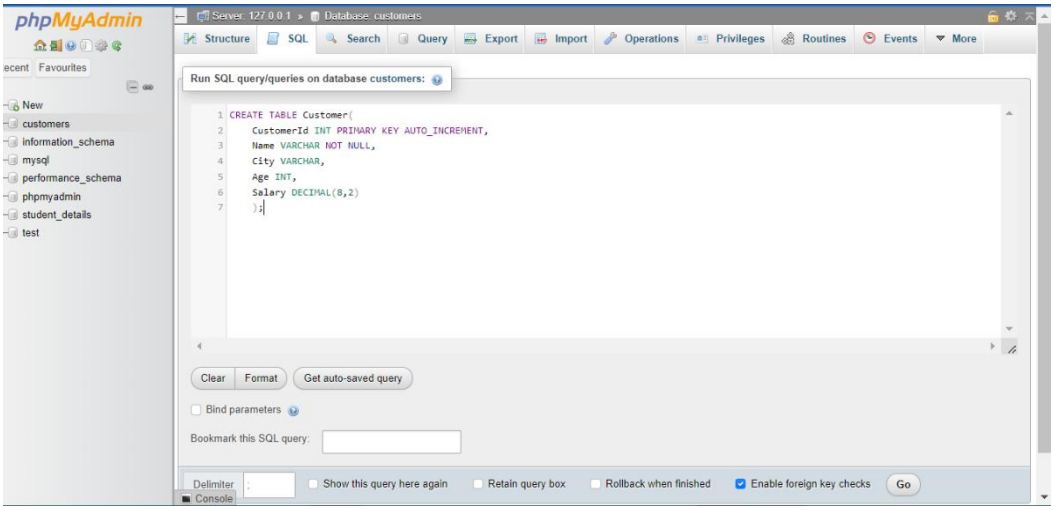
.Examples:Numeric range: WHERE salary BETWEEN 50000 AND 100000Date range: WHERE order_date BETWEEN '2024-01-01' AND '2024-12-31'Text range (lexical): WHERE last_name BETWEEN 'A' AND 'D' (depends on collations)

5. An INNER JOIN returns only rows where the join condition is met in both tables, while a LEFT JOIN returns all rows from the left table and the matching rows from the right table. If there is no match in the right table for a given row in the left table, the columns from the right table will be NULL for that row

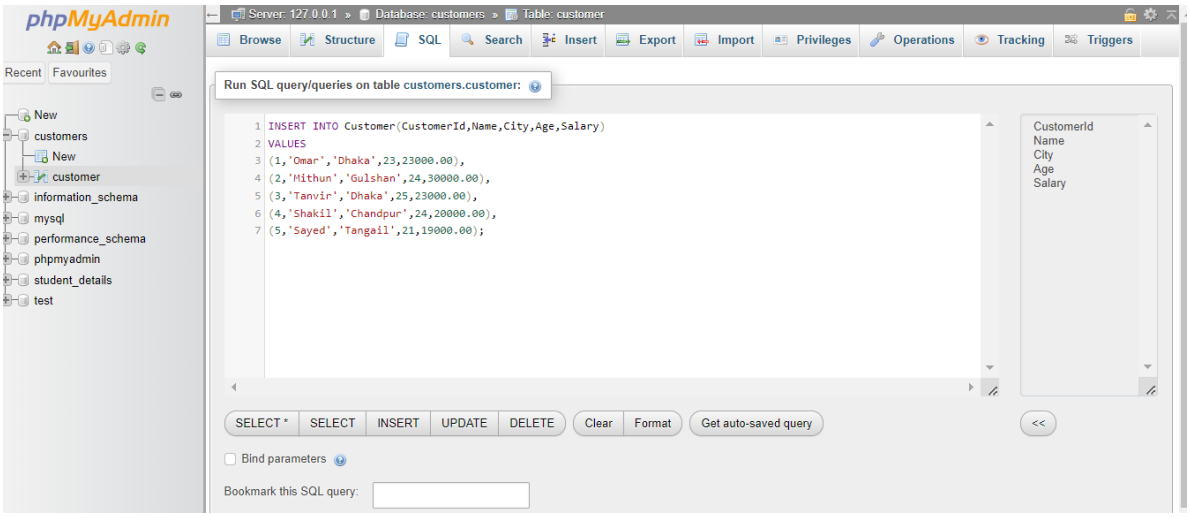
Difference between inner join and LEFT join

INNER JOIN	LEFT JOIN
Return only the rows that have matching values in both tables	Returns all rows form the left tbale and the matching rows from the table
Excludes non matching fows form both tables	Including non matching rows from the left table, filling right side columns with NULL values
To find customers who have placed at least one order	To list all customers including those who haven't placed any orders
Combines rows from both tables only when there is a match in the join condition	Returns all rows from the left table, and the matching rows from the right table. Non-matching rows from the left table will still appear with NULL values for right-table columns.
NULL Handling Does not return any NULLs for unmatched rows (since unmatched rows are not included).	Returns NULL values for right-table columns where there's no match

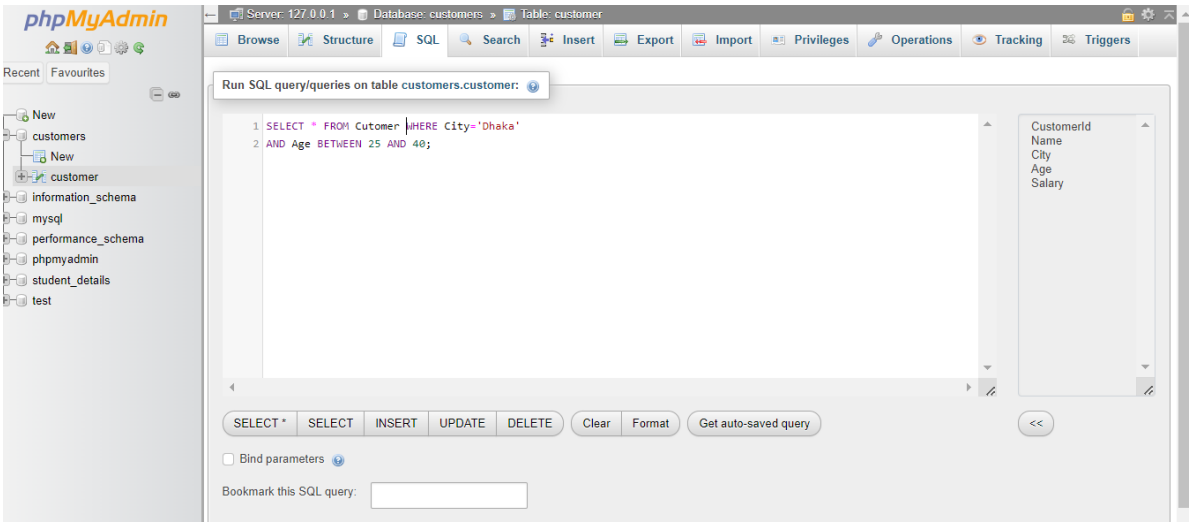
6.Create a table named Customer with the following fields:

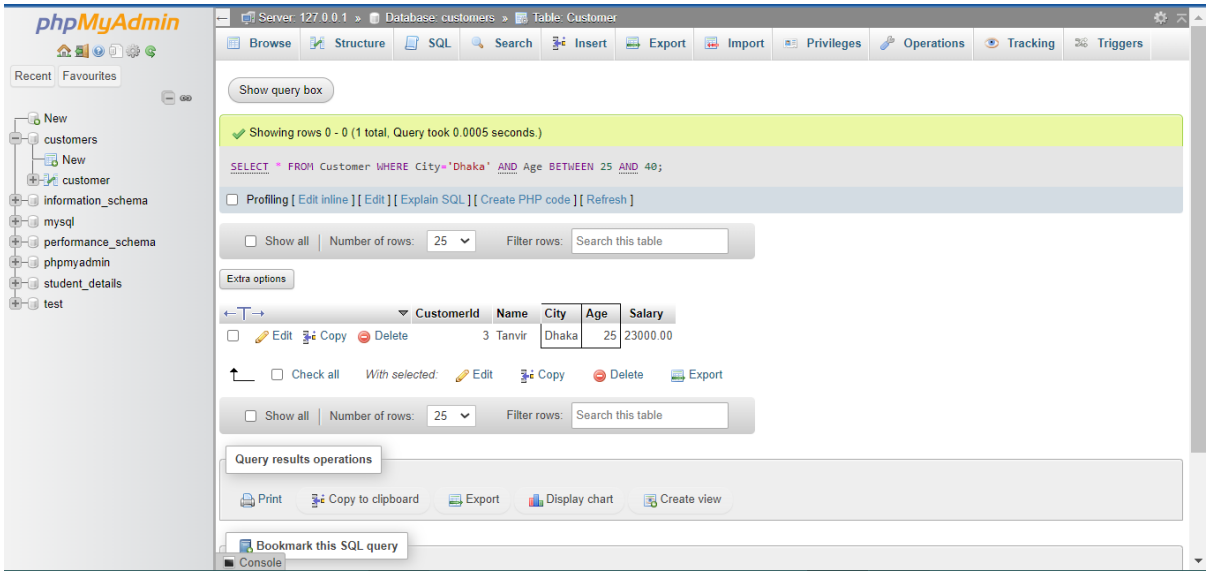


Insert at least 5 records into the table.

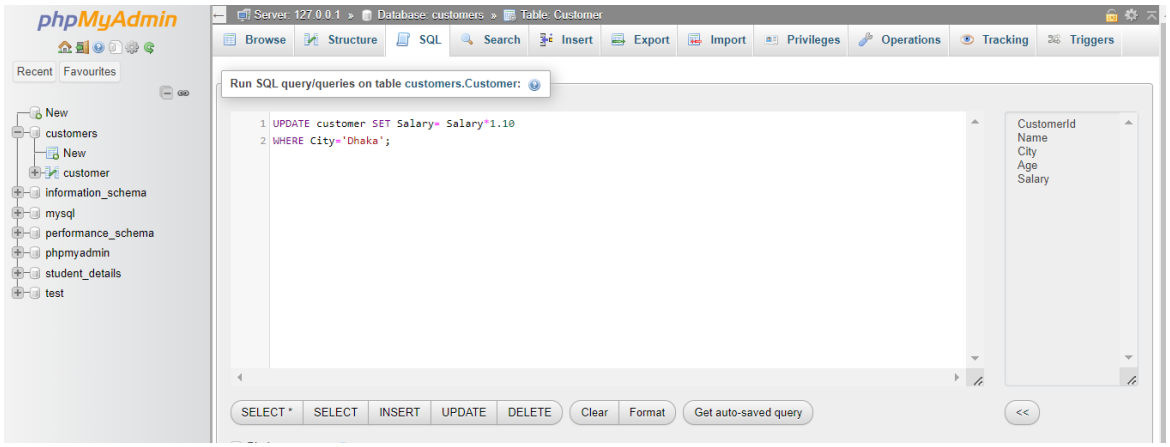


Display all customers whose city is ‘Dhaka’ and age is between 25 and 40.

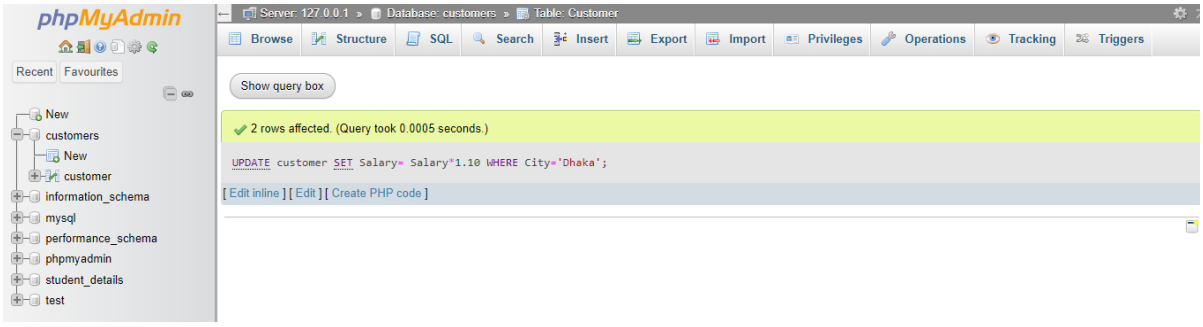




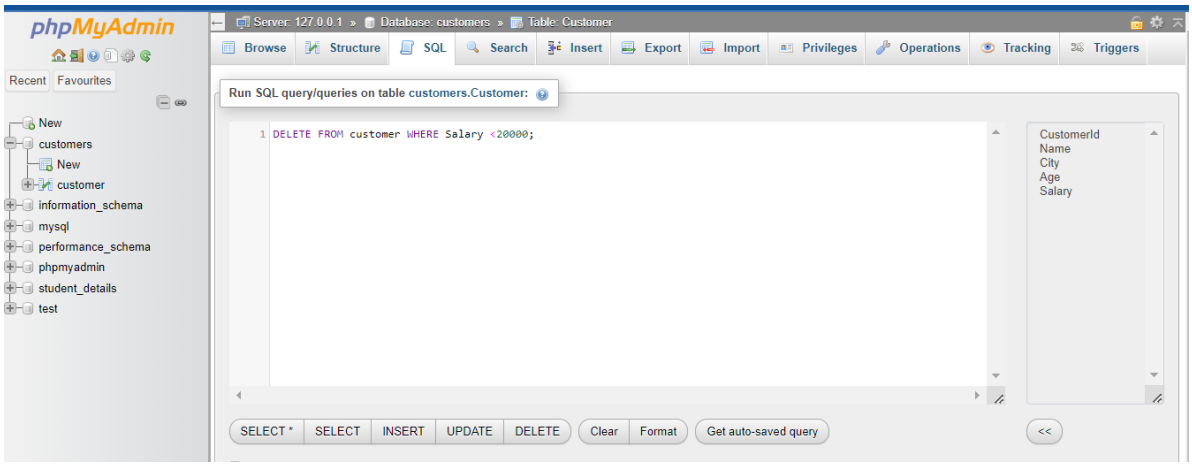
Increase the salary of all customar living in ‘Gulshan’ by 10%



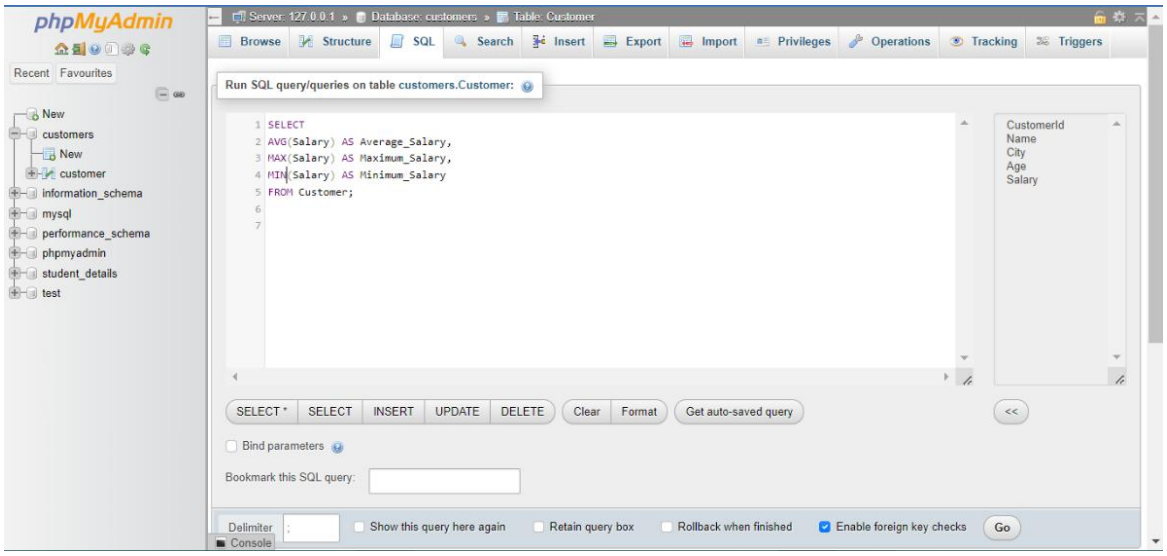
OUTPUT:



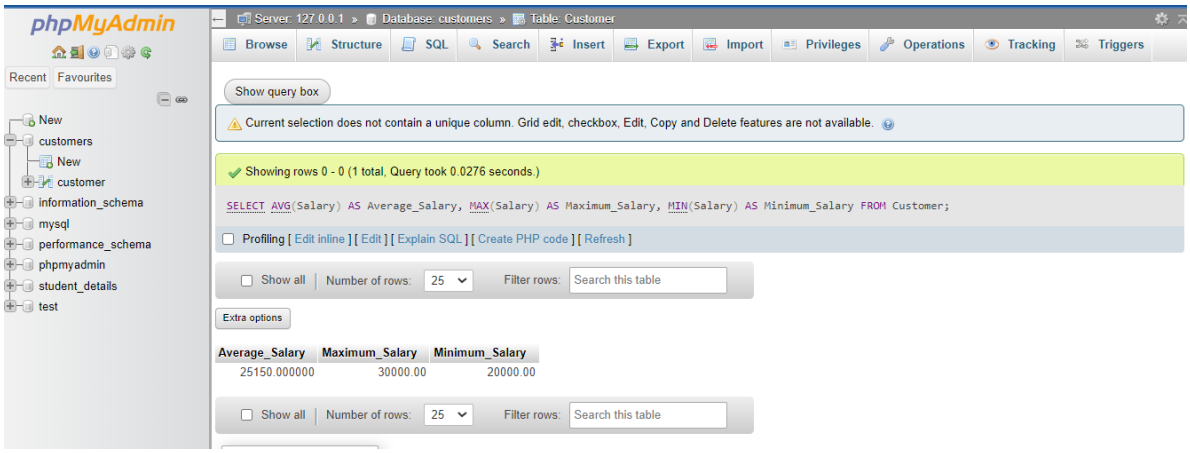
Delete cutomers whose asalary is less then 20000



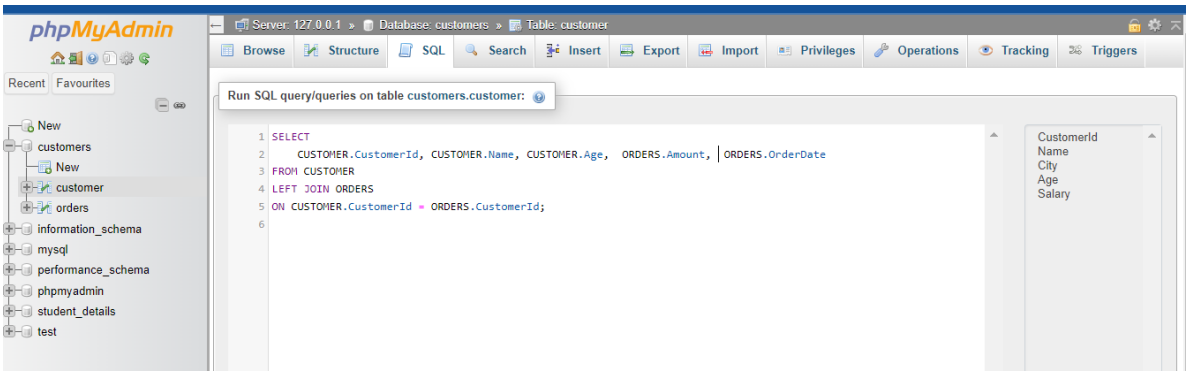
Display the average, maximum, and minimum salary using aggregate functions.

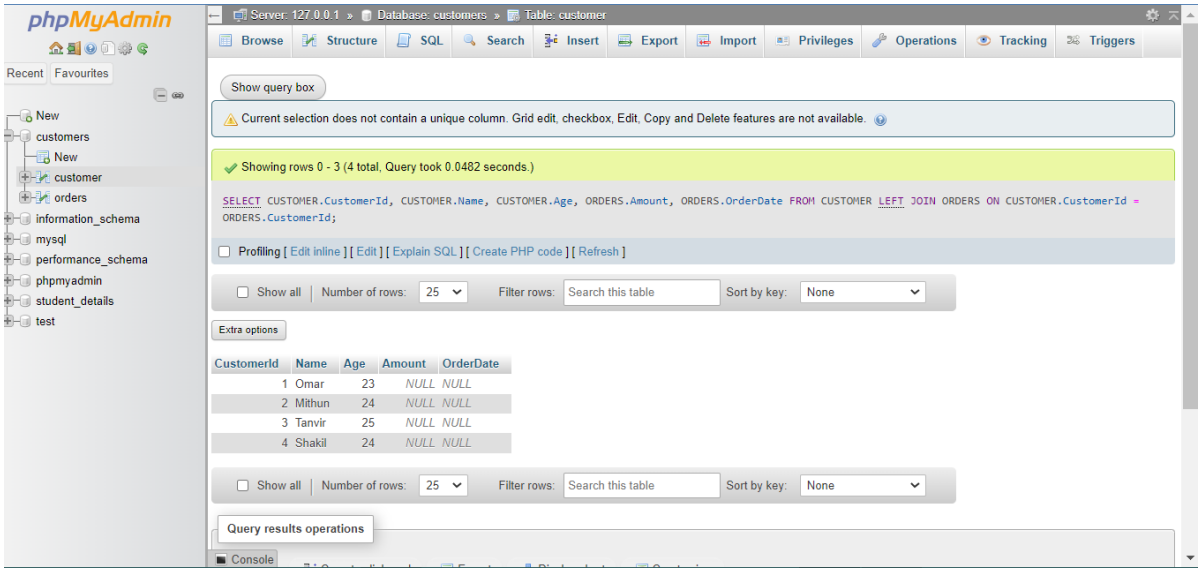


OUTPUT:

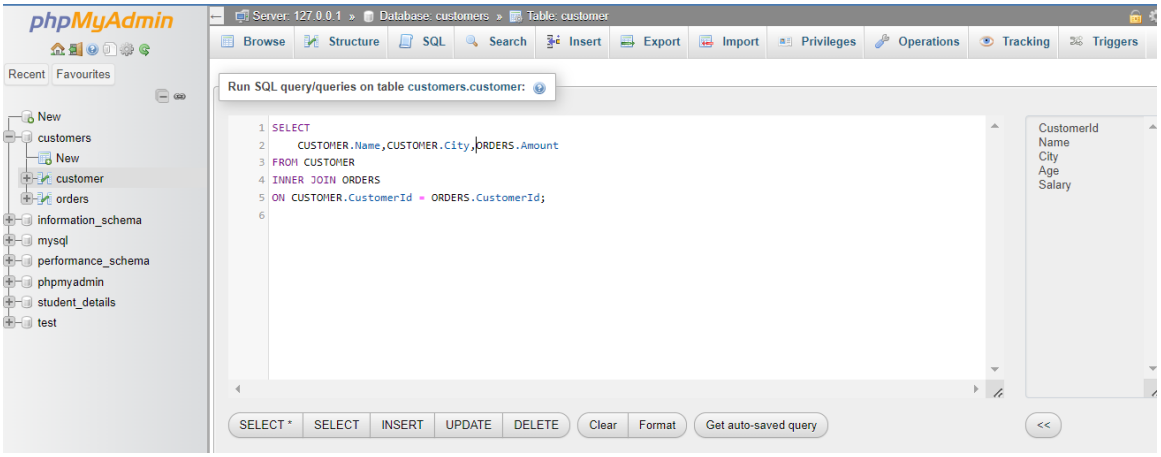


(a) LEFT JOIN to display all customers and their order amounts (if any)

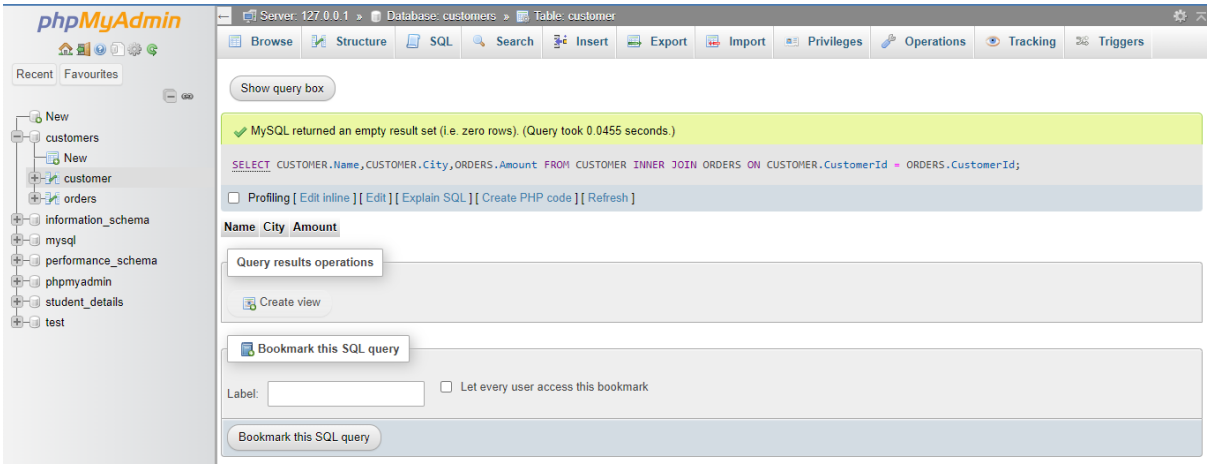




(b) INNER JOIN to display cutomerName,City and OrderAmount

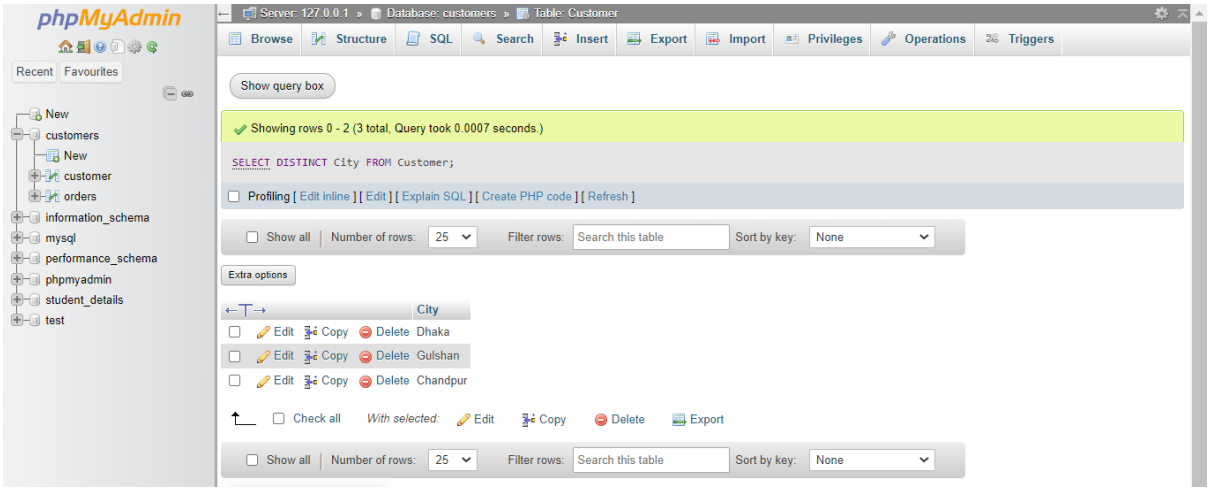


Result:

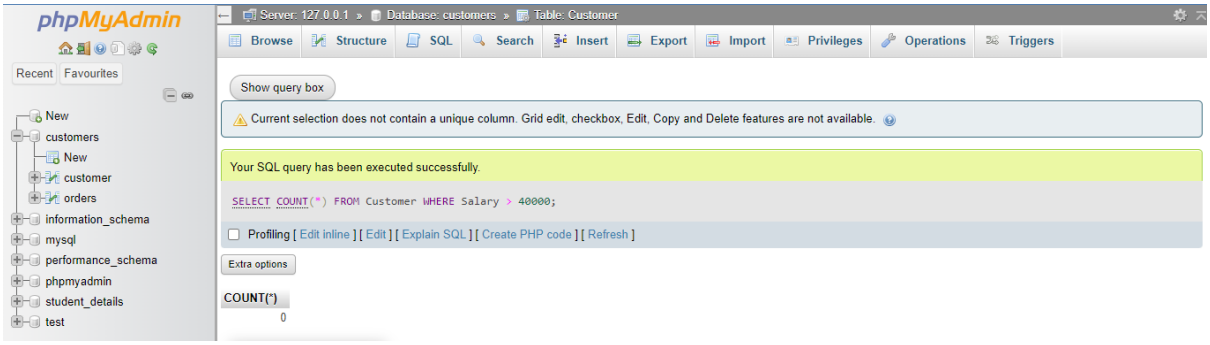


8. Write the SQL output or reasoning for the following queries (each 2 marks):

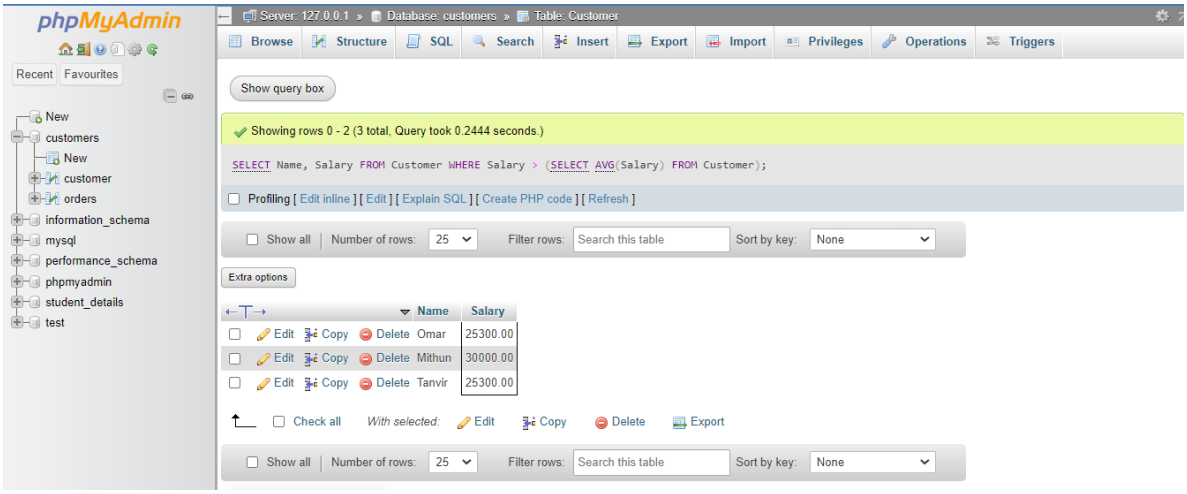
(1)



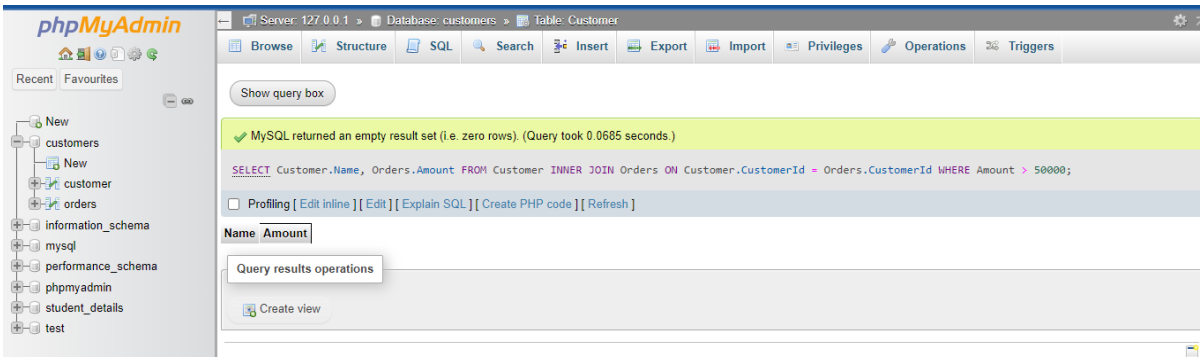
(2)



(3)



(4)



(5) The GROUP BY and HAVING clauses in SQL work in conjunction to aggregate and filter data based on groups.

1. GROUP BY Clause:

The GROUP BY clause is used to arrange rows that have the same values in one or more specified columns into summary rows. It is typically used with aggregate functions (e.g., COUNT(), SUM(), AVG(), MIN(), MAX()) to perform calculations on each group.

Code:

SELECT column1, aggregate_function(column2)

FROM table_name

GROUP BY column1;

In this example, the rows are grouped based on the unique values in column1, and the aggregate_function is applied to column2 within each of those groups

2. HAVING Clause:

The HAVING clause is used to filter the groups created by the GROUP BY clause. Unlike the WHERE clause, which filters individual rows before

grouping, HAVING filters groups after they have been formed and aggregate functions have been applied. This means HAVING can use aggregate functions in its conditions

Code:

```
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING condition_on_aggregate;
```

Here, after the data is grouped by column1 and the aggregate_function is calculated for each group, the HAVING clause then filters these groups, keeping only those that satisfy condition_on_aggregate.

How they work together:

The GROUP BY clause first creates logical groups of rows based on the specified columns. Then, aggregate functions are applied to these groups to produce a single summary row for each group. Finally, the HAVING clause evaluates a condition on these aggregated results or group properties, effectively filtering out groups that do not meet the specified criteria. This allows for precise analysis of grouped data by filtering based on aggregate values.