

## Ex No 2 :Implementation of SQL commands DDL,DML, DCL and TCL

### PROBLEM STATEMENT

In a growing company, the Marketing department embarks on new projects to enhance brand visibility. To streamline project management, a MarketingProjects table is created to track project details. Initially, it captures ProjectID, ProjectName, and Budget. As the team expands, a new requirement arises to monitor project statuses. The table structure is altered to include a Status column. Upon successful implementation, the MarketingProjects table facilitates efficient tracking of ongoing marketing initiatives. Eventually, when the need to restructure arises, the table is dropped, ensuring a clean database environment. This seamless process ensures effective project management and database maintenance.

- **Create Table:** Create a table named MarketingProjects to store project details including ProjectID, ProjectName, and Budget.
- **Insert Table:** Insert data into the MarketingProjects table to capture the details of new marketing projects, including ProjectID, ProjectName, Budget, and Status if available.
- **Alter Table:** Modify the table structure to include a new column named Status to track project statuses as the team expands.
- Retrieve the data stored in the table MarketingProjects.

### Input Table:

ProjectID	ProjectName	Budget
1	Project A	50000.00
2	Project B	75000.00
3	Project C	100000.00

### QUERY

```
create table MarketingProjects
```

```
(
```

```
ProjectID int,
```

```
ProjectName varchar(20),
```

```
Budget decimal(10,2)
```

```
);
```

```
insert into MarketingProjects values(1,'Project A',50000.00);
```

```
insert into MarketingProjects values(2,'Project B',75000.00);
```

```
insert into MarketingProjects values(3,'Project C',100000.00);
```

```
alter table MarketingProjects add Status varchar(10);
```

```
select*from MarketingProjects;
```

```
drop table MarketingProjects;
```

#### OUTPUT

ProjectID	ProjectName	Budget	Status
1	Project A	50000.00	Null
2	Project B	75000.00	Null
3	Project C	100000.00	Null

#### PROBLEM STATEMENT

In a bustling office environment, a company establishes its employee database to efficiently manage personnel information. The Employees table is created to centralize data on staff, capturing crucial details such as ID, Name, and Age. As the workforce grows, the system handles seamless data entry, populating the table with employee profiles. HR swiftly accesses and updates employee records, ensuring accuracy and compliance. The structured database enables quick retrieval and analysis of workforce demographics. This robust system optimizes HR operations, fostering a well-organized and productive workplace.

- Create a table named Employees to store employee information, including ID, Name, and Age.

#### Input Table:

ID	Name	Age
1	John Doe	30
2	Jane Smith	25
3	Michael Johnson	35
4	Emily Davis	28

#### QUERY

```
create table Employees
```

```
(
```

```
  ID int Primary Key,
```

```
  Name varchar(255),
```

```
  Age int
```

```
);
```

#### OUTPUT

ID	Name	Age
1	John Doe	30
2	Jane Smith	25
3	Michael Johnson	35
4	Emily Davis	28

## PROBLEM STATEMENT

Imagine you're developing a web application for a restaurant. The application needs to handle menu items and their corresponding prices. To achieve this, you decide to create a database table to store menu item details. Each menu item should have a unique identifier, and prices. Initially the prices are stored as a integer. But for the customer's wish the price should be stored precisely with two decimal places to ensure accurate billing.

- Given the existing "products" table in your database, your task is to alter the table to change the data type of the "price" column from INTEGER to DECIMAL
- After altering the table you need to display the table.

### Input Table:

product_id	product_name	price
1	Idly	20
2	Dosa	50
3	Biryani	80
4	Curd rice	40

## QUERY

```
alter table products modify column price decimal(10,2);
```

```
select *from products;
```

## OUTPUT

product_id	product_name	price
1	Idly	20.00
2	Dosa	50.00
3	Biryani	80.00
4	Curd rice	40.00

## PROBLEM STATEMENT

You are developing a contact management system for a small business. The system needs to store contact information for employees, clients, and suppliers. Each contact entry should include a unique identifier, first name, last name, email address, and phone number. Additionally, the system should enforce that each contact has a unique identifier. Due to privacy reasons the management decided to remove phone number from the database.

- Given the existing "contacts" table in your database, your task is to drop the "phone\_number" column from the table. Assume that the business requirements have changed, and there's no longer a need to store phone numbers for contacts.
- After dropping the column you need to display the table.

### Input Table:

contact_id	first_name	last_name	email	phone_number
1	Joe	doe	John.doe@example.com	1234567890

#### QUERY

```
alter table contacts drop column phone_number;
```

```
select * from contacts;
```

#### OUTPUT

contact_id	first_name	last_name	email
1	Joe	doe	John.doe@example.com

#### PROBLEM STATEMENT

In your role as a database analyst for a multinational corporation, you are preparing a report for the sales department that requires extracting specific data from the company's sales database. The report needs to focus on the performance metrics of salespeople, specifically highlighting their names and commission rates. This data will assist the management team in analyzing sales strategies and commission structures across various regions.

- Access the salesman table, and retrieve the names and commission rates of all salespeople to help in the compilation of the sales performance report.

#### Input Table:

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5003	Lausen Hen	San Jose	0.12
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5007	Paul Adam	Rome	0.13

#### QUERY

```
select name,commission from salesman;
```

#### OUTPUT

name	commission
James Hoog	0.15
Nail Knite	0.13

Lausen Hen	0.12
Pit Alex	0.11
Mc Lyon	0.14
Paul Adam	0.13

## PROBLEM STATEMENT

As part of routine database maintenance for an online retailer, you have been given the responsibility of managing the archival and cleanup of old transaction records. This involves removing entries from the order\_details table that are dated before January 1, 2020. This task is critical to ensure the database remains efficient and storage costs are minimized, while keeping the system's performance optimal for current transaction processing.

- Execute an SQL command to remove all order entries from the order\_details table where the order\_date falls before January 1, 2020, to maintain a streamlined database.

### Input Table:

order_id	customer_id	product_id	quantity	order_date
1	101	501	2	2019-12-25
2	102	502	1	2020-01-15
3	103	503	5	2018-11-03
4	104	504	1	2020-05-21
5	105	505	3	2019-07-19

## QUERY

```
delete from order_details where order_date < '2020-01-01';
```

## OUTPUT

order_id	customer_id	product_id	quantity	order_date
2	102	502	1	2020-01-15
4	104	504	1	2020-05-21

## PROBLEM STATEMENT

As an HR analyst at your company, you are tasked with preparing a report for an upcoming departmental review meeting. The focus of this report is on department 57, which has been undergoing significant changes and requires a detailed examination of its workforce. Your job is to extract the details of all employees assigned to this department to assist the management team in assessing staffing levels and planning for potential reassignments or further recruitment.

- Extract employee details from the emp\_details table for all individuals working in department 57.
- The required details are employee ID, first name, last name, and department number.

**Input Table:**

EMP_IDNO	EMP_FNAME	EMP_LNAME	EMP_DEPT
127323	Michale	Robbin	57
526689	Carlos	Snares	63
843795	Enric	Dosia	57
328717	Jhon	Snares	63
444527	Joseph	Dosni	47
659831	Zanifer	Emily	47
847674	Kuleswar	Sitaraman	57
748681	Henrey	Gabriel	47
555935	Alex	Manuel	57
539569	George	Mardy	27
733843	Mario	Saule	63
631548	Alan	Snappy	27
839139	Maria	Foster	57

**QUERY**

```
select * from emp_details where EMP_DEPT=57;
```

**OUTPUT**

EMP_IDNO	EMP_FNAME	EMP_LNAME	EMP_DEPT
127323	Michale	Robbin	57
843795	Enric	Dosia	57
847674	Kuleswar	Sitaraman	57
555935	Alex	Manuel	57
839139	Maria	Foster	57

**PROBLEM STATEMENT**

As a database administrator, you are tasked with demonstrating to new trainees how SQL can handle arithmetic operations directly within queries. This is particularly useful for generating calculated results on the fly, which is a common requirement in reporting and data analysis tasks. Your objective is to show how to perform a basic arithmetic operation, such as adding two numbers together, using SQL.

- Set up a table named Calculations, populate it with sample data, and then execute a SQL query that displays the result of an arithmetic expression involving the data in the table.

**Input Table:**

id	num1	num2
1	50	150

#### QUERY

```
select num1,num2,(num1+num2) as AdditionResult from Calculations;
```

#### OUTPUT

Num1	Num2	AdditionResult
50	150	200

#### PROBLEM STATEMENT

As a data manager for an e-commerce platform, you are preparing for a major seasonal sale event. Part of the marketing strategy is to highlight premium products to attract high-end buyers. You are assigned to curate a list of the top five most expensive items currently available, which will be prominently featured on the homepage to draw attention and drive sales.

- Extract the names and prices of the five most expensive products listed in the product\_data table to assist the marketing team in creating visually appealing content for the homepage.

#### Input Table:

product_id	name	description	price	category_id
1	Luxury Watch	A luxury wristwatch	9999.99	1
2	Designer Bag	A high-end designer bag	7500	2
3	High-End Laptop	Top-spec performance laptop.	3000	3
4	Smartphone	Latest model smartphone	1200	4
5	Gourmet Coffee Maker	State-of-the-art coffee machine	450	5
6	Elegant Vase	Antique decorative vase	200	6

#### QUERY

```
SELECT name, price FROM product_data ORDER BY price DESC LIMIT 5;
```

#### OUTPUT

name	price
------	-------

Luxury Watch	9999.99
Designer Bag	7500
High-End Laptop	3000
Smartphone	1200
Gourmet Coffee Maker	450

## PROBLEM STATEMENT

You are developing a financial application that manages bank accounts. As part of the application, you need to implement a feature that allows fund transfers between two accounts. To ensure data integrity and consistency, you decide to execute the fund transfer operation within a transaction.

This involves two accounts, Account A and Account B. Account A has an ID of 123 and an initial balance of \$1000, while Account B has an ID of 456 and an initial balance of \$2000. The goal is to transfer \$500 from Account A to Account B.

To achieve this, you utilize SQL transactions. You start a transaction, deduct \$500 from Account A, add \$500 to Account B, and then commit the transaction to make the changes permanent. Finally, you verify the updated account balances by selecting data from the Accounts table.

- Create Accounts table: Define a table named Accounts with columns for AccountID, AccountName, and Balance.
- Insert sample data: Insert sample records into the Accounts table to represent Account A and Account B, each with their respective AccountID, AccountName, and initial Balance.
- Start a transaction: Begin a transaction to ensure that the fund transfer operation is atomic and isolated from other database operations.
- Deduct \$500 from Account A: Update the Balance of Account A by deducting \$500.
- Add \$500 to Account B: Update the Balance of Account B by adding \$500.
- Commit the transaction: Commit the transaction to make the changes permanent in the database.
- Verify the transaction: Select and display the data from the Accounts table to verify that the fund transfer operation was successful.

### Input Table (Accounts):

AccountID	AccountName	Balance
123	Account A	1000.00
456	Account B	2000.00

## QUERY

**begin;**



```

create table Accounts
(
AccountID int ,
AccountName varchar(20),
Balance decimal(10,2)
);
INSERT INTO Accounts values(123,'Account A',1000.00);
INSERT INTO Accounts values(456,'Account B',2000.00);
update Accounts set Balance=Balance-500 where AccountID=123;
update Accounts set Balance=Balance+500 where AccountID=456;
select* from Accounts;
commit;

```

(or)

```

CREATE TABLE Accounts (
AccountID INT PRIMARY KEY,
AccountName VARCHAR(100),
Balance DECIMAL(10, 2)
);
-- Insert sample data into Accounts table
INSERT INTO Accounts (AccountID, AccountName, Balance) VALUES
(123, 'Account A', 1000.00),
(456, 'Account B', 2000.00);
-- Start a transaction
START TRANSACTION;
-- Deduct $500 from Account A
UPDATE Accounts
SET Balance = Balance - 500
WHERE AccountID = 123;
-- Add $500 to Account B

```

UPDATE Accounts

SET Balance = Balance + 500

WHERE AccountID = 456;

-- Commit the transaction

COMMIT;

-- Select data from Accounts table

SELECT \* FROM Accounts;

**OUTPUT:**

AccountID	AccountName	Balance
123	Account A	500.00
456	Account B	2500.00

**PROBLEM STATEMENT**

A financial institution, needing to empower their analyst, creates a table 'Accounts' to store account details. Sample data is inserted. To facilitate analysis, a user 'fin\_analyst' is established with SELECT privileges on 'Accounts'. Finally, data from 'Accounts' is queried for analysis.

- Set up a database environment for financial analysis.
- Create an 'Accounts' table to store account information.
- Populate it with sample data. Establish a user 'fin\_analyst' with SELECT privileges. Reload privileges to ensure changes take effect.
- Finally, retrieve data from the 'Accounts' table for analysis.

**Input Table: Accounts**

AccountID	AccountName	Balance
1	Savings	5000.00
2	Checking	2500.00
3	Investment	10000.00

**Output Table:**

AccountID	AccountName	Balance
1	Savings	5000.00
2	Checking	2500.00
3	Investment	10000.00

**QUERY**

start transaction;

create table Accounts

```

(
AccountID int,
AccountName varchar(225),
Balance varchar(230)
);

Insert into Accounts values(1, 'Savings',5000.00);

insert into Accounts values (2, 'Checking', 2500.88);

insert into Accounts values (3, 'Investment',10000.00);

select * from Accounts;

commit;

```

## OUTPUT

AccountID	AccountName	Balance
1	Savings	5000.00
2	Checking	2500.00
3	Investment	10000.00

## PROBLEM STATEMENT

As the database administrator at a bustling city library, you face a common issue where multiple librarians often attempt to update the inventory counts of books simultaneously. This situation requires a robust solution to manage concurrency and maintain data integrity. The challenge lies in implementing a system where updates to the book quantities in the library's database are processed in an orderly manner, ensuring that the transaction initiated first by any librarian is completed before others.

### Task:

- Ensure that updates to the book inventory by one librarian do not interfere with or overwrite concurrent updates by another librarian.
- Test the system by simulating a scenario where two librarians update the quantity of the same book at the same time, committing the transaction that started first before the other.

### Input Table:

BookID	Title	Author	Quantity	LastUpdated
1	1984	George Orwell	30	2024-04-15 09:00:00
2	To Kill a Mockingbird	Harper Lee	20	2024-04-15 09:00:00

## QUERY

```

start transaction;

```

update Books set Quantity=28 where BookID=1;

update Books set LastUpdated='2024-04-15 09:02:00' where BookID=1;

select \* from Books where BookID=1;

commit

#### OUTPUT

BookID	Title	Author	Quantity	LastUpdated
1	1984	George Orwell	28	2024-04-15 09:02:00

#### RESULT

### Ex No 3 : Queries to demonstrate implementation of Integrity Constraints

#### PROBLEM STATEMENT

You've been tasked with enhancing the database schema for a school management system. Currently, the system has a table named "students" that stores student information, including their student ID, name, and marks. Now, the management wants to associate each student with the

department they attend. To achieve this, you need to add a new column to the "students" table named "department\_id" to store the foreign key referencing the "departments".

- Given the existing "students" table in your database, your task is to alter the table by adding a column "department\_id"
- After altering you need to display the table

#### Input Table:

Name	Marks
John Doe	90
Jane Smith	85
Michael Johnson	65
Emily Davis	88

#### QUERY:

```
create table students(ID int, Name varchar(20),Marks Int,department_id varchar(20));
```

```
insert into students value(1,'John Doe',90,null);
```

```
insert into students value(2,'Jane Smith',85,null);
```

```
insert into students value(3,'Michael Johnson',65,null);
```

```
insert into students value(4,'Emily Davis',88,null);
```

```
select * from students;
```

#### OUTPUT:

**Problem Statement:**

In a company, the HR department is updating the employee database schema to enhance data management. They aim to remove the EmergencyContactName column from the Employees table while preserving existing data. To achieve this, they plan to create a temporary table, copy relevant data, and then drop the original column.

- Drop any existing Employees table to avoid conflicts.
- Create a new Employees table with necessary columns, including EmployeeID, Name, DepartmentID, EmergencyContactName, and EmergencyContactPhone.
- Remove the EmergencyContactName column from the Employees table by creating a temporary table (temp\_Employees) and copying data from the original table, excluding the EmergencyContactName column.

**Input Table:**

- EmployeeID: INT, Primary Key
- Name: VARCHAR(255)
- DepartmentID: INT
- EmergencyContactName: VARCHAR(255)
- EmergencyContactPhone: VARCHAR(20)

**Query:**

```
create table Employees(EmployeeID int primary key,Name varchar(225),  
DepartmentID int,EmergencyContactName varchar(255),EmergencyContactPhone varchar(20));
```

**Output:****Result:**

## Ex No 4 : Practice of Inbuilt functions

### Problem Statement

In an expansive digital marketplace, an e-commerce conglomerate manages a vast database of customer transactions. Amidst the dynamic landscape of online shopping, the company endeavors to maintain a pulse on consumer activity. Within this ecosystem, there arises a requirement to periodically evaluate recent customer purchases to glean insights into evolving consumer behaviors and trends.

- Develop a SQL query to analyze the "Purchases" table within the existing database infrastructure, aiming to identify customers who have made purchases within the last 30 days.

### Input Table:

PurchaseID	CustomerID	ProductID	PurchaseDate
1	1	1	2024-03-15
2	2	2	2024-04-10
3	1	3	2024-04-20

### Query:

```
select
CASE
WHEN EXISTS(
  select *
  from Purchases
  where purchaseDate >=DATE_SUB(CURRENT_DATE(),INTERVAL 30 DAY)
)
THEN 'Yes'
else 'No'
END
as Purchases_Last_30_Days;
```

### Output:

Purchases_Last_30_Days
Yes

### Problem Statement:

In a retail database, tables for products, categories, and sales are set up. Products are categorized into Electronics, Books, and Stationery, each with distinct items and prices. Sales records show transactions for various products. Queries are executed to find products without sales, combine names and prices of Electronics and Books, and identify electronics priced higher than any book. These queries help monitor inventory, analyze sales trends, and adjust pricing strategies for maximizing profits.

- Find products with no sales records.
- Perform a union of names and prices from Electronics and Books categories.
- Select electronics with rounded prices higher than any book's price.

### Input Table:

#### Category Table:

category_name
Electronics
Books
Stationery

#### Product Table:

product_id	category	name	price
1	Electronics	Laptop	1200
2	Electronics	Camera	450
3	Books	Database Systems	55
4	Books	SQL For Dummies	20
5	Stationery	Notebook	5
6	Stationery	Pen Set	18.5
7	Electronics	Smartphone	700
8	Electronics	Tablet	350
9	Books	Modern SQL	45
10	Books	Data Science for Business	40
11	Stationery	Stapler	12
12	Stationery	Marker Set	6.5

#### Sales Table:

sale_id	product_id	sale_date	quantity
1	1	2023-01-15	2
2	1	2023-02-20	1
3	2	2023-03-25	3
4	7	2023-04-15	4
5	7	2023-05-10	1
6	9	2023-06-12	2

### Query:



```

select * from products where product_id not in(select product_id from sales);

select name,price from products where category in('Electronics','Books') order by category desc ;

select product_id,name,round(price) as rounded_price from products where
category='Electronics';

```

**Output:**

## OUTPUT

### PROBLEM STATEMENT

In a retail e-commerce platform, orders are managed through a database. Each order has a unique ID, customer ID, order date, and status. For instance, a customer places an order for various items, such as

electronics or clothing, with different statuses like "Pending," "Shipped," or "Delivered." To maintain operational efficiency, the company needs to track the total number of orders for each status. This helps in monitoring the progress of orders, ensuring timely delivery, and addressing any issues promptly.

- Your task should use the COUNT function to calculate the number of orders for each status.

**Input Table:**

order_id	customer_id	order_date	order_status
1	101	2023-01-15	Delivered
2	102	2023-01-16	Shipped
3	103	2023-01-17	Pending
4	104	2023-01-18	Shipped
5	105	2023-01-19	Delivered
6	106	2023-01-20	Pending
7	107	2023-01-21	Delivered

### QUERY

```

select order_status ,count(order_status)as total_orders
from orders group by order_status order by order_status;

```

### OUTPUT

Order_status	Total_orders
Delivered	3
Pending	2
Shipped	2

## PROBLEM STATEMENT

In an online marketplace, a retail platform monitors product prices across categories. Using a database, they track product details such as category, name, and price. To analyze pricing strategies, they calculate the median price for products in each category. This helps in understanding the pricing distribution within categories, enabling the platform to adjust prices competitively and optimize revenue. The query fetches median prices dynamically, aiding in strategic decision-making for pricing and product placement.

- Calculate the median price of products within each category to assess pricing distributions and optimize pricing strategies for a retail platform.

### Input Table:

product_id	category	name	price
1	Electronics	Laptop	1200
2	Electronics	Camera	450
3	Books	Database Systems	55
4	Books	SQL for Dummies	20
5	Stationery	Notebook	5
6	Stationery	Pen Set	18.5
7	Electronics	Smartphone	700
8	Electronics	Tablet	350
9	Books	Modern SQL	45
10	Books	Data Science for Business	40
11	Stationery	Stapler	12
12	Stationery	Marker set	6.5

## QUERY

```
set @ri:=-1;
select category,avg(price) as median_price from (
select @ri:=if(@pg=Market.category,@ri+1,0) as ri,
@pg:=Market.category as category,
Market.price as price from Market order by Market.price,
Market.category) as p
where p.ri in(floor(@ri/2),ceil(@ri/2))
group by category
order by category;
```

(or)

```
Ln 1: Col 1
1 alter table products modify column price double;
2 select
3 category, avg(price) as median_price
4 from (select p1.category, p1.price,
5 (
6 select count(*)
7 from products p2
8 where p2.category=p1.category and p2.price<=p1.price)
9 as price_rank,
10 (
11 select count(*) from products p3
12 where p3.category=p1.category)
13 as total_rows from products p1) as ranked_products
14 where
15 price_rank between total_rows/2 and (total_rows/2)+1 group by category
16 order by category;
```

## OUTPUT

category	Median_price
Books	42.5
Electronics	575
Stationary	9.25

## PROBLEM STATEMENT

Within the expansive realm of cinema, a renowned movie database meticulously catalogues an extensive array of films and their associated talent. Aspiring filmmakers, cinephiles, and industry professionals alike rely on this comprehensive repository to explore the intricate tapestry of cinematic creations. Amidst this rich tapestry, emerges a desire to uncover the collaborative endeavors between directors and actors, particularly in the context of the visionary filmmaker 'Christopher Nolan'.

Harnessing the power of data analysis, the database aims to unveil the names of actors who have graced the screen in movies directed by this iconic filmmaker. Through this exploration, the database seeks to shed light on the dynamic relationships forged within the realm of filmmaking and celebrate the contributions of actors to cinematic masterpieces.

- Write a query to retrieve the names of actors who have appeared in movies directed by 'Christopher Nolan'.
- The necessary tables have already been created and populated with sample data.
- Utilize subqueries to select the relevant data from the provided tables.
- Ensure that your query returns distinct actor names

**Input Table:**

ActorID	Name
1	Leonardo Dicaprio
2	Tom Hardy
3	Michael Caine
4	Anne Hathaway
5	Christian Bale

**Director Table:**

DirectorID	Name
1	Christopher Nolan

**Movies Table:**

MovieID	Title	DirectorID
1	Inception	1
2	The Dark Knight	1
3	Interstellar	1
4	The Prestige	1
5	Dunkirk	1

**Cast Table:**

MovieID	ActorID
1	1
1	2
1	3
1	4
2	5
2	3
3	1
3	3
3	5
4	1
4	3
4	4
5	2
5	5

## QUERY

```
select Name from Actors where ActorID in
(select distinct ActorID from Cast where MovieID in
(select MovieID from Movies where DirectorID in
(select DirectorID from Directors)) );
```

## OUTPUT

Name
Leonardo Dicaprio
Tom Hardy
Michael Caine
Anne Hathaway
Christian Bale

## PROBLEM STATEMENT

In a retail store, managers aim to identify the best-performing salesperson in terms of revenue generated. By analyzing sales data, they can reward high achievers and provide additional training or support to those who may need improvement. The query calculates the top-performing salesperson based on the total sales amount achieved. This information guides managerial decisions, enhances employee motivation, and fosters a competitive yet supportive work environment conducive to achieving sales targets and maximizing profits.

- Identify the product with the highest total sales amount to determine the top-selling item in the sales data.

### Input Table:

sale_id	product_id	quantity_sold	sale_amount
1	101	5	500.5
2	102	3	300.25
3	101	2	200
4	103	10	3500
5	102	7	1500.5
6	101	3	800
7	104	5	700.75

## QUERY

```
select product_id as top_selling_product,sale_amount as total_sales_amount  
  
from sales where sale_amount=(select max(sale_amount) from sales) ;
```

## OUTPUT

top_selling_product	total_sales_amount
103	3500.00

## RESULT

### Ex No 5 : Implementation of Join and Nested Queries and Set operators

#### PROBLEM STATEMENT

In a multinational retail corporation, sales data for different years is stored in separate tables. The company aims to identify products consistently popular across two consecutive years for strategic stocking decisions. By comparing sales records from 2020 and 2021, they can pinpoint items with consistent demand, ensuring sufficient inventory levels. This approach aids in optimizing procurement, minimizing stockouts, and maintaining customer satisfaction by consistently offering sought-after products.

#### AIM

- Your task Identify products with consistent demand across different years by finding common product IDs present in both the 2020 and 2021 sales tables.

**Input Table:****sales\_2020**

sale_id	product_id
1	101
2	102
3	103
4	104
5	105

**sales\_2021**

sale_id	product_id
1	102
2	103
3	105
4	106
5	107

**QUERY**

```
select product_id from sales_2021 where product_id in(select product_id from sales_2020);
```

(or)

```
select product_id from sales_2020  
INTERSECT  
select product_id from sales_2021;
```

**OUTPUT**

Product_id
102
103
105

PROBLEM STATEMENT

You are managing the database for an online store. The tasks involve creating tables for customer information, product data, and order details, establishing relationships among them, and ensuring efficient data retrieval with indexes. You also need to manage data entry for new customers, products, and orders.

- Create tables to store information about customers, products, and orders, with necessary relationships and constraints.
- **customer\_information:** Stores details about customers, including their ID, name, email, phone number, and address.
- **product\_data:** Holds information about products, such as product ID, name, description, price, and category ID.
- **order\_details:** Records details of customer orders, linking products and customers through foreign key.

Input Table:

Customer Information:

customer_id	name	email	phone_number	address
1	John Doe	john@example.com	1234567890	123 Main Street, City, Country

Product Data:

product_id	name	description	price	category_id
101	Smartphone	High-quality smartphone with advanced features	599.99	1

Order Details:

order_id	customer_id	product_id	quantity	order_date
1001	1	101	2	2024-04-15

QUERY



```
Ln 1: Col 1
1 - CREATE TABLE customer_information(
2   customer_id int primary key,
3   name varchar(100),
4   email varchar(70),
5   phone_number varchar(100),
6   address varchar(30));
7
8 - CREATE TABLE product_data(
9   product_id int primary key,
10  name varchar(100),
11  description varchar(400),
12  price decimal(10,2),
13  category_id int);
14
15
16 - CREATE TABLE order_details(
17   order_id int primary key,
18   customer_id int,
19   product_id int,
20   quantity int,
21   order_date date,
22   foreign key(customer_id) references customer_information(customer_id),
23   foreign key(product_id) references product_data(product_id));
```

## OUTPUT

.....

## PROBLEM STATEMENT

Imagine you're managing a university's student database. You have two tables, Students and Courses, storing information about students' grades and the courses they are enrolled in. The students table contains data about each student, including their FirstName, Grade, and the CourseID they are enrolled in. The Courses table stores information about various courses offered by the university, including the CourseID and CourseName. Finding Students with 'A' Grades. In this scenario, you want to identify all students who have achieved an 'A' grade in their courses.

- Must use inner join with where clause

**Input Table:**

**Students**

**Table**

StudentID	FirstName	Grade	CourseID
1	Alice	A	101

2	Bob	B	102
3	Charlie	C	103
4	David	D	102
5	Eve	E	101

#### Course Table

CourseID	CourseName
101	Math
102	Science
103	English
104	History

#### QUERY

select Students.FirstName,Courses.CourseName from Students inner join

Courses On Students.CourseId=Courses.CourseID

where Students.Grade='A';

#### OUTPUT

FirstName	CourseName
Alice	Math
Charlie	English

#### PROBLEM STATEMENT

Imagine you work at a medium-sized company, and the HR department recently assigned employees to different departments. You want to retrieve a list of employees along with their

corresponding department names. This information will help you understand which employees belong to which departments. Note: Department, Employees are keep in two different table.

- Must use inner join.

#### **Input Table:**

##### **Employee**

#### **Table:**

EmployeeID	FirstName	LastName	DeptID
1	John	Doe	101
2	Jane	Smith	102
5	Robert	Brown	103

#### **Department Table:**

DeptID	DepartmentName
101	Sales
102	Marketing
103	HR

#### **QUERY**

```
select Employees.FirstName,Department.DepartmentName from Employees join  
Department on Employees.DeptID=Department.DeptID;
```

#### **OUTPUT**

#### **PROBLEM STATEMENT**

In a bustling city, a food delivery service employs several delivery personnel to ensure prompt and efficient delivery of orders from various restaurants to customers' doorsteps. As part of their commitment to customer satisfaction, the company regularly collects feedback through ratings provided by customers after each delivery. The management understands the importance of

assessing the performance of their delivery personnel to maintain high service standards.

- Implement query to calculate the average rating received by each delivery person in the food delivery service.
- The query should retrieve the average rating for each delivery person by averaging the ratings provided by multiple customers for each delivery person.

**Input Table:**

FoodID	FoodName	DeliveryBoyName	Rating
1	Pizza	John	4
2	Sushi	Lisa	5
3	Burger	John	4
4	Pasta	Mike	3
5	Tacos	Lisa	4
6	Salad	John	5
7	Sandwich	Mike	3
8	Stir Fry	Lisa	4
9	Curry	John	5
10	Ramen	Mike	4

**QUERY**

Ln 2: Col 1

```
1 select DeliveryBoyName, AVG(Rating) as AverageRating
2 from DeliveryRatings group by DeliveryBoyName;
```

**OUTPUT**

DeliveryBoyName	AverageRating
John	4.5000
Lisa	4.3333
Mike	4.3333

## PROBLEM STATEMENT

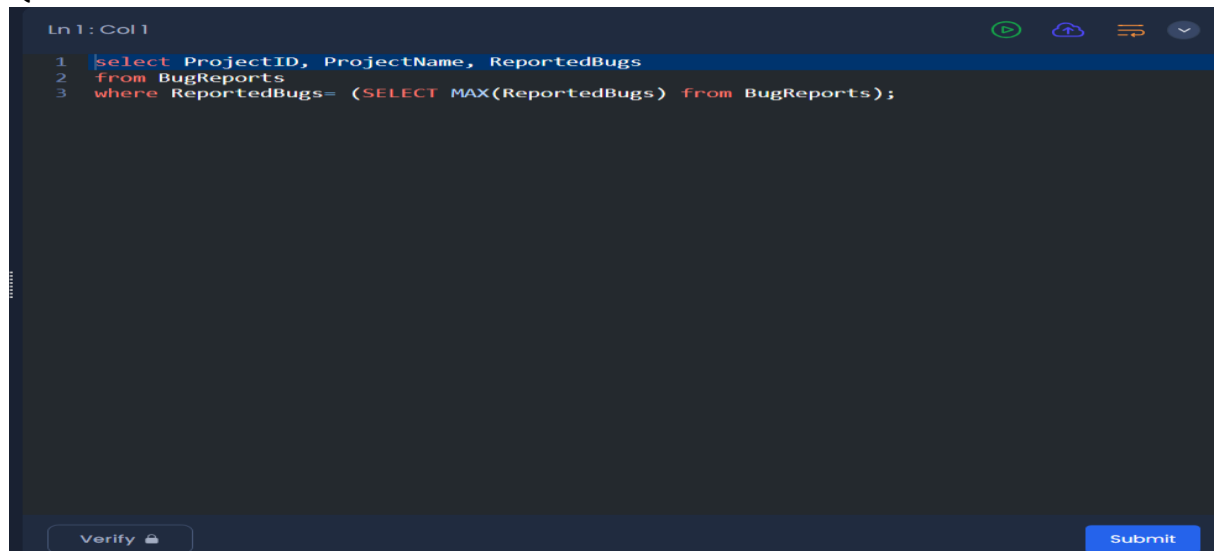
In a software development company, ensuring product quality is paramount. The company meticulously tracks the number of bugs reported for each project to prioritize bug fixes effectively. The project management team recognizes the criticality of promptly addressing projects with the highest number of reported bugs to maintain customer satisfaction and uphold the company's reputation for delivering reliable software solutions.

- Design a query to identify the project(s) with the highest number of reported bugs.
- The query should retrieve the ProjectID, ProjectName, and ReportedBugs for the project(s) with the maximum number of reported bugs.

**Input Table:**

ProjectID	ProjectName	ReportedBugs
1	Project A	10
2	Project B	8
3	Project C	15
4	Project D	12
5	Project E	20
6	Project F	5

## QUERY

A screenshot of a SQL query editor interface. The editor has a dark background with a light blue header bar. The header bar contains the text 'Ln1: Col1' and several icons (a green play button, a blue cloud icon, a red and blue icon, and a dropdown arrow). The main area of the editor shows a SQL query with line numbers 1, 2, and 3 on the left. The query is: 

```
1 select ProjectID, ProjectName, ReportedBugs
2 from BugReports
3 where ReportedBugs= (SELECT MAX(ReportedBugs) from BugReports);
```

 At the bottom of the editor, there is a 'Verify' button with a lock icon and a 'Submit' button.

## OUTPUT

ProjectID	ProjectName	ReportedBugs
4	Project D	12

## PROBLEM STATEMENT

In a healthcare organization, the quality assurance team analyzes surgical procedure durations to ensure optimal patient care. By examining data from the procedure and surgery tables, they identify the shortest and longest durations for each surgical procedure type, enabling process optimization and maintaining high standards of efficiency and safety.

- Write an SQL query to identify the shortest and longest duration of surgeries for each surgical procedure type in the healthcare organization's database.

### Input Table:

ProcedureID	ProcedureType
1	Appendectomy
2	Hernia Repair
3	Knee Surgery
4	Cataract Surgery

### Surgery Table:

SurgeryID	ProcedureID	SurgeryDuration (in minutes)
1	1	60
2	1	75
3	2	90
4	2	120
5	3	180
6	3	150
7	4	45
8	4	60
9	4	30

## QUERY

```
select Procedures.ProcedureType,
min(Surgery.SurgeryDuration)      as
MinDuration,
max(Surgery.SurgeryDuration)      as
MaxDuration from Procedures
join Surgery on Procedures.ProcedureID=
Surgery.ProcedureID
group      by
ProcedureType;
```

## OUTPUT

## PROBLEM STATEMENT

Operating within a niche of personalized travel experiences, a travel agency prioritizes understanding customer booking behavior to deliver tailored travel packages. The marketing team's objective is to design precise promotional offers by gauging the frequency of bookings made by each customer. This approach empowers the agency to align its marketing strategies with the varied preferences and behaviors of its clientele, thereby enhancing customer satisfaction and fostering loyalty.

- Implement SQL query to determine the total number of bookings made by each customer in a travel agency's database.

### Input Table:

#### Customer Table:

CustomerID	CustomerName
1	John Smith
2	Emma Johnson
3	Michael Lee
4	Sarah Brown

#### Booking Table:

BookingID	CustomerID	BookingDate
1	1	2024-04-01
2	1	2024-04-03
3	2	2024-04-02
4	3	2024-04-01
5	3	2024-04-03
6	3	2024-04-04
7	4	2024-04-02
8	4	2024-04-03
9	4	2024-04-04

## QUERY

```
SELECT c.CustomerID, c.CustomerName, COUNT(b.BookingID) AS TotalBookings FROM Customer c
JOIN Booking b ON c.CustomerID = b.CustomerID GROUP BY c.CustomerID, c.CustomerName;
```

**OUTPUT**

**RESULT**

**Ex No : 6 Implementation of virtual tables using Views**



## PROBLEM STATEMENT

You are a manager at a company and you need to review the details of all employees frequently. You've created a view named "EmployeeDetails" that displays the EmployeeID, FirstName, and LastName columns from the Employees table for easy access to this information.

### Tasks:

- Create a view named EmployeeDetails that displays the EmployeeID, FirstName, and LastName columns from the Employees table.
- Test the view by selecting all rows from it.

### Input Table:

EmployeeID	FirstName	LastName
1	John	Doe
2	Jane	Smith
3	Alice	Johnson

### Output Table:

EmployeeID	FirstName	LastName
1	John	Doe
2	Jane	Smith
3	Alice	Johnson

### QUERY:

```
create table Employees
```

```
(
```

```
EmployeeID int,
```

```
FirstName varchar(20),
```

```
LastName varchar(20)
```

```
);
```

```
insert into Employees values(1,'John','Doe');
```

```
insert into Employees values(2,'Jane','Smith');
```

```
insert into Employees values(3,'Alice','Johnson');
```

```
create view EmployeeDetails AS
```

```
select EmployeeID ,FirstName, LastName from Employees;
```

```
select* From EmployeeDetails;
```

### OUTPUT

EmployeeID	FirstName	LastName
1	John	Doe

2	Jane	Smith
3	Alice	Johnson

## PROBLEM STATEMENT

As a human resources manager, you need to manage employee salaries efficiently. You've created an updatable view named "EmployeeSalary" that displays the EmployeeID, FirstName, LastName, and Salary of all employees in the company. The view allows you to modify the salary of any employee directly through the view.

### Tasks:

- Create an updateable view named "EmployeeSalary" that allows modifying the Salary column of the Employees table.
- Test the view by updating the salary of an employee through the view.

### QUERY:

**create table Employees**

**(**

**EmployeeID int,**

**FirstName varchar(20),**

**LastName varchar(20),**

**Salary decimal(10,2)**

**);**

**insert into Employees values**

**(1,'John','Doe',50000.00),**

**(2,'Jane','Smith',60000.00),**

**(3,'Alice','Johnson',55000.00);**

**update Employees set Salary=65000.00 where EmployeeID=1;**

**create view EmployeeSalary as**

**select FirstName,LastName,Salary from Employees;**

**select\* from EmployeeSalary;**

### OUTPUT

FirstName	LastName	Salary
John	Doe	65000.00

Jane	Smith	60000.00
Alice	Johnson	55000.00

**RESULT:**

### **Ex No 7 : Practice of Procedural extensions (Procedure,Function, Cursors, Triggers)**

#### **PROBLEM STATEMENT**

Imagine you're tasked with creating a payroll system for a company. As part of this project, you need to retrieve employee names and their corresponding salaries from a database table named "employees." You decide to use a SQL cursor named "employee\_cursor" to fetch this data. The cursor will iterate through each row of the table, fetching the employee name and salary one by one. As a developer, you need to write a SQL query to accomplish this task efficiently, ensuring accurate retrieval of employee information for the payroll system's processing.

**Tasks:**

- Declare and open a cursor named "employee\_cursor" to fetch the employee names and salaries from the "employees" table.
- Fetch the data using a loop and print each employee name and salary.

**Input Table:**

employee_id	employee_name	salary
1	John Doe	50000.00
2	Jane Smith	60000.00
3	Michael Johnson	70000.00
4	Emily Davis	55000.00

**QUERY**

-- Declare cursor

DECLARE employee\_cursor CURSOR FOR

SELECT name, salary FROM employees;

-- Open cursor

OPEN employee\_cursor;

-- Variables to store fetched data

DECLARE employee\_name TEXT;

DECLARE employee\_salary NUMERIC;

-- Fetch data using loop

LOOP

-- Fetch next row into variables

FETCH employee\_cursor INTO employee\_name, employee\_salary;

-- Exit loop if no more rows

EXIT WHEN NOT FOUND;

-- Print employee name and salary

RAISE NOTICE 'Employee: %, Salary: %', employee\_name, employee\_salary;

END LOOP;

```
-- Close cursor
```

```
CLOSE employee_cursor;
```

## OUTPUT:

## PROBLEM STATEMENT

In a corporate setting, you've been tasked with enhancing the payroll system for a company's HR department. As part of this project, you're required to implement a mechanism that ensures the "last\_salary\_updated" column in the "employee" table is always current whenever an employee's salary changes.

### Tasks:

- Create a trigger named "update\_salary\_trigger" that automatically updates the "last\_salary\_updated" column in the "employee" table with the current date whenever the "salary" column for an employee is updated.

### Input Table:

employee_id	employee_name	salary	Last_salary_updated
1	John Doe	50000.00	2024-04-15
2	Jane Smith	60000.00	2024-04-14
3	Michael Johnson	70000.00	2024-04-15
4	Emily Davis	55000.00	2024-04-13

### Output Table:

employee_id	employee_name	salary	Last_salary_updated
1	John Doe	55000.00	2024-04-18
2	Jane Smith	60000.00	2024-04-14
3	Michael Johnson	72000.00	2024-04-18
4	Emily Davis	55000.00	2024-04-13

## QUERY

```
CREATE TRIGGER update_salary_trigger
```

```
BEFORE UPDATE ON employee
```

```
FOR EACH ROW
```

```
SET NEW.last_salary_updated := CURRENT_DATE
```

## OUTPUT

**RESULT:**

## Ex No 8 : Mini Project (Application Development)

### PROBLEM STATEMENT

You are the database administrator at an IT training center that offers various specialized programs, including "Web Development" and "Data Analysis." You have been tasked with maintaining the center's database, which includes managing program details and tracking trainee enrollments.

#### Task:

- The Programs table should have fields for ProgramID (primary key), Title, and Description. It will store information about different training programs offered by the organization.

#### Insert sample data into the Programs table:

- ProgramID 1: Title "Web Development", Description "Learn to build websites using HTML, CSS, and JavaScript."
- ProgramID 2: Title "Data Analysis", Description "Learn how to analyze data using Python and SQL."

#### Insert sample data into the Trainees table:

- TraineeID 1: FirstName "Alice", LastName "Johnson", Email "alice.johnson@example.com", ProgramID 1 (enrolled in Web Development program)
- TraineeID 2: FirstName "Bob", LastName "Brown", Email "bob.brown@example.com", ProgramID 2 (enrolled in Data Analysis program)
- Remove the trainee with TraineeID 2 from the Trainees table.
- Finally, retrieve a list of trainees along with their first name, last name, email, and the title of the program they are enrolled in.

#### Input Table:

ProgramID	Title	Description
1	Web Development	Learn to build websites using HTML, CSS and JavaScript
2	Data Analysis	Learn how to analyze data using Python and SQL.

TraineeID	FirstName	LastName	Email	ProgramID
1	Alice	Johnson	alice.johnson@example.com	1
2	Bob	Brown	bob.brown@example.com	2

TraineeID	FirstName	LastName	Email	ProgramID
1	Alice	Johnson	alice.johnson@example.com	1

#### Query:

Ln1:Col1

```
1 create table Programs
2 (
3   ProgramID int Primary Key,
4   Title varchar (20),
5   Description varchar(100)
6 );
7 insert into Programs values(1,'Web Development','Learn to build websites using HTML,CSS and JavaScript');
8 insert into Programs values(2,'Data Analysis','Learn how to analyze data using Python ans SQL. ');
9 -- alter table Programs CHANGE COLUMN Title Program varchar(20);
10 create table Trainees
11 (
12   TraineeID int,
13   FirstName varchar(20),
14   LastName varchar(20),
15   Email varchar(30),
16   ProgramID int
17 );
18 insert into Trainees values(1,'Alice','Johnson','alice.johnson@example.com',1);
19 insert into Trainees values(2,'Bob','Brown','bob.brown@example.com',2);
20 delete from Trainees where TraineeID=2;
21 select t.FirstName as 'First Name',t.LastName as 'Last Name',t.Email,p.Title as Program from Programs p INNER JOIN Trainees t on
22 p.ProgramID=t.ProgramID;
```

## OUTPUT

First Name	Last Name	Email	Program
Alice	Johnson	alice.johnson@example.com	Web Development

## PROBLEM STATEMENT

In a dynamic corporation, the payroll processing system is the cornerstone of employee compensation management. The HR department aims to streamline the calculation of overtime pay for employees exceeding regular work hours. To achieve this, the payroll system should incorporate a feature that automatically computes overtime pay using predefined rates and employee work logs. By analyzing work hours logged beyond standard thresholds and applying the appropriate overtime rates, the system ensures accurate and efficient compensation processing for overtime work.

### Task:

- Create Table: Set up a table named "EmployeeWorkLogs" to track employee work hours, including fields for EmployeeID (unique identifier), Date, RegularHours, and OvertimeHours.
- Insert Data: Populate the "EmployeeWorkLogs" table with sample work hour data for different employees, including regular and overtime hours worked on various dates.
- Calculate Overtime Pay: Define an overtime rate of \$20 per hour. Calculate the overtime pay for each employee based on the overtime hours worked,output format displays the total overtime pay.

### Query:

-- Create Employee Work Logs table

CREATE TABLE EmployeeWorkLogs (

EmployeeID INT,



```

Date DATE,
RegularHours INT,
OvertimeHours INT
);

-- Insert sample data into Employee Work Logs table
INSERT INTO EmployeeWorkLogs (EmployeeID, Date, RegularHours, OvertimeHours)
VALUES
(101, '2024-05-01', 8, 2),
(102, '2024-05-01', 8, 3),
(103, '2024-05-01', 8, 1),
(104, '2024-05-02', 8, 2),
(105, '2024-05-02', 8, 3),
(106, '2024-05-02', 8, 1),
(107, '2024-05-03', 8, 2),
(108, '2024-05-03', 8, 3),
(109, '2024-05-03', 8, 1),
(110, '2024-05-04', 8, 2);

-- Define overtime rate
SET @OvertimeRate = 20; -- Assuming $20 per overtime hour

-- Query to calculate overtime pay
SELECT
EmployeeID,
CONCAT('$', OvertimeHours * @OvertimeRate) AS OvertimePay
FROM
EmployeeWorkLogs;

```

#### OUTPUT

EmployeeID	OvertimePay
101	\$40
102	\$60
103	\$20
104	\$40
105	\$60
106	\$20
107	\$40

108	\$60
109	\$20
110	\$40

## PROBLEM STATEMENT

A local blood donation center manages data related to its donors and the donations they make. The center uses a database to store information about each donor, including their name, blood type, and contact details, as well as records of each donation event.

### Tasks:

- Setup Database Tables: Create tables to store donor information (Donors) and donation events (Donations).
- Enter Initial Data: Populate the database with initial data for a couple of donors and their respective donation records.
- Remove Records: John Doe decides to retract his consent for using his personal data, prompting the removal of his records from the database.
- Update Contact Information: Jane Doe updates her contact number in the donor records.
- Generate Report: Generate a report detailing all remaining donations and associated donor information for review at a monthly meeting.

### Input Table:

DonorID	FirstName	LastName	BloodType	ContactNumber
1	John	Doe	O+	123-456-7890
2	Jane	Doe	A+	987-654-3210

DonationID	DonorID	DonationDate	AmountDonated
1	1	2024-04-01	500
2	2	2024-04-02	450

### Output Table:

#### Donors Table (After Deletion and Update)

DonorID	FirstName	LastName	BloodType	ContactNumber
2	Jane	Doe	A+	555-123-4567

#### Donations Table (After Deletion)

DonationID	DonorID	DonationDate	AmountDonated
2	2	2024-04-02	450

### Final Report (After Selection):

#### Current Donations Report

FirstName	LastName	BloodType	DonationDate	Amount
Jane	Doe	A+	2024-04-02	450

Query :

Ln1:Col1

```

1 create table Donors
2 (
3   DonorID int,
4   FirstName varchar(20),
5   LastName varchar(20),
6   BloodType varchar(10),
7   ContactNumber varchar(20)
8 );
9 insert into Donors values(1,'John','Doe','O+','123-456-7890');
10 insert into Donors values(2,'Jane','Doe','A+','987-654-3210');
11
12 create table Donations
13 (
14   DonationID int,
15   DonorID int,
16   DonationDate date,
17   AmountDonated int
18 );
19 insert into Donations values(1,1,'2024-04-01',500);
20 insert into Donations values(2,2,'2024-04-02',450);
21 delete from Donors where DonorID=1;
22 update Donors set ContactNumber='555-123-4567' where DonorID=2;
23 delete from Donations where DonorID=1;
24 select d.FirstName as 'First Name',d.LastName as 'Last Name',d.BloodType as 'Blood Type',
25   D.DonationDate as 'Donation Date',D.AmountDonated as 'Amount Donated'
26 from Donors d INNER JOIN
27   Donations D on d.DonorID=D.DonorID;

```

OUTPUT

FirstName	LastName	BloodType	DonationDate	Amount
Jane	Doe	A+	2024-04-02	450

## PROBLEM STATEMENT

You are a database manager for a city's traffic management department. Your responsibility is to develop a database that records details about each traffic light's location, operational status, and maintenance records. Additionally, the system should allow for reporting on lights needing maintenance, traffic light functionality rates, and incident reports due to traffic light failures.

Task:

- Create Database Tables: Design tables for traffic lights, intersections, maintenance logs, and incident reports.

- Populate Database: Insert hypothetical data into these tables.
- Perform SQL Queries:
- Report on traffic lights that are currently non-operational.
- Summarize the number of incidents caused by faulty traffic lights by location.
- List upcoming maintenance schedules for traffic lights based on their last service date.

**Input Table:**

**Intersections Table:**

IntersectionID	Location
1	1st Ave & Main St
2	2nd Ave & Main St

**TrafficLights Table:**

LightID	IntersectionID	Status	LastServiceDate
101	1	Operational	2023-01-10
102	1	Non-Operational	2023-03-15
201	2	Operational	2023-02-20

**MaintenanceLogs Table:**

LogID	LightID	ServiceDate	Details
1	101	2023-01-10	Routine Check
2	102	2023-03-15	Replaced unit

**IncidentReports Table:**

ReportID	LightID	ReportDate	IncidentDescription
1	102	2023-03-16	Failure led to minor accident

**Output Table:**

**Non-Operational Traffic Lights**

Location	LightID	Status
1st Ave & Main St	102	Non-Operational

**Incident Reports by Location**

Location	NumberOfIncidents
1st Ave & Main St	1

**Next Service Due for Traffic Lights**

LightID	Location	LastServiceDate	NextServiceDue
101	1st Ave & Main St	2023-01-10	2023-07-10
102	1st Ave & Main St	2023-03-15	2023-09-15

Query:

Ln1: Col1

```
1 create table Intersection
2 (
3     IntersectionID int,
4     Location varchar(20)
5 );
6 insert into Intersection values
7 (1,'1st Ave & Main St'),
8 (2,'2nd Ave & Main St');
9 create table TrafficLights
10 (
11     LightID int,
12     IntersectionID int,
13     Status varchar(20),
14     LastServiceDate date
15 );
16 insert into TrafficLights values
17 (101,1,'Operational','2023-01-10'),
18 (102,1,'Non-Operational','2023-03-15'),
19 (103,2,'Operational','2023-02-20');
20 create table MaintenanceLogs
21 (
22     LogID int,
23     LightID int,
24     ServiceDate date,
25     Details varchar(20)
26 );
27 insert into MaintenanceLogs values
28 (1,101,'2023-01-10','Routine Check'),
29 (2,102,'2023-03-15','Replaced unit');
30 create table IncidentReports
31 (
32     ReportID int,
33     LightID int,
34     ReportDate date,
35     IncidentDescription varchar(50)
36 );
```

Ln 23 : Col 15

```
16 insert into TrafficLights values
17 (101,1,'Operational','2023-01-10'),
18 (102,1,'Non-Operational','2023-03-15'),
19 (103,2,'Operational','2023-02-20');
20 create table MaintenanceLogs
21 (
22     LogID int,
23     LightID int,
24     ServiceDate date,
25     Details varchar(20)
26 );
27 insert into MaintenanceLogs values
28 (1,101,'2023-01-10','Routine Check'),
29 (2,102,'2023-03-15','Replaced unit');
30 create table IncidentReports
31 (
32     ReportID int,
33     LightID int,
34     ReportDate date,
35     IncidentDescription varchar(50)
36 );
37 insert into IncidentReports values(1,102,'2023-03-16','Failure led to minor accident');
38
39 select i.Location, t.LightID,t.Status from Intersection i inner join TrafficLights t
40 on i.IntersectionID=t.IntersectionID where t.LightID=102;
41
42 select i.Location ,COUNT(ir.ReportID) AS NumberOfIncidents
43 From Intersection i Join TrafficLights t ON i.IntersectionID=t.IntersectionID
44 join IncidentReports ir ON t.LightID=ir.LightID
45 where i.IntersectionID=1
46 Group by i.Location;
47
48 select t.LightID ,i.Location,t.LastServiceDate, DATE_ADD(t.LastServiceDate, INTERVAL 6 MONTH)
49 AS NextServiceDue FROM TrafficLights t
50 Join Intersection i ON t.IntersectionID=i.IntersectionID
51 where t.IntersectionID=1 order by t.LightID desc;
```

**RESULT:**