Practical No.:- 1

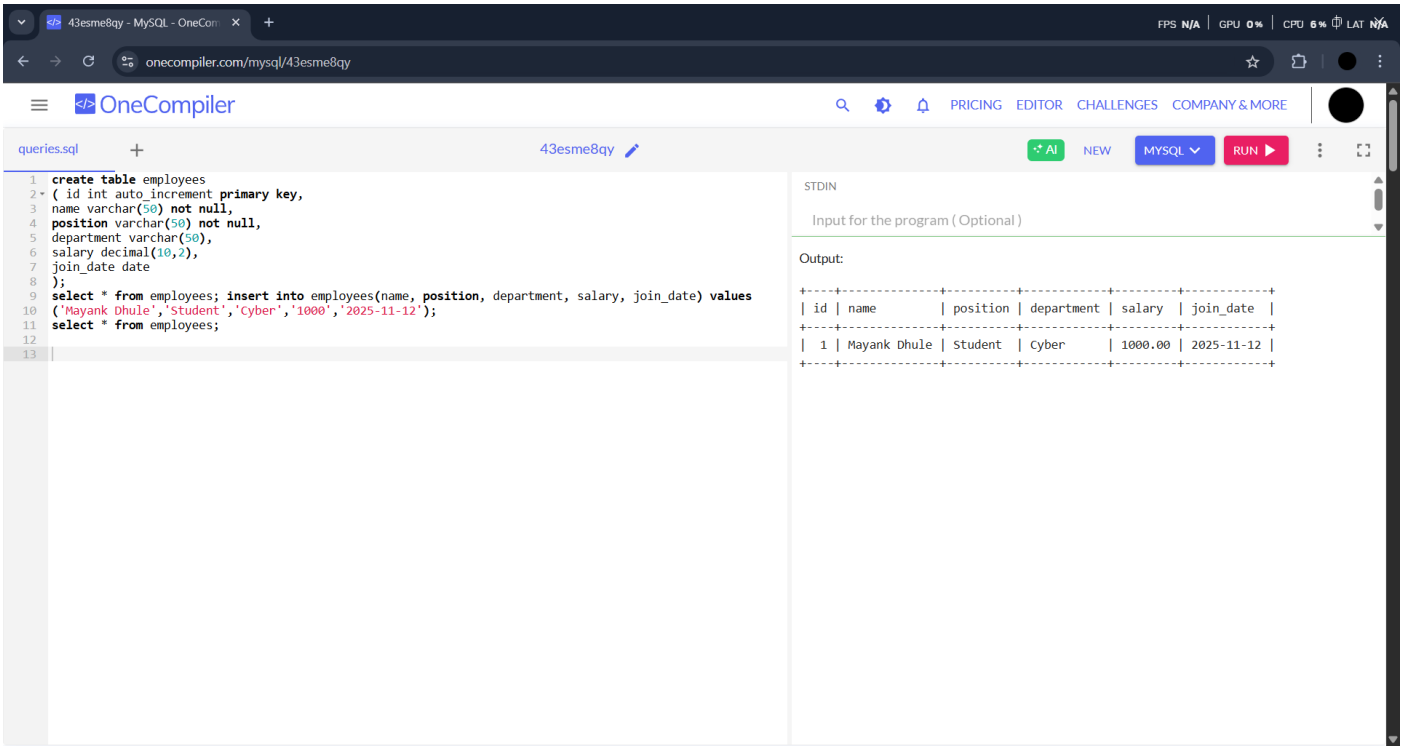
Aim:-

To create a database called company2 and define an employees table with appropriate attributes like id, name, position, department, salary, and join_date. Insert sample data and fetch records using SQL queries.

Code:-

```
create database company2; use
company2; create table
employees( id int auto_increment
primary key, name varchar(50)
not null, position varchar(50) not
null, department varchar(50),
salary decimal(10,2), join_date
date
);
select * from employees; insert into employees(name, position,
department, salary, join_date) values
('Mayank Dhule','Student','Cyber','1000','2025-11-12');
select * from employees;
```

Screenshot of Output:-



Practical No.:- 2

Aim:-

To establish a database called Employee and make an Employee table with PRIMARY KEY, NOT NULL, CHECK, and DEFAULT constraints. The table will have employee information such as EmployeeID, Name, Salary, JoiningDate, and ActiveStatus. Insert data and retrieve it using SQL queries as well.

Code:-

Create Database

Employee;

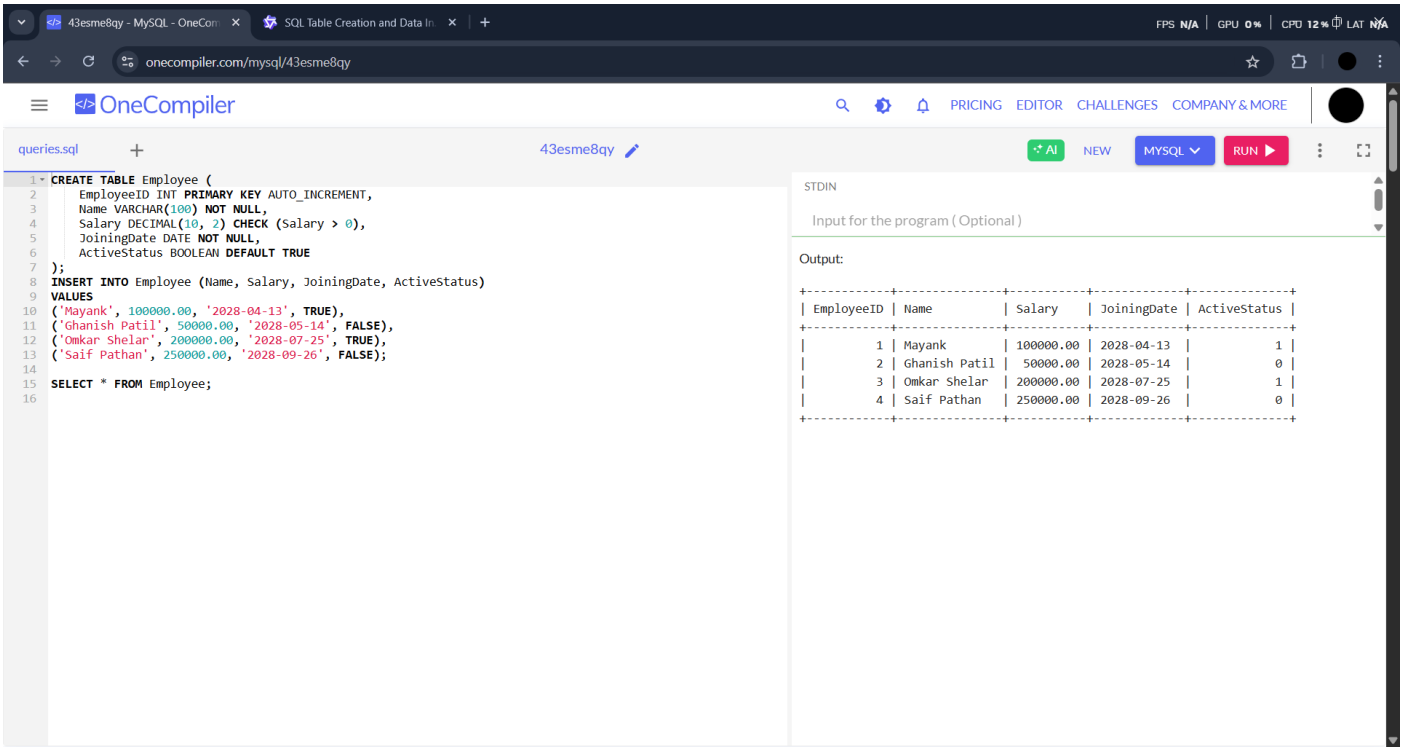
use Employee;

CREATE TABLE

Employee (

```
EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Salary DECIMAL(10, 2) CHECK (Salary > 0),
    JoiningDate DATE NOT NULL,
    ActiveStatus BOOLEAN DEFAULT TRUE
);
INSERT INTO Employee (Name, Salary, JoiningDate, ActiveStatus)
VALUES
('Mayank', 100000.00, '2028-04-13', TRUE),
('Ghanish Patil', 50000.00, '2028-05-14', FALSE),
('Omkar Shelar', 200000.00, '2028-07-25', TRUE),
('Saif Pathan', 250000.00, '2028-09-26', FALSE);
SELECT * FROM Employee;
```

Screenshot of Output:-



Practical No.:- 3

Aim:-

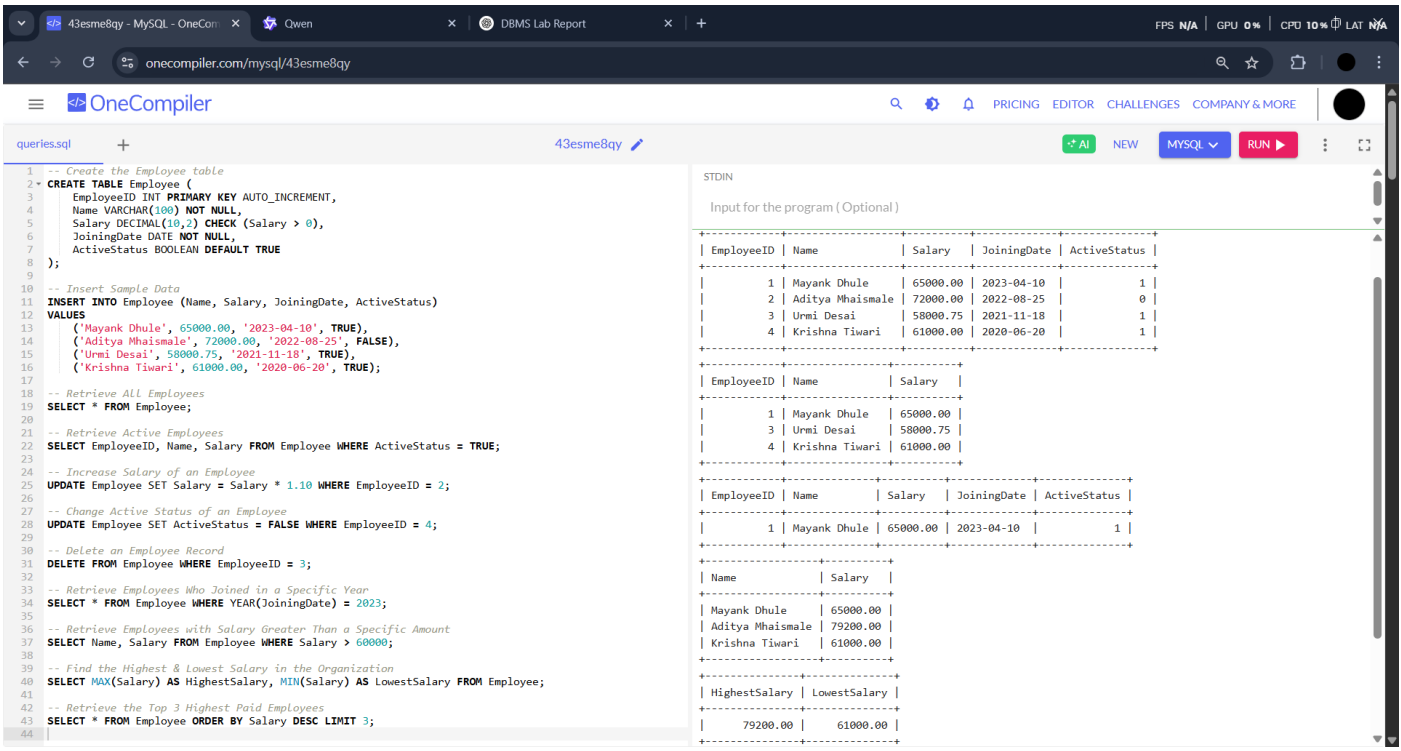
Create a table with columns for EmployeeID, Name, Salary, JoiningDate, and ActiveStatus using different data types. Insert sample data and perform queries to manipulate and retrieve data.

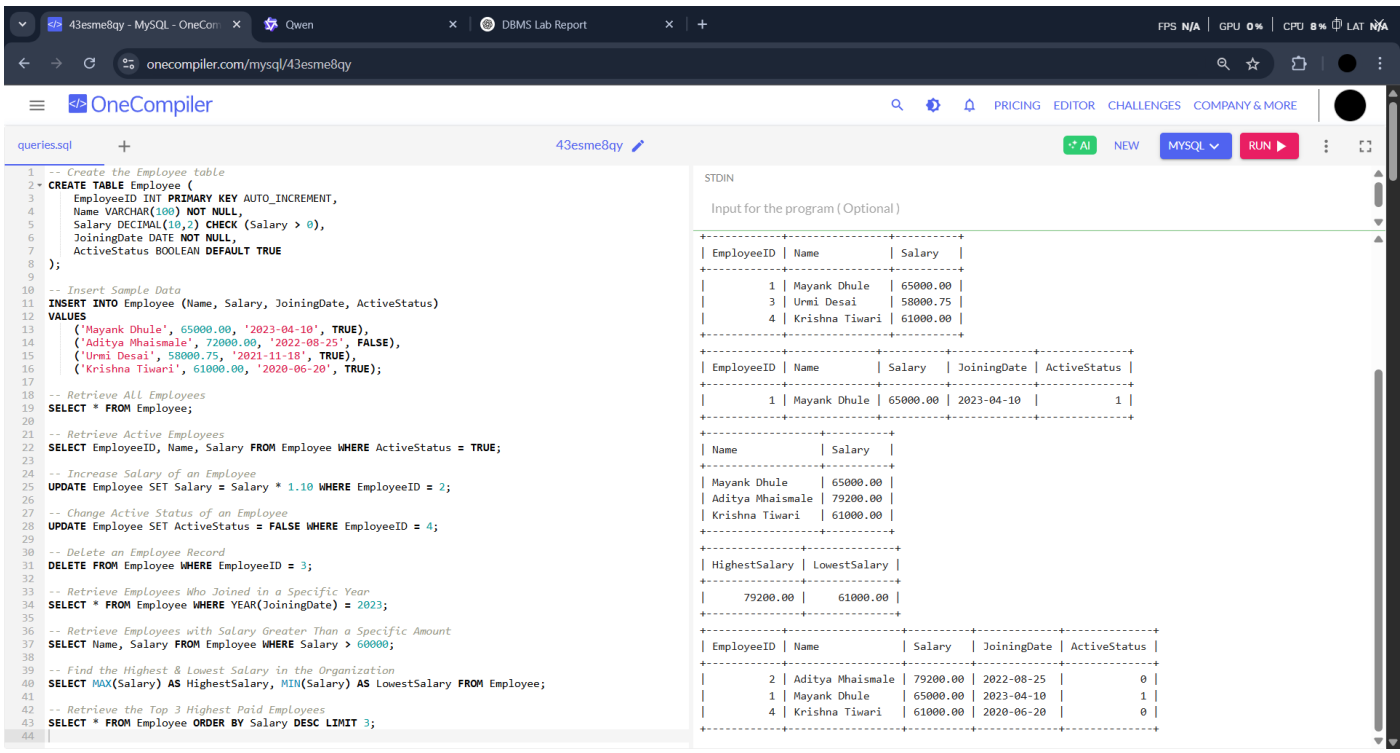
Code:-

```
-- Create the Employee table
CREATE TABLE Employee (
    EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100) NOT NULL,
    Salary DECIMAL(10,2) CHECK (Salary > 0),
    JoiningDate DATE NOT NULL,
    ActiveStatus BOOLEAN DEFAULT TRUE
);

-- Insert Sample Data
INSERT INTO Employee (Name, Salary, JoiningDate, ActiveStatus)
VALUES
    ('Mayank Dhule', 65000.00, '2023-04-10', TRUE),
    ('Aditya Mhaismale', 72000.00, '2022-08-25', FALSE),
    ('Urmi Desai', 58000.75, '2021-11-18', TRUE),
    ('Krishna Tiwari', 61000.00, '2020-06-20', TRUE);

-- Retrieve All Employees
SELECT * FROM Employee;
```

```
-- Retrieve Active Employees

SELECT EmployeeID, Name, Salary FROM Employee WHERE ActiveStatus
= TRUE;


-- Increase Salary of an Employee

UPDATE Employee SET Salary = Salary * 1.10 WHERE EmployeeID = 2;


-- Change Active Status of an Employee

UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID = 4;


-- Delete an Employee Record

DELETE FROM Employee WHERE EmployeeID = 3;


-- Retrieve Employees Who Joined in a Specific Year

SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;


-- Retrieve Employees with Salary Greater Than a Specific Amount

SELECT Name, Salary FROM Employee WHERE Salary > 60000;


-- Find the Highest & Lowest Salary in the Organization

SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary
FROM Employee;


-- Retrieve the Top 3 Highest Paid Employees

SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
```

Screenshot of Output:-





Practical No.:- 4

Creating Employee Table with Constraints

Aim:-

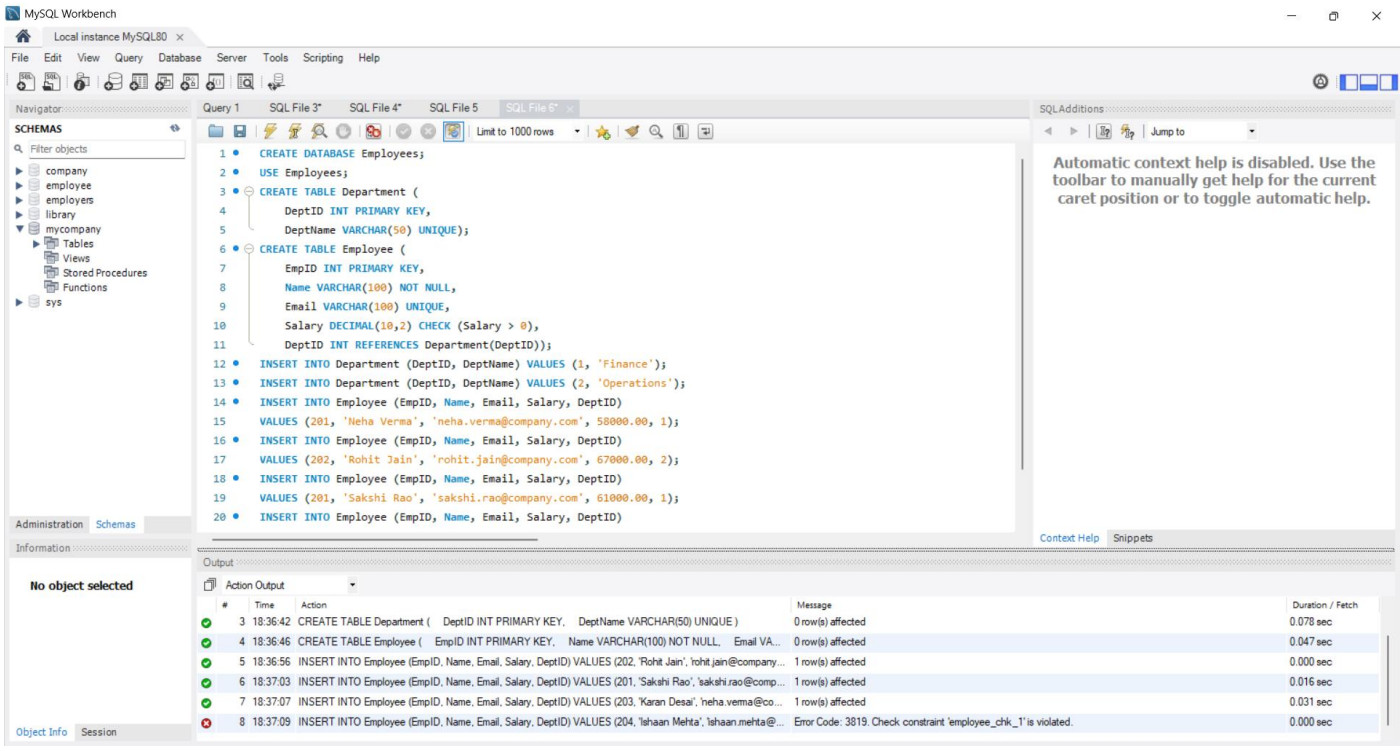
Create a table to store employee information with constraints like Primary Key, Foreign Key, and Unique.

Code:-

```
-- Create and use the Employees database
CREATE DATABASE Employees;
USE Employees;

-- Create Department table
CREATE TABLE Department (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(50) UNIQUE
);

-- Create Employee table with constraints
CREATE TABLE Employee (
    EmpID INT PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    Salary DECIMAL(10,2) CHECK (Salary > 0),
    DeptID INT REFERENCES Department(DeptID)
);

-- Insert Valid Data
INSERT INTO Department (DeptID, DeptName) VALUES (1, 'Finance');
INSERT INTO Department (DeptID, DeptName) VALUES (2, 'Operations');

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
VALUES (201, 'Neha Verma', 'neha.verma@company.com', 58000.00, 1);
```

```
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
VALUES (202, 'Rohit Jain', 'rohit.jain@company.com', 67000.00, 2);

-- Insert Invalid Data to Test Constraints

-- Duplicate Primary Key
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
VALUES (201, 'Sakshi Rao', 'sakshi.rao@company.com', 61000.00, 1);

-- Duplicate Unique Email
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
VALUES (203, 'Karan Desai', 'neha.verma@company.com', 62000.00, 2);

-- Salary Check Constraint Violation
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)
VALUES (204, 'Ishaan Mehta', 'ishaan.mehta@company.com', -45000.00, 1);
```

Screenshot of Output:-



Practical No.:- 5

Testing Employee Constraints

Aim:-

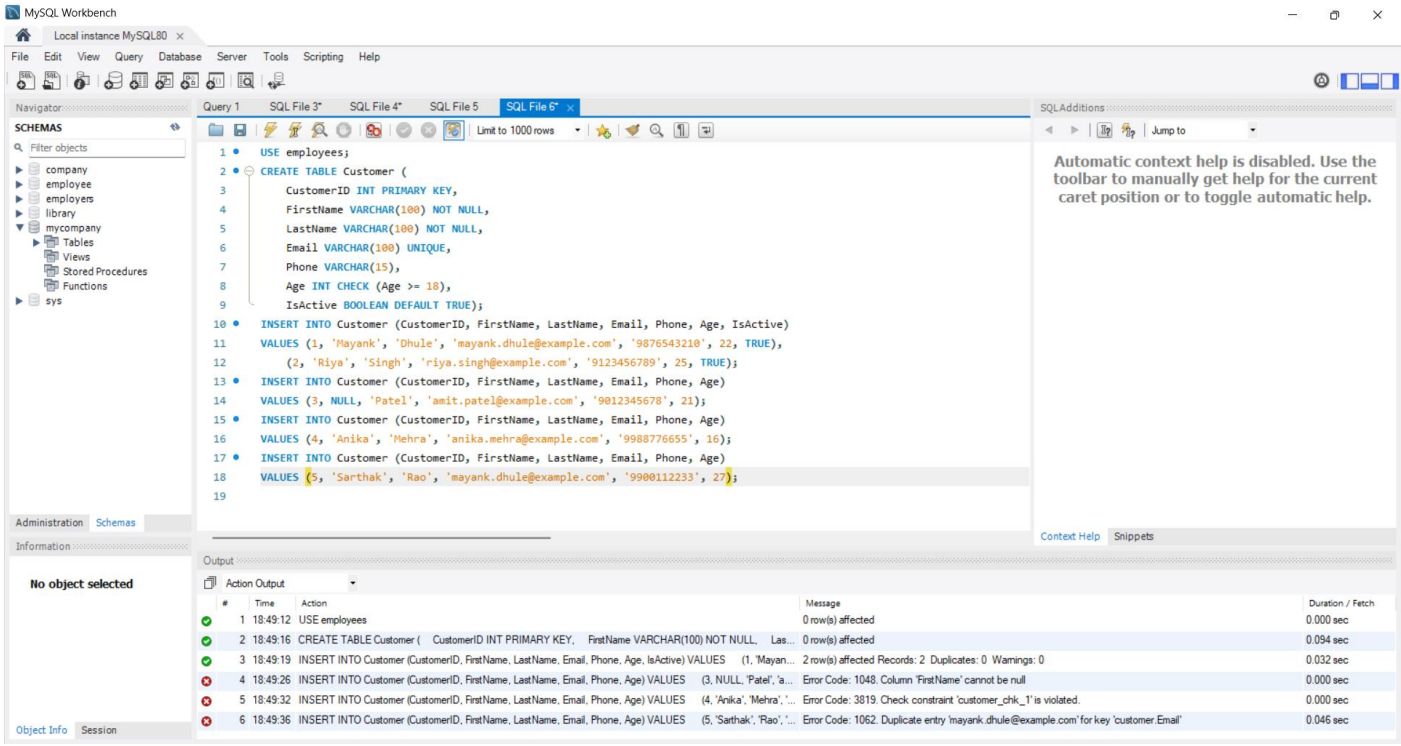
To test constraints like PRIMARY KEY, UNIQUE, and CHECK by inserting invalid data into the Employee table.

Code:-

```
 -- Use the employees database -- Use the employees database
USE employees;


-- Create Customer table
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    FirstName VARCHAR(100) NOT NULL,
    LastName VARCHAR(100) NOT NULL,
    Email VARCHAR(100) UNIQUE,
    Phone VARCHAR(15),
    Age INT CHECK (Age >= 18),
    IsActive BOOLEAN DEFAULT TRUE
);


-- Insert Valid Data
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age, IsActive)
```

```
VALUES
    (1, 'Mayank', 'Dhule', 'mayank.dhule@example.com', '9876543210', 22, TRUE),
    (2, 'Riya', 'Singh', 'riya.singh@example.com', '9123456789', 25, TRUE);

-- Insert Invalid Data to Test Constraints

-- NULL value in NOT NULL column (FirstName)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
VALUES
    (3, NULL, 'Patel', 'amit.patel@example.com', '9012345678', 21);

-- Age less than 18 (violates CHECK constraint)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
VALUES
    (4, 'Anika', 'Mehra', 'anika.mehra@example.com', '9988776655', 16);

-- Duplicate email (violates UNIQUE constraint)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
VALUES
    (5, 'Sarthak', 'Rao', 'mayank.dhule@example.com', '9900112233', 27);
```

Screenshot of Output:-



Practical No.:- 6

Aim:-

Use DDL commands to create tables and DML commands to insert, update, and delete data. Write SELECT queries to retrieve and verify data changes.

Code:-

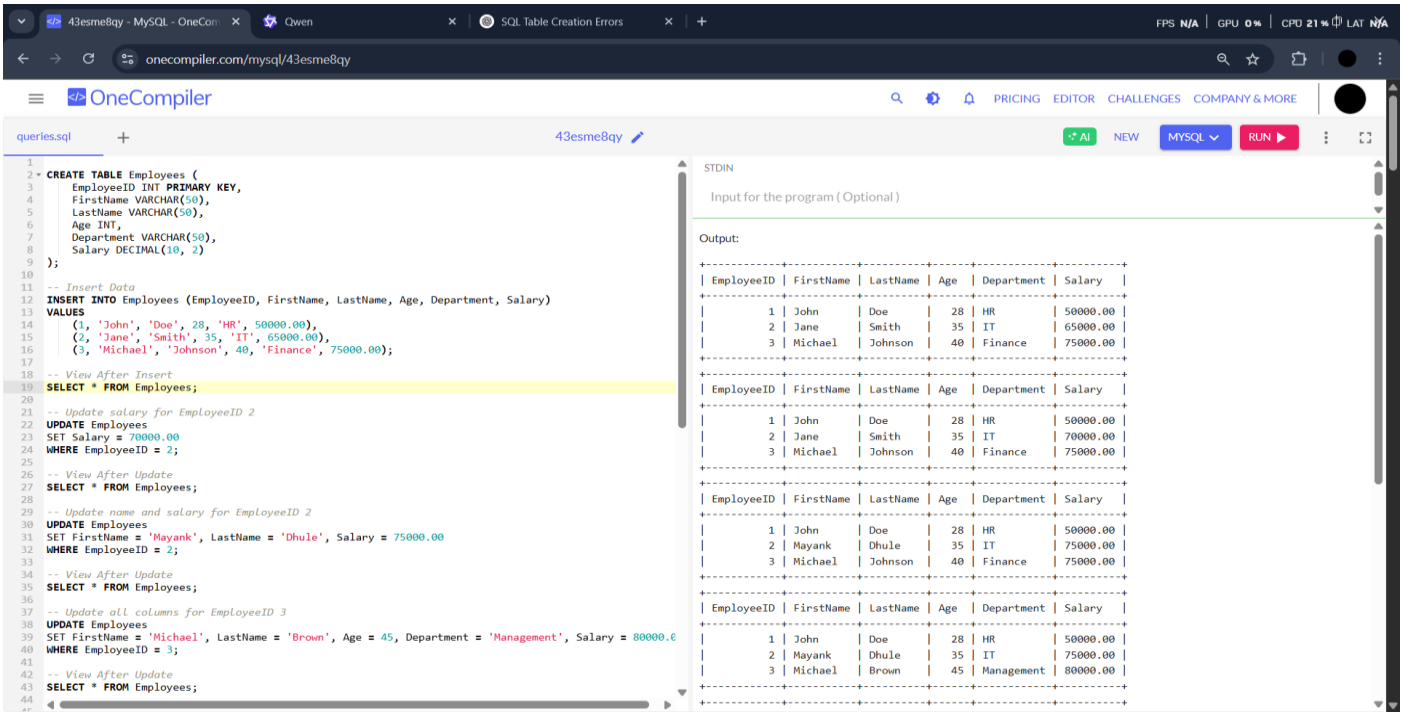
```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Age INT,
    Department VARCHAR(50),
    Salary DECIMAL(10, 2)
);

-- Insert Data
INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)
VALUES
    (1, 'John', 'Doe', 28, 'HR', 50000.00),
    (2, 'Jane', 'Smith', 35, 'IT', 65000.00),
    (3, 'Michael', 'Johnson', 40, 'Finance', 75000.00);

-- View After Insert
SELECT * FROM Employees;

-- Update salary for EmployeeID 2
UPDATE Employees
SET Salary = 70000.00
WHERE EmployeeID = 2;
```

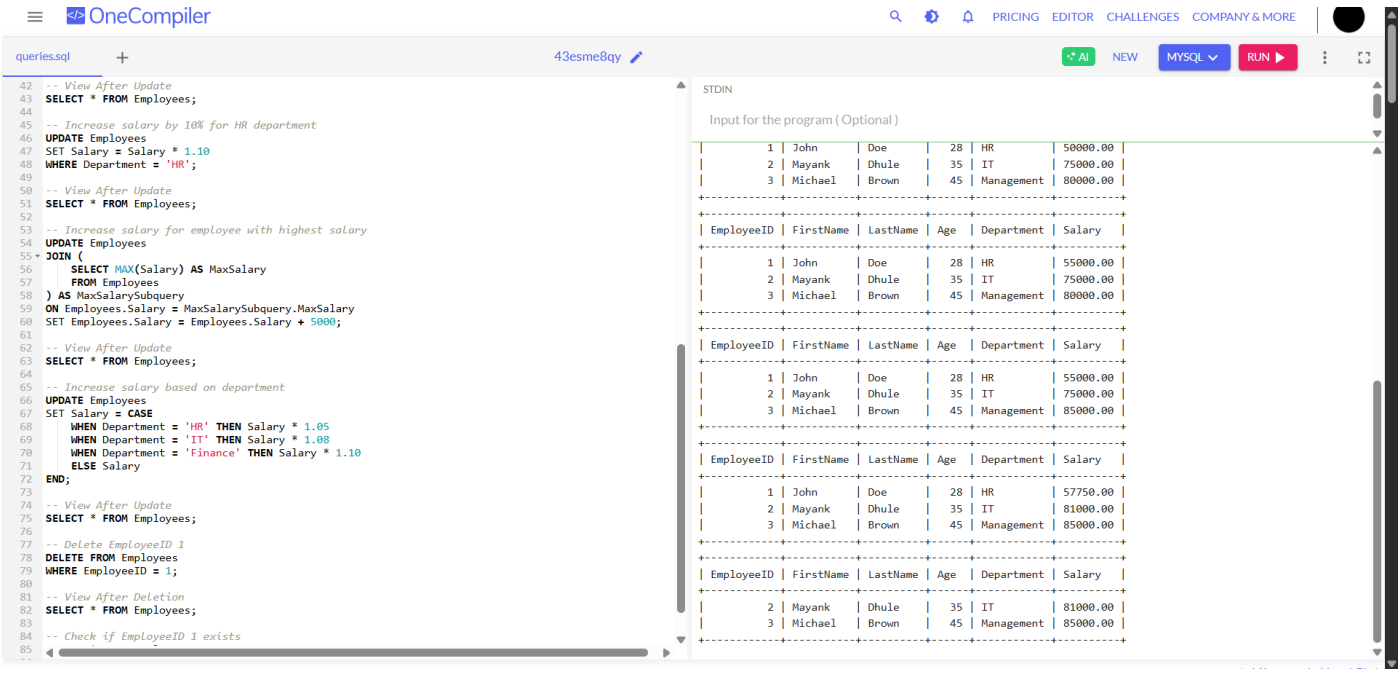
```
-- View After Update
SELECT * FROM Employees;

-- Update name and salary for EmployeeID 2
UPDATE Employees
SET FirstName = 'Mayank', LastName = 'Dhule', Salary = 75000.00
WHERE EmployeeID = 2;

-- View After Update
SELECT * FROM Employees;

-- Update all columns for EmployeeID 3
UPDATE Employees
SET FirstName = 'Michael', LastName = 'Brown', Age = 45, Department = 'Management',
Salary = 80000.00
WHERE EmployeeID = 3;

-- View After Update
SELECT * FROM Employees;

-- Increase salary by 10% for HR department
UPDATE Employees
SET Salary = Salary * 1.10
WHERE Department = 'HR';

-- View After Update
```

```
SELECT * FROM Employees;

-- Increase salary for employee with highest salary
UPDATE Employees
JOIN (
    SELECT MAX(Salary) AS MaxSalary
    FROM Employees
) AS MaxSalarySubquery
ON Employees.Salary = MaxSalarySubquery.MaxSalary
SET Employees.Salary = Employees.Salary + 5000;

-- View After Update
SELECT * FROM Employees;

-- Increase salary based on department
UPDATE Employees
SET Salary = CASE
    WHEN Department = 'HR' THEN Salary * 1.05
    WHEN Department = 'IT' THEN Salary * 1.08
    WHEN Department = 'Finance' THEN Salary * 1.10
    ELSE Salary
END;

-- View After Update
SELECT * FROM Employees;

-- Delete EmployeeID 1
```

```
DELETE FROM Employees
WHERE EmployeeID = 1;

-- View After Deletion
SELECT * FROM Employees;

-- Check if EmployeeID 1 exists
SELECT * FROM Employees
WHERE EmployeeID = 1;
```

Screenshot of Output:-





Practical No.:- 7

Aim:-

Create a Sales table and use aggregate functions like COUNT, SUM, AVG, MIN, and MAX to summarize sales data and calculate statistics.

Code:-

```
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY AUTO_INCREMENT,
    Product VARCHAR(50),
    Quantity INT,
    Price DECIMAL(10,2),
    SaleDate DATE
);
```

```
INSERT INTO Sales (Product, Quantity, Price, SaleDate)
VALUES
    ('Laptop', 2, 75000.00, '2025-02-01'),
    ('Mobile', 5, 20000.00, '2025-02-02'),
    ('Tablet', 3, 30000.00, '2025-02-03'),
    ('Laptop', 1, 78000.00, '2025-02-04'),
    ('Mobile', 4, 22000.00, '2025-02-05'),
    ('Tablet', 2, 32000.00, '2025-02-06');

-- View all records
SELECT * FROM Sales;

-- COUNT Queries
-- 1. Count the total number of sales records
SELECT COUNT(*) AS Total_Sales FROM Sales;

-- 2. Sum of total revenue generated
SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;

-- 3. Average price of products sold
SELECT AVG(Price) AS Average_Price FROM Sales;

-- 4. Minimum and Maximum price of a product sold
SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;
```

```
-- 5. Count the number of distinct products sold
SELECT COUNT(DISTINCT Product) AS Unique_Products FROM Sales;

-- 6. Count the number of sales per product
SELECT Product, COUNT(*) AS Sales_Count
FROM Sales
GROUP BY Product;

-- 7. Count the number of sales per day
SELECT SaleDate, COUNT(*) AS Sales_Per_Day
FROM Sales
GROUP BY SaleDate;

-- 8. Count the number of sales where more than 2 units were sold
SELECT COUNT(*) AS High_Quantity_Sales
FROM Sales
WHERE Quantity > 2;

-- 9. Count the number of sales in the current month
SELECT COUNT(*) AS Sales_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
```

```
-- 10. Count the number of sales transactions where total sale value was more than ₹50,000

SELECT COUNT(*) AS High_Value_Sales

FROM Sales

WHERE (Quantity * Price) > 50000;


-- 11. Count the number of sales records for each product where total sale value is greater than ₹40,000

SELECT Product, COUNT(*) AS High_Value_Transactions

FROM Sales

WHERE (Quantity * Price) > 40000

GROUP BY Product;


-- 12. Count the number of sales made after a specific date (e.g., Feb 3, 2025)

SELECT COUNT(*) AS Sales_After_Date

FROM Sales

WHERE SaleDate > '2025-02-03';


-- SUM Queries

-- 1. Sum of total revenue generated

SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;


-- 2. Sum of total quantity of products sold

SELECT SUM(Quantity) AS Total_Quantity_Sold FROM Sales;


-- 3. Sum of total revenue per product
```

```
SELECT Product, SUM(Quantity * Price) AS Revenue_Per_Product
FROM Sales
GROUP BY Product;

-- 4. Sum of total revenue per day
SELECT SaleDate, SUM(Quantity * Price) AS Revenue_Per_Day
FROM Sales
GROUP BY SaleDate;

-- 5. Sum of total revenue in the current month
SELECT SUM(Quantity * Price) AS Revenue_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

-- 6. Sum of revenue for sales where quantity sold is greater than 2
SELECT SUM(Quantity * Price) AS High_Quantity_Revenue
FROM Sales
WHERE Quantity > 2;

-- 7. Sum of total revenue generated after a specific date (e.g., Feb 3, 2025)
SELECT SUM(Quantity * Price) AS Revenue_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';
```

```
-- 8. Sum of revenue per product where the total revenue per transaction is greater than ₹40,000
SELECT Product, SUM(Quantity * Price) AS High_Value_Revenue
FROM Sales
WHERE (Quantity * Price) > 40000
GROUP BY Product;

-- AVG Queries
-- 1. Average price of products sold
SELECT AVG(Price) AS Average_Price FROM Sales;

-- 2. Average quantity of products sold per transaction
SELECT AVG(Quantity) AS Average_Quantity_Sold FROM Sales;

-- 3. Average revenue per transaction
SELECT AVG(Quantity * Price) AS Average_Revenue_Per_Transaction
FROM Sales;

-- 4. Average price per product
SELECT Product, AVG(Price) AS Average_Price_Per_Product
FROM Sales
GROUP BY Product;

-- 5. Average revenue per product
SELECT Product, AVG(Quantity * Price) AS Average_Revenue_Per_Product
FROM Sales
```

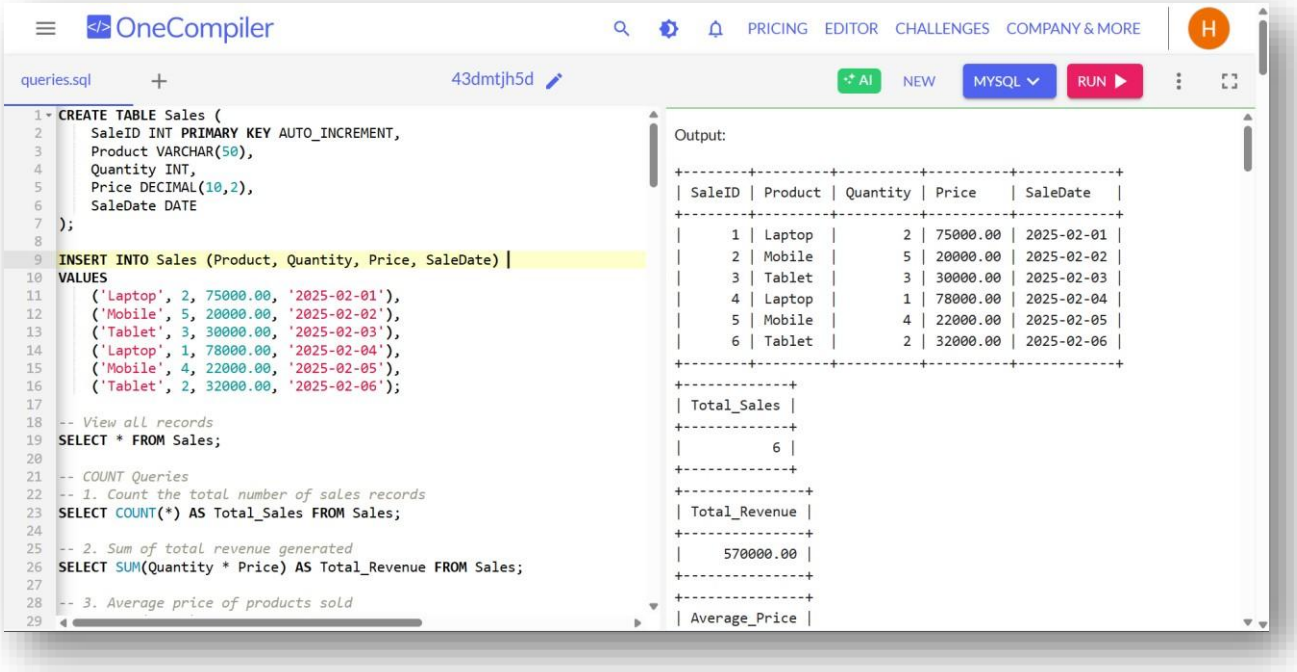
```
GROUP BY Product;

-- 6. Average quantity sold per product
SELECT Product, AVG(Quantity) AS Average_Quantity_Per_Product
FROM Sales
GROUP BY Product;

-- 7. Average revenue per day
SELECT SaleDate, AVG(Quantity * Price) AS Average_Revenue_Per_Day
FROM Sales
GROUP BY SaleDate;

-- 8. Average revenue in the current month
SELECT AVG(Quantity * Price) AS Average_Revenue_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

-- 9. Average price of products where more than 2 units were sold
SELECT AVG(Price) AS Avg_Price_High_Quantity_Sales
FROM Sales
WHERE Quantity > 2;

-- 10. Average revenue after a specific date (e.g., Feb 3, 2025)
SELECT AVG(Quantity * Price) AS Average_Revenue_After_Date
```

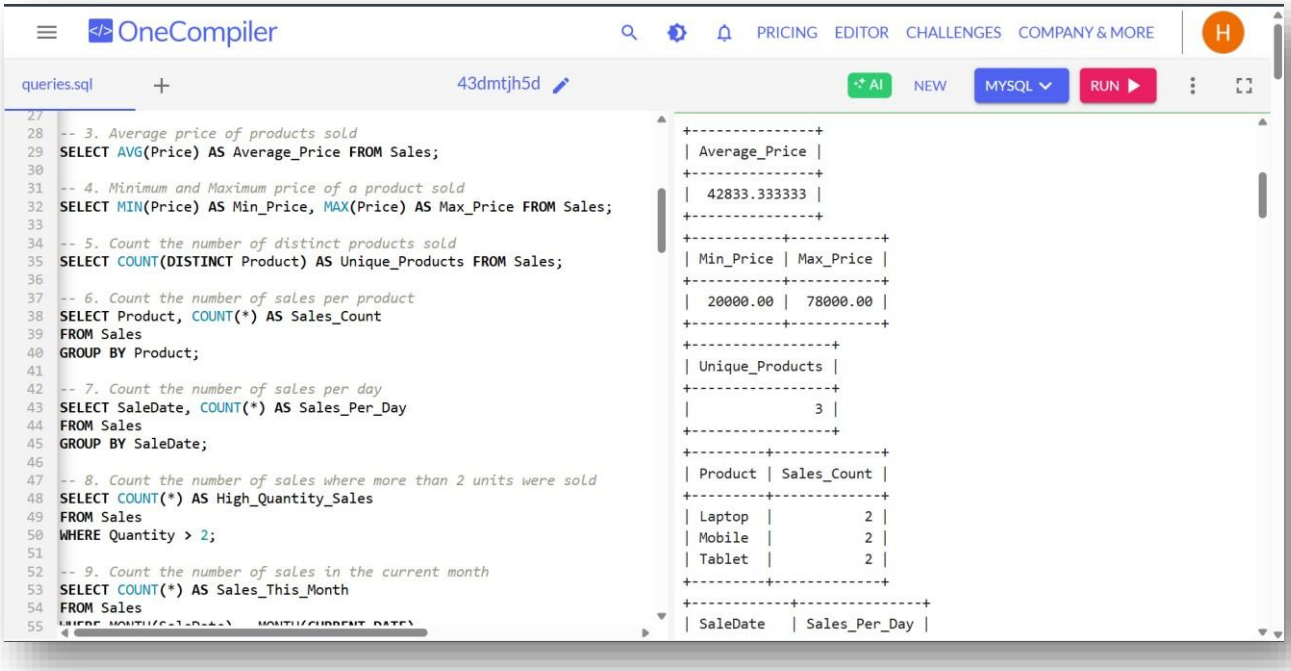
```
FROM Sales
WHERE SaleDate > '2025-02-03';

-- MIN/MAX Queries
-- 1. Minimum and Maximum price of a product sold
SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;

-- 2. Minimum and Maximum quantity of products sold in a single transaction
SELECT MIN(Quantity) AS Min_Quantity_Sold, MAX(Quantity) AS Max_Quantity_Sold FROM Sales;

-- 3. Minimum and Maximum revenue generated from a single transaction
SELECT MIN(Quantity * Price) AS Min_Revenue, MAX(Quantity * Price) AS Max_Revenue FROM Sales;

-- 4. Minimum and Maximum price per product
SELECT Product, MIN(Price) AS Min_Price_Per_Product, MAX(Price) AS Max_Price_Per_Product
FROM Sales
GROUP BY Product;

-- 5. Minimum and Maximum revenue per product
SELECT Product, MIN(Quantity * Price) AS Min_Revenue_Per_Product, MAX(Quantity * Price) AS Max_Revenue_Per_Product
FROM Sales
GROUP BY Product;
```

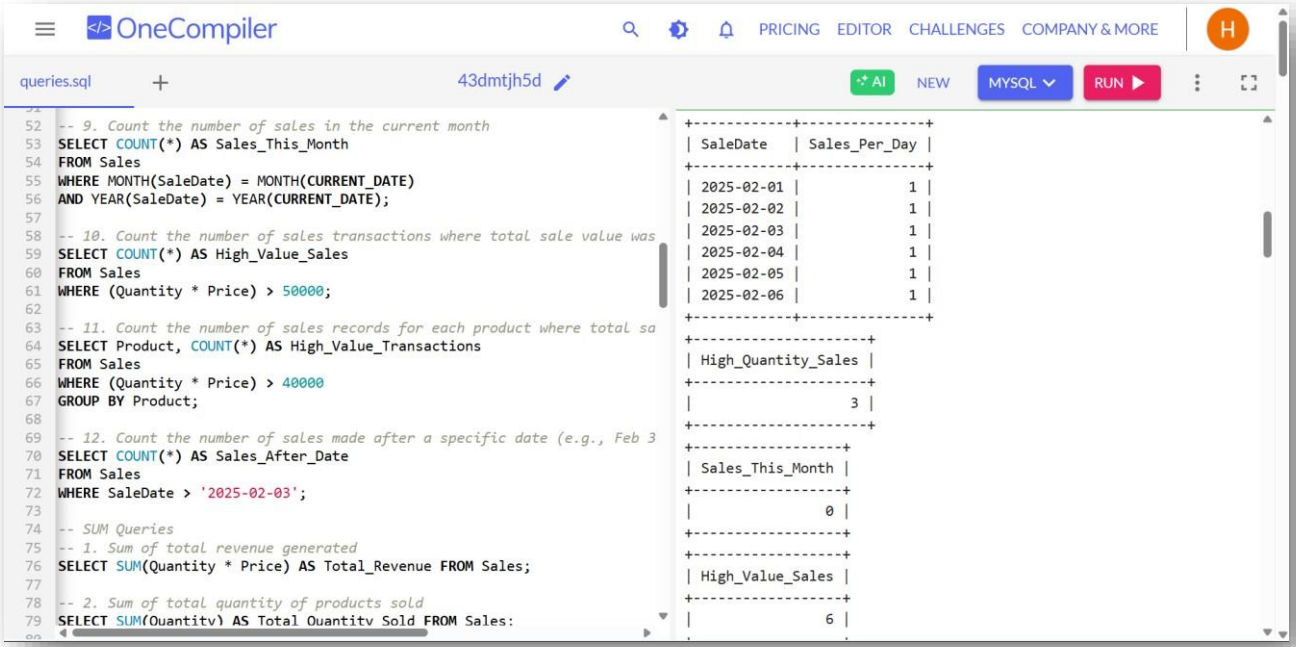
```
-- 6. Minimum and Maximum quantity sold per product
SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity) AS Max_Quantity_Per_Product
FROM Sales
GROUP BY Product;

-- 7. Minimum and Maximum revenue per day
SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quantity * Price) AS Max_Revenue_Per_Day
FROM Sales
GROUP BY SaleDate;

-- 8. Minimum and Maximum revenue in the current month
SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity * Price) AS Max_Revenue_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

-- 9. Minimum and Maximum price of products where more than 2 units were sold
SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_Price_High_Quantity_Sales
FROM Sales
WHERE Quantity > 2;

-- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3, 2025)
SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity
```

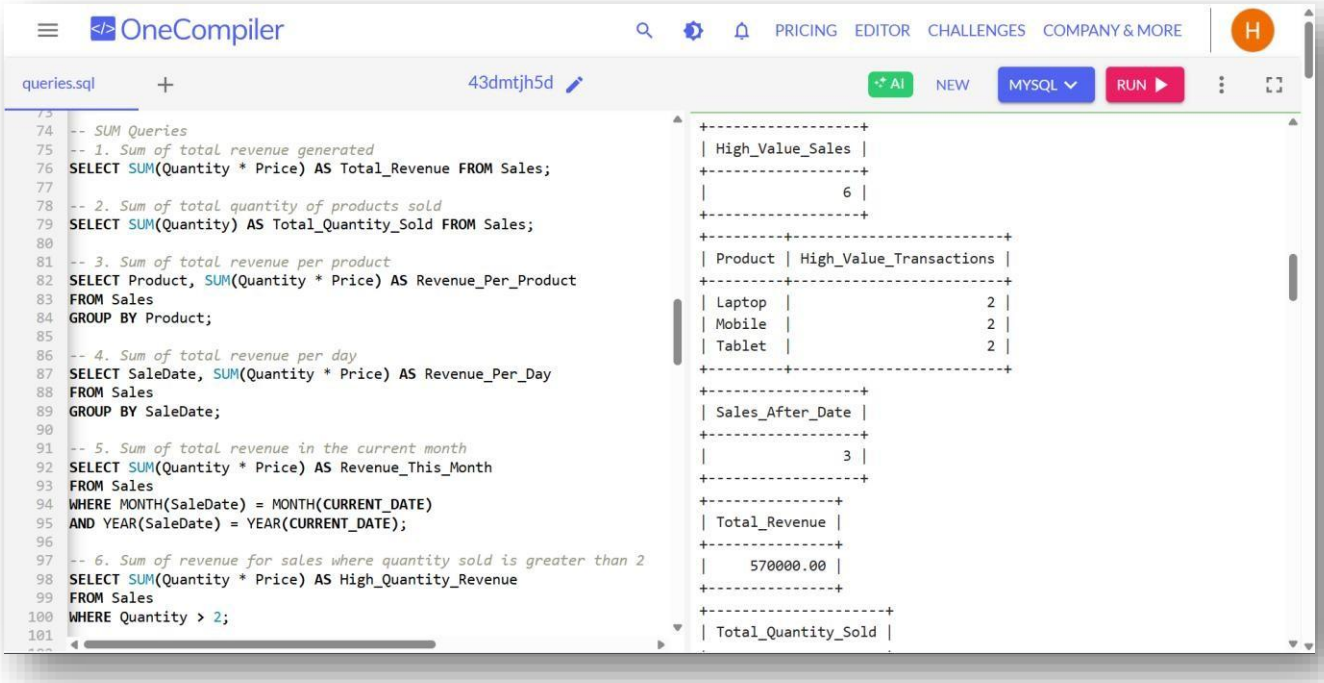
```
* Price) AS Max_Revenue_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';

-- View final data
SELECT * FROM Sales;
```

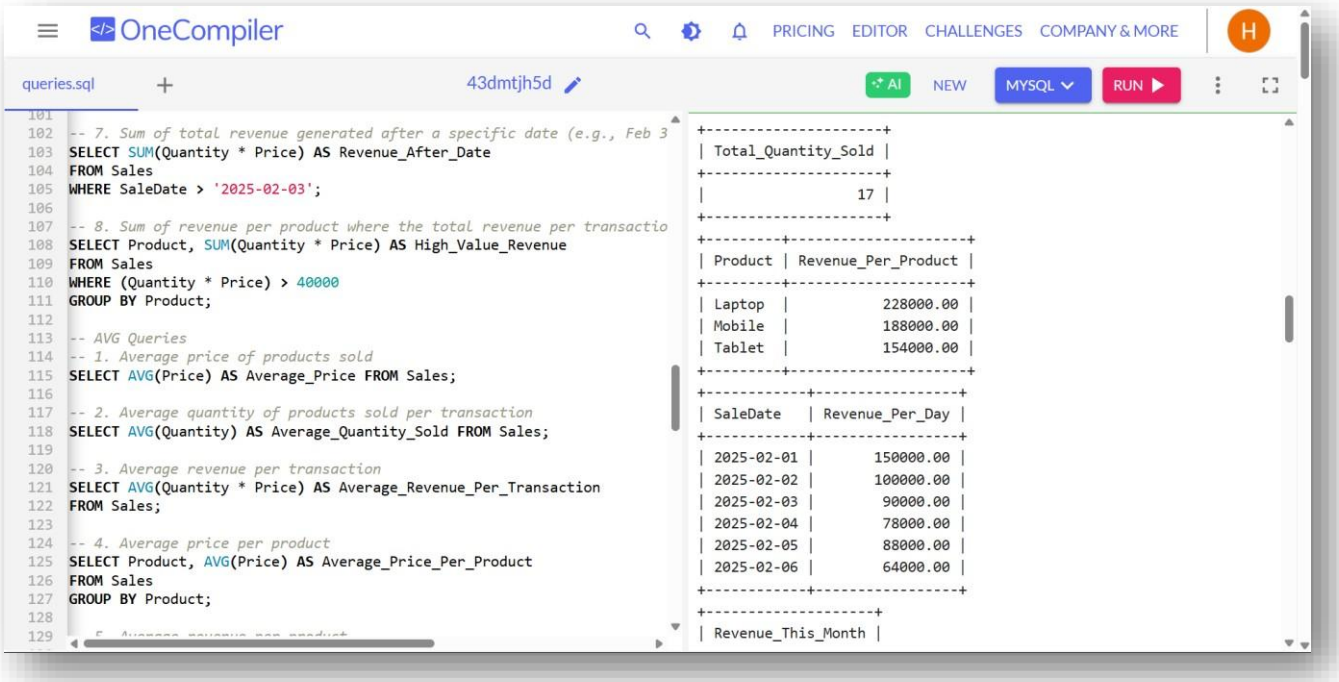
Screenshot of Output:-

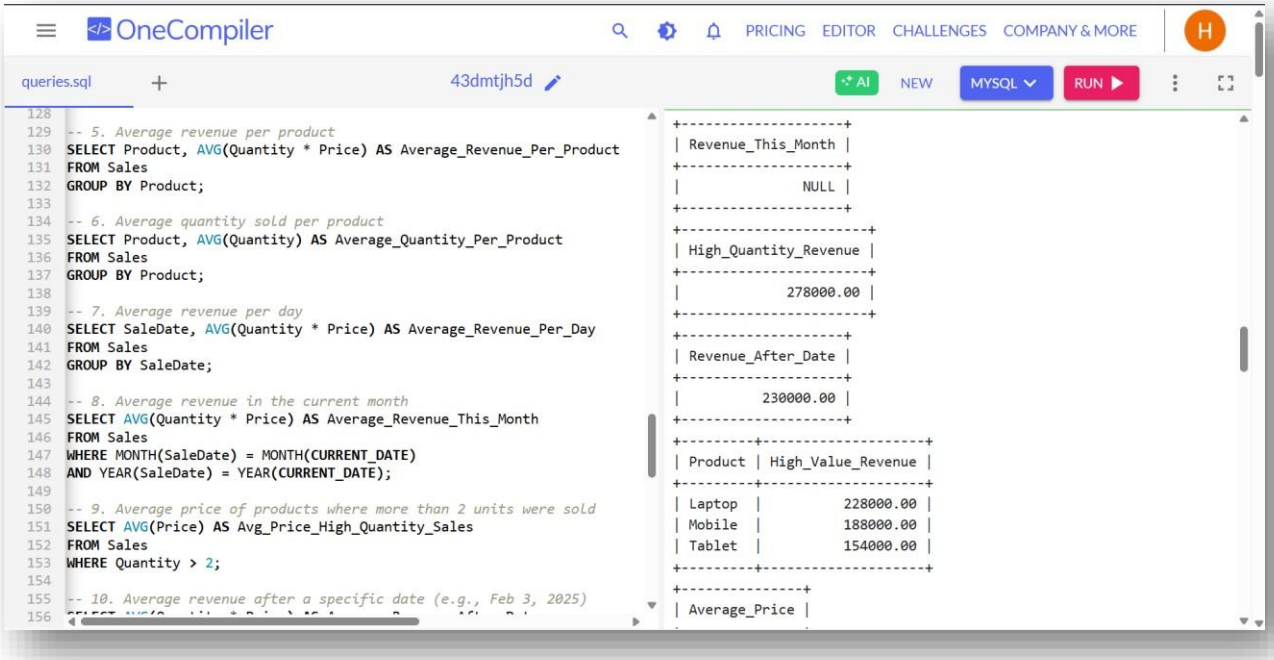


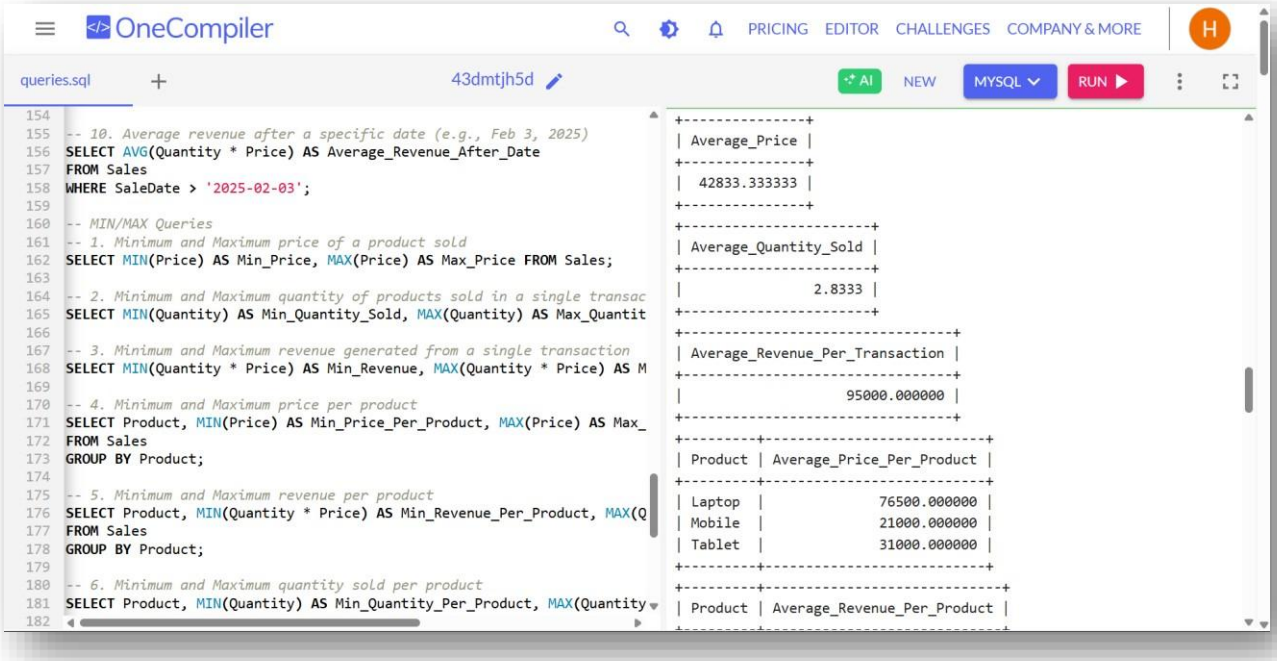

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 3. Average price of products sold
SELECT AVG(Price) AS Average_Price FROM Sales;
-- 4. Minimum and Maximum price of a product sold
SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;
-- 5. Count the number of distinct products sold
SELECT COUNT(DISTINCT Product) AS Unique_Products FROM Sales;
-- 6. Count the number of sales per product
SELECT Product, COUNT(*) AS Sales_Count
FROM Sales
GROUP BY Product;
-- 7. Count the number of sales per day
SELECT SaleDate, COUNT(*) AS Sales_Per_Day
FROM Sales
GROUP BY SaleDate;
-- 8. Count the number of sales where more than 2 units were sold
SELECT COUNT(*) AS High_Quantity_Sales
FROM Sales
WHERE Quantity > 2;
-- 9. Count the number of sales in the current month
SELECT COUNT(*) AS Sales_This_Month
FROM Sales
| Average_Price |
| 42833.333333 |
| Min_Price | Max_Price |
| 20000.00 | 78000.00 |
| Unique_Products |
| 3 |
Product	Sales_Count
Laptop	2
Mobile	2
Tablet	2
SaleDate	Sales_Per_Day

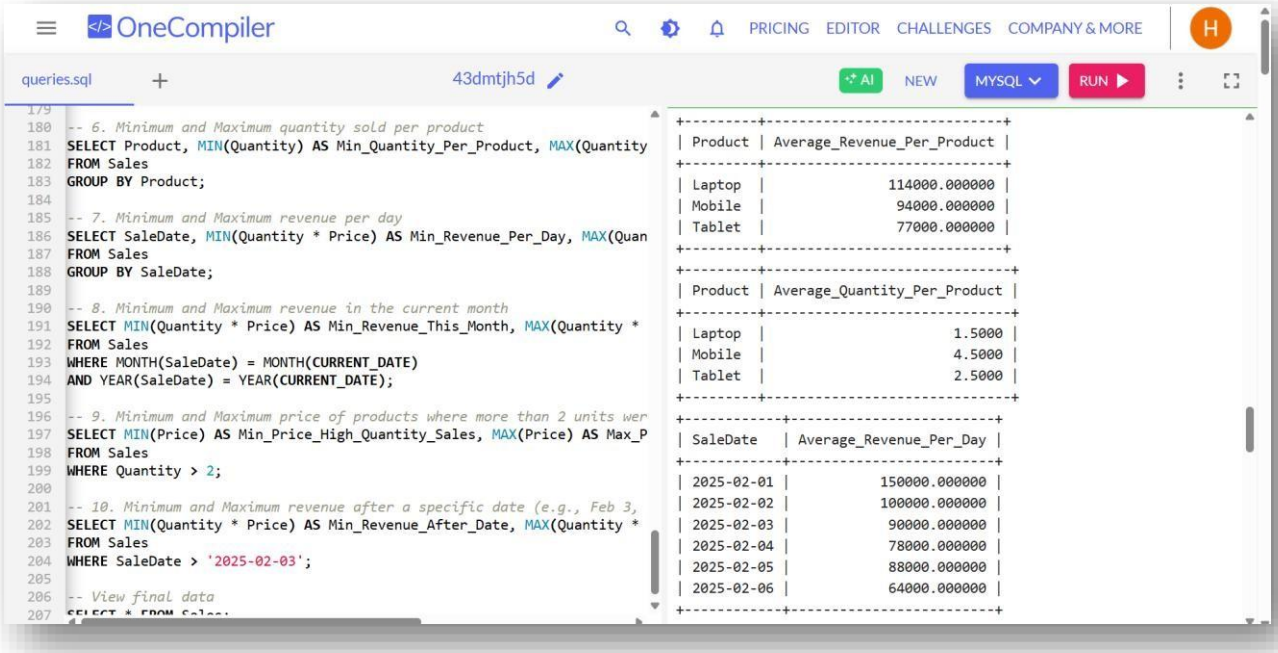

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 9. Count the number of sales in the current month
SELECT COUNT(*) AS Sales_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
-- 10. Count the number of sales transactions where total sale value was
SELECT COUNT(*) AS High_Value_Sales
FROM Sales
WHERE (Quantity * Price) > 50000;
-- 11. Count the number of sales records for each product where total sa
SELECT Product, COUNT(*) AS High_Value_Transactions
FROM Sales
WHERE (Quantity * Price) > 40000
GROUP BY Product;
-- 12. Count the number of sales made after a specific date (e.g., Feb 3
SELECT COUNT(*) AS Sales_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';
-- SUM Queries
-- 1. Sum of total revenue generated
SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;
-- 2. Sum of total quantity of products sold
SELECT SUM(Quantity) AS Total_Quantity_Sold FROM Sales;
SaleDate	Sales_Per_Day
2025-02-01	1
2025-02-02	1
2025-02-03	1
2025-02-04	1
2025-02-05	1
2025-02-06	1
High_Quantity_Sales
3
Sales_This_Month
0
High_Value_Sales
6

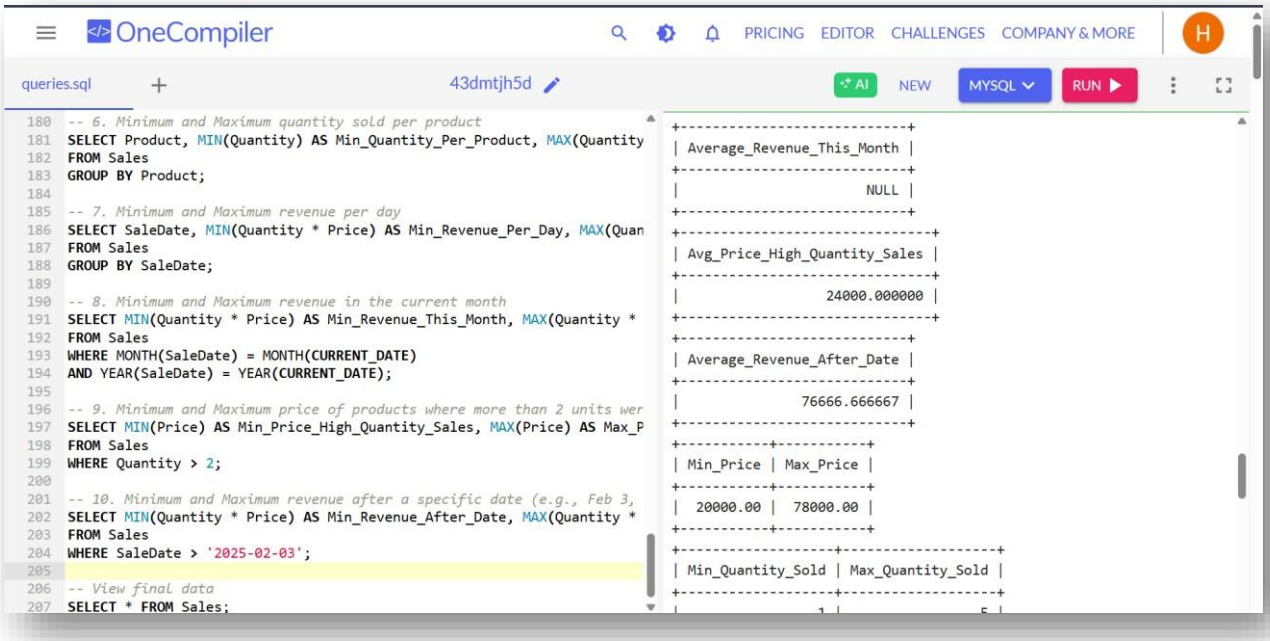

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- SUM Queries
-- 1. Sum of total revenue generated
SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;
-- 2. Sum of total quantity of products sold
SELECT SUM(Quantity) AS Total_Quantity_Sold FROM Sales;
-- 3. Sum of total revenue per product
SELECT Product, SUM(Quantity * Price) AS Revenue_Per_Product
FROM Sales
GROUP BY Product;
-- 4. Sum of total revenue per day
SELECT SaleDate, SUM(Quantity * Price) AS Revenue_Per_Day
FROM Sales
GROUP BY SaleDate;
-- 5. Sum of total revenue in the current month
SELECT SUM(Quantity * Price) AS Revenue_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
-- 6. Sum of revenue for sales where quantity sold is greater than 2
SELECT SUM(Quantity * Price) AS High_Quantity_Revenue
FROM Sales
WHERE Quantity > 2;
| High_Value_Sales |
| 6 |
Product	High_Value_Transactions
Laptop	2
Mobile	2
Tablet	2
Sales_After_Date
3
Total_Revenue
570000.00
Total_Quantity_Sold

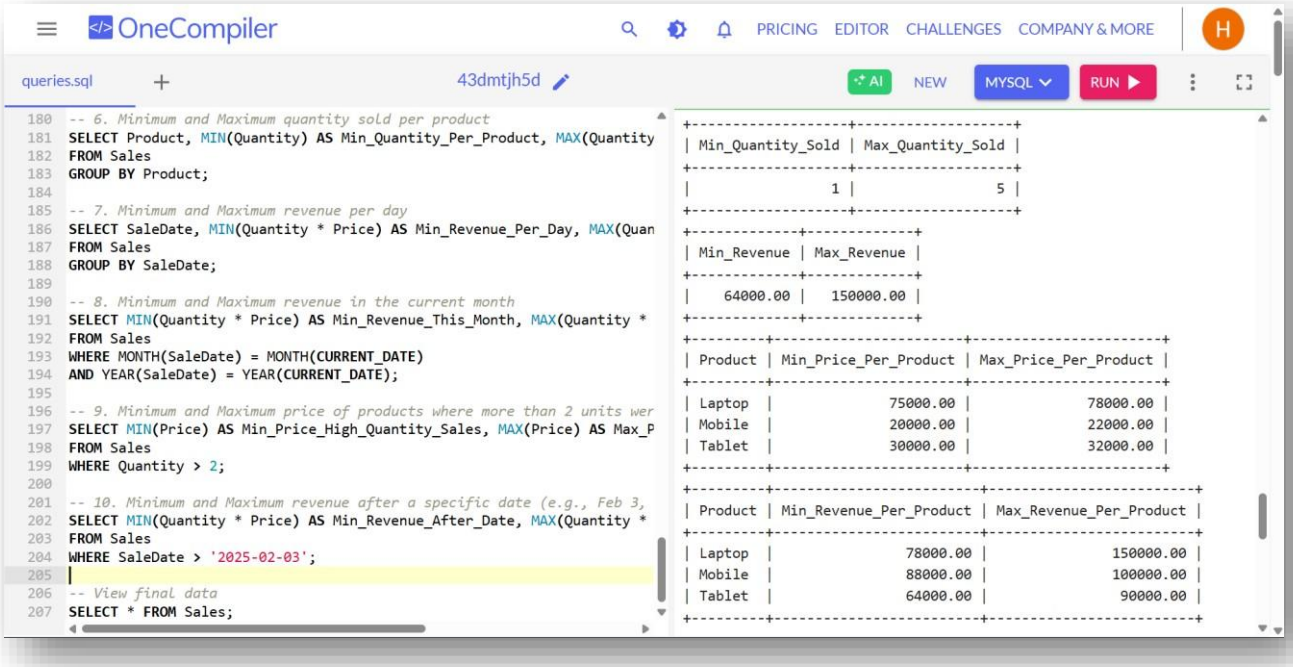

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 7. Sum of total revenue generated after a specific date (e.g., Feb 3
SELECT SUM(Quantity * Price) AS Revenue_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';
-- 8. Sum of revenue per product where the total revenue per transactio
SELECT Product, SUM(Quantity * Price) AS High_Value_Revenue
FROM Sales
WHERE (Quantity * Price) > 40000
GROUP BY Product;
-- AVG Queries
-- 1. Average price of products sold
SELECT AVG(Price) AS Average_Price FROM Sales;
-- 2. Average quantity of products sold per transaction
SELECT AVG(Quantity) AS Average_Quantity_Sold FROM Sales;
-- 3. Average revenue per transaction
SELECT AVG(Quantity * Price) AS Average_Revenue_Per_Transaction
FROM Sales;
-- 4. Average price per product
SELECT Product, AVG(Price) AS Average_Price_Per_Product
FROM Sales
GROUP BY Product;
| Total_Quantity_Sold |
| 17 |
Product	Revenue_Per_Product
Laptop	228000.00
Mobile	188000.00
Tablet	154000.00
SaleDate	Revenue_Per_Day
2025-02-01	150000.00
2025-02-02	100000.00
2025-02-03	90000.00
2025-02-04	78000.00
2025-02-05	88000.00
2025-02-06	64000.00
Revenue_This_Month

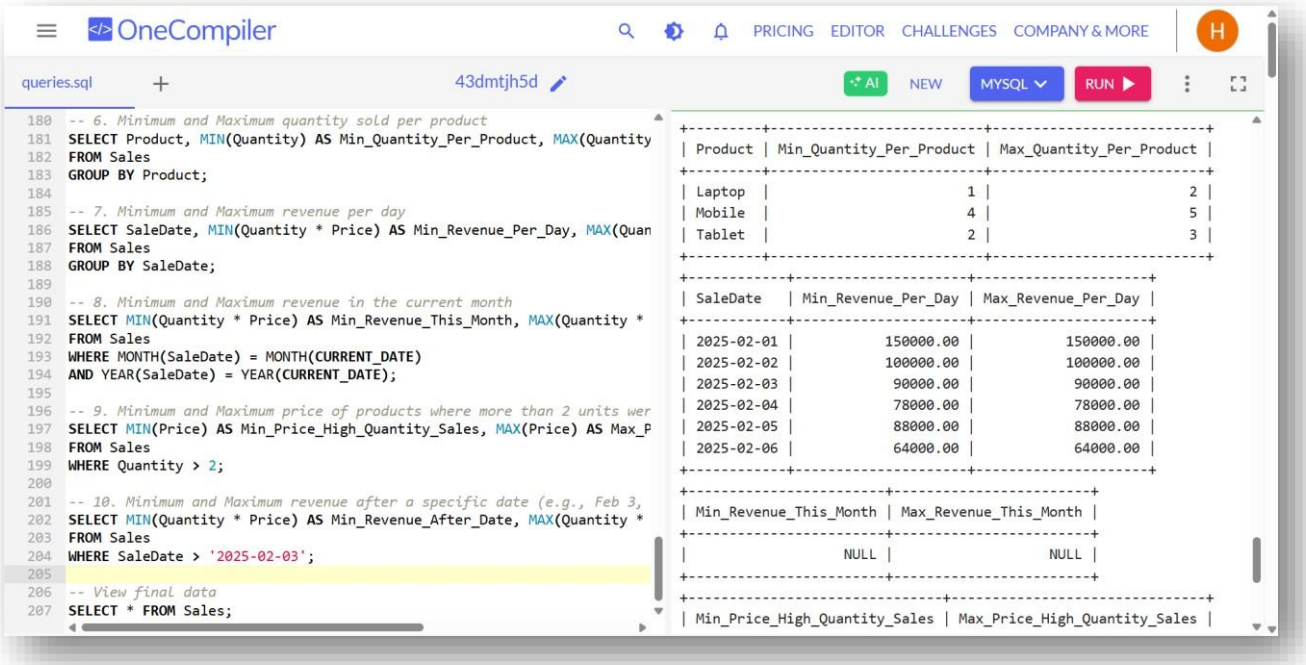

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 5. Average revenue per product
SELECT Product, AVG(Quantity * Price) AS Average_Revenue_Per_Product
FROM Sales
GROUP BY Product;
-- 6. Average quantity sold per product
SELECT Product, AVG(Quantity) AS Average_Quantity_Per_Product
FROM Sales
GROUP BY Product;
-- 7. Average revenue per day
SELECT SaleDate, AVG(Quantity * Price) AS Average_Revenue_Per_Day
FROM Sales
GROUP BY SaleDate;
-- 8. Average revenue in the current month
SELECT AVG(Quantity * Price) AS Average_Revenue_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
-- 9. Average price of products where more than 2 units were sold
SELECT AVG(Price) AS Avg_Price_High_Quantity_Sales
FROM Sales
WHERE Quantity > 2;
-- 10. Average revenue after a specific date (e.g., Feb 3, 2025)
| Revenue_This_Month |
| NULL |
| High_Quantity_Revenue |
| 278000.00 |
| Revenue_After_Date |
| 230000.00 |
Product	High_Value_Revenue
Laptop	228000.00
Mobile	188000.00
Tablet	154000.00
Average_Price

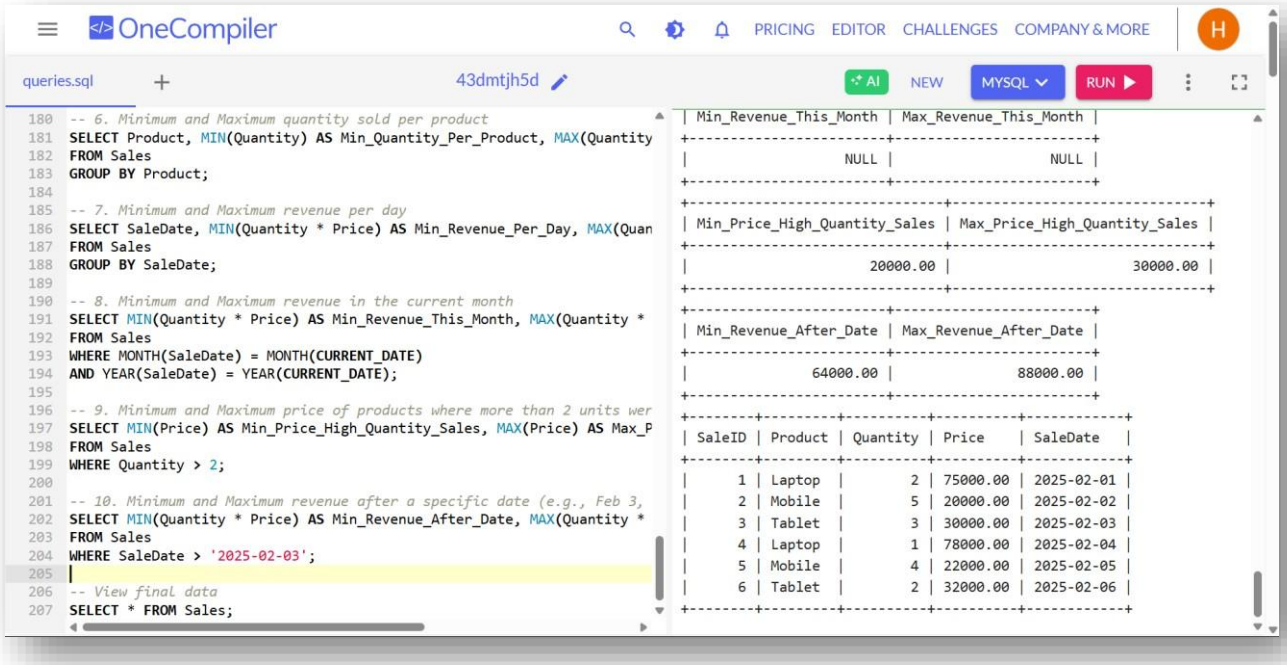

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 10. Average revenue after a specific date (e.g., Feb 3, 2025)
SELECT AVG(Quantity * Price) AS Average_Revenue_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';
-- MIN/MAX Queries
-- 1. Minimum and Maximum price of a product sold
SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;
-- 2. Minimum and Maximum quantity of products sold in a single transac
SELECT MIN(Quantity) AS Min_Quantity_Sold, MAX(Quantity) AS Max_Quantit
-- 3. Minimum and Maximum revenue generated from a single transaction
SELECT MIN(Quantity * Price) AS Min_Revenue, MAX(Quantity * Price) AS M
-- 4. Minimum and Maximum price per product
SELECT Product, MIN(Price) AS Min_Price_Per_Product, MAX(Price) AS Max_
FROM Sales
GROUP BY Product;
-- 5. Minimum and Maximum revenue per product
SELECT Product, MIN(Quantity * Price) AS Min_Revenue_Per_Product, MAX(Q
FROM Sales
GROUP BY Product;
-- 6. Minimum and Maximum quantity sold per product
SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity
| Average_Price |
| 42833.333333 |
| Average_Quantity_Sold |
| 2.8333 |
| Average_Revenue_Per_Transaction |
| 95000.000000 |
Product	Average_Price_Per_Product
Laptop	76500.000000
Mobile	21000.000000
Tablet	31000.000000
Product	Average_Revenue_Per_Product


OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 6. Minimum and Maximum quantity sold per product
SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity
FROM Sales
GROUP BY Product;
-- 7. Minimum and Maximum revenue per day
SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quan
FROM Sales
GROUP BY SaleDate;
-- 8. Minimum and Maximum revenue in the current month
SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity *
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
-- 9. Minimum and Maximum price of products where more than 2 units wer
SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_P
FROM Sales
WHERE Quantity > 2;
-- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3,
SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity *
FROM Sales
WHERE SaleDate > '2025-02-03';
-- View final data
Product	Average_Revenue_Per_Product
Laptop	114000.000000
Mobile	94000.000000
Tablet	77000.000000
Product	Average_Quantity_Per_Product
Laptop	1.5000
Mobile	4.5000
Tablet	2.5000
SaleDate	Average_Revenue_Per_Day
2025-02-01	150000.000000
2025-02-02	100000.000000
2025-02-03	90000.000000
2025-02-04	78000.000000
2025-02-05	88000.000000
2025-02-06	64000.000000


OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 6. Minimum and Maximum quantity sold per product
SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity
FROM Sales
GROUP BY Product;
-- 7. Minimum and Maximum revenue per day
SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quan
FROM Sales
GROUP BY SaleDate;
-- 8. Minimum and Maximum revenue in the current month
SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity *
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
-- 9. Minimum and Maximum price of products where more than 2 units wer
SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_P
FROM Sales
WHERE Quantity > 2;
-- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3,
SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity *
FROM Sales
WHERE SaleDate > '2025-02-03';
-- View final data
SELECT * FROM Sales;
| Average_Revenue_This_Month |
| NULL |
| Avg_Price_High_Quantity_Sales |
| 24000.000000 |
| Average_Revenue_After_Date |
| 76666.666667 |
Min_Price	Max_Price
20000.00	78000.00
Min_Quantity_Sold	Max_Quantity_Sold


OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 6. Minimum and Maximum quantity sold per product
SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity
FROM Sales
GROUP BY Product;
-- 7. Minimum and Maximum revenue per day
SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quan
FROM Sales
GROUP BY SaleDate;
-- 8. Minimum and Maximum revenue in the current month
SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity *
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
-- 9. Minimum and Maximum price of products where more than 2 units wer
SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_P
FROM Sales
WHERE Quantity > 2;
-- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3,
SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity *
FROM Sales
WHERE SaleDate > '2025-02-03';
-- View final data
SELECT * FROM Sales;
Min_Quantity_Sold	Max_Quantity_Sold
1	5
Min_Revenue	Max_Revenue
64000.00	150000.00
Product	Min_Price_Per_Product	Max_Price_Per_Product
Laptop	75000.00	78000.00
Mobile	20000.00	22000.00
Tablet	30000.00	32000.00
Product	Min_Revenue_Per_Product	Max_Revenue_Per_Product
Laptop	78000.00	150000.00
Mobile	88000.00	100000.00
Tablet	64000.00	90000.00


OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 6. Minimum and Maximum quantity sold per product
SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity
FROM Sales
GROUP BY Product;
-- 7. Minimum and Maximum revenue per day
SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quan
FROM Sales
GROUP BY SaleDate;
-- 8. Minimum and Maximum revenue in the current month
SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity *
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
-- 9. Minimum and Maximum price of products where more than 2 units wer
SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_P
FROM Sales
WHERE Quantity > 2;
-- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3,
SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity *
FROM Sales
WHERE SaleDate > '2025-02-03';
-- View final data
SELECT * FROM Sales;
Product	Min_Quantity_Per_Product	Max_Quantity_Per_Product
Laptop	1	2
Mobile	4	5
Tablet	2	3
SaleDate	Min_Revenue_Per_Day	Max_Revenue_Per_Day
2025-02-01	150000.00	150000.00
2025-02-02	100000.00	100000.00
2025-02-03	90000.00	90000.00
2025-02-04	78000.00	78000.00
2025-02-05	88000.00	88000.00
2025-02-06	64000.00	64000.00
Min_Revenue_This_Month	Max_Revenue_This_Month
NULL	NULL
Min_Price_High_Quantity_Sales	Max_Price_High_Quantity_Sales


OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 6. Minimum and Maximum quantity sold per product
SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity
FROM Sales
GROUP BY Product;
-- 7. Minimum and Maximum revenue per day
SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quan
FROM Sales
GROUP BY SaleDate;
-- 8. Minimum and Maximum revenue in the current month
SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity *
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
-- 9. Minimum and Maximum price of products where more than 2 units wer
SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_P
FROM Sales
WHERE Quantity > 2;
-- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3,
SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity *
FROM Sales
WHERE SaleDate > '2025-02-03';
-- View final data
SELECT * FROM Sales;
Min_Revenue_This_Month	Max_Revenue_This_Month
NULL	NULL
Min_Price_High_Quantity_Sales	Max_Price_High_Quantity_Sales
20000.00	30000.00
Min_Revenue_After_Date	Max_Revenue_After_Date
64000.00	88000.00
SaleID	Product	Quantity	Price	SaleDate
1	Laptop	2	75000.00	2025-02-01
2	Mobile	5	20000.00	2025-02-02
3	Tablet	3	30000.00	2025-02-03
4	Laptop	1	78000.00	2025-02-04
5	Mobile	4	22000.00	2025-02-05
6	Tablet	2	32000.00	2025-02-06

Practical No.:- 8

Aim:-

Given Customers and Orders tables, write SQL queries to perform INNER JOIN, LEFT JOIN, and RIGHT JOIN to retrieve combined data for customer orders.

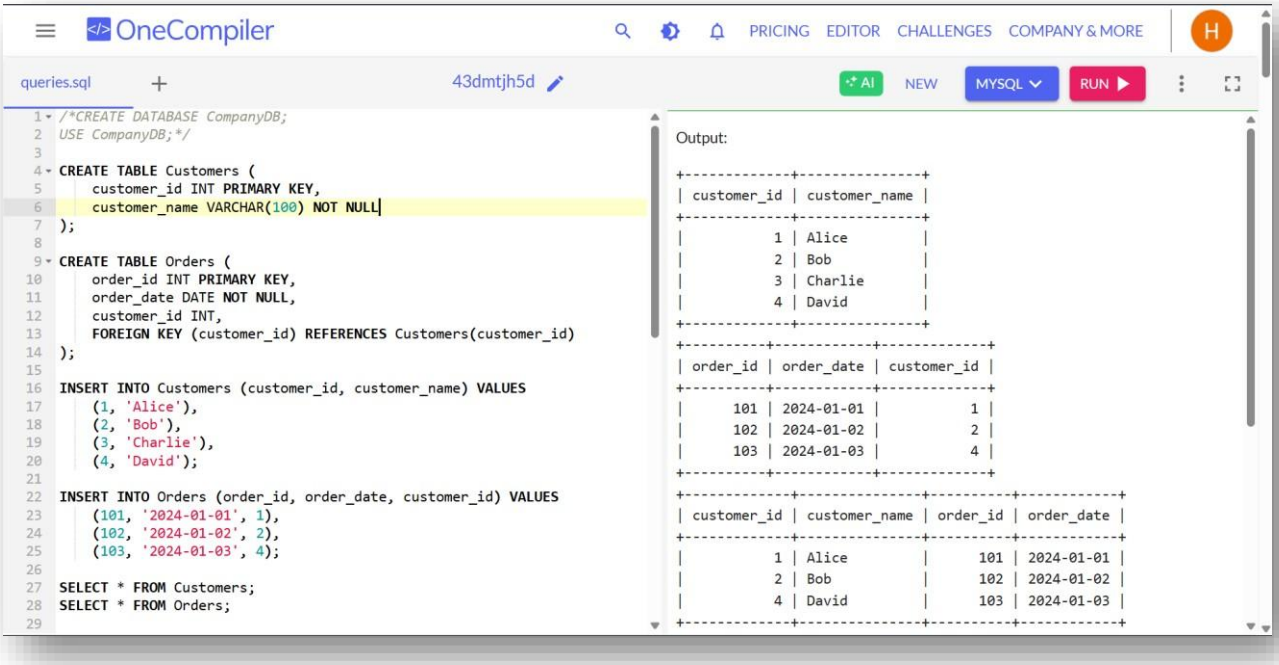
Code:-

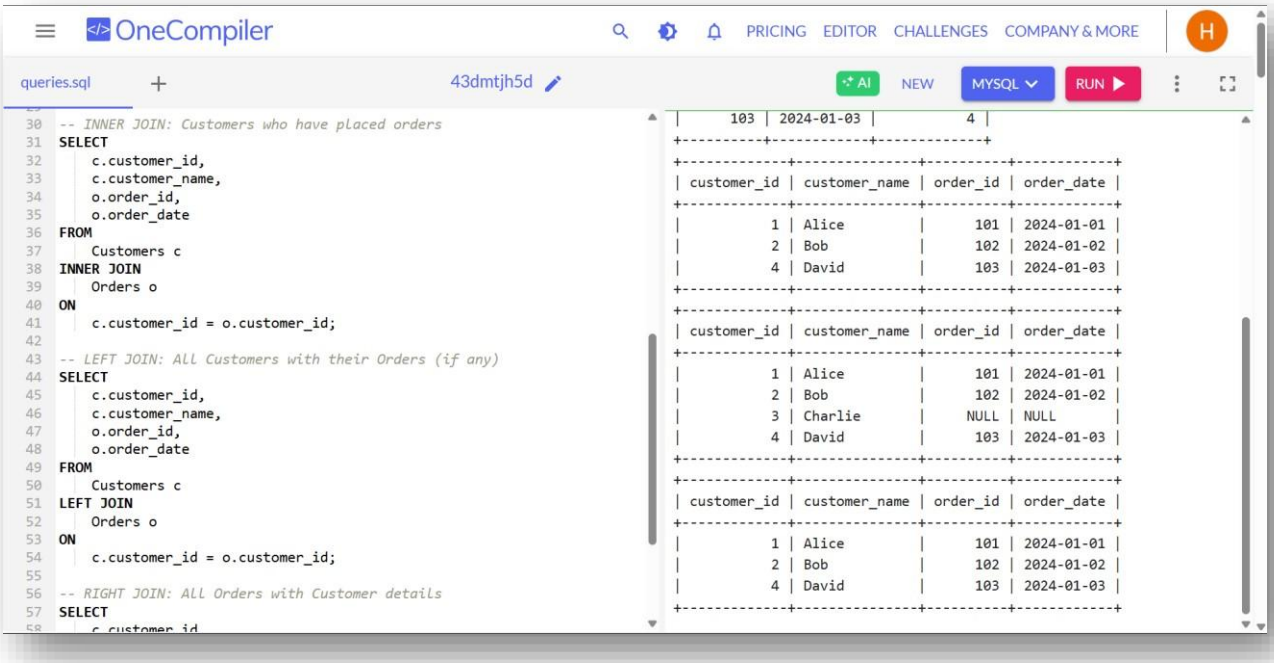
```
CREATE DATABASE CompanyDB;
USE CompanyDB;

CREATE TABLE Customers ( customer_id
INT PRIMARY KEY, customer_name
VARCHAR(100) NOT NULL
);
CREATE TABLE Orders (
order_id INT PRIMARY KEY,
order_date DATE NOT NULL,
   customer_id INT,
   FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);

INSERT INTO Customers (customer_id, customer_name) VALUES
   (1, 'Alice'),
   (2, 'Bob'),
```

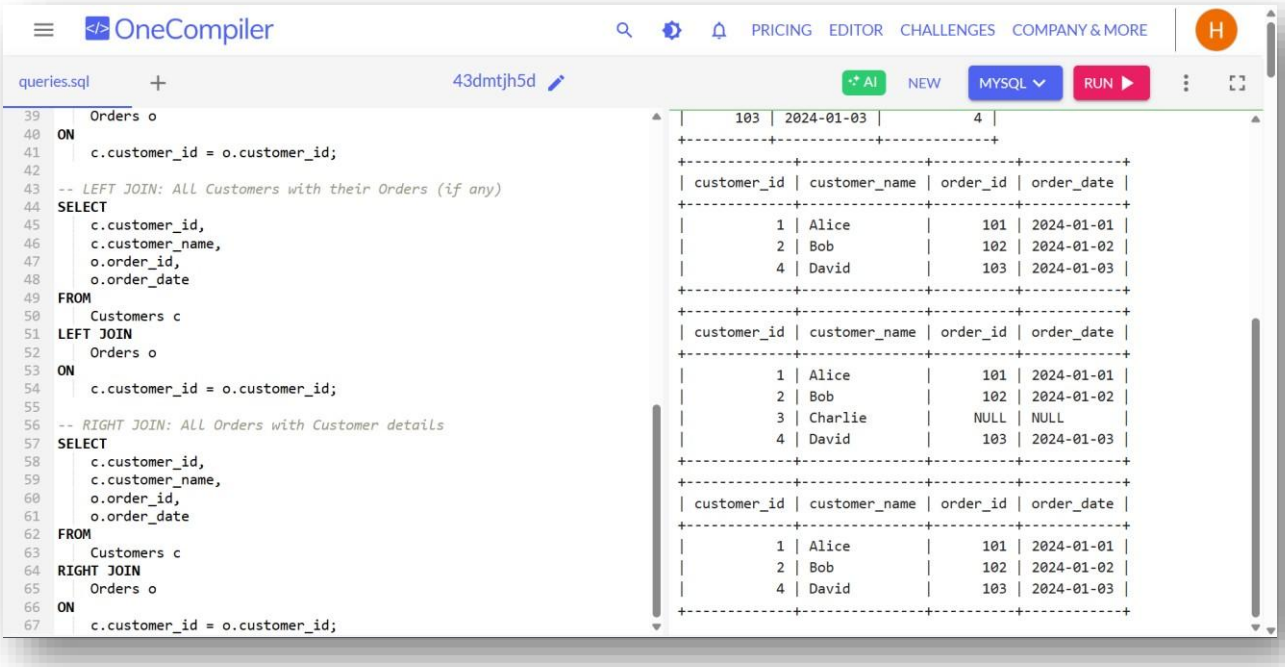
```
    (3, 'Charlie'),
    (4, 'David');

INSERT INTO Orders (order_id, order_date, customer_id) VALUES
    (101, '2024-01-01', 1),
    (102, '2024-01-02', 2),
    (103, '2024-01-03', 4);

SELECT * FROM Customers;
SELECT * FROM Orders;

-- INNER JOIN: Customers who have placed orders
SELECT
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_date
FROM
    Customers c
INNER JOIN
    Orders o
ON
    c.customer_id = o.customer_id;

-- LEFT JOIN: All Customers with their Orders (if any)
```

```
SELECT
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_date
FROM
    Customers c
LEFT JOIN
    Orders o
ON
    c.customer_id = o.customer_id;

-- RIGHT JOIN: All Orders with Customer details
SELECT
    c.customer_id,
    c.customer_name,
    o.order_id,
    o.order_date
FROM
    Customers c
RIGHT JOIN
    Orders o
ON
    c.customer_id = o.customer_id;
```

Screenshot of Output:-







Name:- Mayank Dhule

PRN:- 2124UCSM1027

Dept.:- Cyber Security