Practical No.:- 1

Install and set up MySQL.

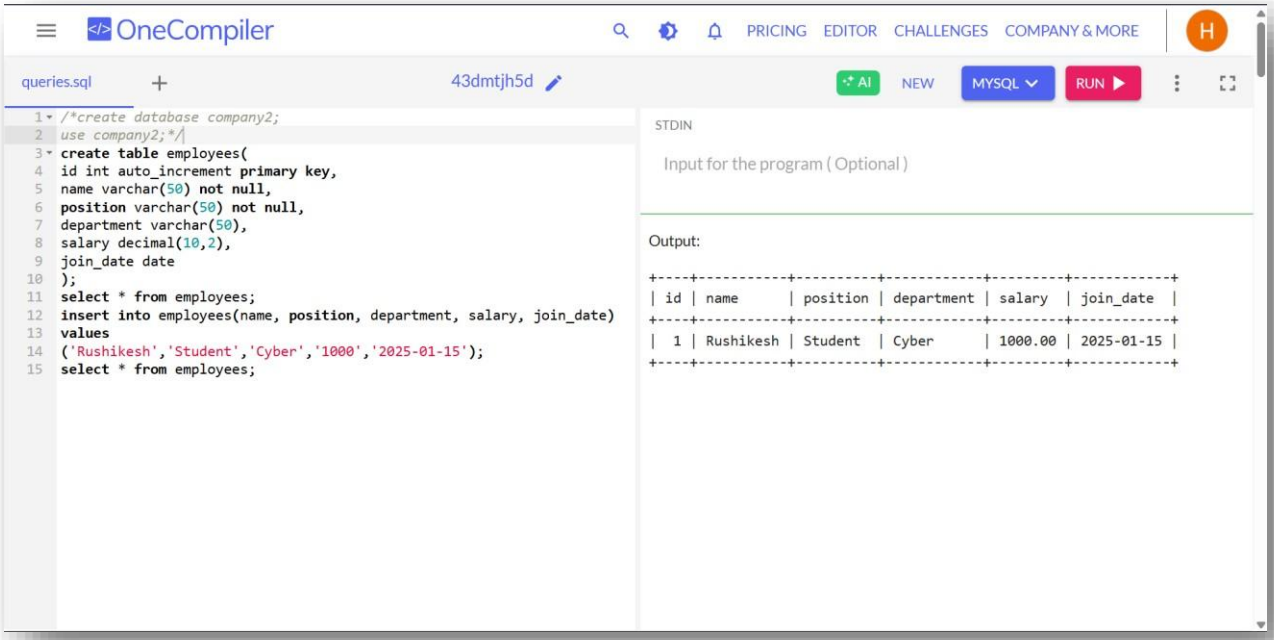
Aim:-

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
('Rushikesh','Student','Cyber','1000','2025-01-15'); select
* from employees;
```

Screenshot of Output:-



Practical No.:- 2

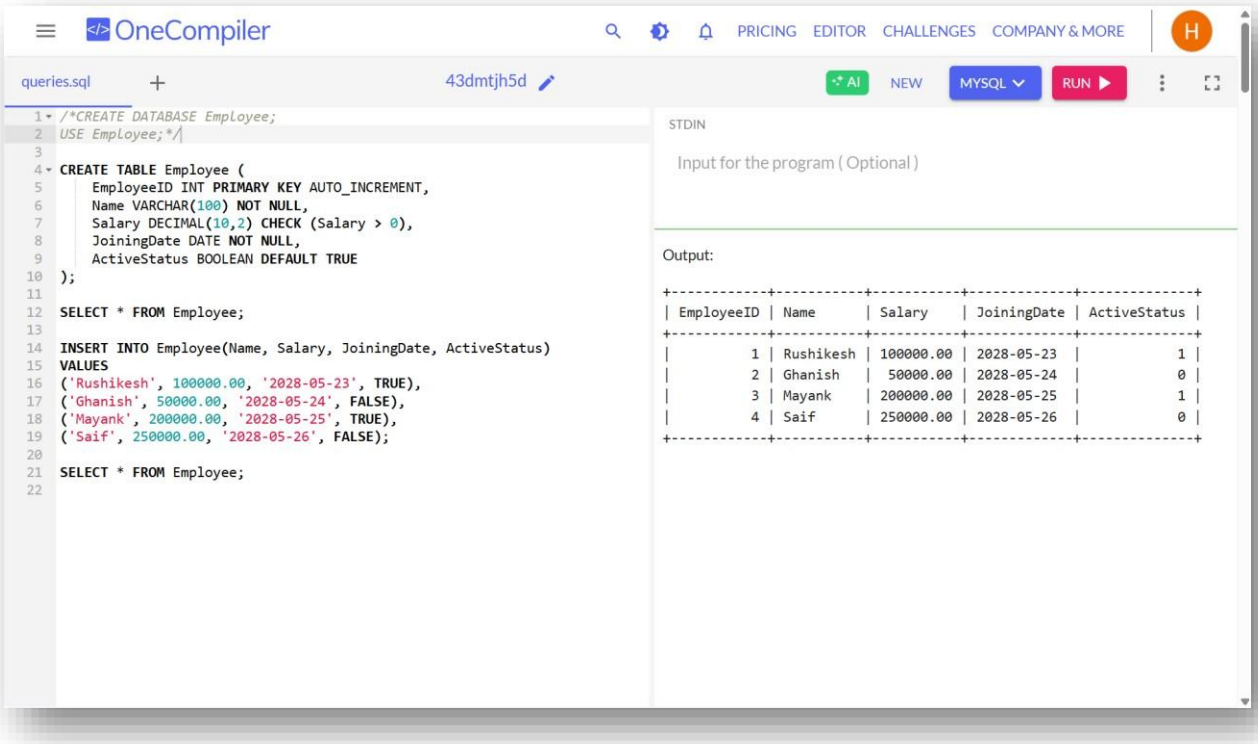
Aim:-

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

);

select * from Employee; insert into

Employee(Name,salary,JoiningDate,ActiveStatus) values

('Rushikesh',100000.00,'23-05-2028',true),

('Ghanish',50000.00,'24-05-2028',false),

('Mayank',200000.00,'25-05-2028',true),

('Saif',250000.00,'26-05-2028',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 DATABASE COMPANY;

USE COMPANY;

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

);

select * from Employee;


INSERT INTO Employee(Name,Salary, JoiningDate,ActiveStatus)

VALUES

('Rushikesh Patil',55000.00,'2023-06-15',TRUE),

('Ghanish Patil',60000.00,'2022-06-15',TRUE),

('mayank dhule',70000.00,'2021-06-15',FALSE),

('Omkar Shelar',1000000.00,'2020-06-15',TRUE);


select * from Employee;
```

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

select * from Employee;

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

select * from Employee;

UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID =
4;

select * from Employee;

DELETE FROM Employee WHERE EmployeeID = 3;

select * from Employee;

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

select * from Employee;

SELECT Name, Salary FROM Employee WHERE Salary > 60000;
```

select * from Employee;

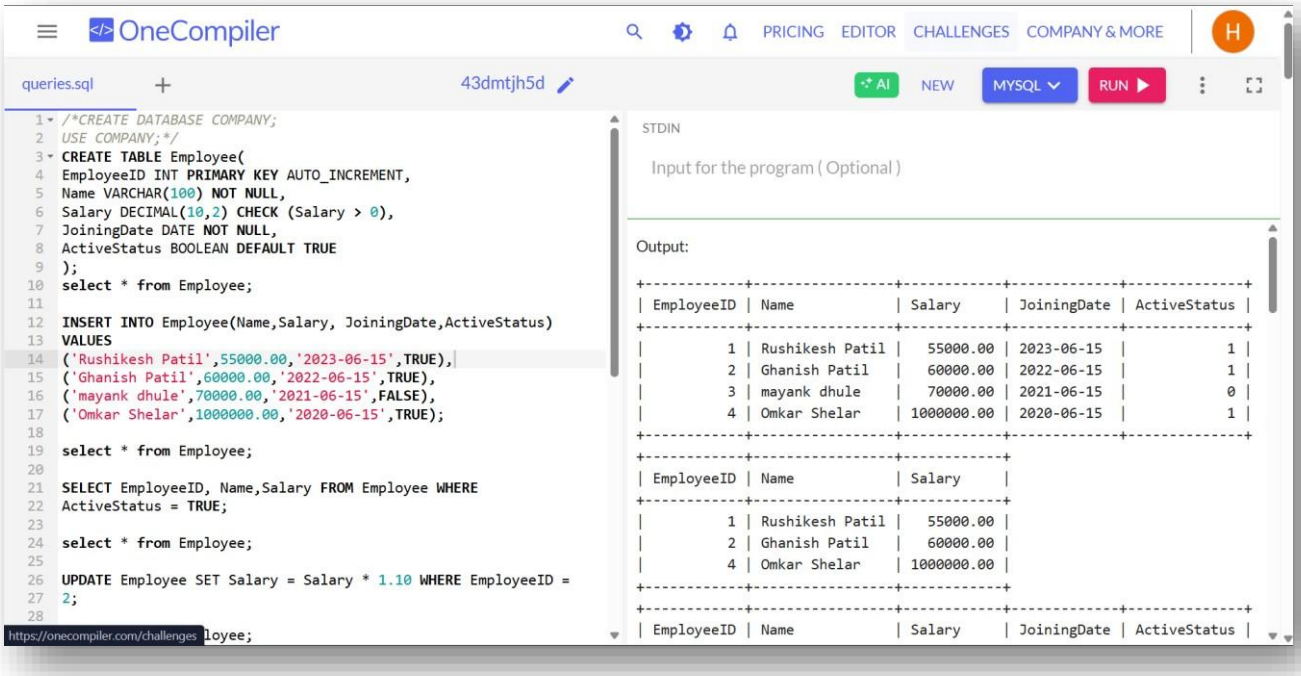
SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary FROM Employee;

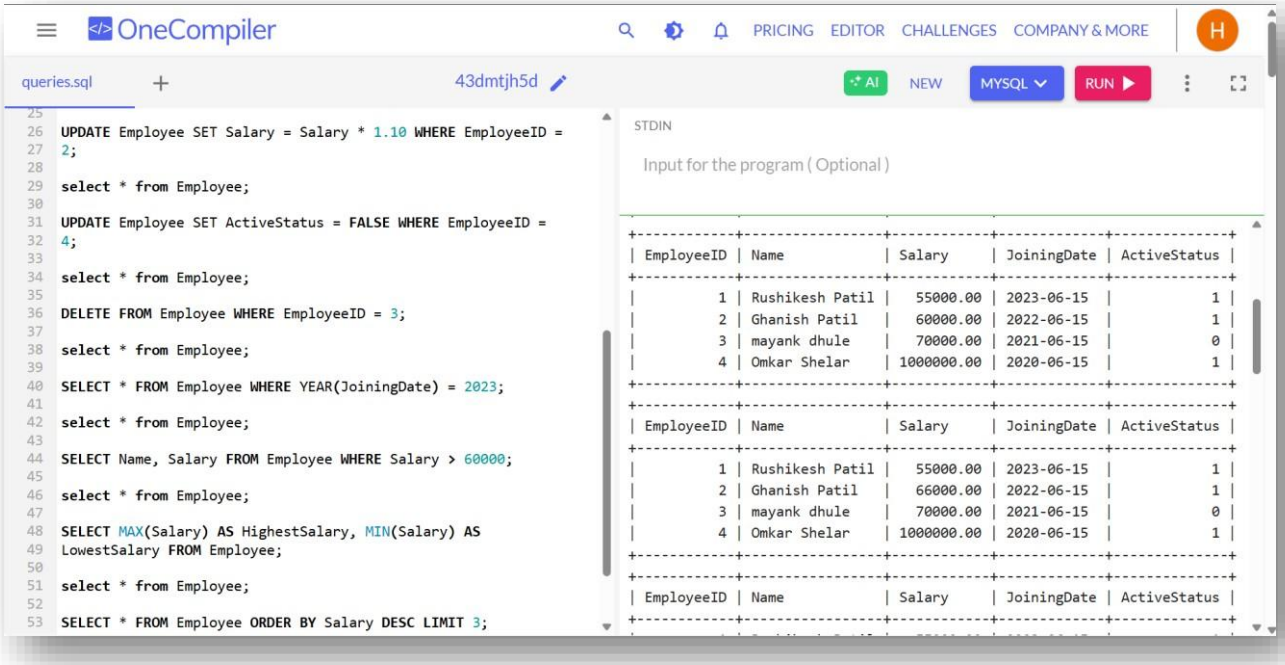
select * from Employee;

SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;

select * from Employee;

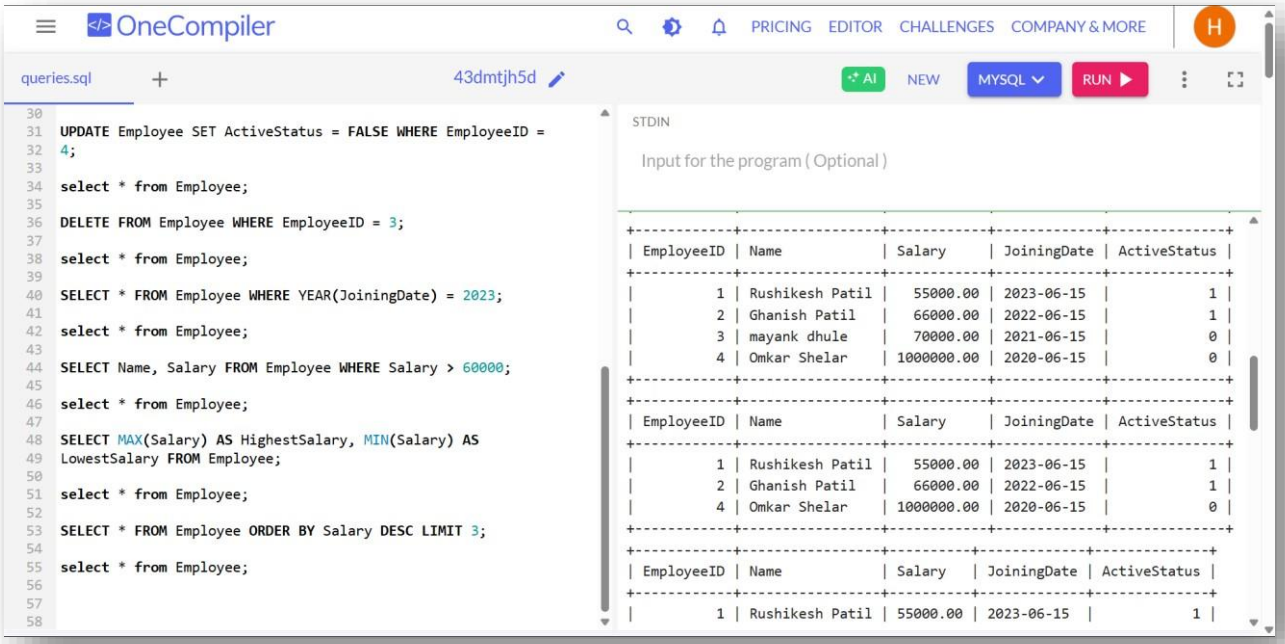
Screenshot of Output:-





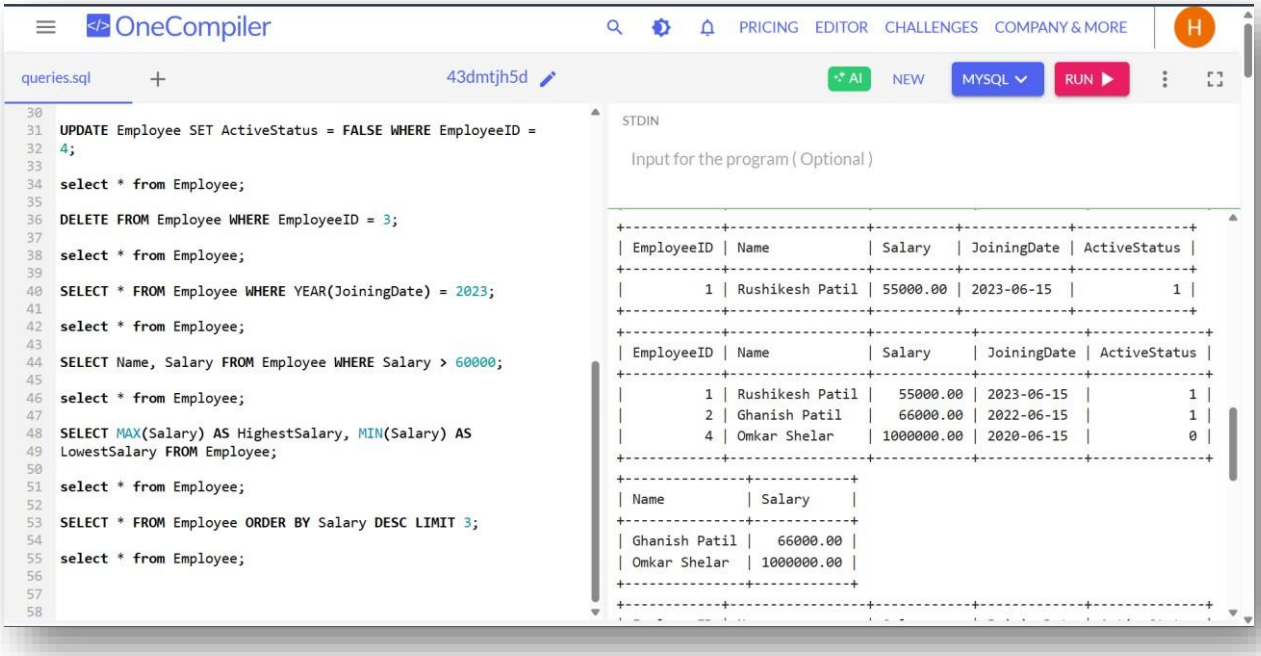

```
UPDATE Employee SET Salary = Salary * 1.10 WHERE EmployeeID =
2;

select * from Employee;

UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID =
4;

select * from Employee;

DELETE FROM Employee WHERE EmployeeID = 3;

select * from Employee;

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

select * from Employee;

SELECT Name, Salary FROM Employee WHERE Salary > 60000;

select * from Employee;

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

select * from Employee;

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

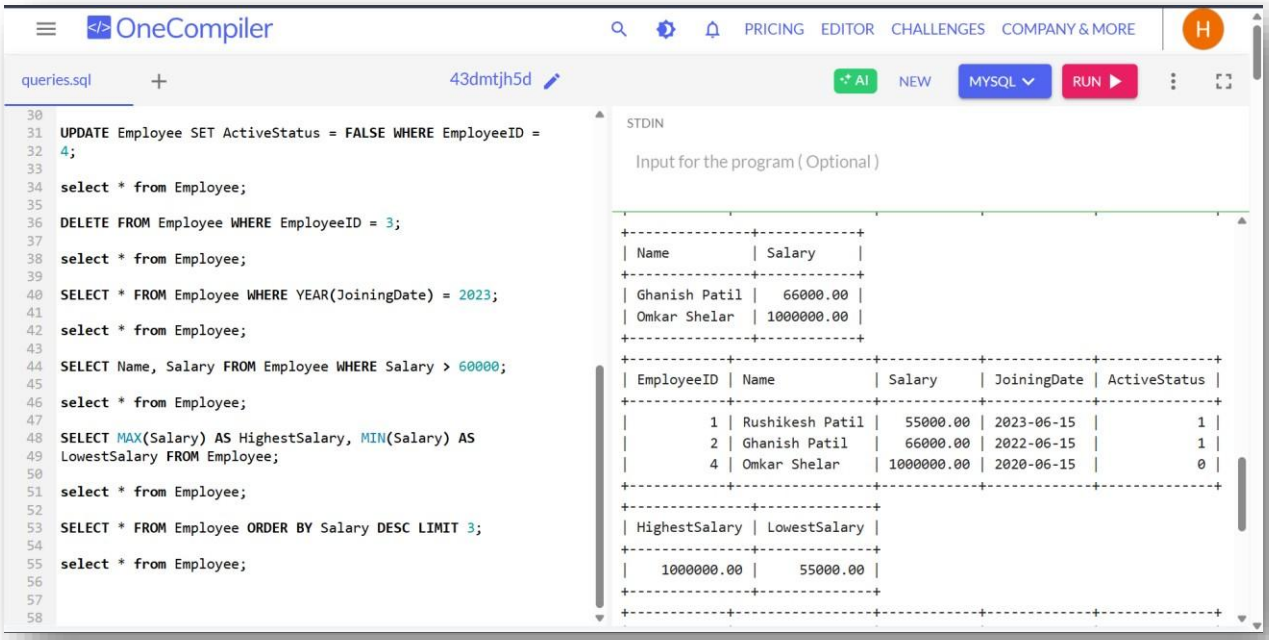
```
+------------+----------------+------------+-------------+--------------+
| EmployeeID | Name           | Salary     | JoiningDate | ActiveStatus |
+------------+----------------+------------+-------------+--------------+
|          1 | Rushikesh Patil |   55000.00 | 2023-06-15  |            1 |
|          2 | Ghanish Patil  |   60000.00 | 2022-06-15  |            1 |
|          3 | mayank dhule   |   70000.00 | 2021-06-15  |            0 |
|          4 | Omkar Shelar   | 1000000.00 | 2020-06-15  |            1 |
+------------+----------------+------------+-------------+--------------+
+------------+----------------+------------+-------------+--------------+
| EmployeeID | Name           | Salary     | JoiningDate | ActiveStatus |
+------------+----------------+------------+-------------+--------------+
|          1 | Rushikesh Patil |   55000.00 | 2023-06-15  |            1 |
|          2 | Ghanish Patil  |   66000.00 | 2022-06-15  |            1 |
|          3 | mayank dhule   |   70000.00 | 2021-06-15  |            0 |
|          4 | Omkar Shelar   | 1000000.00 | 2020-06-15  |            1 |
+------------+----------------+------------+-------------+--------------+
+------------+----------------+------------+-------------+--------------+
| EmployeeID | Name           | Salary     | JoiningDate | ActiveStatus |
+------------+----------------+------------+-------------+--------------+
```

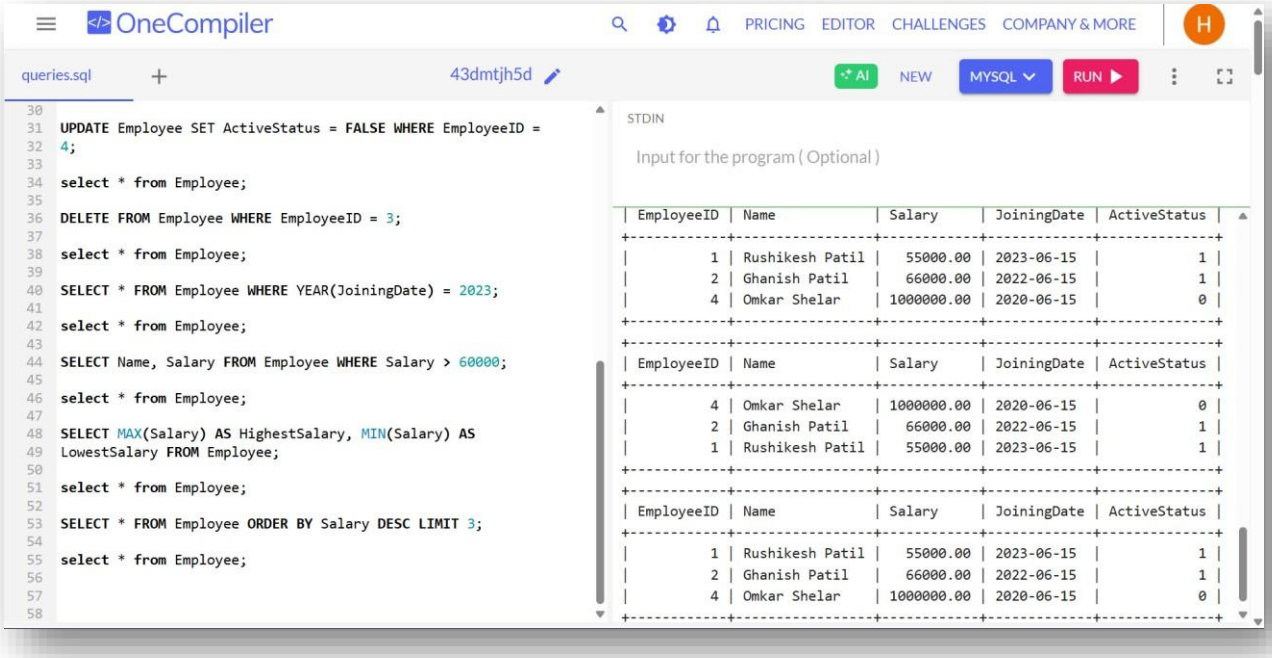



```
UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID =
4;

select * from Employee;

DELETE FROM Employee WHERE EmployeeID = 3;

select * from Employee;

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

select * from Employee;

SELECT Name, Salary FROM Employee WHERE Salary > 60000;

select * from Employee;

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

select * from Employee;

SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;

select * from Employee;
```

```
+------------+----------------+------------+-------------+--------------+
| EmployeeID | Name           | Salary     | JoiningDate | ActiveStatus |
+------------+----------------+------------+-------------+--------------+
|          1 | Rushikesh Patil |   55000.00 | 2023-06-15  |            1 |
|          2 | Ghanish Patil  |   66000.00 | 2022-06-15  |            1 |
|          3 | mayank dhule   |   70000.00 | 2021-06-15  |            0 |
|          4 | Omkar Shelar   | 1000000.00 | 2020-06-15  |            0 |
+------------+----------------+------------+-------------+--------------+
+------------+----------------+------------+-------------+--------------+
| EmployeeID | Name           | Salary     | JoiningDate | ActiveStatus |
+------------+----------------+------------+-------------+--------------+
|          1 | Rushikesh Patil |   55000.00 | 2023-06-15  |            1 |
|          2 | Ghanish Patil  |   66000.00 | 2022-06-15  |            1 |
|          4 | Omkar Shelar   | 1000000.00 | 2020-06-15  |            0 |
+------------+----------------+------------+-------------+--------------+
+------------+----------------+----------+-------------+--------------+
| EmployeeID | Name           | Salary   | JoiningDate | ActiveStatus |
+------------+----------------+----------+-------------+--------------+
|          1 | Rushikesh Patil | 55000.00 | 2023-06-15  |            1 |
```


OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
STDIN
Input for the program (Optional)
UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID = 4;
select * from Employee;
DELETE FROM Employee WHERE EmployeeID = 3;
select * from Employee;
SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
select * from Employee;
SELECT Name, Salary FROM Employee WHERE Salary > 60000;
select * from Employee;
SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary FROM Employee;
select * from Employee;
SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
select * from Employee;
EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Rushikesh Patil	55000.00	2023-06-15	1
EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Rushikesh Patil	55000.00	2023-06-15	1
2	Ghanish Patil	66000.00	2022-06-15	1
4	Omkar Shelar	1000000.00	2020-06-15	0
Name	Salary
Ghanish Patil	66000.00
Omkar Shelar	1000000.00


OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
STDIN
Input for the program (Optional)
UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID = 4;
select * from Employee;
DELETE FROM Employee WHERE EmployeeID = 3;
select * from Employee;
SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
select * from Employee;
SELECT Name, Salary FROM Employee WHERE Salary > 60000;
select * from Employee;
SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary FROM Employee;
select * from Employee;
SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
select * from Employee;
Name	Salary
Ghanish Patil	66000.00
Omkar Shelar	1000000.00
EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Rushikesh Patil	55000.00	2023-06-15	1
2	Ghanish Patil	66000.00	2022-06-15	1
4	Omkar Shelar	1000000.00	2020-06-15	0
HighestSalary	LowestSalary
1000000.00	55000.00


OneCompiler
PRICING
EDITOR
CHALLENGES
COMPANY & MORE
queries.sql
43dmtjh5d
AI
NEW
MYSQL
RUN
STDIN
Input for the program (Optional)
UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID = 4;
select * from Employee;
DELETE FROM Employee WHERE EmployeeID = 3;
select * from Employee;
SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
select * from Employee;
SELECT Name, Salary FROM Employee WHERE Salary > 60000;
select * from Employee;
SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary FROM Employee;
select * from Employee;
SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
select * from Employee;
EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Rushikesh Patil	55000.00	2023-06-15	1
2	Ghanish Patil	66000.00	2022-06-15	1
4	Omkar Shelar	1000000.00	2020-06-15	0
EmployeeID	Name	Salary	JoiningDate	ActiveStatus
4	Omkar Shelar	1000000.00	2020-06-15	0
2	Ghanish Patil	66000.00	2022-06-15	1
1	Rushikesh Patil	55000.00	2023-06-15	1
EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Rushikesh Patil	55000.00	2023-06-15	1
2	Ghanish Patil	66000.00	2022-06-15	1
4	Omkar Shelar	1000000.00	2020-06-15	0

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 TABLE Department (
DeptID INT PRIMARY KEY,
DeptName VARCHAR(50) UNIQUE
);
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,'HR');
INSERT INTO Department (DeptID, DeptName) VALUES (2,'IT');

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (101,
```

'Alice', 'alice@example.com', 50000.00, 1);
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (102, 'Bob','bob@example.com', 60000.00, 2);

select * from Employee;

--Insert Invalid Data to Test Constraints
--Duplicate Primary Key

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (101, 'Charlie', 'charlie@example.com', 55000.00, 1);

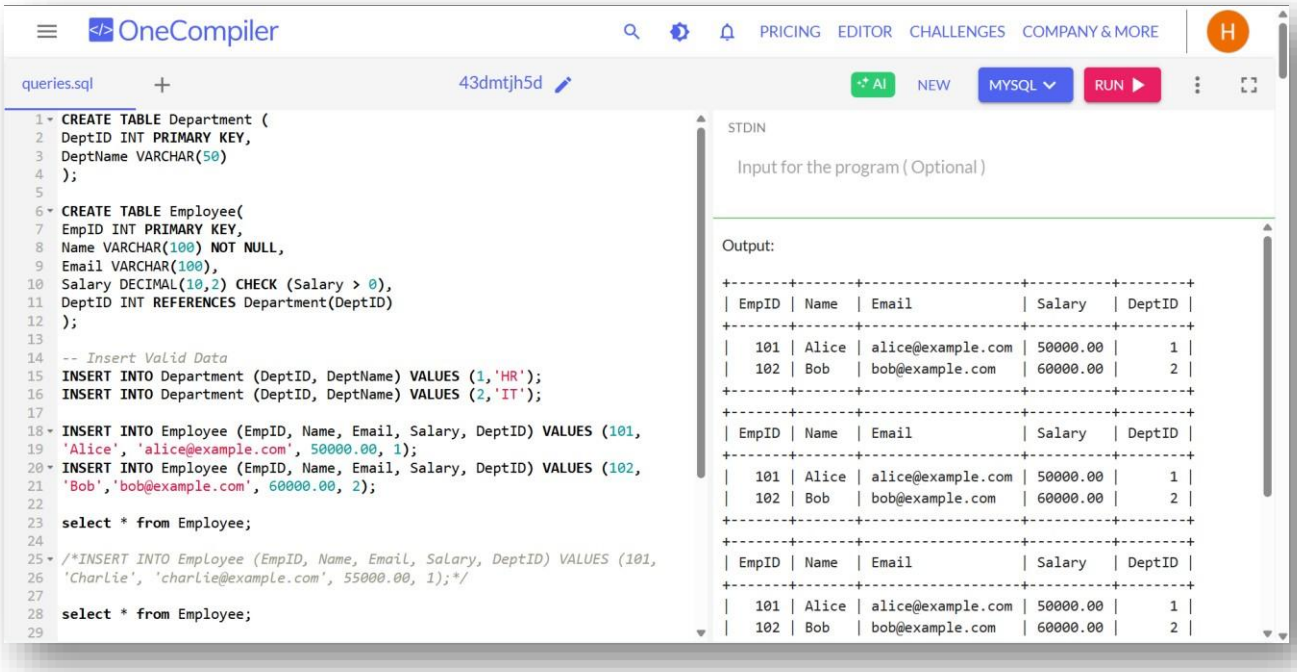
select * from Employee;

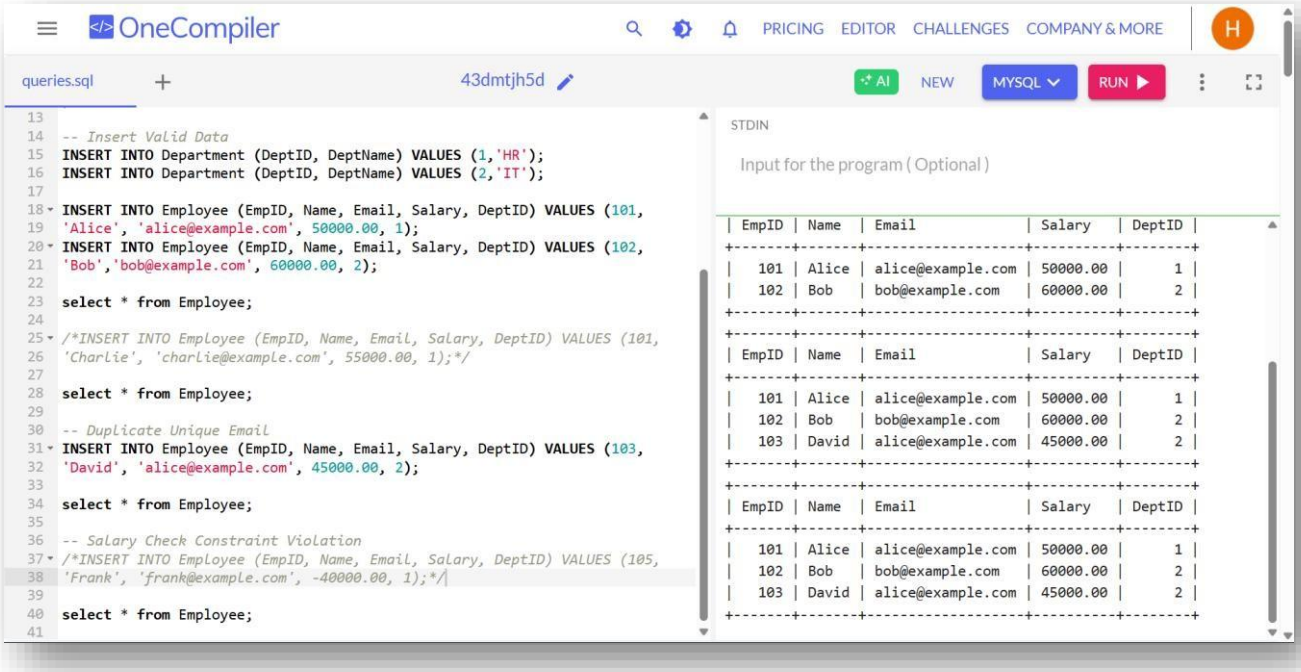
-- Duplicate Unique Email
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (103, 'David', 'alice@example.com', 45000.00, 2);

select * from Employee;

-- Salary Check Constraint Violation
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (105, 'Frank', 'frank@example.com', -40000.00, 1); select
* from Employee;

Screenshot of Output:-





Practical No.:- 5

Testing Employee Constraints

Aim:-

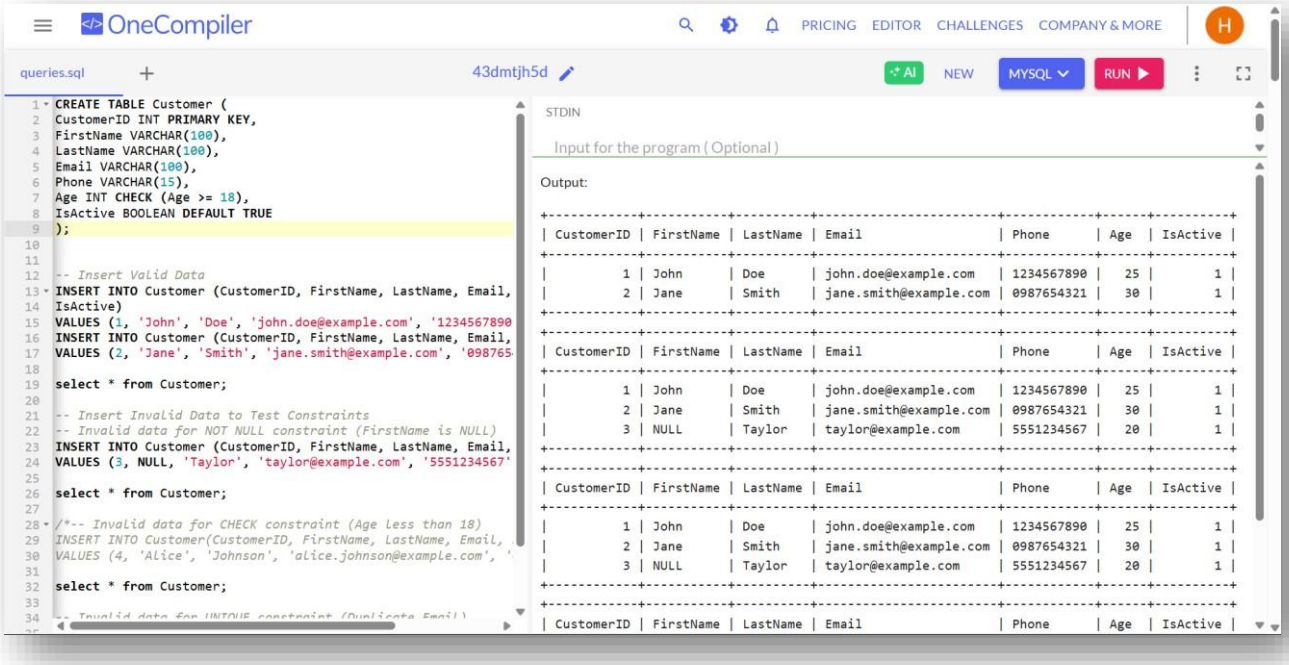
To test constraints like PRIMARY KEY, UNIQUE, and CHECK by inserting invalid data into the Employee table.

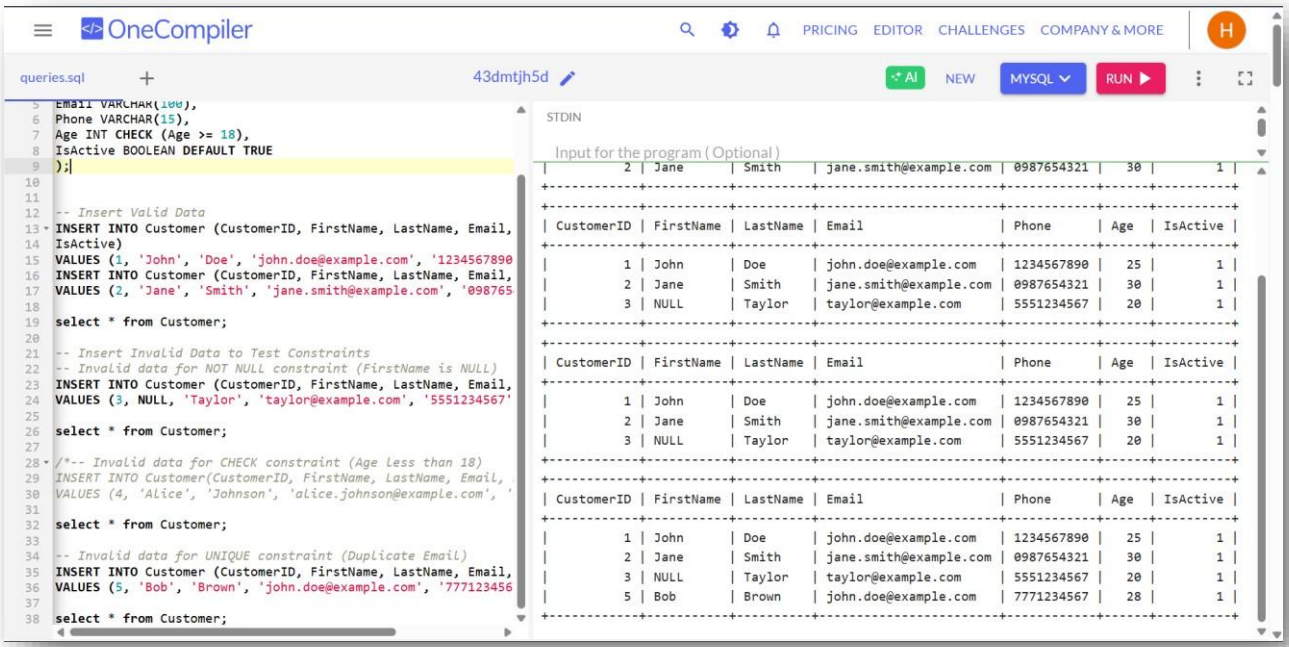
Code:-

```
CREATE TABLE Customer (
CustomerID INT PRIMARY KEY,
FirstName VARCHAR(100),
LastName VARCHAR(100),
Email VARCHAR(100),
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, 'John', 'Doe', 'john.doe@example.com', '1234567890', 25, TRUE);
```

```
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone,
Age)

VALUES (2, 'Jane', 'Smith', 'jane.smith@example.com', '0987654321', 30);


select * from Customer;


-- Insert Invalid Data to Test Constraints

-- Invalid data for NOT NULL constraint (FirstName is NULL)

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone,
Age)

VALUES (3, NULL, 'Taylor', 'taylor@example.com', '5551234567', 20);


select * from Customer;


-- Invalid data for CHECK constraint (Age less than 18)

INSERT INTO Customer(CustomerID, FirstName, LastName, Email, Phone,
Age)

VALUES (4, 'Alice', 'Johnson', 'alice.johnson@example.com', '6669876543', 16);


select * from Customer;


-- Invalid data for UNIQUE constraint (Duplicate Email)

INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone,
Age)

VALUES (5, 'Bob', 'Brown', 'john.doe@example.com', '7771234567', 28);


select * from Customer;
```

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:-

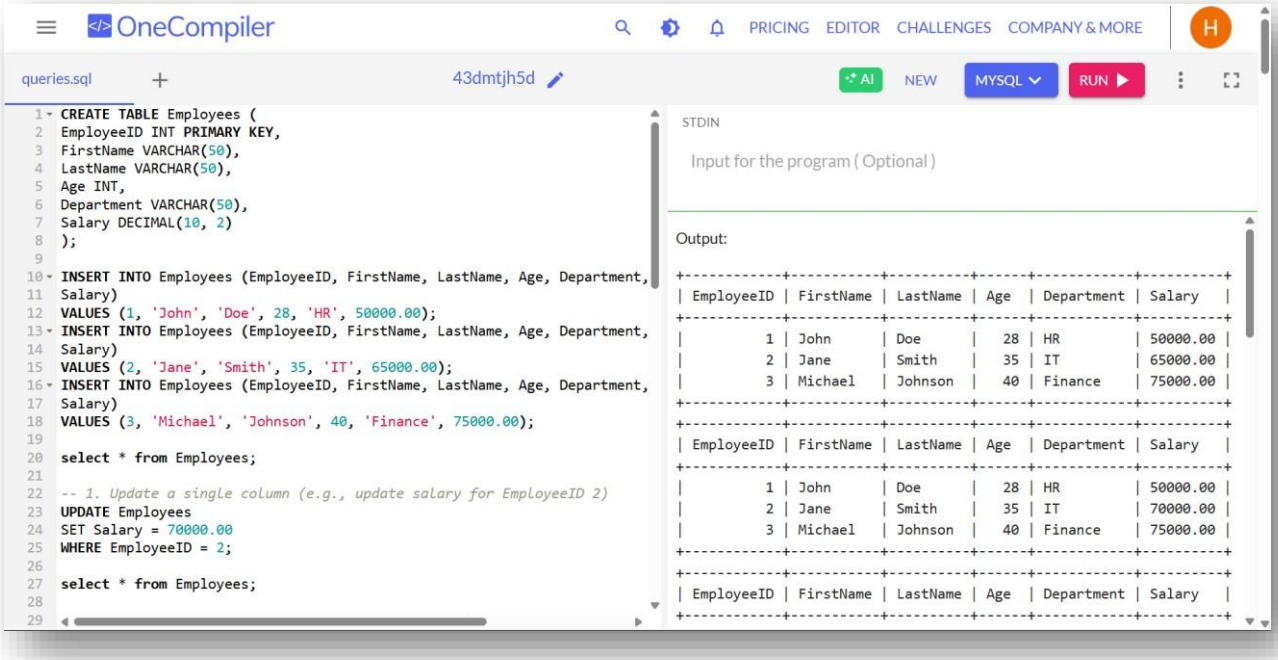
```
CREATE TABLE Employees (
EmployeeID INT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Age INT,
Department VARCHAR(50),
Salary DECIMAL(10, 2)
);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Age,
Department,
Salary)
VALUES (1, 'John', 'Doe', 28, 'HR', 50000.00);
```

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, Age,
Department,

Salary)

VALUES (2, 'Jane', 'Smith', 35, 'IT', 65000.00);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Age,
Department,

Salary)

VALUES (3, 'Michael', 'Johnson', 40, 'Finance', 75000.00);


select * from Employees;


-- 1. Update a single column (e.g., update salary for EmployeeID 2)

UPDATE Employees

SET Salary = 70000.00

WHERE EmployeeID = 2;


select * from Employees;


-- 2. Update multiple columns for a specific row (e.g., update name and salary for
EmployeeID 2)


SET FirstName = 'Janet', LastName = 'Williams', Salary = 75000.00

WHERE EmployeeID = 2;


select * from Employees;


-- 3. Update entire tuple (all columns for EmployeeID 3)
```

```
UPDATE Employees
SET FirstName = 'Michael', LastName = 'Brown', Age = 45, Department =
'Management', Salary = 80000.00
WHERE EmployeeID = 3;

select * from Employees;

-- 4. Update with a condition (e.g., increase salary by 10% for all employees in
HR)
SET Salary = Salary * 1.10
WHERE Department = 'HR';

select * from Employees;

-- 5. Update with a subquery (e.g., increase salary for Employee with highest
salary)
UPDATE Employees
SET Salary = Salary + 5000
WHERE Salary = (SELECT MAX(Salary) FROM Employees);

select * from Employees;

-- 6. Update using a CASE statement (e.g., increase salary based on department)
UPDATE Employees
SET Salary = CASE
WHEN Department = 'HR' THEN Salary * 1.05
WHEN Department = 'IT' THEN Salary * 1.08
```

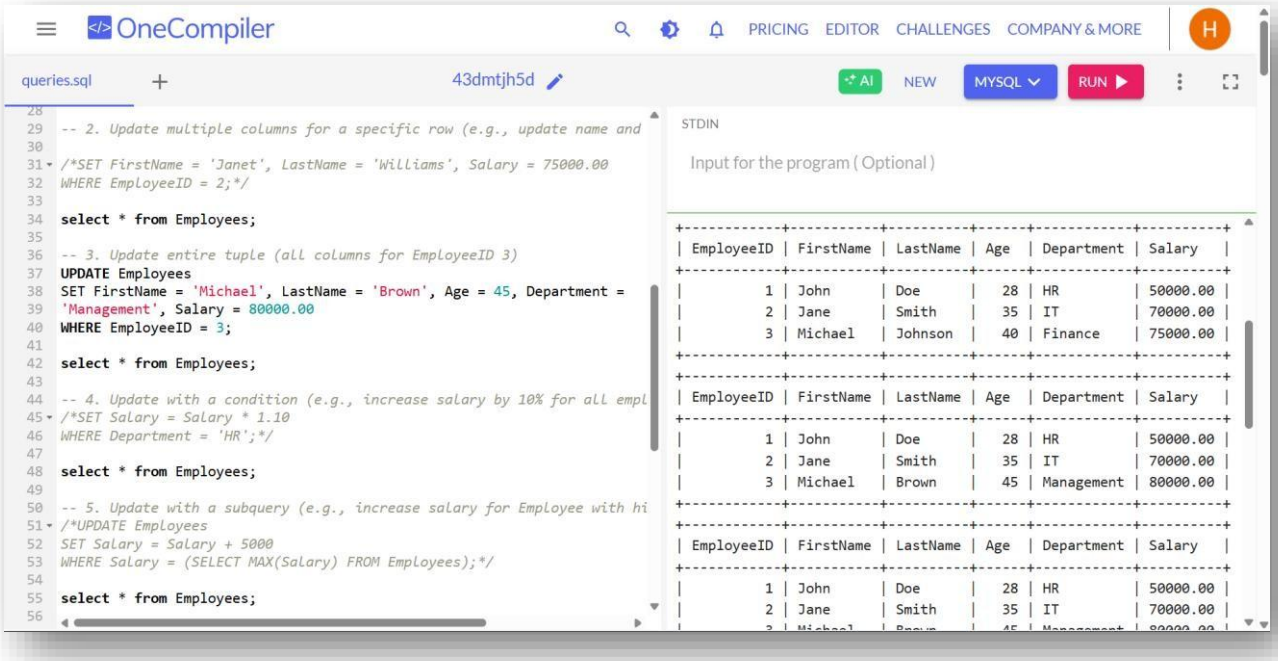
```
WHEN Department = 'Finance' THEN Salary * 1.10
ELSE Salary
END;

select * from Employees;

-- Delete Data from the Table (DML Command)
DELETE FROM Employees
WHERE EmployeeID = 1;

-- Select and Verify Data (SELECT Query)
-- To retrieve all data from the table
SELECT * FROM Employees;

-- To verify the update (checking updated values for EmployeeID 2)
SELECT * FROM Employees

WHERE EmployeeID = 2;
-- To verify the deletion (checking if EmployeeID 1 exists)
SELECT * FROM Employees

WHERE EmployeeID = 1; select
* from Employees;
```

Screenshot of Output:-



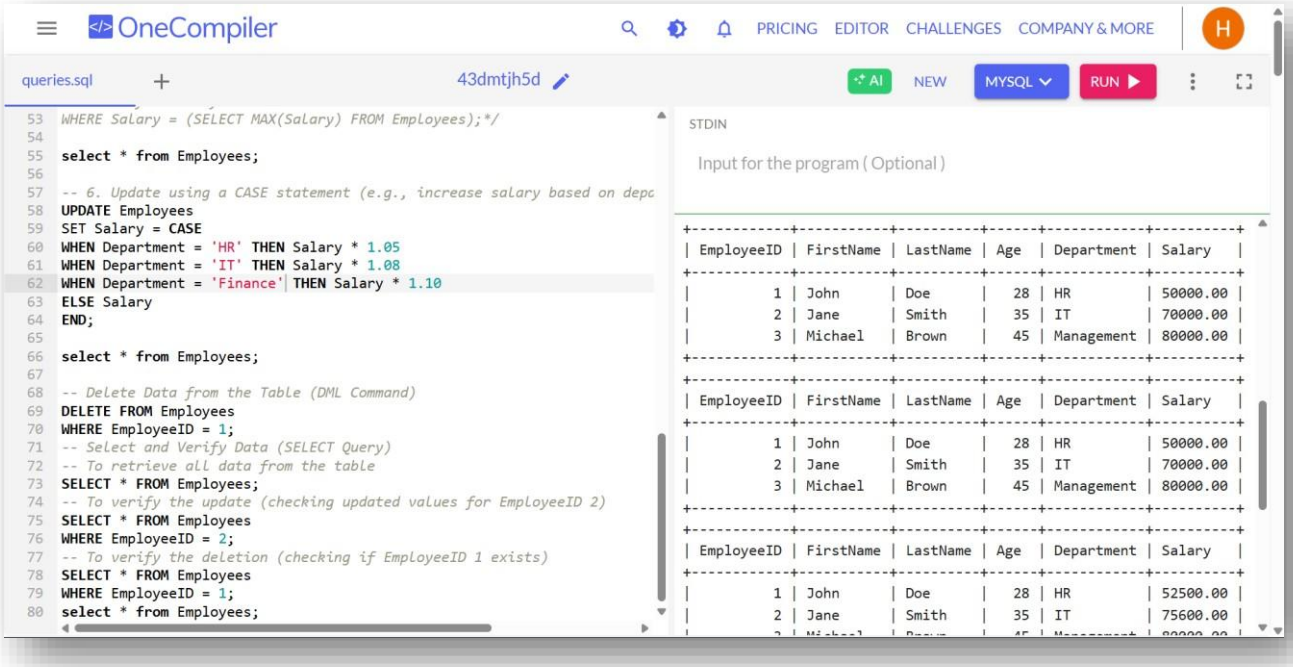

```
CREATE TABLE Employees (
EmployeeID INT PRIMARY KEY,
FirstName VARCHAR(50),
LastName VARCHAR(50),
Age INT,
Department VARCHAR(50),
Salary DECIMAL(10, 2)
);

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

select * from Employees;

-- 1. Update a single column (e.g., update salary for EmployeeID 2)
UPDATE Employees
SET Salary = 70000.00
WHERE EmployeeID = 2;

select * from Employees;
```

```
+------------+-----------+----------+-----+------------+----------+
| EmployeeID | FirstName | LastName | Age | Department | Salary   |
+------------+-----------+----------+-----+------------+----------+
|          1 | John      | Doe      |  28 | HR         | 50000.00 |
|          2 | Jane      | Smith    |  35 | IT         | 65000.00 |
|          3 | Michael   | Johnson  |  40 | Finance    | 75000.00 |
+------------+-----------+----------+-----+------------+----------+
+------------+-----------+----------+-----+------------+----------+
| EmployeeID | FirstName | LastName | Age | Department | Salary   |
+------------+-----------+----------+-----+------------+----------+
|          1 | John      | Doe      |  28 | HR         | 50000.00 |
|          2 | Jane      | Smith    |  35 | IT         | 70000.00 |
|          3 | Michael   | Johnson  |  40 | Finance    | 75000.00 |
+------------+-----------+----------+-----+------------+----------+
+------------+-----------+----------+-----+------------+----------+
| EmployeeID | FirstName | LastName | Age | Department | Salary   |
+------------+-----------+----------+-----+------------+----------+
```



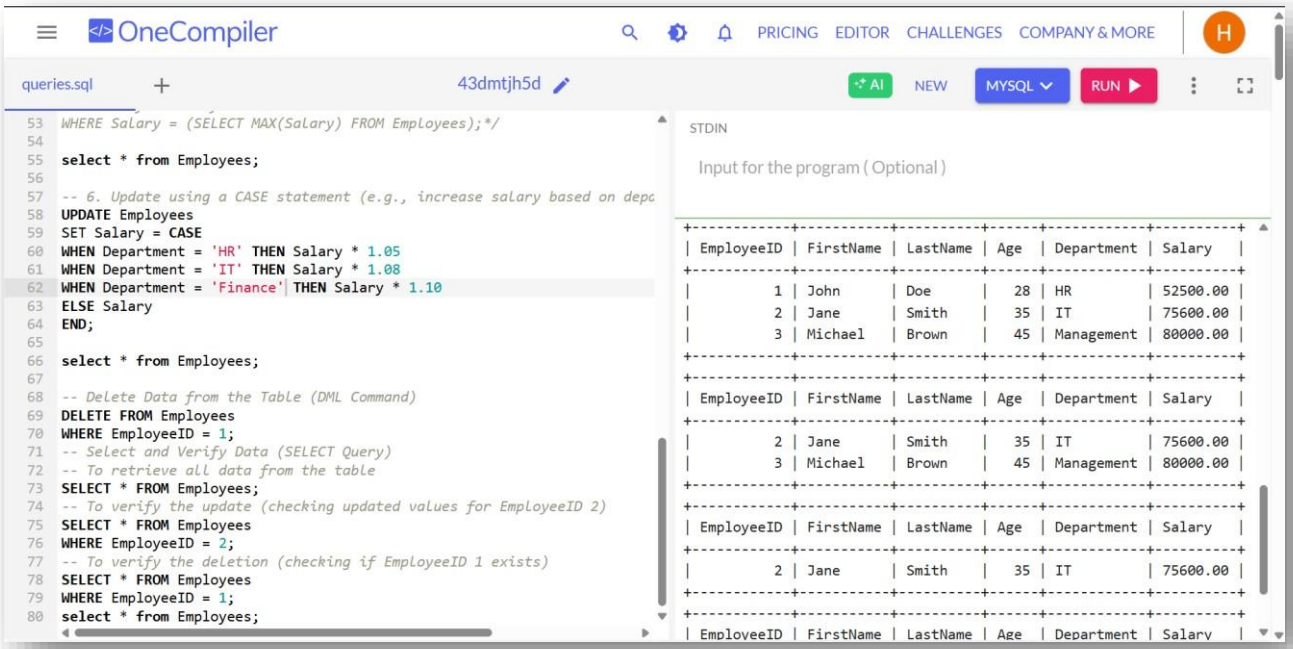

```
-- 2. Update multiple columns for a specific row (e.g., update name and

/*SET FirstName = 'Janet', LastName = 'Williams', Salary = 75000.00
WHERE EmployeeID = 2;*/

select * from Employees;

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

select * from Employees;

-- 4. Update with a condition (e.g., increase salary by 10% for all empl
/*SET Salary = Salary * 1.10
WHERE Department = 'HR';*/

select * from Employees;

-- 5. Update with a subquery (e.g., increase salary for Employee with hi
/*UPDATE Employees
SET Salary = Salary + 5000
WHERE Salary = (SELECT MAX(Salary) FROM Employees);*/

select * from Employees;
```

```
+------------+-----------+----------+-----+------------+----------+
| EmployeeID | FirstName | LastName | Age | Department | Salary   |
+------------+-----------+----------+-----+------------+----------+
|          1 | John      | Doe      |  28 | HR         | 50000.00 |
|          2 | Jane      | Smith    |  35 | IT         | 70000.00 |
|          3 | Michael   | Johnson  |  40 | Finance    | 75000.00 |
+------------+-----------+----------+-----+------------+----------+
+------------+-----------+----------+-----+------------+----------+
| EmployeeID | FirstName | LastName | Age | Department | Salary   |
+------------+-----------+----------+-----+------------+----------+
|          1 | John      | Doe      |  28 | HR         | 50000.00 |
|          2 | Jane      | Smith    |  35 | IT         | 70000.00 |
|          3 | Michael   | Brown    |  45 | Management | 80000.00 |
+------------+-----------+----------+-----+------------+----------+
+------------+-----------+----------+-----+------------+----------+
| EmployeeID | FirstName | LastName | Age | Department | Salary   |
+------------+-----------+----------+-----+------------+----------+
|          1 | John      | Doe      |  28 | HR         | 50000.00 |
|          2 | Jane      | Smith    |  35 | IT         | 70000.00 |
```



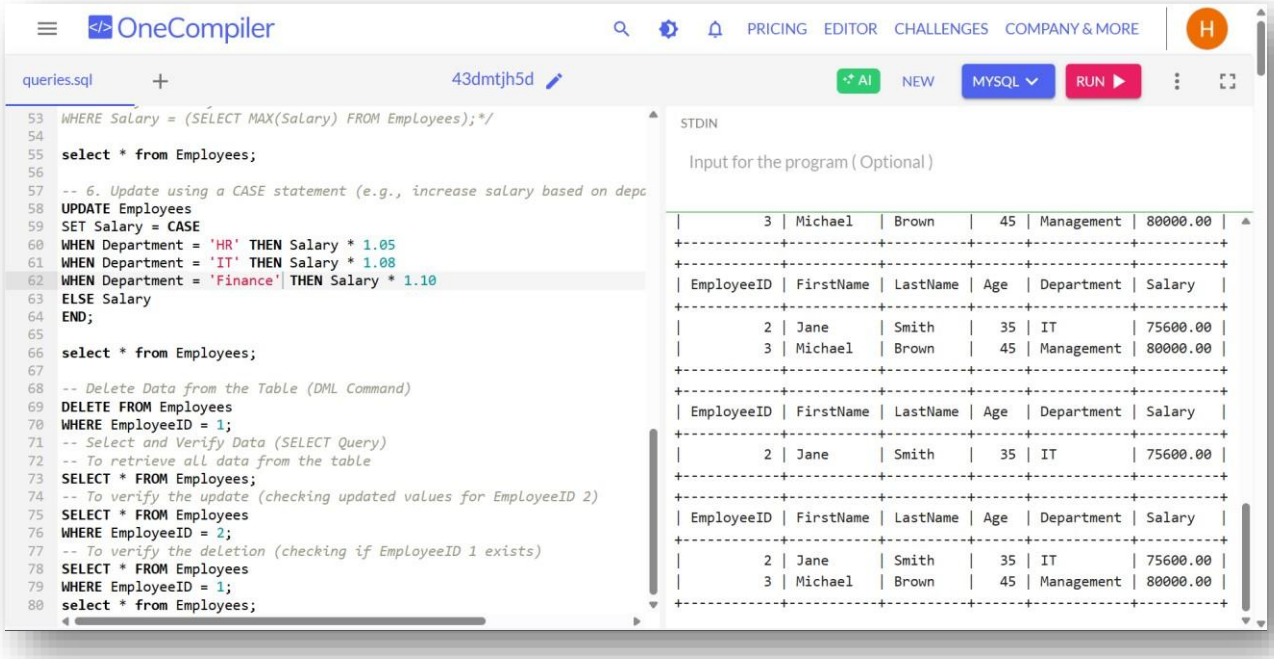

```
53 WHERE Salary = (SELECT MAX(Salary) FROM Employees);*/
54
55 select * from Employees;
56
57 -- 6. Update using a CASE statement (e.g., increase salary based on depa
58 UPDATE Employees
59 SET Salary = CASE
60 WHEN Department = 'HR' THEN Salary * 1.05
61 WHEN Department = 'IT' THEN Salary * 1.08
62 WHEN Department = 'Finance' THEN Salary * 1.10
63 ELSE Salary
64 END;
65
66 select * from Employees;
67
68 -- Delete Data from the Table (DML Command)
69 DELETE FROM Employees
70 WHERE EmployeeID = 1;
71 -- Select and Verify Data (SELECT Query)
72 -- To retrieve all data from the table
73 SELECT * FROM Employees;
74 -- To verify the update (checking updated values for EmployeeID 2)
75 SELECT * FROM Employees
76 WHERE EmployeeID = 2;
77 -- To verify the deletion (checking if EmployeeID 1 exists)
78 SELECT * FROM Employees
79 WHERE EmployeeID = 1;
80 select * from Employees;
```

```
+------------+-----------+----------+------+------------+----------+
| EmployeeID | FirstName | LastName | Age  | Department | Salary   |
+------------+-----------+----------+------+------------+----------+
|          1 | John      | Doe      |   28 | HR         | 50000.00 |
|          2 | Jane      | Smith    |   35 | IT         | 70000.00 |
|          3 | Michael   | Brown    |   45 | Management | 80000.00 |
+------------+-----------+----------+------+------------+----------+
+------------+-----------+----------+------+------------+----------+
| EmployeeID | FirstName | LastName | Age  | Department | Salary   |
+------------+-----------+----------+------+------------+----------+
|          1 | John      | Doe      |   28 | HR         | 50000.00 |
|          2 | Jane      | Smith    |   35 | IT         | 70000.00 |
|          3 | Michael   | Brown    |   45 | Management | 80000.00 |
+------------+-----------+----------+------+------------+----------+
+------------+-----------+----------+------+------------+----------+
| EmployeeID | FirstName | LastName | Age  | Department | Salary   |
+------------+-----------+----------+------+------------+----------+
|          1 | John      | Doe      |   28 | HR         | 52500.00 |
|          2 | Jane      | Smith    |   35 | IT         | 75600.00 |
```




```
53 WHERE Salary = (SELECT MAX(Salary) FROM Employees);*/
54
55 select * from Employees;
56
57 -- 6. Update using a CASE statement (e.g., increase salary based on depa
58 UPDATE Employees
59 SET Salary = CASE
60 WHEN Department = 'HR' THEN Salary * 1.05
61 WHEN Department = 'IT' THEN Salary * 1.08
62 WHEN Department = 'Finance' THEN Salary * 1.10
63 ELSE Salary
64 END;
65
66 select * from Employees;
67
68 -- Delete Data from the Table (DML Command)
69 DELETE FROM Employees
70 WHERE EmployeeID = 1;
71 -- Select and Verify Data (SELECT Query)
72 -- To retrieve all data from the table
73 SELECT * FROM Employees;
74 -- To verify the update (checking updated values for EmployeeID 2)
75 SELECT * FROM Employees
76 WHERE EmployeeID = 2;
77 -- To verify the deletion (checking if EmployeeID 1 exists)
78 SELECT * FROM Employees
79 WHERE EmployeeID = 1;
80 select * from Employees;
```

```
+------------+-----------+----------+------+------------+----------+
| EmployeeID | FirstName | LastName | Age  | Department | Salary   |
+------------+-----------+----------+------+------------+----------+
|          2 | Jane      | Smith    |   35 | IT         | 75600.00 |
|          3 | Michael   | Brown    |   45 | Management | 80000.00 |
+------------+-----------+----------+------+------------+----------+
+------------+-----------+----------+------+------------+----------+
| EmployeeID | FirstName | LastName | Age  | Department | Salary   |
+------------+-----------+----------+------+------------+----------+
|          2 | Jane      | Smith    |   35 | IT         | 75600.00 |
+------------+-----------+----------+------+------------+----------+
+------------+-----------+----------+------+------------+----------+
| EmployeeID | FirstName | LastName | Age  | Department | Salary   |
```


OneCompiler
PRICING
EDITOR
CHALLENGES
COMPANY & MORE
queries.sql
43dmtjh5d
AI
NEW
MYSQL
RUN
STDIN
Input for the program (Optional)
WHERE Salary = (SELECT MAX(Salary) FROM Employees);*/
select * from Employees;
-- 6. Update using a CASE statement (e.g., increase salary based on depc
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;
select * from Employees;
-- Delete Data from the Table (DML Command)
DELETE FROM Employees
WHERE EmployeeID = 1;
-- Select and Verify Data (SELECT Query)
-- To retrieve all data from the table
SELECT * FROM Employees;
-- To verify the update (checking updated values for EmployeeID 2)
SELECT * FROM Employees
WHERE EmployeeID = 2;
-- To verify the deletion (checking if EmployeeID 1 exists)
SELECT * FROM Employees
WHERE EmployeeID = 1;
select * from Employees;
3	Michael	Brown	45	Management	80000.00
EmployeeID	FirstName	LastName	Age	Department	Salary
2	Jane	Smith	35	IT	75600.00
3	Michael	Brown	45	Management	80000.00
EmployeeID	FirstName	LastName	Age	Department	Salary
2	Jane	Smith	35	IT	75600.00
EmployeeID	FirstName	LastName	Age	Department	Salary
2	Jane	Smith	35	IT	75600.00
3	Michael	Brown	45	Management	80000.00

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;
```

```sql
-- 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;
```

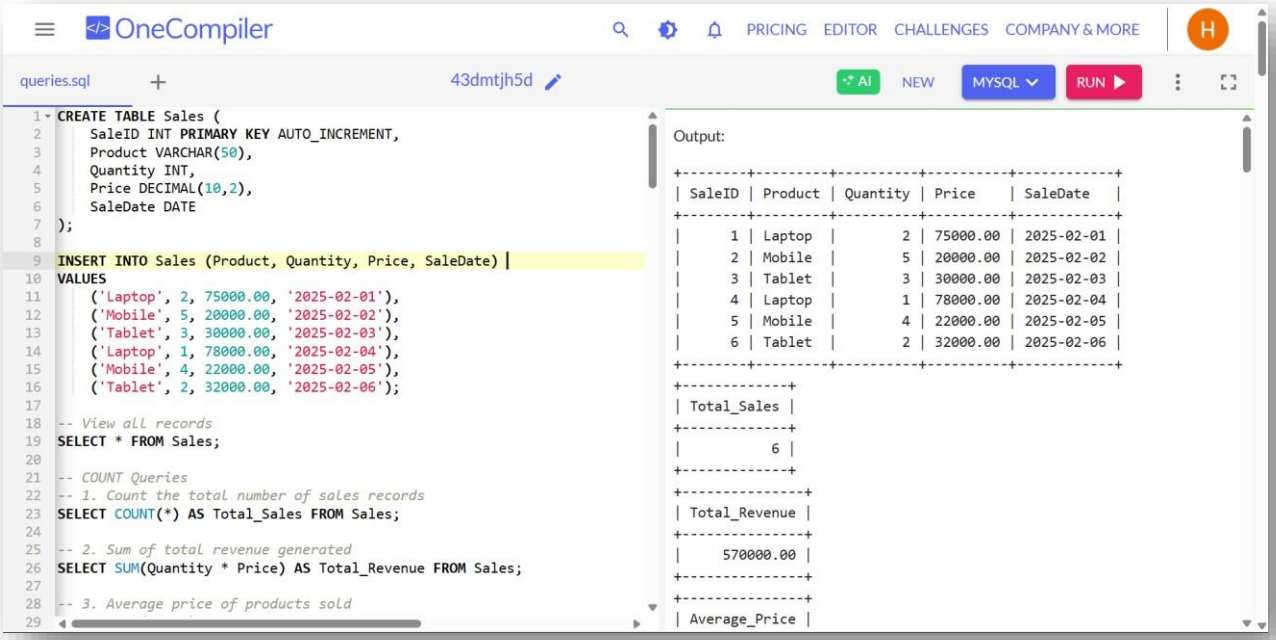
```
-- 3. Average revenue per transaction
SELECT AVG(Quantity * Price) AS Average_Revenue_Per_Transaction
FROM Sales;

-- 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;
```

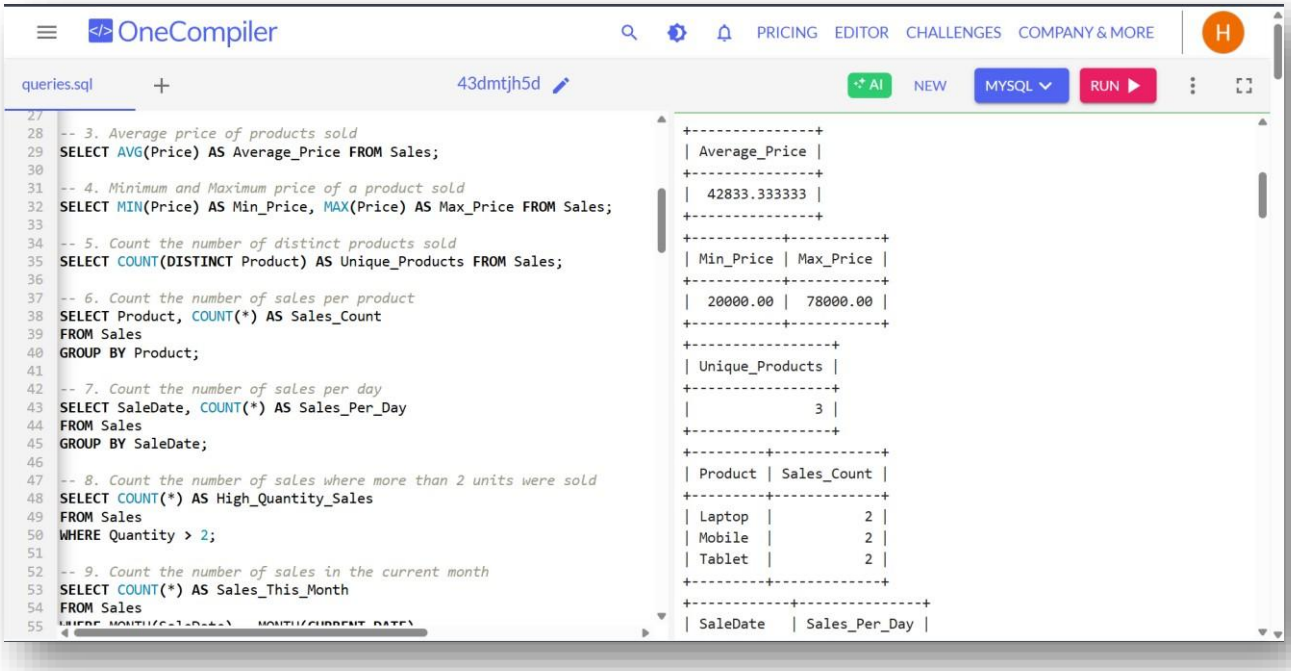
```
-- 8. Average revenue in the current month
SELECT AVG(Quantity * Price) AS Average_Revenue_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

-- 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
```

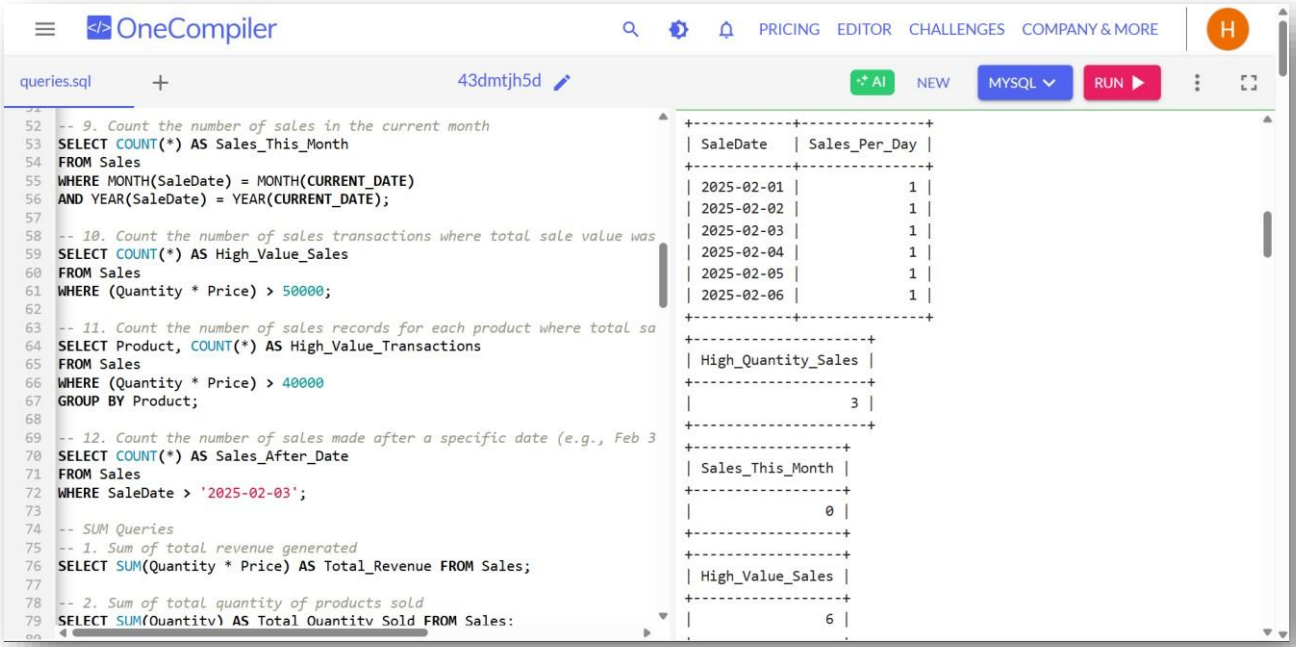
```
SELECT MIN(Quantity * Price) AS Min_Revenue, MAX(Quantity * Price) AS Max_Revenue FROM Sales;


-- 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;
```

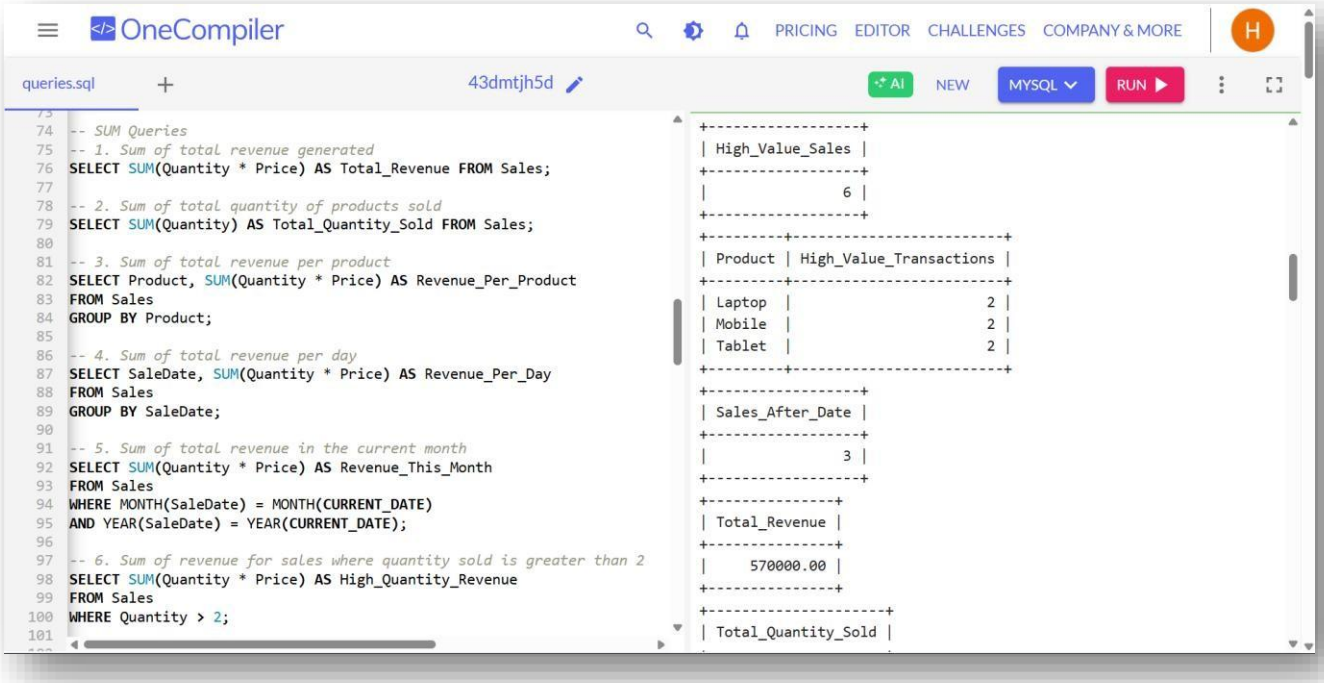
```
-- 8. Minimum and Maximum revenue in the current month
SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity * Price) AS Max_Revenue_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);

-- 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;
```

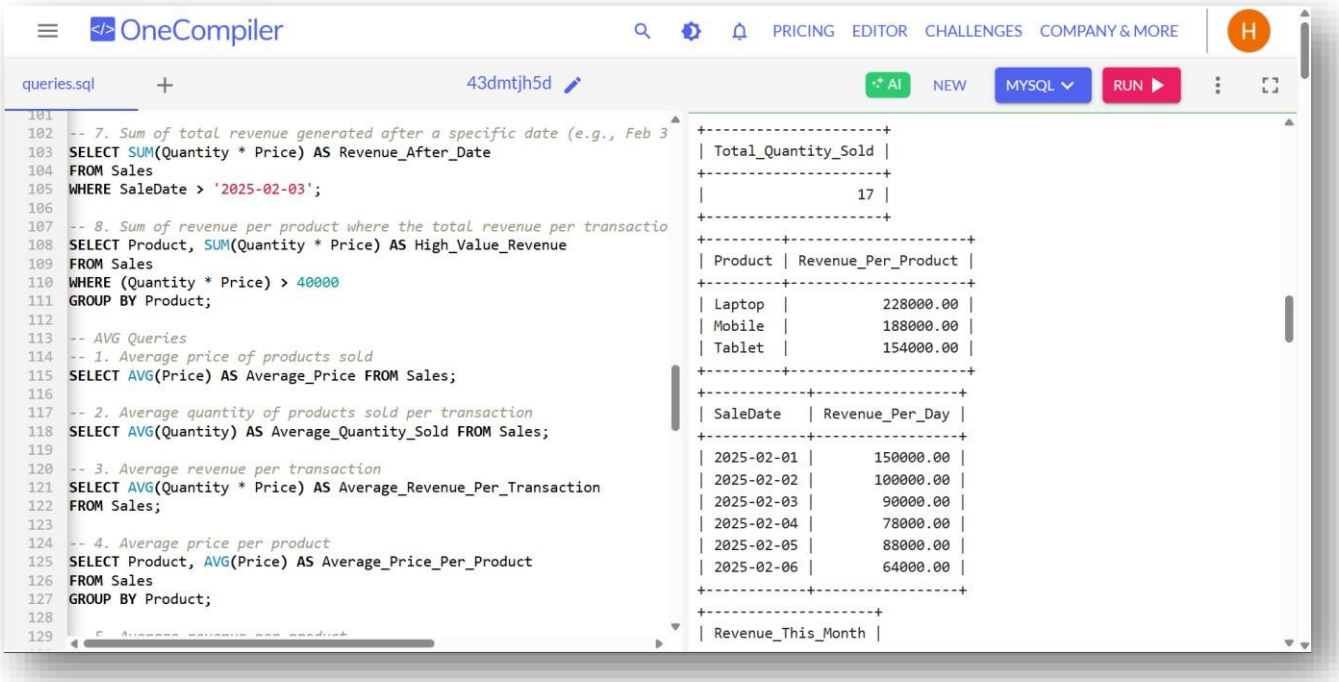
Screenshot of Output:-

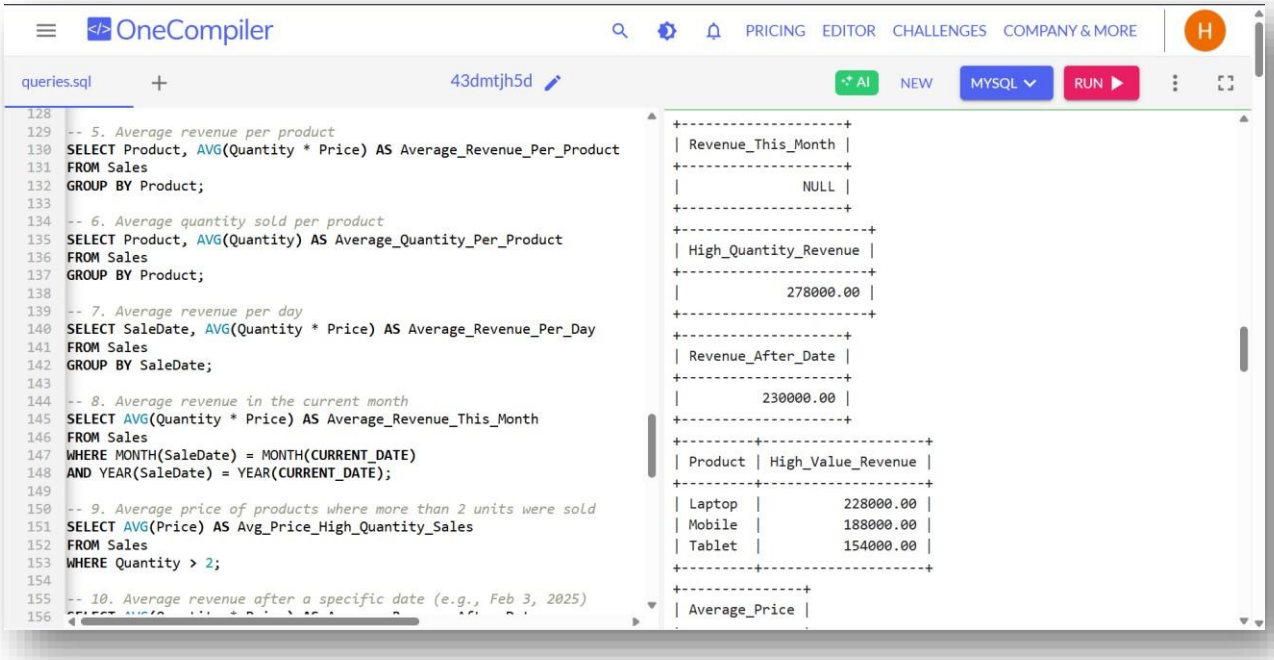


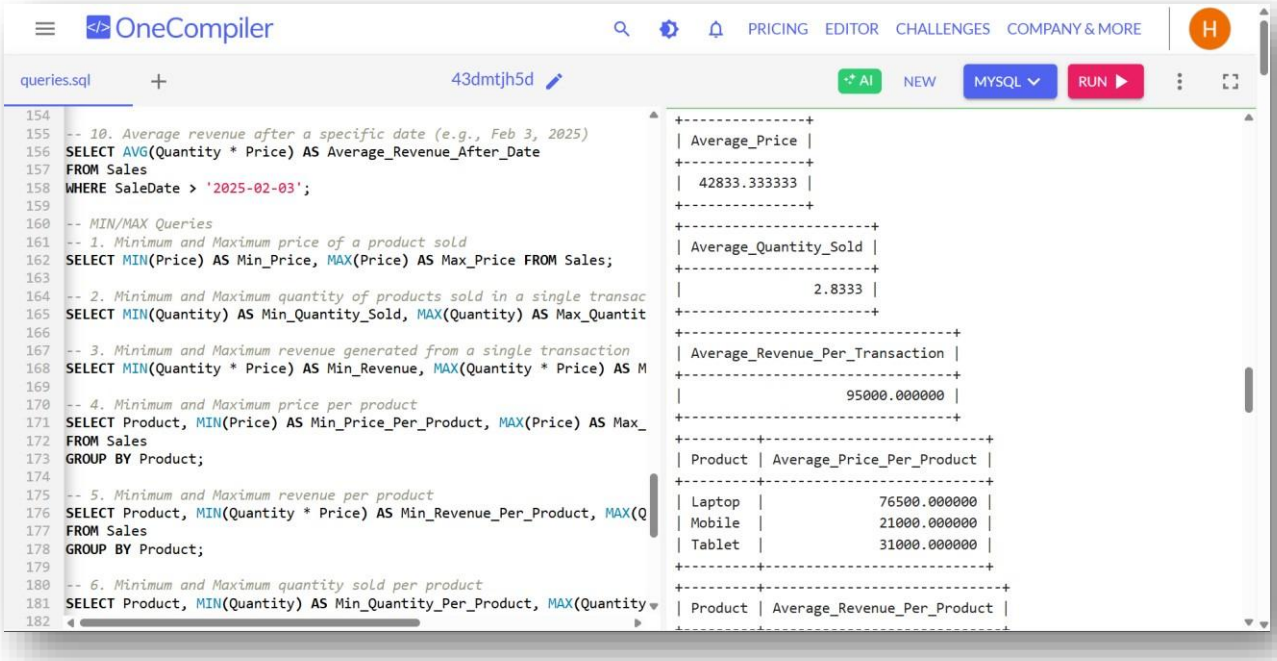

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 3. Average price of products sold
SELECT AVG(Price) AS Average_Price FROM Sales;
-- 4. Minimum and Maximum price of a product sold
SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;
-- 5. Count 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

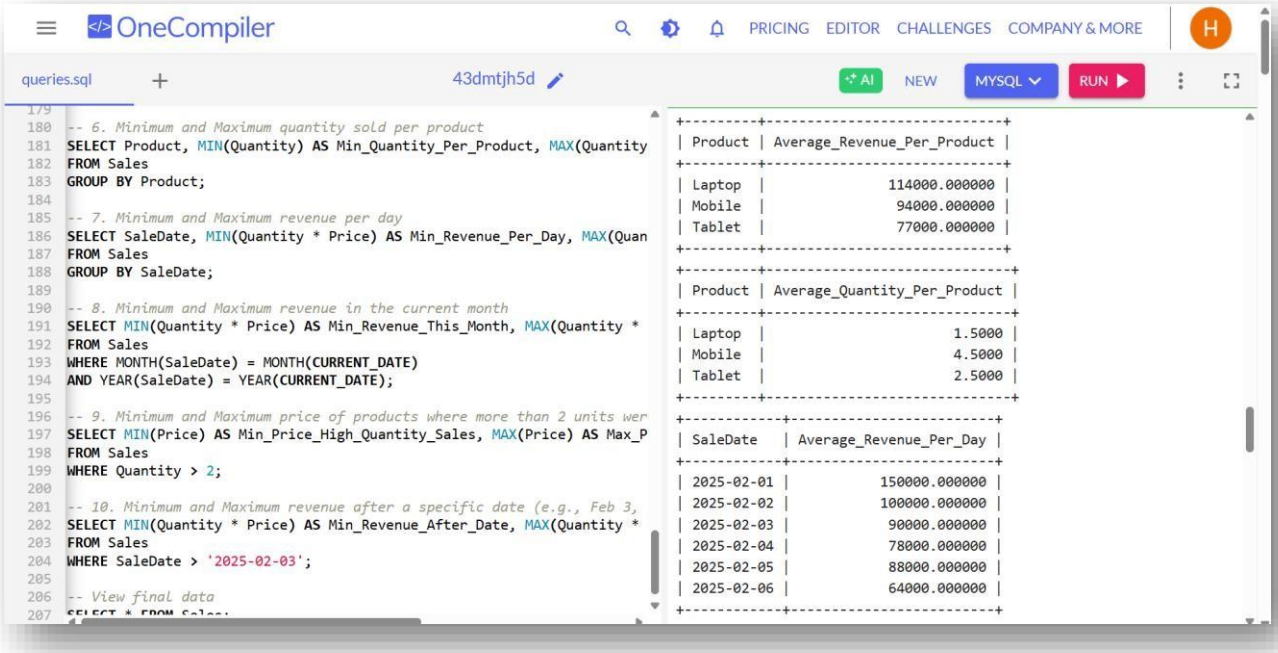

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 9. Count the number of sales in the current month
SELECT COUNT(*) AS Sales_This_Month
FROM Sales
WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
-- 10. Count the number 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

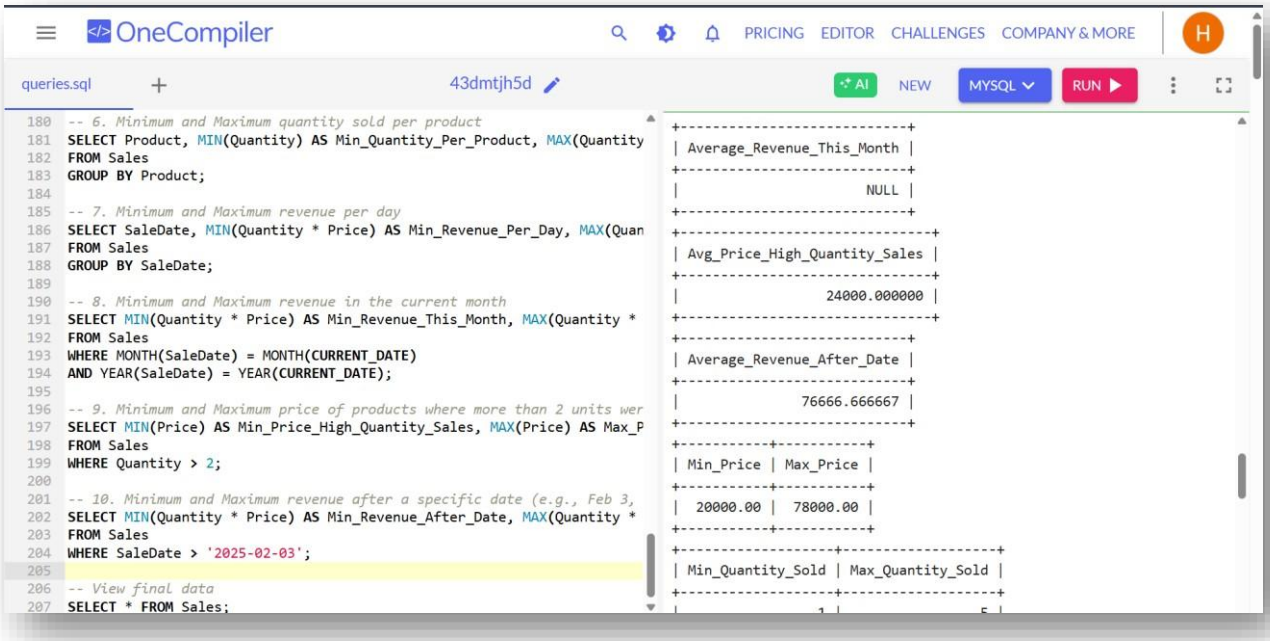

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- SUM Queries
-- 1. Sum of total revenue generated
SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;
-- 2. Sum of total quantity of products sold
SELECT SUM(Quantity) AS Total_Quantity_Sold FROM Sales;
-- 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

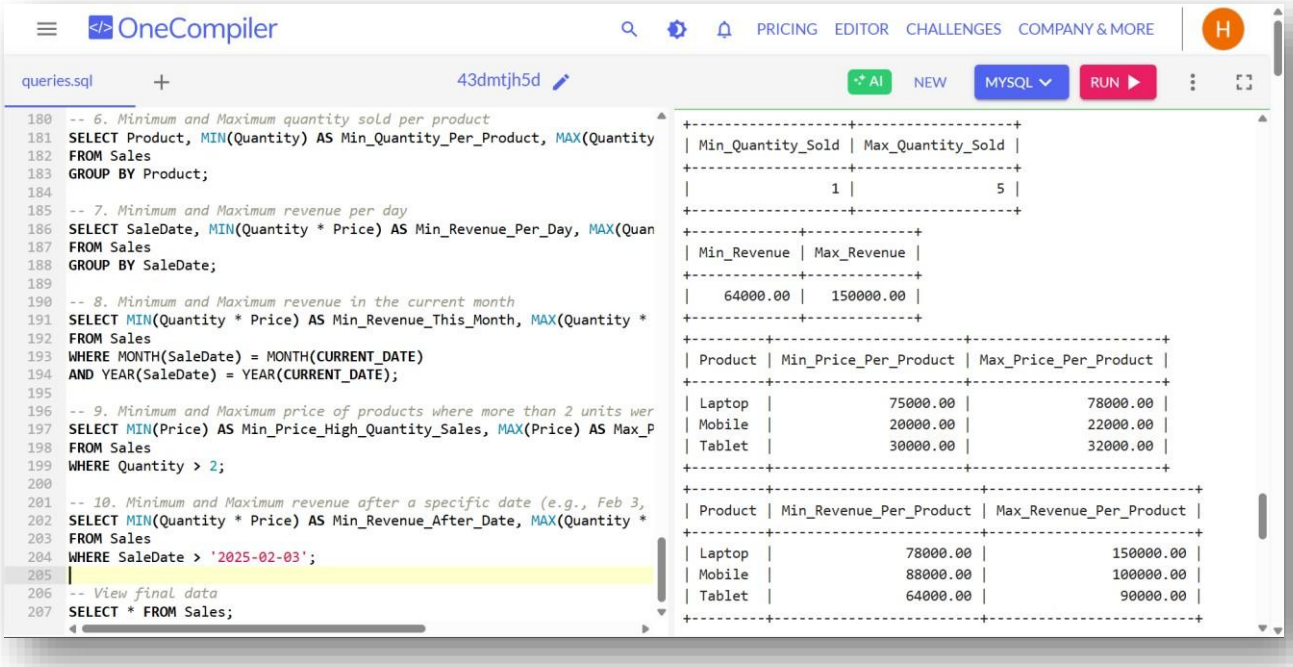

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 7. Sum of total revenue generated after a specific date (e.g., Feb 3
SELECT SUM(Quantity * Price) AS Revenue_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';
-- 8. Sum of revenue per product where the total 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

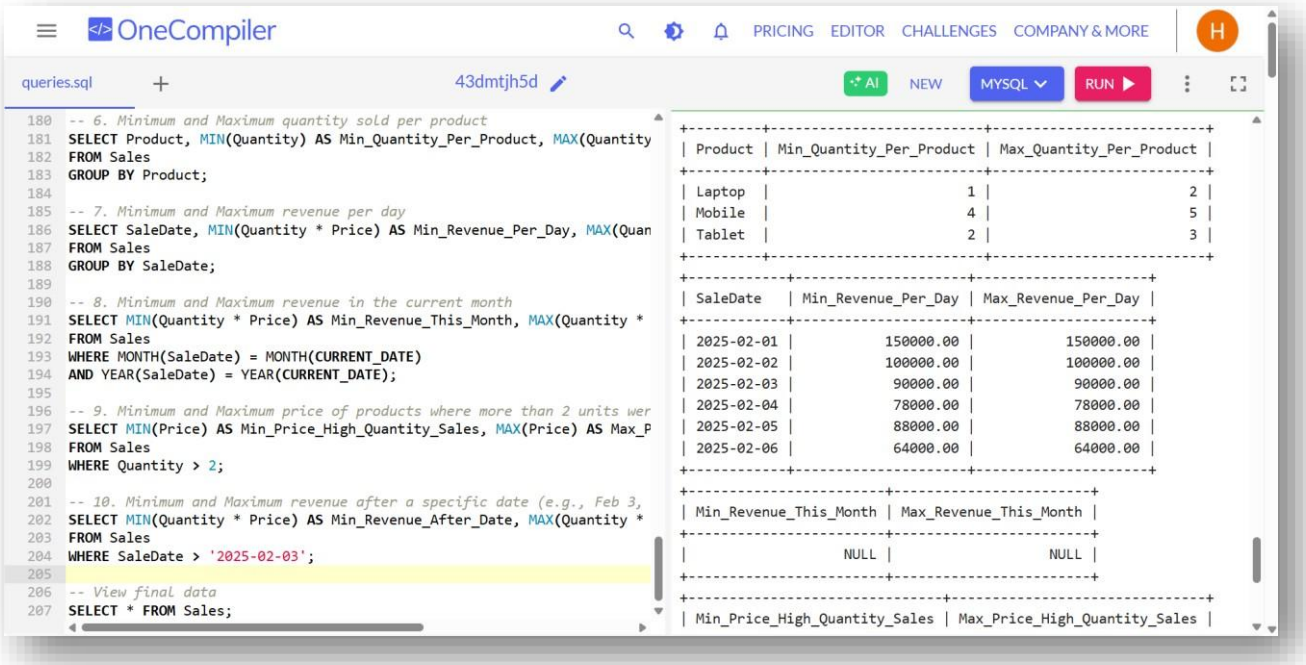

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 5. Average revenue per product
SELECT Product, AVG(Quantity * Price) AS Average_Revenue_Per_Product
FROM Sales
GROUP BY Product;
-- 6. Average quantity sold per product
SELECT Product, AVG(Quantity) AS Average_Quantity_Per_Product
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

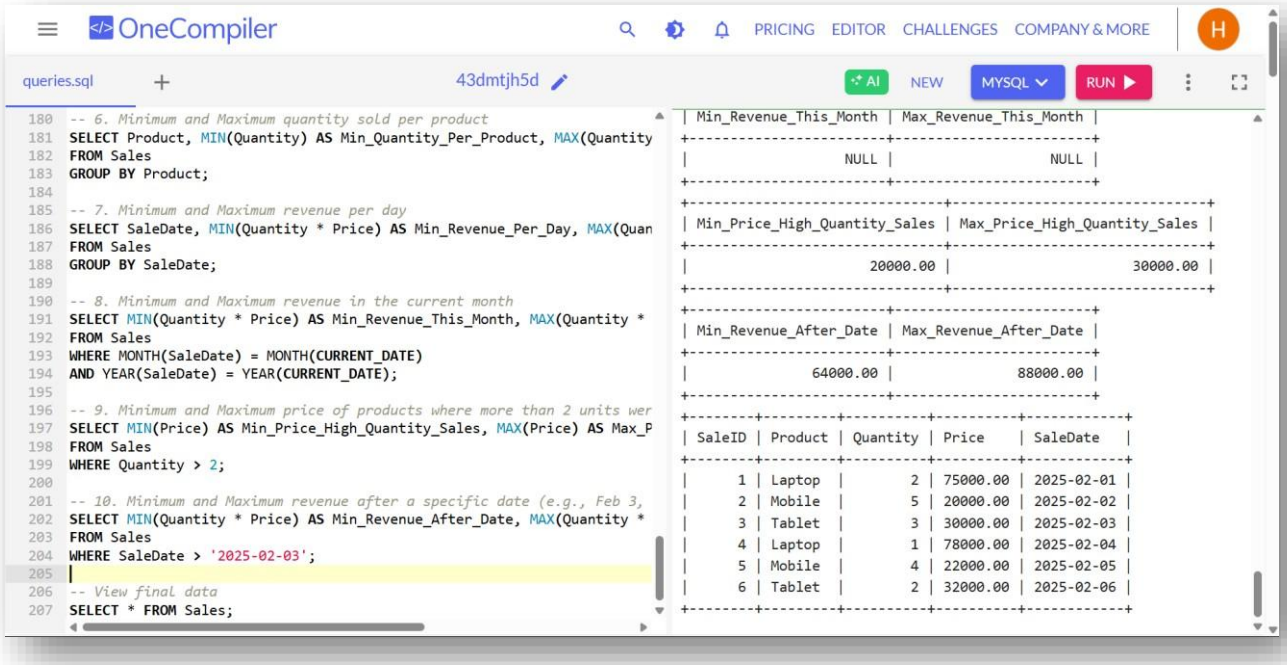

OneCompiler
PRICING EDITOR CHALLENGES COMPANY & MORE
queries.sql
43dmtjh5d
AI NEW MYSQL RUN
-- 10. Average revenue after a specific date (e.g., Feb 3, 2025)
SELECT AVG(Quantity * Price) AS Average_Revenue_After_Date
FROM Sales
WHERE SaleDate > '2025-02-03';
-- MIN/MAX Queries
-- 1. Minimum and Maximum 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.

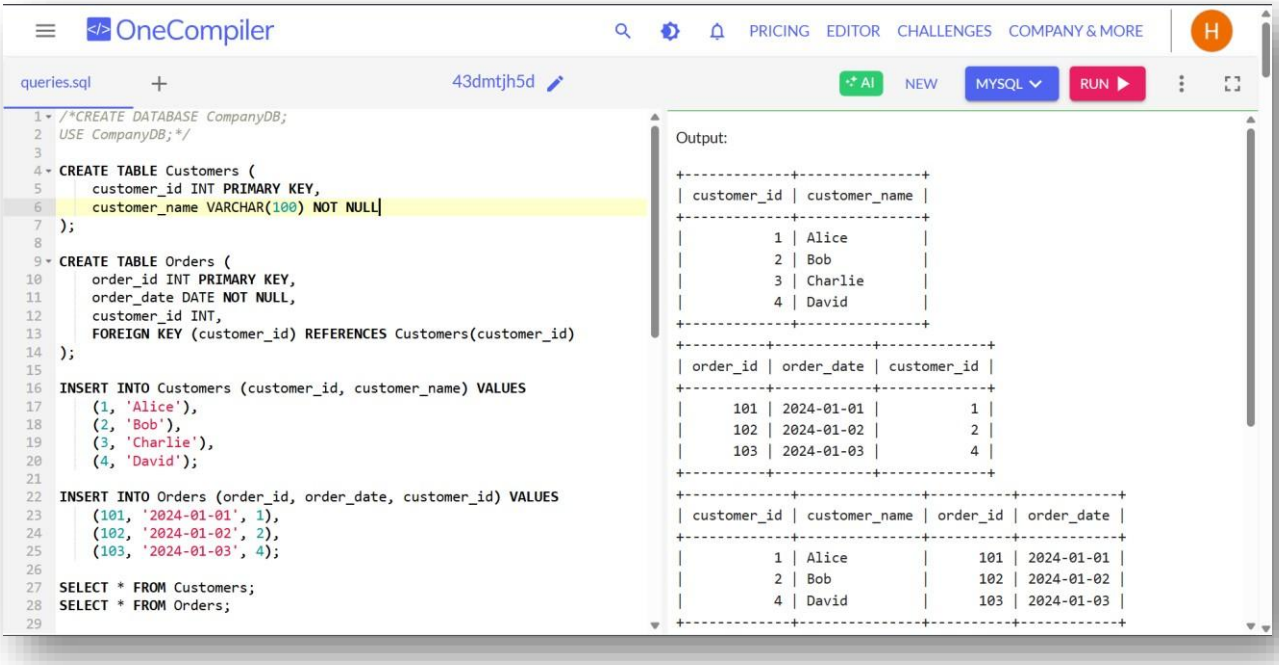
Code:-

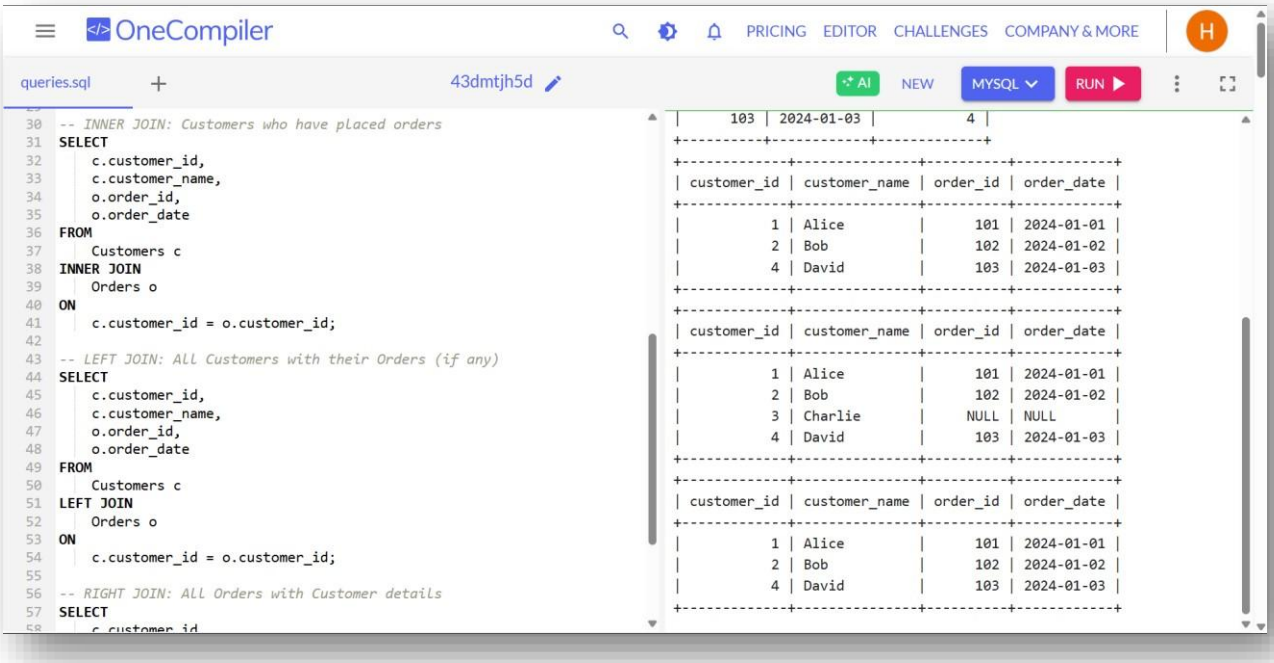
```
CREATE DATABASE CompanyDB;
USE CompanyDB;

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'),
```

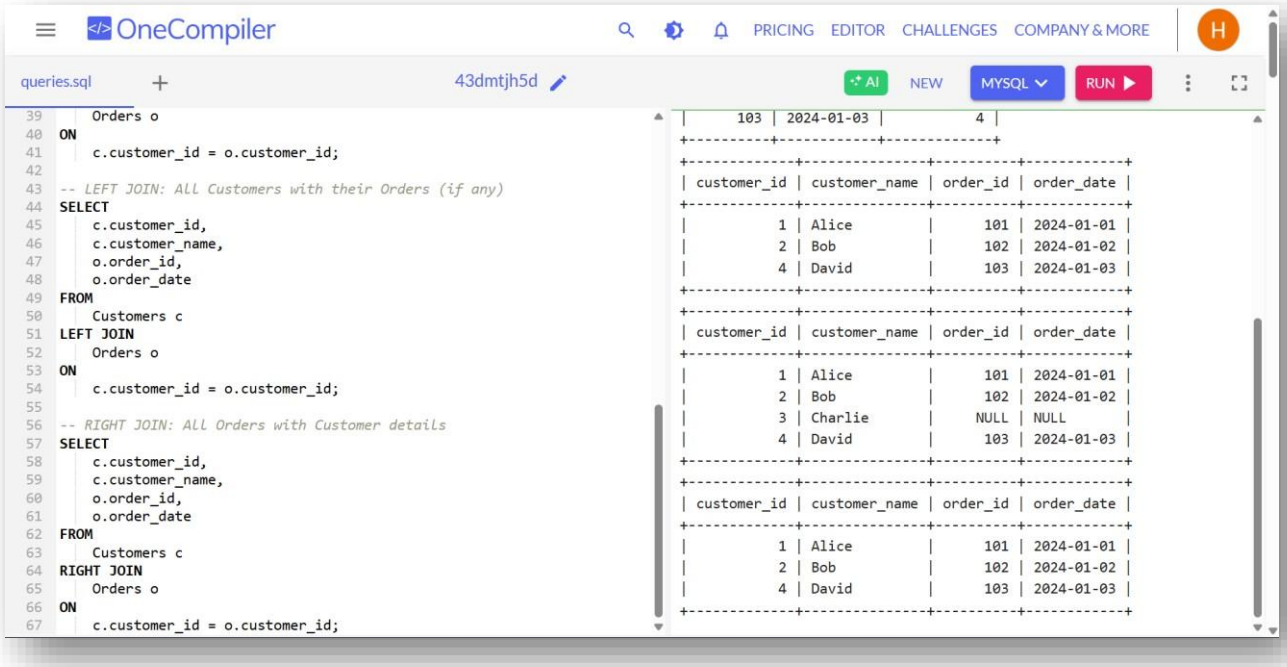
```
    (3, 'Charlie'),
    (4, 'David');

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:-







Nmae: Vishnu Deepak Varma

PRN: 2124UCSM1067

Dept: Cyber security