**“DBMS LAB FILES”**

SUBMITTED BY

**NAME=SAITEJ AHER**

**PRN=2124UCSM1077**

**{*Data Base Management System* }**

Under the Guidance of

**MR. Utkarsh Varshney Sir**

****

**Department of**

**“Computer Science and Engineering”**

**Sanjivani Rural Education Society**

**SANJIVANI UNIVERSITY**

**KOPARGAON – 423603,**

**DIST : AHMEDNAGAR 2024-25**

**INDEX**

| **Sr. No.** | **Practical Title** | **Date** | **Signature** | **Page No.** |
| --- | --- | --- | --- | --- |
| **1** | **Install and Set Up MySQL. Create Database and Table for Employee Details. Perform INSERT & DELETE.** |  |  |  |
| **2** | **Create Student Table. Perform INSERT, UPDATE, DELETE, and SELECT Operations.** |  |  |  |
| **3** | **Create Employee Table with Data Types. Insert Data and Perform Queries.** |  |  |  |
| **4** | **Create Employee Table with Constraints (Primary Key, Foreign Key, Unique). Test with Valid/Invalid Data.** |  |  |  |
| **5** | **Create Customer Table with NOT NULL, CHECK, DEFAULT Constraints. Test with Valid/Invalid Data.** |  |  |  |
| **6** | **Use DDL and DML Commands. Perform Insert, Update, Delete, and Data Retrieval Using SELECT Queries.** |  |  |  |
| **7** | **Create Sales Table. Use Aggregate Functions: COUNT, SUM, AVG, MIN, MAX.** |  |  |  |
| **8** | **Use INNER JOIN, LEFT JOIN, and RIGHT JOIN on Customers and Orders Tables.** |  |  |  |

**Lab File – Practical 1**

**AIM :-** Install and set up MySQL. Create a database and a table to store employee details.

* **CODE FOR INSERTION :-**

CREATE DATABASE PRACTICAL1;

USE PRACTICAL1;

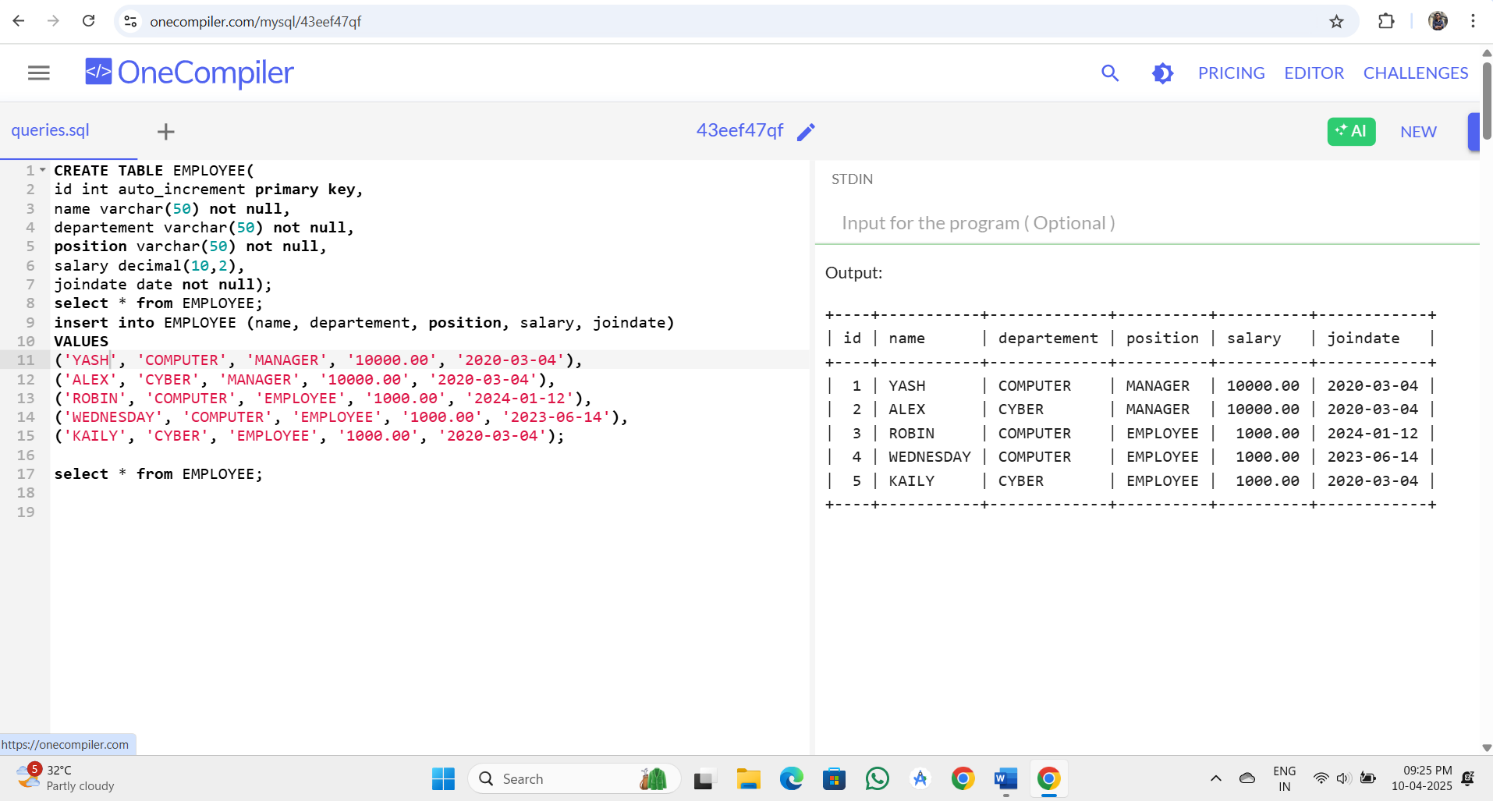
CREATE TABLE EMPLOYEE(  
id int auto\_increment primary key,  
name varchar(50) not null,  
departement varchar(50) not null,  
position varchar(50) not null,  
salary decimal(10,2),  
joindate date not null);

select \* from EMPLOYEE;

insert into EMPLOYEE (name, departement, position, salary, joindate)

VALUES  
('YASH', 'COMPUTER', 'MANAGER', '10000.00', '2020-03-04'),  
('ALEX', 'CYBER', 'MANAGER', '10000.00', '2020-03-04'),  
('ROBIN', 'COMPUTER', 'EMPLOYEE', '1000.00', '2024-01-12'),  
('ROCKEY', 'COMPUTER', 'EMPLOYEE', '1000.00', '2023-06-14'),  
('KAILY', 'CYBER', 'EMPLOYEE', '1000.00', '2020-03-04');

select \* from EMPLOYEE;

* **OUTPUT :-**
* **CODE FOR DELETION :-**

CREATE TABLE EMPLOYEE(

id int auto\_increment primary key,

name varchar(50) not null,

departement varchar(50) not null,

position varchar(50) not null,

salary decimal(10,2),

joindate date not null);

select \* from EMPLOYEE;

insert into EMPLOYEE (name, departement, position, salary, joindate)

VALUES

('YASH', 'COMPUTER', 'MANAGER', '10000.00', '2020-03-04'),

('ALEX', 'CYBER', 'MANAGER', '10000.00', '2020-03-04'),

('ROBIN', 'COMPUTER', 'EMPLOYEE', '1000.00', '2024-01-12'),

('ROCKEY', 'COMPUTER', 'EMPLOYEE', '1000.00', '2023-06-14'),

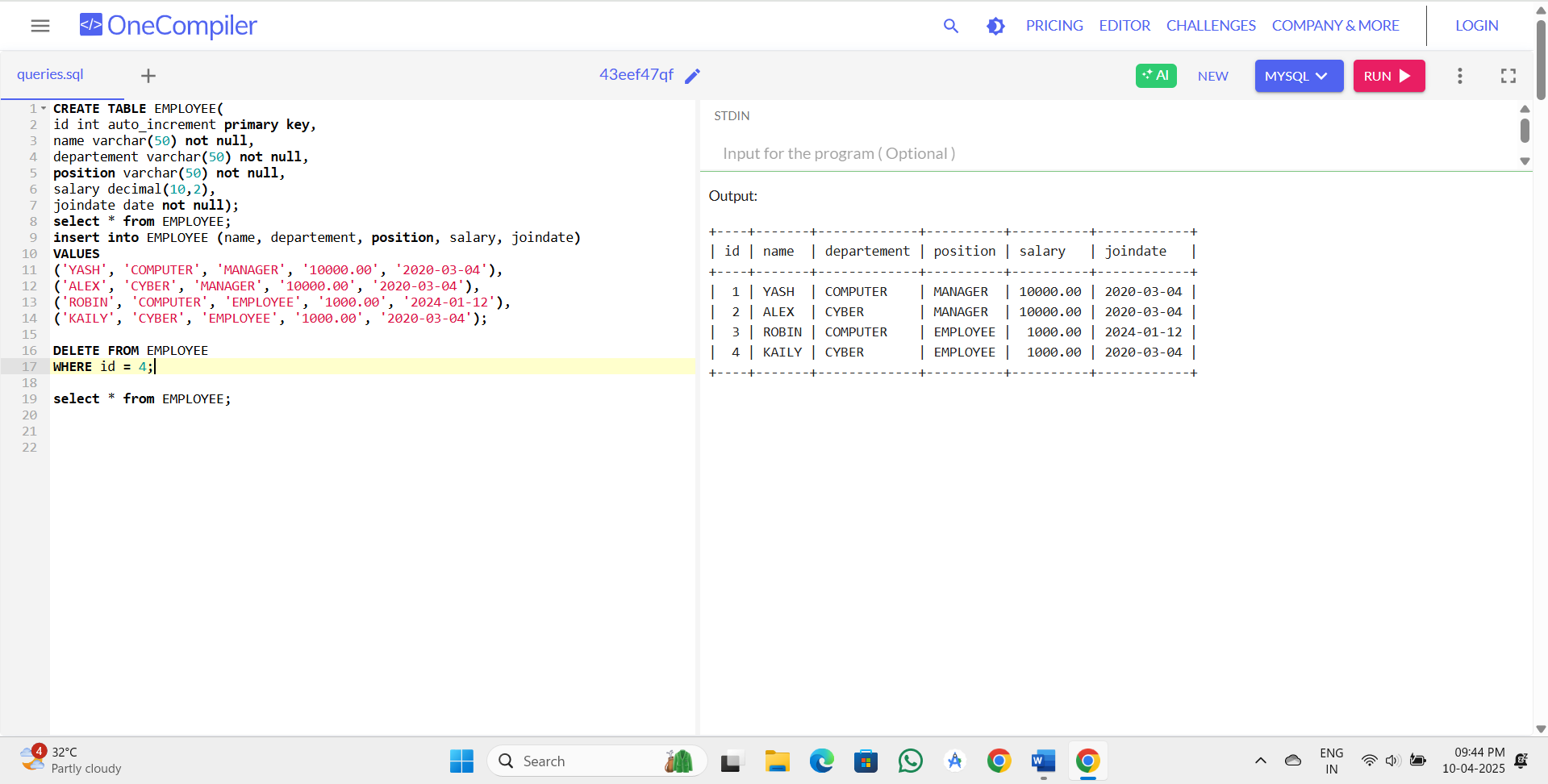
('KAILY', 'CYBER', 'EMPLOYEE', '1000.00', '2020-03-04');

DELETE FROM EMPLOYEE

WHERE id = 4;

select \* from EMPLOYEE;

**OUTPUT :-**

****

**Lab File – Practical 2**

**Aim:** Create a table for storing student information. Insert sample data and perform basic Operations: INSERT, UPDATE, DELETE, and SELECT.

* **CODE FOR TABLE CREATION :-**

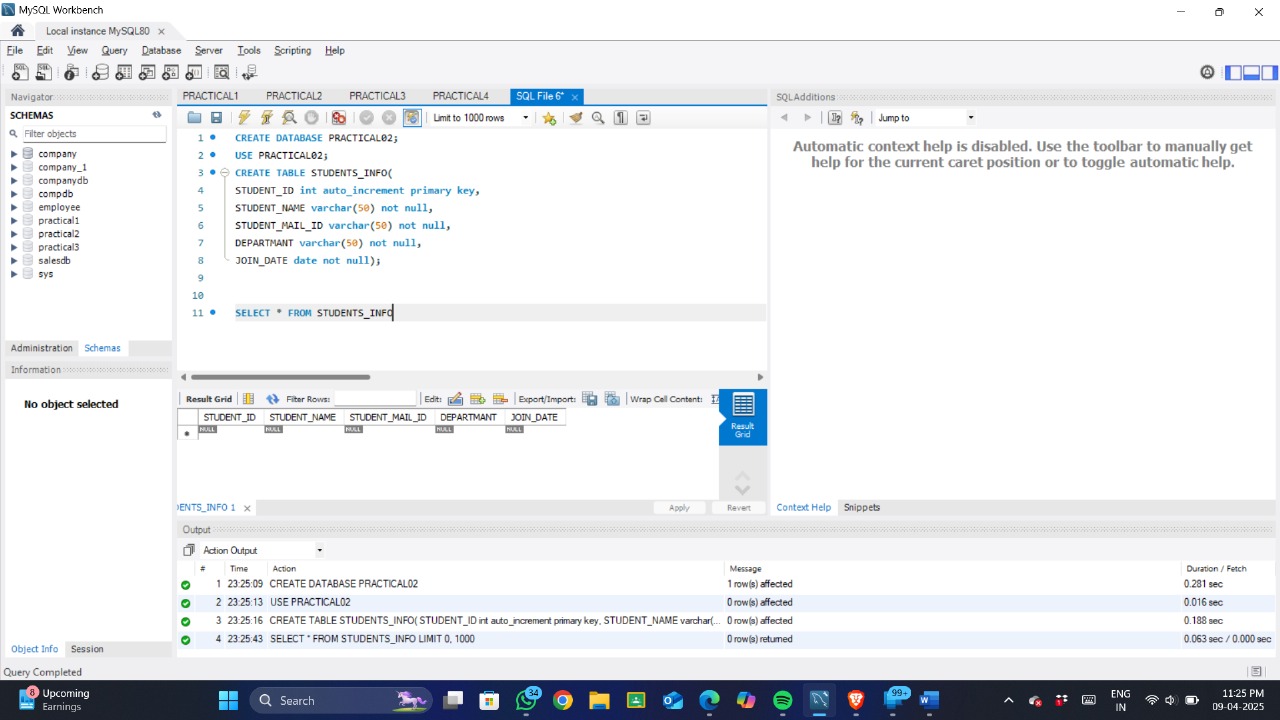
CREATE DATABASE PRACTICAL2;

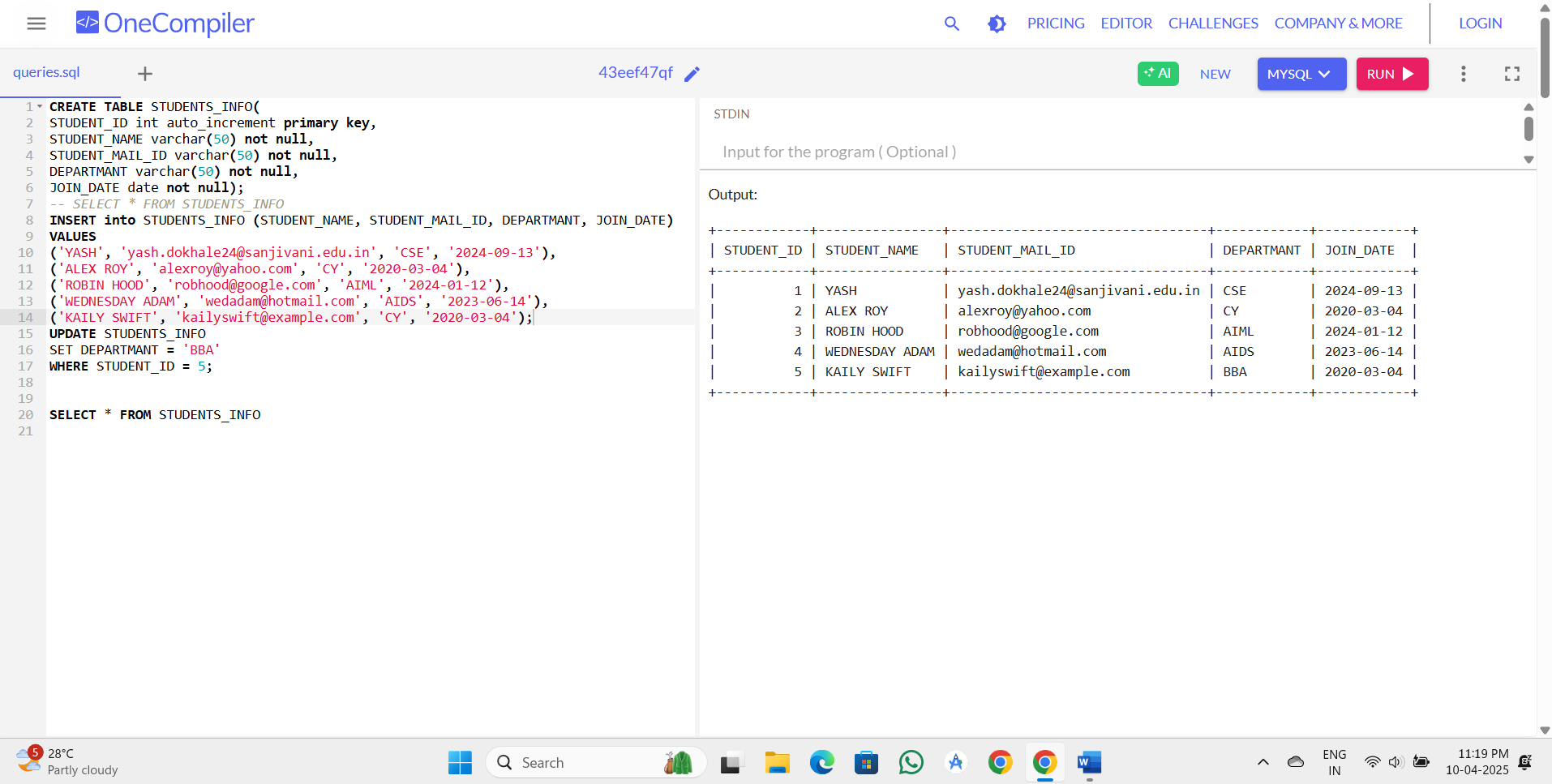
USE PRACTICAL2;

CREATE TABLE STUDENTS\_INFO(  
STUDENT\_ID int auto\_increment primary key,  
STUDENT\_NAME varchar(50) not null,  
STUDENT\_MAIL\_ID varchar(50) not null,  
DEPARTMANT varchar(50) not null,  
JOIN\_DATE date not null);

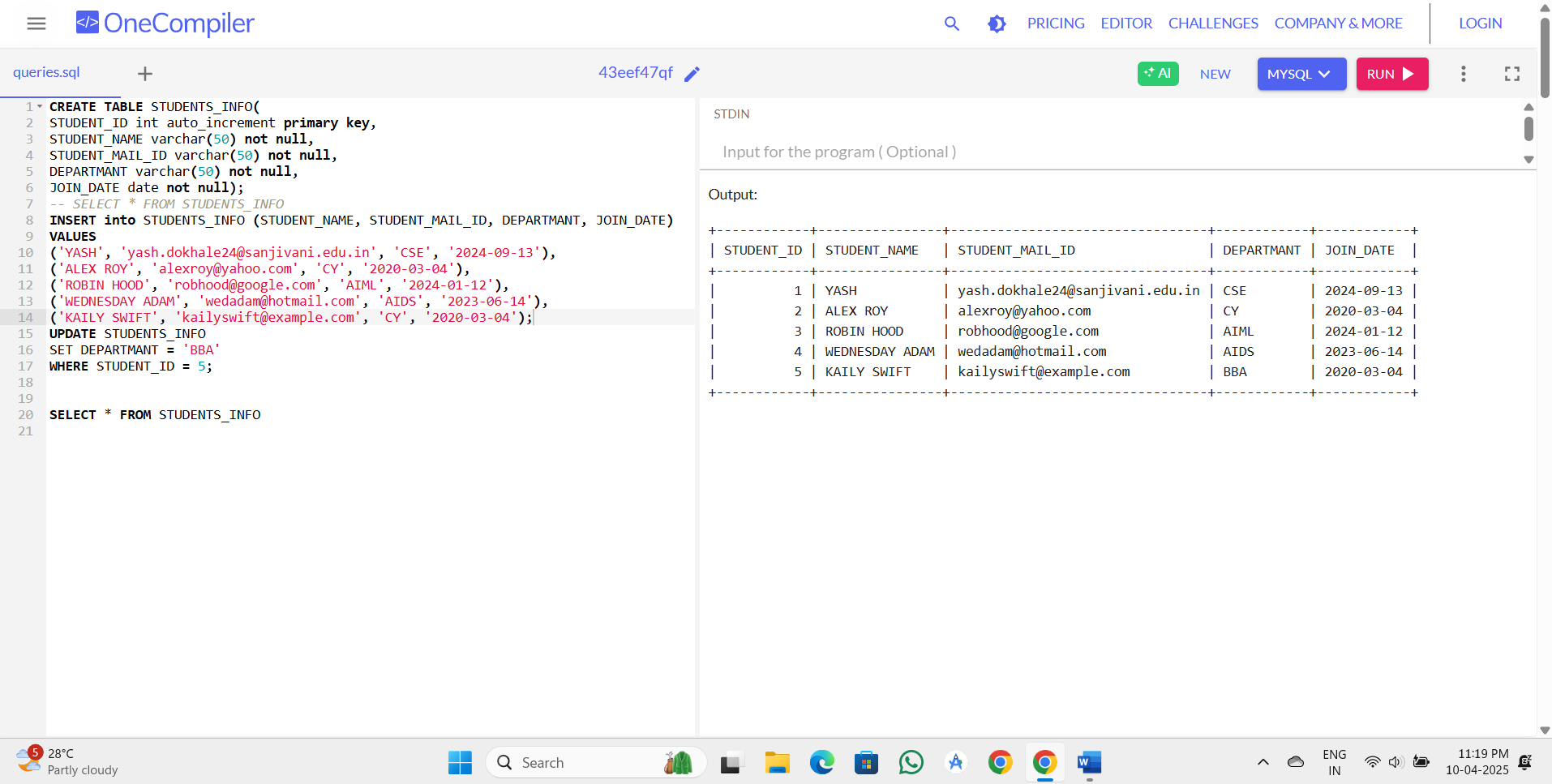
SELECT \* FROM STUDENTS\_INFO

* **OUTPUT :-**

****

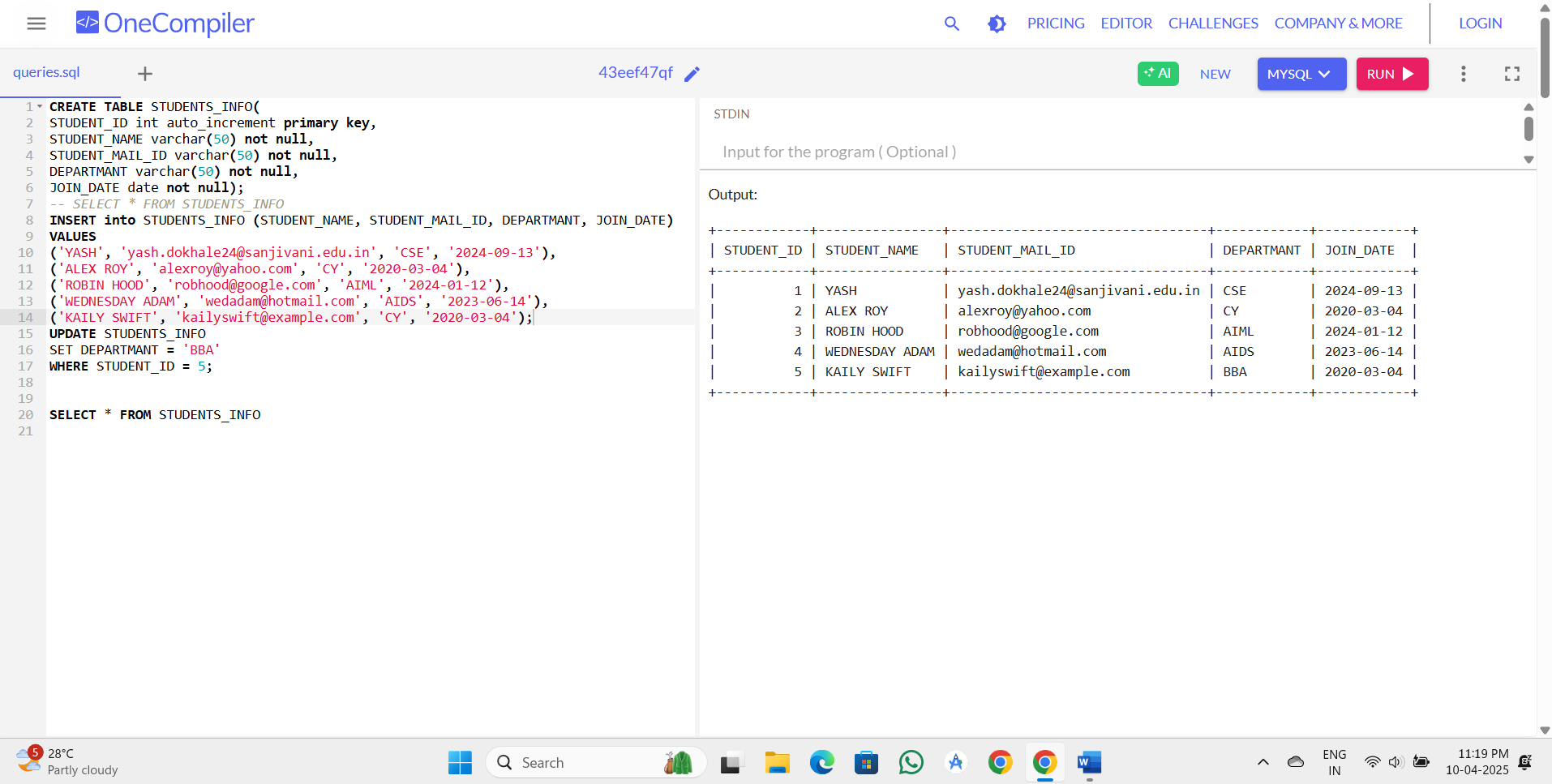
* **CODE FOR SOME SAMPLE DATA :-**
* CREATE TABLE STUDENTS\_INFO(
* STUDENT\_ID int auto\_increment primary key,
* STUDENT\_NAME varchar(50) not null,
* STUDENT\_MAIL\_ID varchar(50) not null,
* DEPARTMANT varchar(50) not null,
* JOIN\_DATE date not null);
* -- SELECT \* FROM STUDENTS\_INFO
* INSERT into STUDENTS\_INFO (STUDENT\_NAME, STUDENT\_MAIL\_ID, DEPARTMANT, JOIN\_DATE)
* VALUES
* ('YASH', 'yash.dokhale24@sanjivani.edu.in', 'CSE', '2024-09-13'),
* ('ALEX ROY', 'alexroy@yahoo.com', 'CY', '2020-03-04'),
* ('ROBIN HOOD', 'robhood@google.com', 'AIML', '2024-01-12'),
* ('WEDNESDAY ADAM', 'wedadam@hotmail.com', 'AIDS', '2023-06-14'),
* ('KAILY SWIFT', 'kailyswift@example.com', 'CY', '2020-03-04');
* **OUTPUT:-**
* **CODE FOR UPDATE THE TABLE :-**
* UPDATE STUDENTS\_INFO
* SET DEPARTMANT = 'BBA'
* WHERE STUDENT\_ID = 5;
* SELECT \* FROM STUDENTS\_INFO

**OUTPUT :-**



* **CODE FOR DELETE THE RECORD FROM THE TABLE :-**

DELETE FROM STUDENTS\_INFO  
WHERE STUDENT\_ID = 3;  
SELECT \* FROM STUDENTS\_INFO

**OUTPUT :-**

**Lab File – Practical 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 FOR CREATION OF TABLE :-**

CREATE DATABASE PRACTICAll3;

USE PRACTICAll3;

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

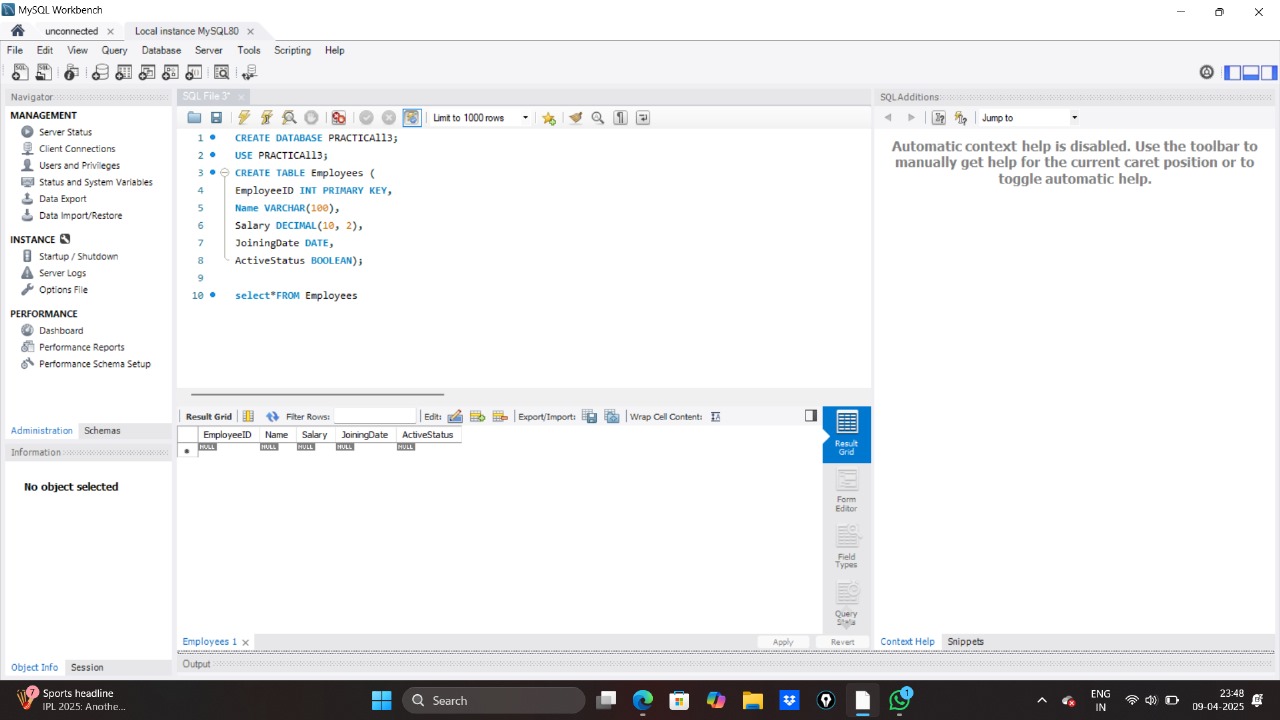
Name VARCHAR(100),

Salary DECIMAL(10, 2),

JoiningDate DATE,

ActiveStatus BOOLEAN);

select\*FROM Employees

* **OUTPUT :-**
* **CODE FOR INSERTION OF SAMPLE DATA :-**

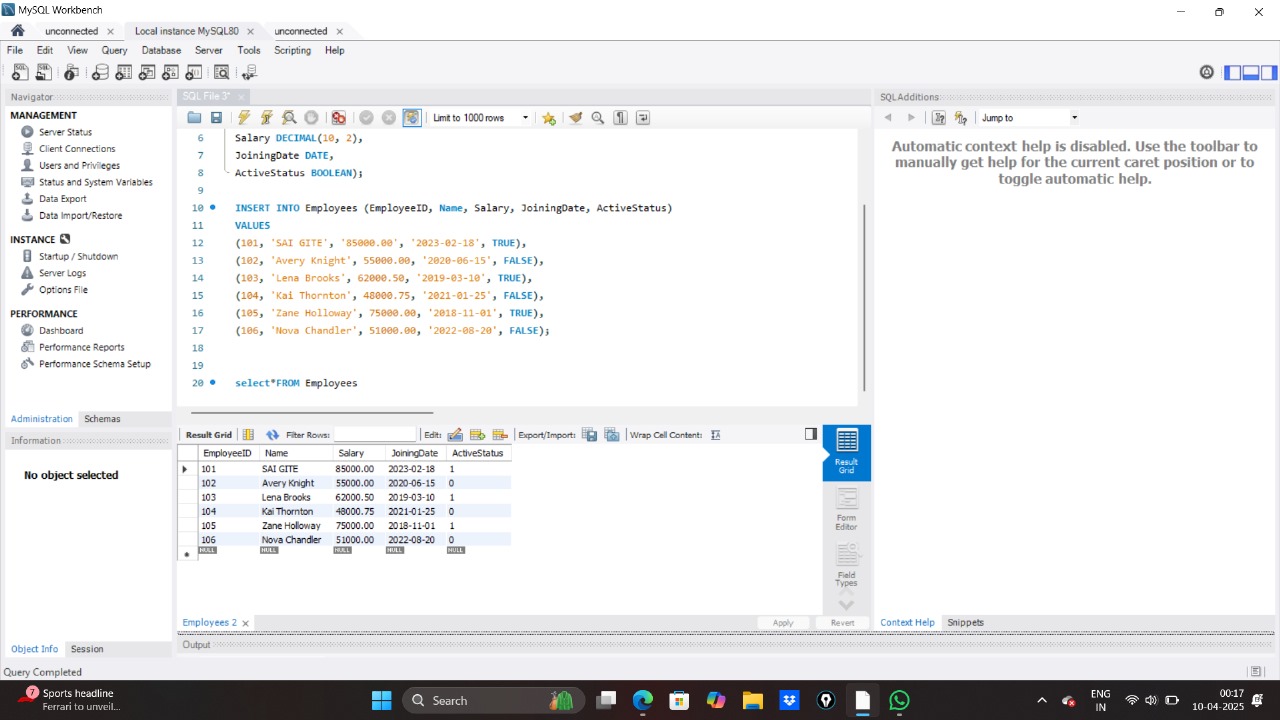
INSERT INTO Employees (EmployeeID, Name, Salary, JoiningDate, ActiveStatus)

VALUES

(101, 'YASH DOKHALE', '85000.00', '2023-02-18', TRUE),  
(102, 'Avery Knight', 55000.00, '2020-06-15', FALSE),  
(103, 'Lena Brooks', 62000.50, '2019-03-10', TRUE),  
(104, 'Kai Thornton', 48000.75, '2021-01-25', FALSE),  
(105, 'Zane Holloway', 75000.00, '2018-11-01', TRUE),  
(106, 'Nova Chandler', 51000.00, '2022-08-20', FALSE);

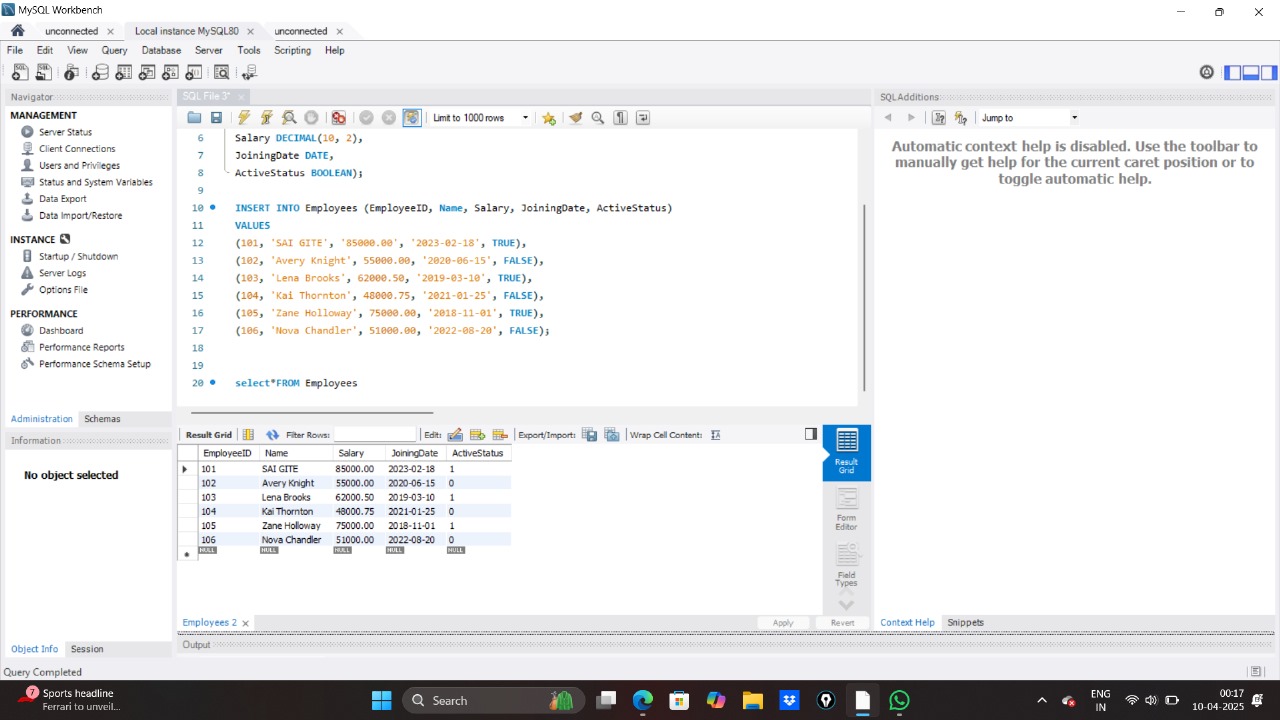
SELECT \* FROM Employees

* **OUTPUT :-**

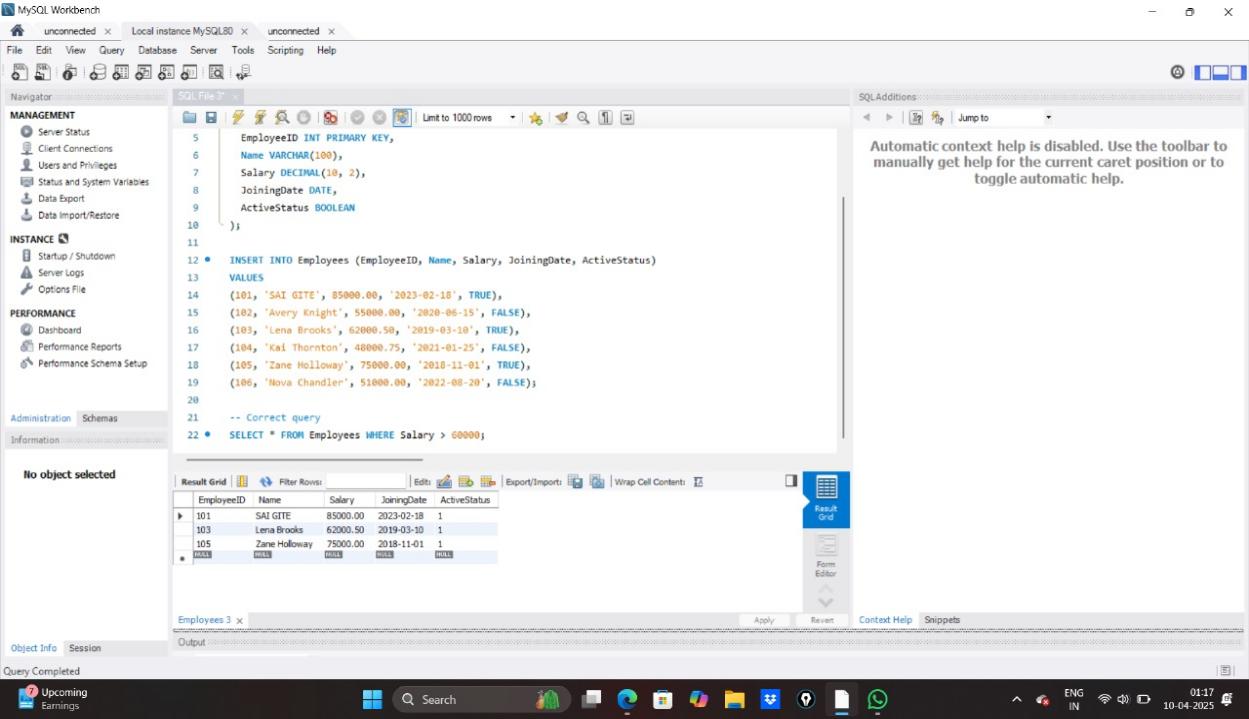
****

* **CODE FOR PERFORMING QUERIES TO MANIPULATE AND RETRIEVE DATA :-**

1. SELECT \* FROM Employees

* **OUTPUT :-**

1. SELECT \* FROM Employees WHERE Salary > 60000 ;

**OUTPUT :-**

5.UPDATE Employees

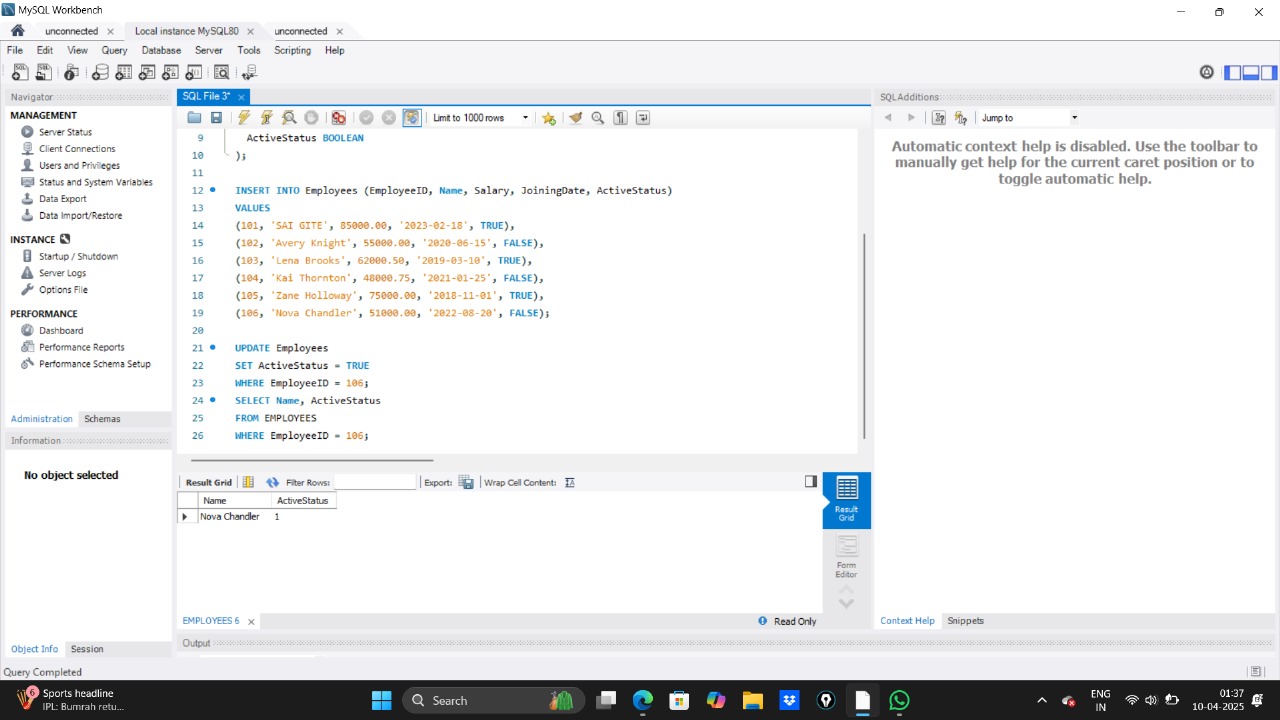
SET ActiveStatus = TRUE

WHERE EmployeeID = 106;

SELECT Name, ActiveStatus

FROM EMPLOYEES

WHERE EmployeeID = 106;

* **OUTPUT :-**

**Lab File – Practical 4**

**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 Employee Table

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,

FOREIGN KEY (DeptID) REFERENCES Department(DeptID)

);

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, 'Yash', 'yash.d@example.com', 50000.00, 1);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)

VALUES (102, 'PRATIk', 'bob@example.com', 60000.00, 2);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)

VALUES (103, 'Charlie', 'charlie@example.com', 55000.00, 1);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)

VALUES (104, 'David', 'david@example.com', 45000.00, 2);

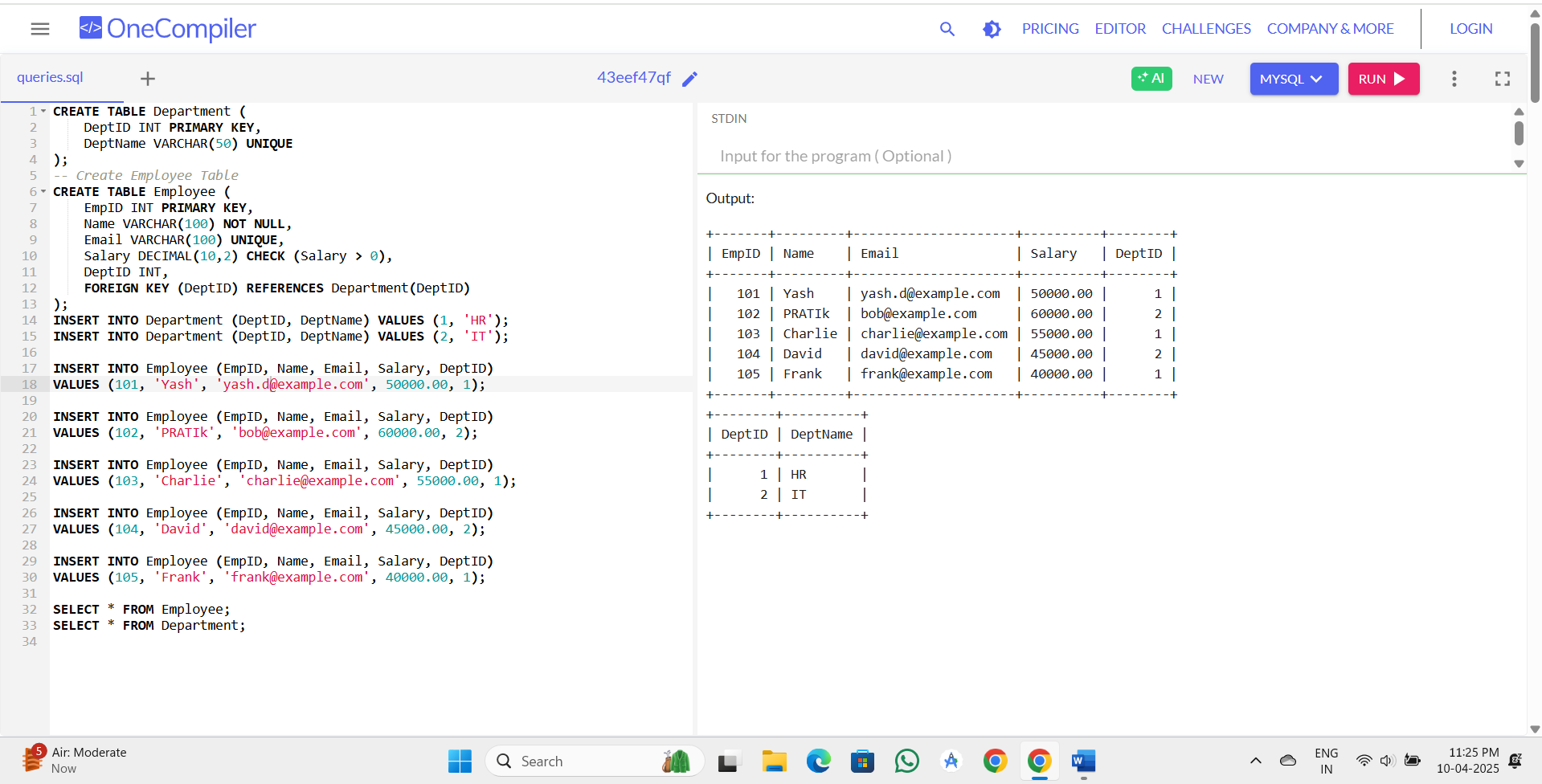
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)

VALUES (105, 'Frank', 'frank@example.com', 40000.00, 1);

SELECT \* FROM Employee;

SELECT \* FROM Department;

**Output :**

****

**Lab File – Practical 5**

**Aim:** Create a table for Customer details with various integrity constraints like NOT NULL, CHECK, and DEFAULT. Insert valid and invalid data to test these constraints and ensure data integrity.

**Code :**

CREATE TABLE Department (

DeptID INT PRIMARY KEY,

DeptName VARCHAR(50) UNIQUE

);

-- Create Employee Table

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,

FOREIGN KEY (DeptID) REFERENCES Department(DeptID)

);

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, 'Yash', 'yash.d@example.com', 50000.00, 1);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)

VALUES (102, 'PRATIk', 'bob@example.com', 60000.00, 2);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)

VALUES (103, 'Charlie', 'charlie@example.com', 55000.00, 1);

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)

VALUES (104, 'David', 'david@example.com', 45000.00, 2);

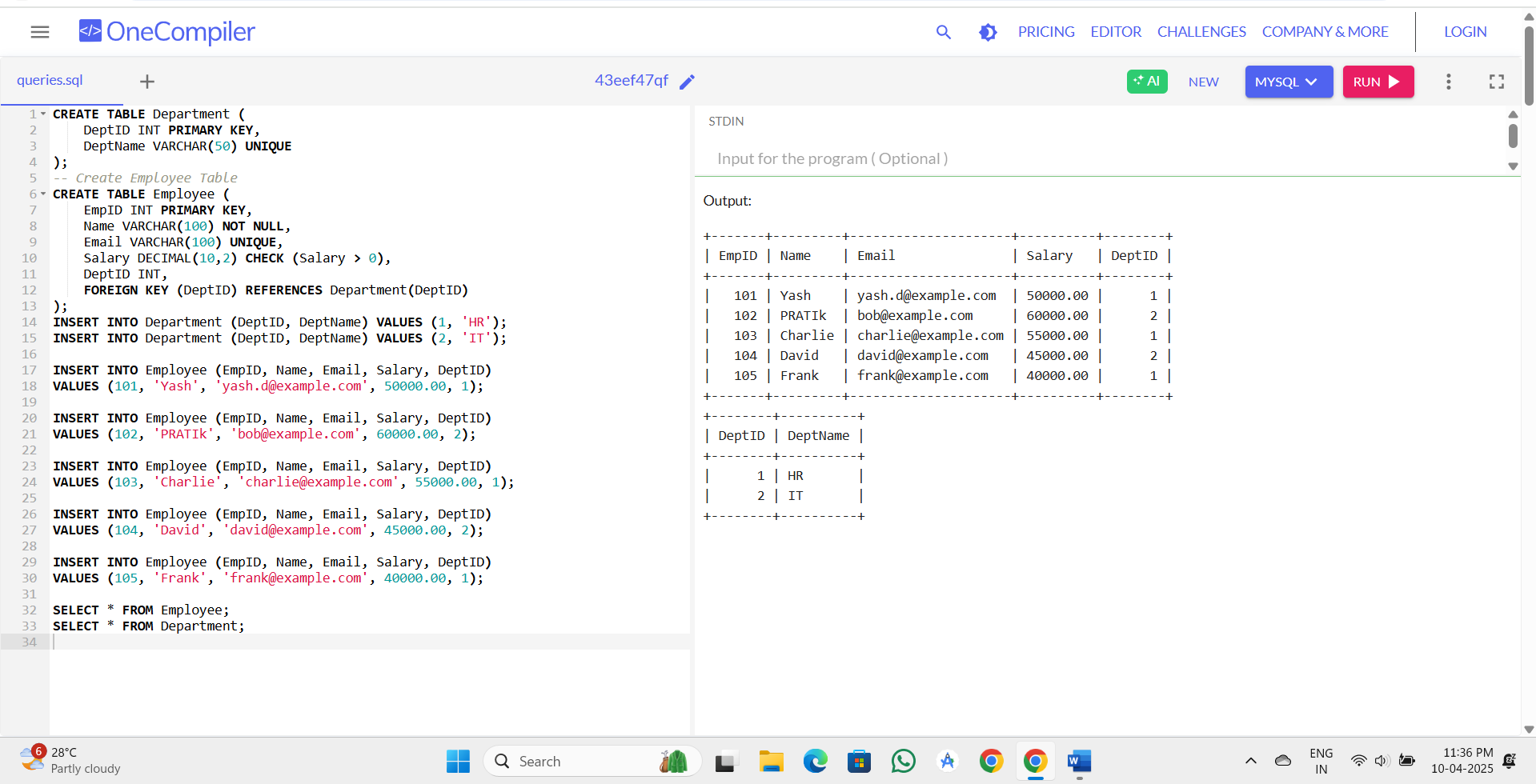
INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID)

VALUES (105, 'Frank', 'frank@example.com', 40000.00, 1);

SELECT \* FROM Employee;

SELECT \* FROM Department;

**Output :**



**Lab File – Practical 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

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Age INT,

Department VARCHAR(50),

Salary DECIMAL(10, 2)

);

-- DML Commands - Insert Data

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

VALUES (1, 'YASH', 'DOKHALE', 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);

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

UPDATE Employees

SET Salary = 70000.00

WHERE EmployeeID = 2;

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

UPDATE Employees

SET FirstName = 'YASH', LastName = 'DOKHALE', Salary = 75000.00

WHERE EmployeeID = 2;

-- 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;

-- 4. Update with a condition (e.g., increase salary by 10% for all employees in HR)

UPDATE Employees

SET Salary = Salary \* 1.10

WHERE Department = 'HR';

-- 5. Update with a subquery (increase salary for Employee with highest salary)

UPDATE Employees

SET Salary = Salary + 5000

WHERE Salary = (

SELECT max\_sal FROM (

SELECT MAX(Salary) AS max\_sal FROM Employees

) AS temp

);

-- 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;

-- Delete Data from the Table

DELETE FROM Employees

WHERE EmployeeID = 1;

-- Select and Verify Data

-- Retrieve all data

SELECT \* FROM Employees;

-- Check update for EmployeeID = 2

SELECT \* FROM Employees

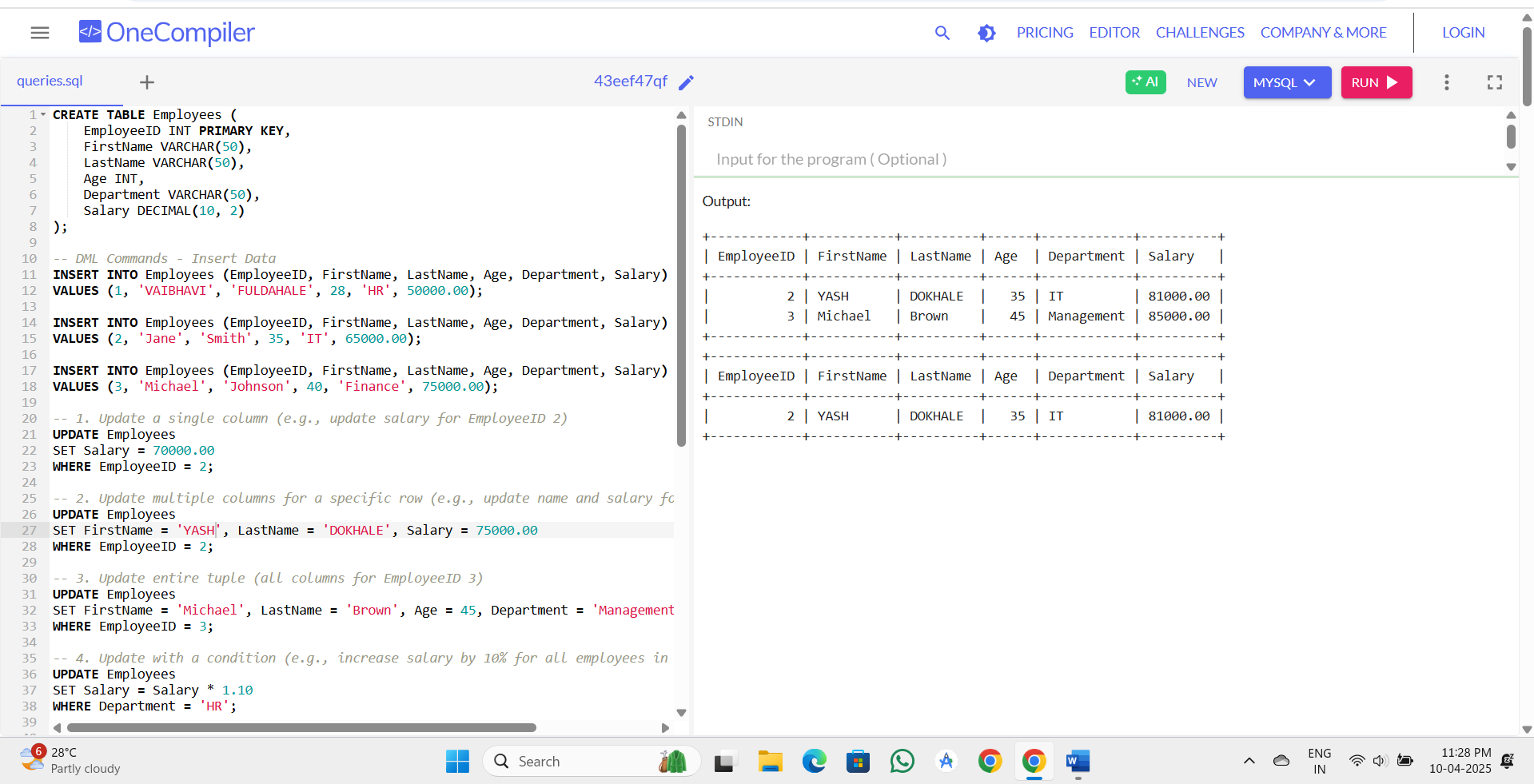
WHERE EmployeeID = 2;

-- Check if EmployeeID = 1 still exists

SELECT \* FROM Employees

WHERE EmployeeID = 1;

**Output :**

****

**Lab File – Practical 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 the Sales table

CREATE TABLE Sales (

SaleID INT PRIMARY KEY AUTO\_INCREMENT,

Product VARCHAR(50),

Quantity INT,

Price DECIMAL(10,2),

SaleDate DATE

);

-- Insert valid sales data

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');

-- COUNT queries

SELECT COUNT(\*) AS Total\_Sales FROM Sales;

SELECT COUNT(DISTINCT Product) AS Unique\_Products FROM Sales;

SELECT Product, COUNT(\*) AS Sales\_Count FROM Sales GROUP BY Product;

SELECT SaleDate, COUNT(\*) AS Sales\_Per\_Day FROM Sales GROUP BY SaleDate;

SELECT COUNT(\*) AS High\_Quantity\_Sales FROM Sales WHERE Quantity > 2;

SELECT COUNT(\*) AS Sales\_This\_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT\_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT\_DATE);

SELECT COUNT(\*) AS High\_Value\_Sales FROM Sales WHERE (Quantity \* Price) > 50000;

SELECT Product, COUNT(\*) AS High\_Value\_Transactions

FROM Sales

WHERE (Quantity \* Price) > 40000

GROUP BY Product;

SELECT COUNT(\*) AS Sales\_After\_Date FROM Sales WHERE SaleDate > '2025-02-03';

-- SUM queries

SELECT SUM(Quantity \* Price) AS Total\_Revenue FROM Sales;

SELECT SUM(Quantity) AS Total\_Quantity\_Sold FROM Sales;

SELECT Product, SUM(Quantity \* Price) AS Revenue\_Per\_Product FROM Sales GROUP BY Product;

SELECT SaleDate, SUM(Quantity \* Price) AS Revenue\_Per\_Day FROM Sales GROUP BY SaleDate;

SELECT SUM(Quantity \* Price) AS Revenue\_This\_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT\_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT\_DATE);

SELECT SUM(Quantity \* Price) AS High\_Quantity\_Revenue FROM Sales WHERE Quantity > 2;

SELECT SUM(Quantity \* Price) AS Revenue\_After\_Date FROM Sales WHERE SaleDate > '2025-02-03';

SELECT Product, SUM(Quantity \* Price) AS High\_Value\_Revenue

FROM Sales

WHERE (Quantity \* Price) > 40000

GROUP BY Product;

-- AVG queries

SELECT AVG(Price) AS Average\_Price FROM Sales;

SELECT AVG(Quantity) AS Average\_Quantity\_Sold FROM Sales;

SELECT AVG(Quantity \* Price) AS Average\_Revenue\_Per\_Transaction FROM Sales;

SELECT Product, AVG(Price) AS Average\_Price\_Per\_Product FROM Sales GROUP BY Product;

SELECT Product, AVG(Quantity \* Price) AS Average\_Revenue\_Per\_Product FROM Sales GROUP BY Product;

SELECT Product, AVG(Quantity) AS Average\_Quantity\_Per\_Product FROM Sales GROUP BY Product;

SELECT SaleDate, AVG(Quantity \* Price) AS Average\_Revenue\_Per\_Day FROM Sales GROUP BY SaleDate;

SELECT AVG(Quantity \* Price) AS Average\_Revenue\_This\_Month

FROM Sales

WHERE MONTH(SaleDate) = MONTH(CURRENT\_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT\_DATE);

SELECT AVG(Price) AS Avg\_Price\_High\_Quantity\_Sales FROM Sales WHERE Quantity > 2;

SELECT AVG(Quantity \* Price) AS Average\_Revenue\_After\_Date FROM Sales WHERE SaleDate > '2025-02-03';

-- MIN and MAX queries

SELECT MIN(Price) AS Min\_Price, MAX(Price) AS Max\_Price FROM Sales;

SELECT MIN(Quantity) AS Min\_Quantity\_Sold, MAX(Quantity) AS Max\_Quantity\_Sold FROM Sales;

SELECT MIN(Quantity \* Price) AS Min\_Revenue, MAX(Quantity \* Price) AS Max\_Revenue FROM Sales;

SELECT Product, MIN(Price) AS Min\_Price\_Per\_Product, MAX(Price) AS Max\_Price\_Per\_Product FROM Sales GROUP BY Product;

SELECT Product, MIN(Quantity \* Price) AS Min\_Revenue\_Per\_Product, MAX(Quantity \* Price) AS Max\_Revenue\_Per\_Product FROM Sales GROUP BY Product;

SELECT Product, MIN(Quantity) AS Min\_Quantity\_Per\_Product, MAX(Quantity) AS Max\_Quantity\_Per\_Product FROM Sales GROUP BY Product;

SELECT SaleDate, MIN(Quantity \* Price) AS Min\_Revenue\_Per\_Day, MAX(Quantity \* Price) AS Max\_Revenue\_Per\_Day FROM Sales GROUP BY SaleDate;

SELECT MIN(Quantity \* Price) AS Min\_Revenue\_This\_Month, MAX(Quantity \* Price) AS Max\_Revenue\_This\_Month

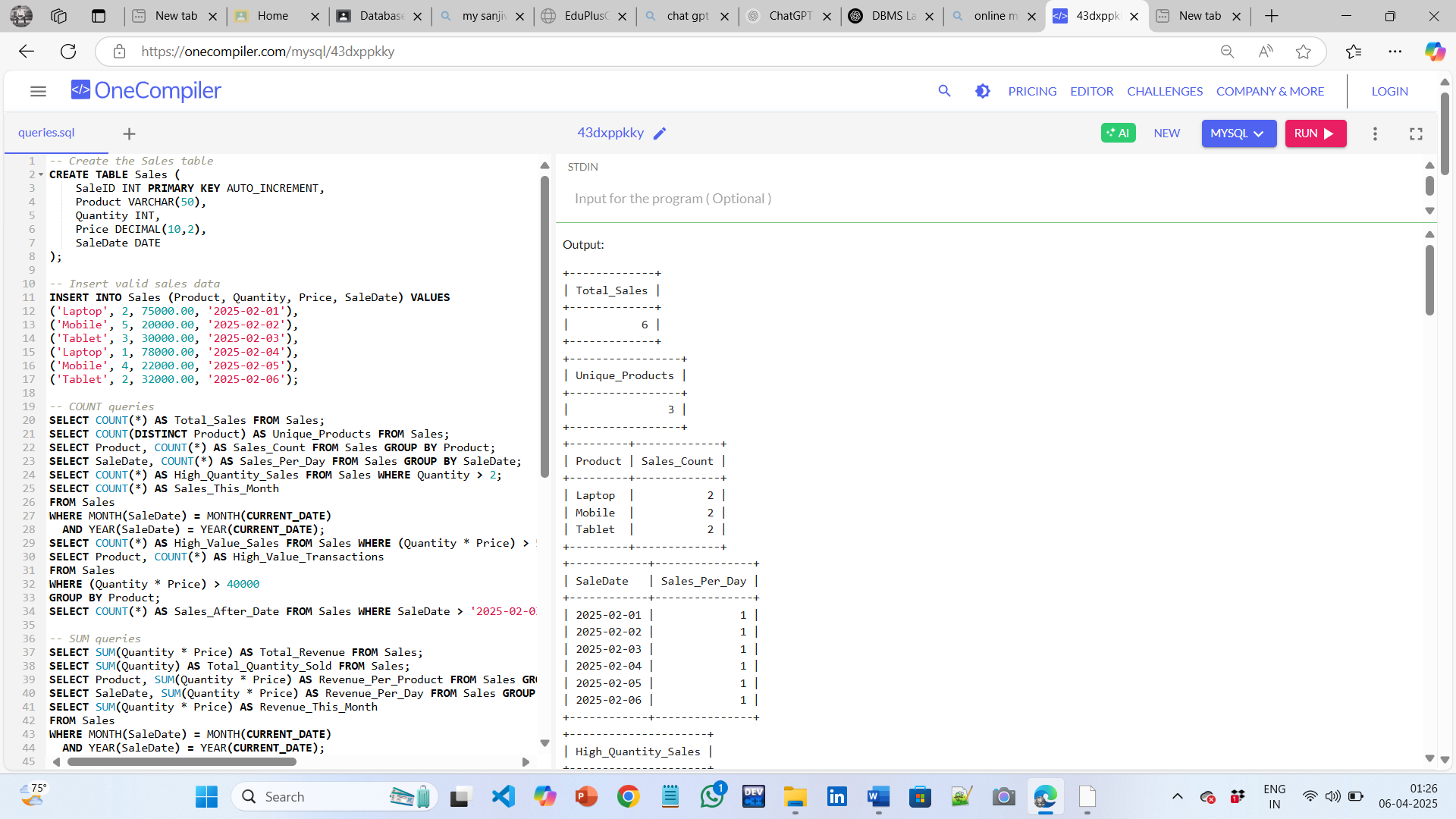
FROM Sales

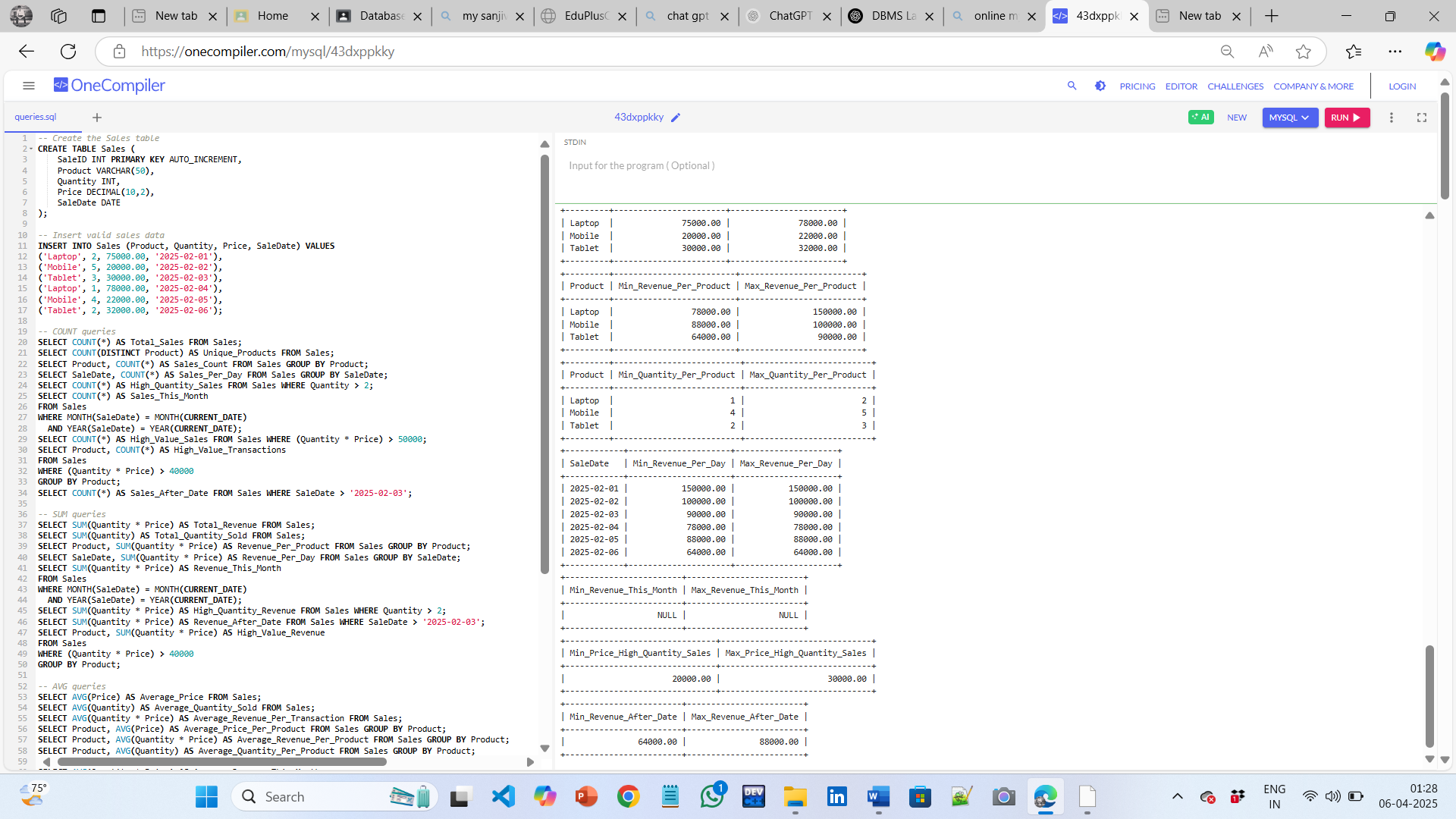
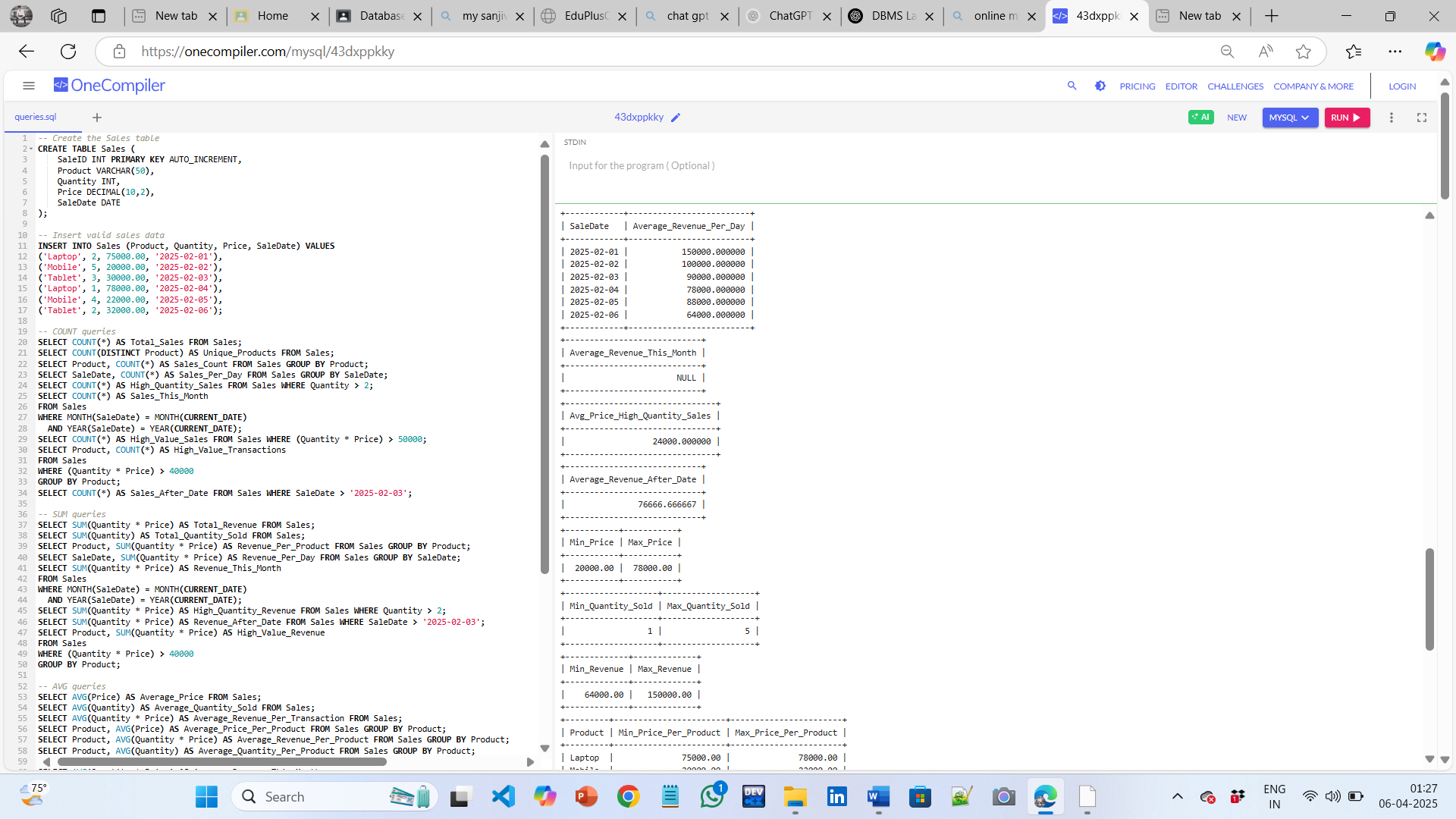
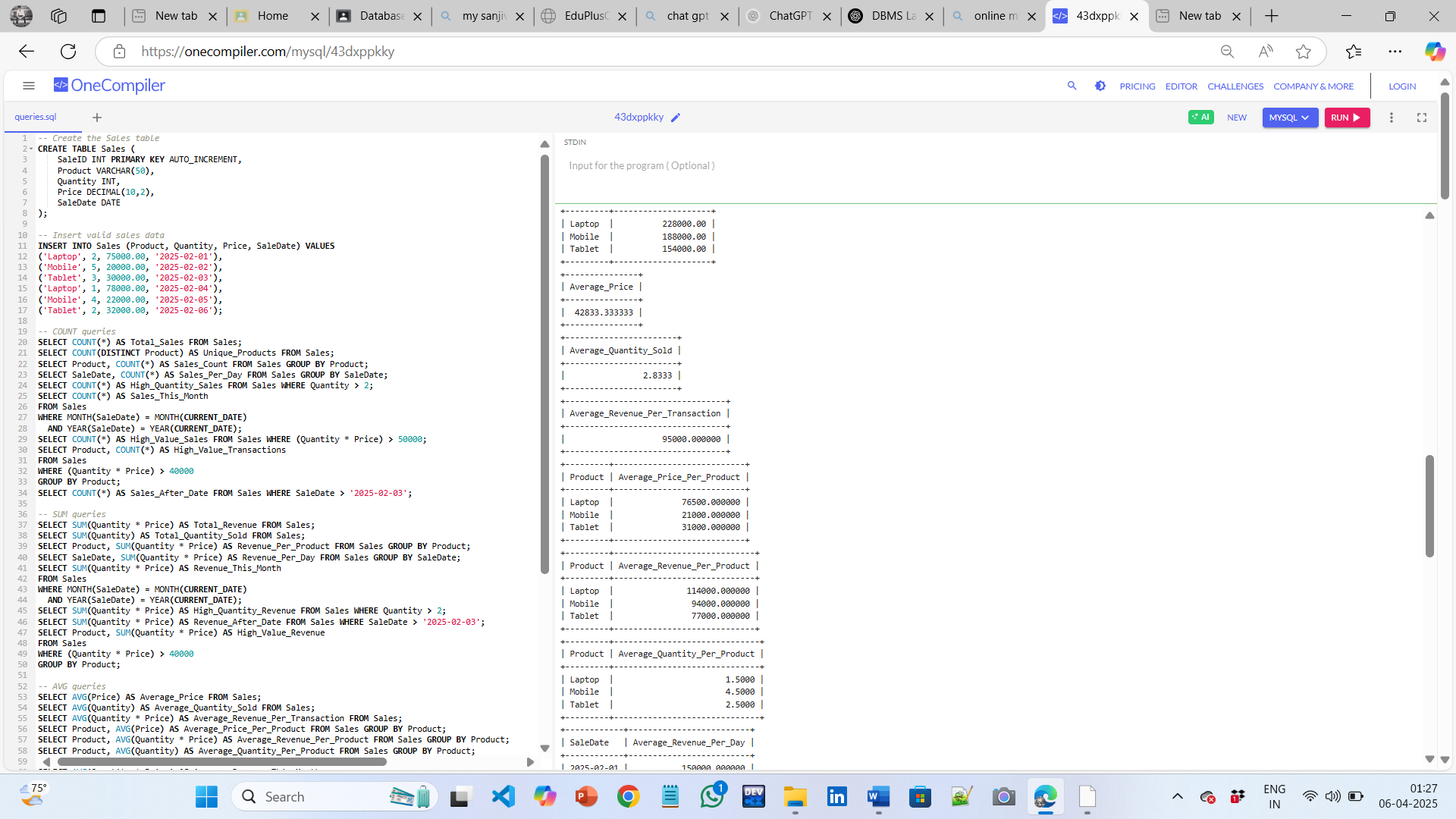
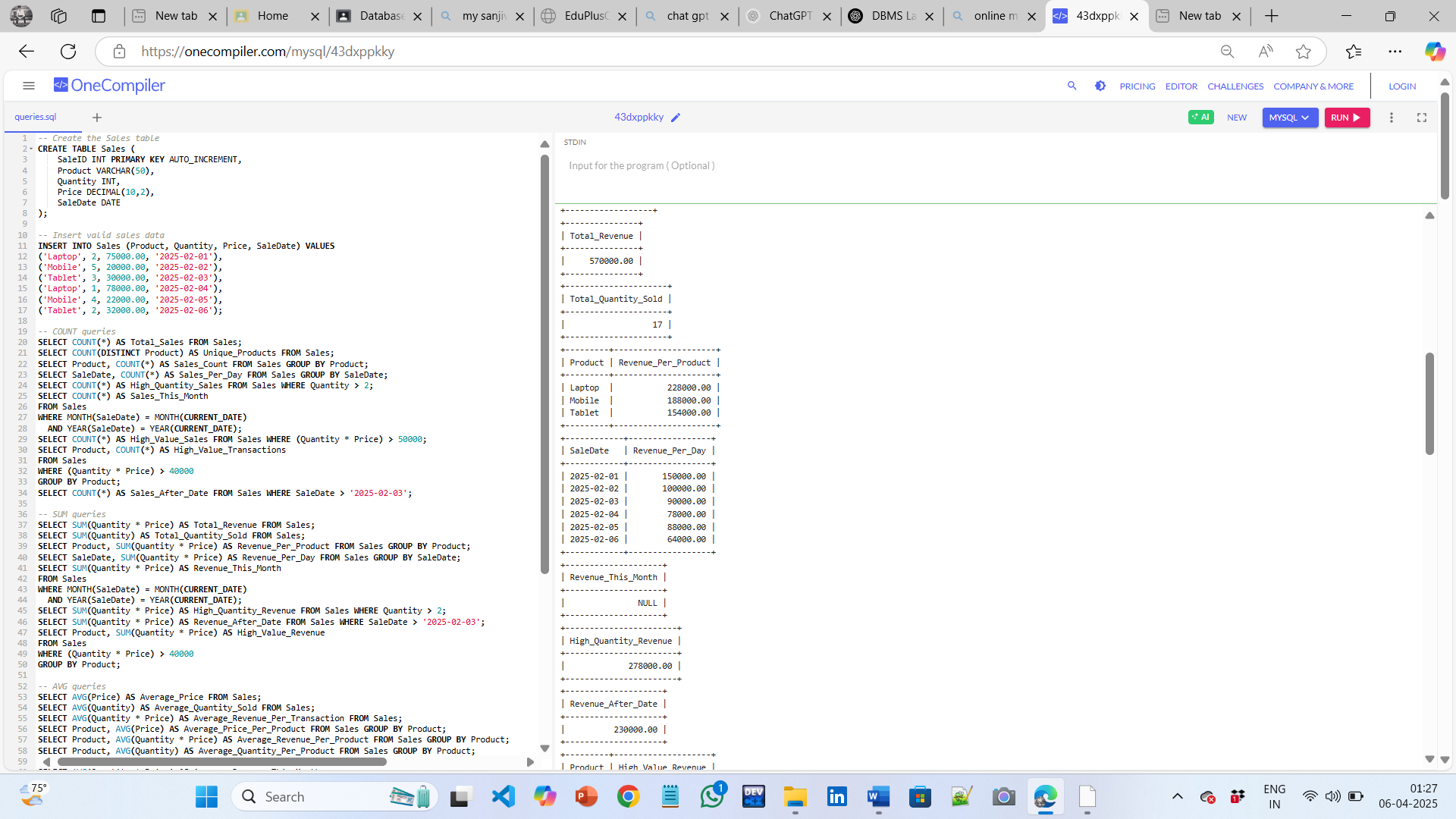
WHERE MONTH(SaleDate) = MONTH(CURRENT\_DATE)

AND YEAR(SaleDate) = YEAR(CURRENT\_DATE);

SELECT MIN(Price) AS Min\_Price\_High\_Quantity\_Sales, MAX(Price) AS Max\_Price\_High\_Quantity\_Sales FROM Sales WHERE Quantity > 2;

SELECT MIN(Quantity \* Price) AS Min\_Revenue\_After\_Date, MAX(Quantity \* Price) AS Max\_Revenue\_After\_Date FROM Sales WHERE SaleDate > '2025-02-03';

**Output :**

****

**Lab File – Practical 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 Customers table

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY,

customer\_name VARCHAR(100) NOT NULL

);

-- Create Orders table with foreign key referencing Customers

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

INSERT INTO Customers (customer\_id, customer\_name) VALUES

(1, 'YASH'),

(2, 'Bob'),

(3, 'Charlie'),

(4, 'David');

-- Insert into Orders

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);

-- Show Customers table

SELECT \* FROM Customers;

-- Show Orders table

SELECT \* FROM Orders;

-- INNER JOIN - Only matching customer & order records

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 (if any)

SELECT

c.customer\_id,

c.customer\_name,

o.order\_id,

o.order\_date

FROM

Customers c

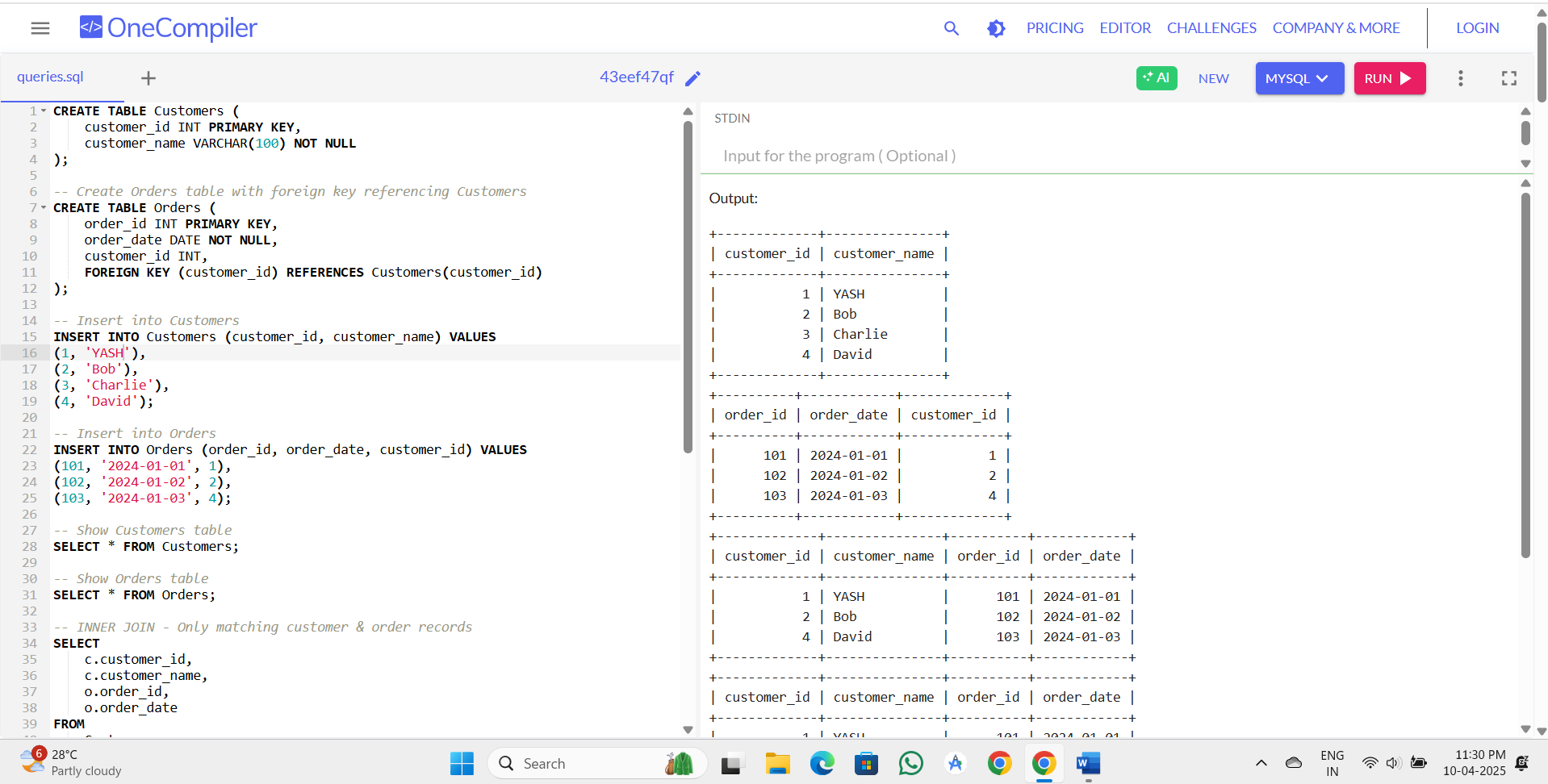
RIGHT JOIN

Orders o

ON

c.customer\_id = o.customer\_id;

**output :**

****