

## Practical No.:- 1

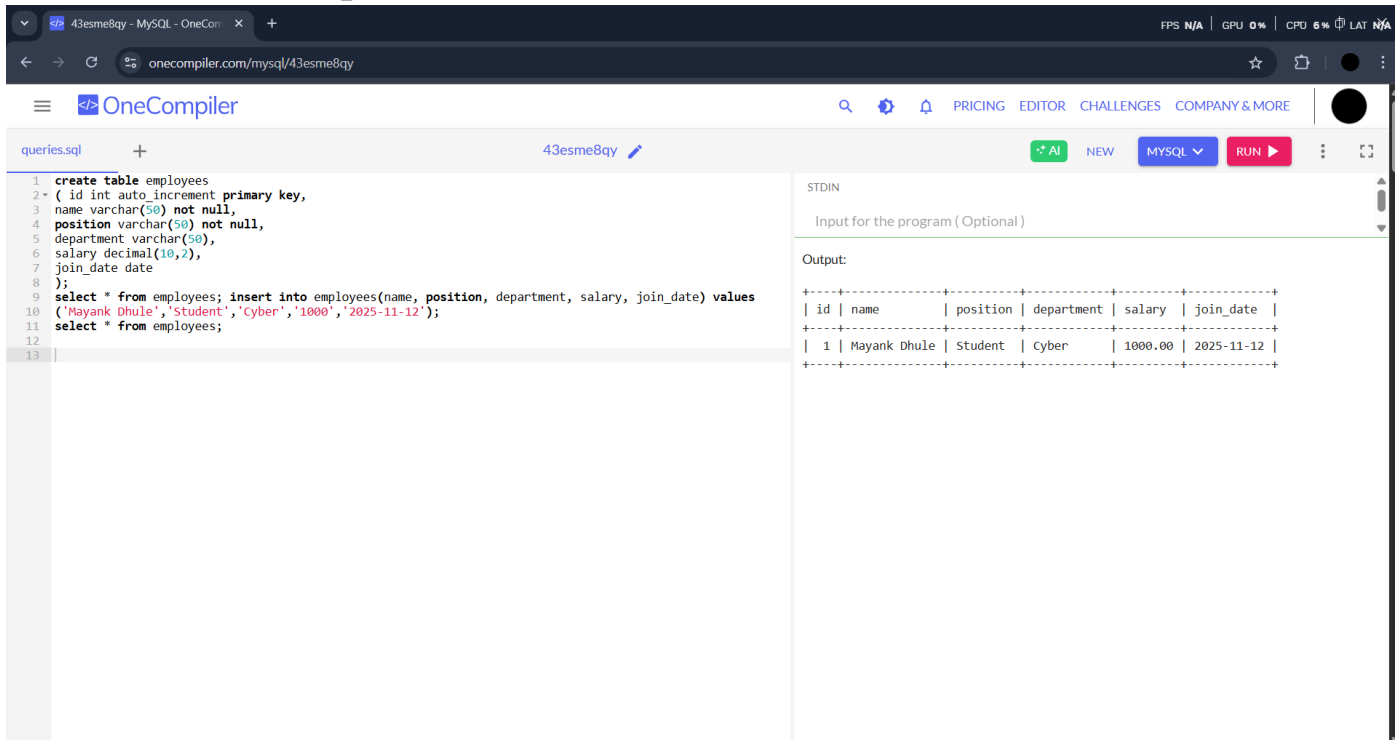
### Aim:-

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:-



The screenshot shows the OneCompiler web interface for MySQL. The editor contains the following SQL code:

```
1 create table employees
2 ( id int auto_increment primary key,
3  name varchar(50) not null,
4  position varchar(50) not null,
5  department varchar(50),
6  salary decimal(10,2),
7  join_date date
8 );
9 select * from employees; insert into employees(name, position, department, salary, join_date) values
10 ('Mayank Dhule', 'Student', 'Cyber', '1000', '2025-11-12');
11 select * from employees;
12
13
```

The output section displays the result of the query, showing a table with one row:

id	name	position	department	salary	join_date
1	Mayank Dhule	Student	Cyber	1000.00	2025-11-12

## 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:-

The screenshot shows the OneCompiler MySQL interface. The SQL editor on the left contains the following code:

```

1 CREATE TABLE Employee (
2   EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
3   Name VARCHAR(100) NOT NULL,
4   Salary DECIMAL(10, 2) CHECK (Salary > 0),
5   JoiningDate DATE NOT NULL,
6   ActiveStatus BOOLEAN DEFAULT TRUE
7 );
8 INSERT INTO Employee (Name, Salary, JoiningDate, ActiveStatus)
9 VALUES
10 ('Mayank', 100000.00, '2028-04-13', TRUE),
11 ('Ghanish Patil', 50000.00, '2028-05-14', FALSE),
12 ('Omkar Shelar', 200000.00, '2028-07-25', TRUE),
13 ('Saif Pathan', 250000.00, '2028-09-26', FALSE);
14
15 SELECT * FROM Employee;
16

```

The output panel on the right displays the result of the query in a table format:

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Mayank	100000.00	2028-04-13	1
2	Ghanish Patil	50000.00	2028-05-14	0
3	Omkar Shelar	200000.00	2028-07-25	1
4	Saif Pathan	250000.00	2028-09-26	0

## **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:-

The screenshot shows the OneCompiler MySQL interface. The left pane contains the following SQL queries:

```
1 -- Create the Employee table
2 CREATE TABLE Employee (
3   EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
4   Name VARCHAR(100) NOT NULL,
5   Salary DECIMAL(10,2) CHECK (Salary > 0),
6   JoiningDate DATE NOT NULL,
7   ActiveStatus BOOLEAN DEFAULT TRUE
8 );
9
10 -- Insert Sample Data
11 INSERT INTO Employee (Name, Salary, JoiningDate, ActiveStatus)
12 VALUES
13   ('Mayank Dhule', 65000.00, '2023-04-10', TRUE),
14   ('Aditya Mhaismale', 72000.00, '2022-08-25', FALSE),
15   ('Urmi Desai', 58000.75, '2021-11-18', TRUE),
16   ('Krishna Tiwari', 61000.00, '2020-06-20', TRUE);
17
18 -- Retrieve All Employees
19 SELECT * FROM Employee;
20
21 -- Retrieve Active Employees
22 SELECT EmployeeID, Name, Salary FROM Employee WHERE ActiveStatus = TRUE;
23
24 -- Increase Salary of an Employee
25 UPDATE Employee SET Salary = Salary * 1.10 WHERE EmployeeID = 2;
26
27 -- Change Active Status of an Employee
28 UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID = 4;
29
30 -- Delete an Employee Record
31 DELETE FROM Employee WHERE EmployeeID = 3;
32
33 -- Retrieve Employees Who Joined in a Specific Year
34 SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
35
36 -- Retrieve Employees with Salary Greater Than a Specific Amount
37 SELECT Name, Salary FROM Employee WHERE Salary > 60000;
38
39 -- Find the Highest & Lowest Salary in the Organization
40 SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary FROM Employee;
41
42 -- Retrieve the Top 3 Highest Paid Employees
43 SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
44
```

The right pane shows the output of the queries:

STDIN

Input for the program (Optional)

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Mayank Dhule	65000.00	2023-04-10	1
2	Aditya Mhaismale	72000.00	2022-08-25	0
3	Urmi Desai	58000.75	2021-11-18	1
4	Krishna Tiwari	61000.00	2020-06-20	1

EmployeeID	Name	Salary
1	Mayank Dhule	65000.00
3	Urmi Desai	58000.75
4	Krishna Tiwari	61000.00

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Mayank Dhule	65000.00	2023-04-10	1

Name	Salary
Mayank Dhule	65000.00
Aditya Mhaismale	72000.00
Krishna Tiwari	61000.00

HighestSalary	LowestSalary
72000.00	61000.00

The screenshot shows the OneCompiler MySQL interface. The left pane contains the following SQL queries:

```
1 -- Create the Employee table
2 CREATE TABLE Employee (
3   EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
4   Name VARCHAR(100) NOT NULL,
5   Salary DECIMAL(10,2) CHECK (Salary > 0),
6   JoiningDate DATE NOT NULL,
7   ActiveStatus BOOLEAN DEFAULT TRUE
8 );
9
10 -- Insert Sample Data
11 INSERT INTO Employee (Name, Salary, JoiningDate, ActiveStatus)
12 VALUES
13   ('Mayank Dhule', 65000.00, '2023-04-10', TRUE),
14   ('Aditya Mhaismale', 72000.00, '2022-08-25', FALSE),
15   ('Urmi Desai', 58000.75, '2021-11-18', TRUE),
16   ('Krishna Tiwari', 61000.00, '2020-06-20', TRUE);
17
18 -- Retrieve All Employees
19 SELECT * FROM Employee;
20
21 -- Retrieve Active Employees
22 SELECT EmployeeID, Name, Salary FROM Employee WHERE ActiveStatus = TRUE;
23
24 -- Increase Salary of an Employee
25 UPDATE Employee SET Salary = Salary * 1.10 WHERE EmployeeID = 2;
26
27 -- Change Active Status of an Employee
28 UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID = 4;
29
30 -- Delete an Employee Record
31 DELETE FROM Employee WHERE EmployeeID = 3;
32
33 -- Retrieve Employees Who Joined in a Specific Year
34 SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
35
36 -- Retrieve Employees with Salary Greater Than a Specific Amount
37 SELECT Name, Salary FROM Employee WHERE Salary > 60000;
38
39 -- Find the Highest & Lowest Salary in the Organization
40 SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS LowestSalary FROM Employee;
41
42 -- Retrieve the Top 3 Highest Paid Employees
43 SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
44
```

The right pane shows the output of the queries:

STDIN

Input for the program (Optional)

EmployeeID	Name	Salary
1	Mayank Dhule	65000.00
3	Urmi Desai	58000.75
4	Krishna Tiwari	61000.00

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Mayank Dhule	65000.00	2023-04-10	1

Name	Salary
Mayank Dhule	65000.00
Aditya Mhaismale	72000.00
Krishna Tiwari	61000.00

HighestSalary	LowestSalary
72000.00	61000.00

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
2	Aditya Mhaismale	72000.00	2022-08-25	0
1	Mayank Dhule	65000.00	2023-04-10	1
4	Krishna Tiwari	61000.00	2020-06-20	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 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:-**

The screenshot displays the MySQL Workbench interface. The central pane shows a series of SQL queries executed in a single session. The queries include creating a database, using it, creating tables for Department and Employee, and inserting data into both. The output pane at the bottom shows the execution results for each query, including the time taken and the number of rows affected. The final query, which attempts to insert a record with a negative salary, results in an error: 'Error Code: 3819. Check constraint 'employee\_chk\_1' is violated.'

#	Time	Action	Message	Duration / Fetch
3	18:36:42	CREATE TABLE Department ( DeptID INT PRIMARY KEY, DeptName VARCHAR(50) UNIQUE )	0 row(s) affected	0.078 sec
4	18:36:46	CREATE TABLE Employee ( EmpID INT PRIMARY KEY, Name VARCHAR(100) NOT NULL, Email VA...	0 row(s) affected	0.047 sec
5	18:36:56	INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (202, 'Rohit Jain', rohit.jain@company...	1 row(s) affected	0.000 sec
6	18:37:03	INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (201, 'Sakshi Rao', 'sakshi.rao@comp...	1 row(s) affected	0.016 sec
7	18:37:07	INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (203, 'Karan Desai', 'neha.verma@co...	1 row(s) affected	0.031 sec
8	18:37:09	INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (204, 'Ishaan Mehta', 'ishaan.mehta@...	Error Code: 3819. Check constraint 'employee_chk_1' is violated.	0.000 sec



## **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, 'Sarathak', 'Rao', 'mayank.dhule@example.com', '9900112233', 27);
```

## Screenshot of Output:-

The screenshot displays the MySQL Workbench interface. The main editor shows a SQL script with the following queries:

```
1 USE employees;
2 CREATE TABLE Customer (
3   CustomerID INT PRIMARY KEY,
4   FirstName VARCHAR(100) NOT NULL,
5   LastName VARCHAR(100) NOT NULL,
6   Email VARCHAR(100) UNIQUE,
7   Phone VARCHAR(15),
8   Age INT CHECK (Age >= 18),
9   IsActive BOOLEAN DEFAULT TRUE);
10 INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age, IsActive)
11 VALUES (1, 'Mayank', 'Dhule', 'mayank.dhule@example.com', '9876543210', 22, TRUE),
12        (2, 'Riya', 'Singh', 'riya.singh@example.com', '9123456789', 25, TRUE);
13 INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
14 VALUES (3, NULL, 'Patel', 'amit.patel@example.com', '9012345678', 21);
15 INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
16 VALUES (4, 'Anika', 'Mehra', 'anika.mehra@example.com', '9988776655', 16);
17 INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)
18 VALUES (5, 'Sarthak', 'Rao', 'mayank.dhule@example.com', '9900112233', 27);
19
```

The Output tab at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	18:49:12	USE employees	0 row(s) affected	0.000 sec
2	18:49:16	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);	0 row(s) affected	0.094 sec
3	18:49:19	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);	2 row(s) affected Records: 2 Duplicates: 0 Warnings: 0	0.032 sec
4	18:49:26	INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age) VALUES (3, NULL, 'Patel', 'amit.patel@example.com', '9012345678', 21);	Error Code: 1048. Column 'FirstName' cannot be null	0.000 sec
5	18:49:32	INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age) VALUES (4, 'Anika', 'Mehra', 'anika.mehra@example.com', '9988776655', 16);	Error Code: 3819 Check constraint 'customer_chk_1' is violated.	0.000 sec
6	18:49:36	INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age) VALUES (5, 'Sarthak', 'Rao', 'mayank.dhule@example.com', '9900112233', 27);	Error Code: 1062 Duplicate entry 'mayank.dhule@example.com' for key 'customer.Email'	0.046 sec

## 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:-**

The screenshot shows the OneCompiler MySQL interface. The left pane contains the following SQL code:

```
1 CREATE TABLE Employees (  
2 EmployeeID INT PRIMARY KEY,  
3 FirstName VARCHAR(50),  
4 LastName VARCHAR(50),  
5 Age INT,  
6 Department VARCHAR(50),  
7 Salary DECIMAL(10, 2)  
8 );  
9  
10  
11 -- Insert Data  
12 INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)  
13 VALUES  
14 (1, 'John', 'Doe', 28, 'HR', 50000.00),  
15 (2, 'Jane', 'Smith', 35, 'IT', 65000.00),  
16 (3, 'Michael', 'Johnson', 40, 'Finance', 75000.00);  
17  
18 -- View After Insert  
19 SELECT * FROM Employees;  
20  
21 -- Update salary for EmployeeID 2  
22 UPDATE Employees  
23 SET Salary = 70000.00  
24 WHERE EmployeeID = 2;  
25  
26 -- View After Update  
27 SELECT * FROM Employees;  
28  
29 -- Update name and salary for EmployeeID 2  
30 UPDATE Employees  
31 SET FirstName = 'Mayank', LastName = 'Dhule', Salary = 75000.00  
32 WHERE EmployeeID = 2;  
33  
34 -- View After Update  
35 SELECT * FROM Employees;  
36  
37 -- Update all columns for EmployeeID 3  
38 UPDATE Employees  
39 SET FirstName = 'Michael', LastName = 'Brown', Age = 45, Department = 'Management', Salary = 80000.00  
40 WHERE EmployeeID = 3;  
41  
42 -- View After Update  
43 SELECT * FROM Employees;  
44
```

The right pane shows the output of the queries. The first query returns three rows of employee data. The second query updates the salary of EmployeeID 2 to 70000.00. The third query updates the name and salary of EmployeeID 2 to Mayank Dhule with a salary of 75000.00. The fourth query updates all columns of EmployeeID 3 to Michael Brown, Age 45, Department Management, and Salary 80000.00.

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
1	John	Doe	28	HR	50000.00
2	Mayank	Dhule	35	IT	75000.00
3	Michael	Johnson	40	Finance	75000.00

EmployeeID	FirstName	LastName	Age	Department	Salary
1	John	Doe	28	HR	50000.00
2	Mayank	Dhule	35	IT	75000.00
3	Michael	Brown	45	Management	80000.00

The screenshot shows the OneCompiler MySQL IDE interface. On the left, a SQL script is being edited, and on the right, the output of the script is displayed.

**SQL Script:**

```

42 -- View After Update
43 SELECT * FROM Employees;
44
45 -- Increase salary by 10% for HR department
46 UPDATE Employees
47 SET Salary = Salary * 1.10
48 WHERE Department = 'HR';
49
50 -- View After Update
51 SELECT * FROM Employees;
52
53 -- Increase salary for employee with highest salary
54 UPDATE Employees
55 JOIN (
56     SELECT MAX(Salary) AS MaxSalary
57     FROM Employees
58 ) AS MaxSalarySubquery
59 ON Employees.Salary = MaxSalarySubquery.MaxSalary
60 SET Employees.Salary = Employees.Salary + 5000;
61
62 -- View After Update
63 SELECT * FROM Employees;
64
65 -- Increase salary based on department
66 UPDATE Employees
67 SET Salary = CASE
68     WHEN Department = 'HR' THEN Salary * 1.05
69     WHEN Department = 'IT' THEN Salary * 1.08
70     WHEN Department = 'Finance' THEN Salary * 1.10
71     ELSE Salary
72 END;
73
74 -- View After Update
75 SELECT * FROM Employees;
76
77 -- Delete EmployeeID 1
78 DELETE FROM Employees
79 WHERE EmployeeID = 1;
80
81 -- View After Deletion
82 SELECT * FROM Employees;
83
84 -- Check if EmployeeID 1 exists
85

```

**Output:**

STDIN

Input for the program (Optional)

EmployeeID	FirstName	LastName	Age	Department	Salary
1	John	Doe	28	HR	50000.00
2	Mayank	Dhule	35	IT	75000.00
3	Michael	Brown	45	Management	80000.00

EmployeeID	FirstName	LastName	Age	Department	Salary
1	John	Doe	28	HR	55000.00
2	Mayank	Dhule	35	IT	75000.00
3	Michael	Brown	45	Management	80000.00

EmployeeID	FirstName	LastName	Age	Department	Salary
1	John	Doe	28	HR	57750.00
2	Mayank	Dhule	35	IT	81000.00
3	Michael	Brown	45	Management	85000.00

EmployeeID	FirstName	LastName	Age	Department	Salary
2	Mayank	Dhule	35	IT	81000.00
3	Michael	Brown	45	Management	85000.00

## 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;
```

-- 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:-

The screenshot shows the OneCompiler MySQL interface. The left pane displays the following SQL queries:

```
1 CREATE TABLE Sales (  
2   SaleID INT PRIMARY KEY AUTO_INCREMENT,  
3   Product VARCHAR(50),  
4   Quantity INT,  
5   Price DECIMAL(10,2),  
6   SaleDate DATE  
7 );  
8  
9 INSERT INTO Sales (Product, Quantity, Price, SaleDate) |  
10 VALUES  
11 ('Laptop', 2, 75000.00, '2025-02-01'),  
12 ('Mobile', 5, 20000.00, '2025-02-02'),  
13 ('Tablet', 3, 30000.00, '2025-02-03'),  
14 ('Laptop', 1, 78000.00, '2025-02-04'),  
15 ('Mobile', 4, 22000.00, '2025-02-05'),  
16 ('Tablet', 2, 32000.00, '2025-02-06');  
17  
18 -- View all records  
19 SELECT * FROM Sales;  
20  
21 -- COUNT Queries  
22 -- 1. Count the total number of sales records  
23 SELECT COUNT(*) AS Total_Sales FROM Sales;  
24  
25 -- 2. Sum of total revenue generated  
26 SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;  
27  
28 -- 3. Average price of products sold
```

The right pane shows the output of these queries:

Output:

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

Total_Sales
6

Total_Revenue
570000.00

Average_Price

OneCompiler

PRICING

EDITOR

CHALLENGES

COMPANY & MORE

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

27
28 -- 3. Average price of products sold
29 SELECT AVG(Price) AS Average_Price FROM Sales;
30
31 -- 4. Minimum and Maximum price of a product sold
32 SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;
33
34 -- 5. Count the number of distinct products sold
35 SELECT COUNT(DISTINCT Product) AS Unique_Products FROM Sales;
36
37 -- 6. Count the number of sales per product
38 SELECT Product, COUNT(*) AS Sales_Count
39 FROM Sales
40 GROUP BY Product;
41
42 -- 7. Count the number of sales per day
43 SELECT SaleDate, COUNT(*) AS Sales_Per_Day
44 FROM Sales
45 GROUP BY SaleDate;
46
47 -- 8. Count the number of sales where more than 2 units were sold
48 SELECT COUNT(*) AS High_Quantity_Sales
49 FROM Sales
50 WHERE Quantity > 2;
51
52 -- 9. Count the number of sales in the current month
53 SELECT COUNT(*) AS Sales_This_Month
54 FROM Sales
55 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE);

```

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

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

52 -- 9. Count the number of sales in the current month
53 SELECT COUNT(*) AS Sales_This_Month
54 FROM Sales
55 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
56 AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
57
58 -- 10. Count the number of sales transactions where total sale value was
59 SELECT COUNT(*) AS High_Value_Sales
60 FROM Sales
61 WHERE (Quantity * Price) > 50000;
62
63 -- 11. Count the number of sales records for each product where total sa
64 SELECT Product, COUNT(*) AS High_Value_Transactions
65 FROM Sales
66 WHERE (Quantity * Price) > 40000
67 GROUP BY Product;
68
69 -- 12. Count the number of sales made after a specific date (e.g., Feb 3
70 SELECT COUNT(*) AS Sales_After_Date
71 FROM Sales
72 WHERE SaleDate > '2025-02-03';
73
74 -- SUM Queries
75 -- 1. Sum of total revenue generated
76 SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;
77
78 -- 2. Sum of total quantity of products sold
79 SELECT SUM(Quantity) AS Total_Quantity_Sold FROM Sales;
80

```

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

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

74 -- SUM Queries
75 -- 1. Sum of total revenue generated
76 SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;
77
78 -- 2. Sum of total quantity of products sold
79 SELECT SUM(Quantity) AS Total_Quantity_Sold FROM Sales;
80
81 -- 3. Sum of total revenue per product
82 SELECT Product, SUM(Quantity * Price) AS Revenue_Per_Product
83 FROM Sales
84 GROUP BY Product;
85
86 -- 4. Sum of total revenue per day
87 SELECT SaleDate, SUM(Quantity * Price) AS Revenue_Per_Day
88 FROM Sales
89 GROUP BY SaleDate;
90
91 -- 5. Sum of total revenue in the current month
92 SELECT SUM(Quantity * Price) AS Revenue_This_Month
93 FROM Sales
94 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
95 AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
96
97 -- 6. Sum of revenue for sales where quantity sold is greater than 2
98 SELECT SUM(Quantity * Price) AS High_Quantity_Revenue
99 FROM Sales
100 WHERE Quantity > 2;
101

```

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

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

101
102 -- 7. Sum of total revenue generated after a specific date (e.g., Feb 3
103 SELECT SUM(Quantity * Price) AS Revenue_After_Date
104 FROM Sales
105 WHERE SaleDate > '2025-02-03';
106
107 -- 8. Sum of revenue per product where the total revenue per transactio
108 SELECT Product, SUM(Quantity * Price) AS High_Value_Revenue
109 FROM Sales
110 WHERE (Quantity * Price) > 40000
111 GROUP BY Product;
112
113 -- AVG Queries
114 -- 1. Average price of products sold
115 SELECT AVG(Price) AS Average_Price FROM Sales;
116
117 -- 2. Average quantity of products sold per transaction
118 SELECT AVG(Quantity) AS Average_Quantity_Sold FROM Sales;
119
120 -- 3. Average revenue per transaction
121 SELECT AVG(Quantity * Price) AS Average_Revenue_Per_Transaction
122 FROM Sales;
123
124 -- 4. Average price per product
125 SELECT Product, AVG(Price) AS Average_Price_Per_Product
126 FROM Sales
127 GROUP BY Product;
128
129

```

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

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

128
129 -- 5. Average revenue per product
130 SELECT Product, AVG(Quantity * Price) AS Average_Revenue_Per_Product
131 FROM Sales
132 GROUP BY Product;
133
134 -- 6. Average quantity sold per product
135 SELECT Product, AVG(Quantity) AS Average_Quantity_Per_Product
136 FROM Sales
137 GROUP BY Product;
138
139 -- 7. Average revenue per day
140 SELECT SaleDate, AVG(Quantity * Price) AS Average_Revenue_Per_Day
141 FROM Sales
142 GROUP BY SaleDate;
143
144 -- 8. Average revenue in the current month
145 SELECT AVG(Quantity * Price) AS Average_Revenue_This_Month
146 FROM Sales
147 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
148 AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
149
150 -- 9. Average price of products where more than 2 units were sold
151 SELECT AVG(Price) AS Avg_Price_High_Quantity_Sales
152 FROM Sales
153 WHERE Quantity > 2;
154
155 -- 10. Average revenue after a specific date (e.g., Feb 3, 2025)
156

```

+-----+-----+	
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

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

154
155 -- 10. Average revenue after a specific date (e.g., Feb 3, 2025)
156 SELECT AVG(Quantity * Price) AS Average_Revenue_After_Date
157 FROM Sales
158 WHERE SaleDate > '2025-02-03';
159
160 -- MIN/MAX Queries
161 -- 1. Minimum and Maximum price of a product sold
162 SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;
163
164 -- 2. Minimum and Maximum quantity of products sold in a single transac
165 SELECT MIN(Quantity) AS Min_Quantity_Sold, MAX(Quantity) AS Max_Quantit
166
167 -- 3. Minimum and Maximum revenue generated from a single transaction
168 SELECT MIN(Quantity * Price) AS Min_Revenue, MAX(Quantity * Price) AS M
169
170 -- 4. Minimum and Maximum price per product
171 SELECT Product, MIN(Price) AS Min_Price_Per_Product, MAX(Price) AS Max_
172 FROM Sales
173 GROUP BY Product;
174
175 -- 5. Minimum and Maximum revenue per product
176 SELECT Product, MIN(Quantity * Price) AS Min_Revenue_Per_Product, MAX(Q
177 FROM Sales
178 GROUP BY Product;
179
180 -- 6. Minimum and Maximum quantity sold per product
181 SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity
182

```

+-----+-----+	
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

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

179 -- 6. Minimum and Maximum quantity sold per product
180 SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity)
181 FROM Sales
182 GROUP BY Product;
183
184 -- 7. Minimum and Maximum revenue per day
185 SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quantity * Price) AS Max_Revenue_Per_Day
186 FROM Sales
187 GROUP BY SaleDate;
188
189 -- 8. Minimum and Maximum revenue in the current month
190 SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity * Price) AS Max_Revenue_This_Month
191 FROM Sales
192 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
193 AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
194
195 -- 9. Minimum and Maximum price of products where more than 2 units were sold
196 SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_Price_High_Quantity_Sales
197 FROM Sales
198 WHERE Quantity > 2;
199
200 -- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3, 2025)
201 SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity * Price) AS Max_Revenue_After_Date
202 FROM Sales
203 WHERE SaleDate > '2025-02-03';
204
205 -- View final data
206 SELECT * FROM Sales;
207

```

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

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

180 -- 6. Minimum and Maximum quantity sold per product
181 SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity)
182 FROM Sales
183 GROUP BY Product;
184
185 -- 7. Minimum and Maximum revenue per day
186 SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quantity * Price) AS Max_Revenue_Per_Day
187 FROM Sales
188 GROUP BY SaleDate;
189
190 -- 8. Minimum and Maximum revenue in the current month
191 SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity * Price) AS Max_Revenue_This_Month
192 FROM Sales
193 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
194 AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
195
196 -- 9. Minimum and Maximum price of products where more than 2 units were sold
197 SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_Price_High_Quantity_Sales
198 FROM Sales
199 WHERE Quantity > 2;
200
201 -- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3, 2025)
202 SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity * Price) AS Max_Revenue_After_Date
203 FROM Sales
204 WHERE SaleDate > '2025-02-03';
205
206 -- View final data
207 SELECT * FROM Sales;
208

```

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
1	5

OneCompiler

PRICING

EDITOR

CHALLENGES

COMPANY & MORE

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

180 -- 6. Minimum and Maximum quantity sold per product
181 SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity)
182 FROM Sales
183 GROUP BY Product;
184
185 -- 7. Minimum and Maximum revenue per day
186 SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quantity * Price) AS Max_Revenue_Per_Day
187 FROM Sales
188 GROUP BY SaleDate;
189
190 -- 8. Minimum and Maximum revenue in the current month
191 SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity * Price) AS Max_Revenue_This_Month
192 FROM Sales
193 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
194 AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
195
196 -- 9. Minimum and Maximum price of products where more than 2 units were sold
197 SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_Price_High_Quantity_Sales
198 FROM Sales
199 WHERE Quantity > 2;
200
201 -- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3, 2025)
202 SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity * Price) AS Max_Revenue_After_Date
203 FROM Sales
204 WHERE SaleDate > '2025-02-03';
205
206 -- View final data
207 SELECT * FROM Sales;

```

Product	Min_Quantity_Sold	Max_Quantity_Sold
Laptop	1	5

SaleDate	Min_Revenue	Max_Revenue
2025-02-01	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

SaleDate	Min_Revenue_Per_Product	Max_Revenue_Per_Product
2025-02-01	78000.00	150000.00
2025-02-02	88000.00	100000.00
2025-02-03	64000.00	90000.00

OneCompiler

PRICING

EDITOR

CHALLENGES

COMPANY & MORE

H

queries.sql

+

43dmtjh5d

AI

NEW

MYSQL

RUN

```

180 -- 6. Minimum and Maximum quantity sold per product
181 SELECT Product, MIN(Quantity) AS Min_Quantity_Per_Product, MAX(Quantity)
182 FROM Sales
183 GROUP BY Product;
184
185 -- 7. Minimum and Maximum revenue per day
186 SELECT SaleDate, MIN(Quantity * Price) AS Min_Revenue_Per_Day, MAX(Quantity * Price) AS Max_Revenue_Per_Day
187 FROM Sales
188 GROUP BY SaleDate;
189
190 -- 8. Minimum and Maximum revenue in the current month
191 SELECT MIN(Quantity * Price) AS Min_Revenue_This_Month, MAX(Quantity * Price) AS Max_Revenue_This_Month
192 FROM Sales
193 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
194 AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
195
196 -- 9. Minimum and Maximum price of products where more than 2 units were sold
197 SELECT MIN(Price) AS Min_Price_High_Quantity_Sales, MAX(Price) AS Max_Price_High_Quantity_Sales
198 FROM Sales
199 WHERE Quantity > 2;
200
201 -- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3, 2025)
202 SELECT MIN(Quantity * Price) AS Min_Revenue_After_Date, MAX(Quantity * Price) AS Max_Revenue_After_Date
203 FROM Sales
204 WHERE SaleDate > '2025-02-03';
205
206 -- View final data
207 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

SaleDate	Min_Revenue_This_Month	Max_Revenue_This_Month
2025-02-01	64000.00	150000.00

Product	Min_Price_High_Quantity_Sales	Max_Price_High_Quantity_Sales
Laptop	75000.00	78000.00
Mobile	20000.00	22000.00
Tablet	30000.00	32000.00

queries.sql43dmtjh5d

AI NEW MySQL RUN

180 -- 6. Minimum and Maximum quantity sold per product

181 SELECT Product, MIN(Quantity) AS Min\_Quantity\_Per\_Product, MAX(Quantity

182 FROM Sales

183 GROUP BY Product;

184

185 -- 7. Minimum and Maximum revenue per day

186 SELECT SaleDate, MIN(Quantity \* Price) AS Min\_Revenue\_Per\_Day, MAX(Quan

187 FROM Sales

188 GROUP BY SaleDate;

189

190 -- 8. Minimum and Maximum revenue in the current month

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

192 FROM Sales

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

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

195

196 -- 9. Minimum and Maximum price of products where more than 2 units wer

197 SELECT MIN(Price) AS Min\_Price\_High\_Quantity\_Sales, MAX(Price) AS Max\_P

198 FROM Sales

199 WHERE Quantity > 2;

200

201 -- 10. Minimum and Maximum revenue after a specific date (e.g., Feb 3,

202 SELECT MIN(Quantity \* Price) AS Min\_Revenue\_After\_Date, MAX(Quantity \*

203 FROM Sales

204 WHERE SaleDate > '2025-02-03';

205

206 -- View final data

207 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

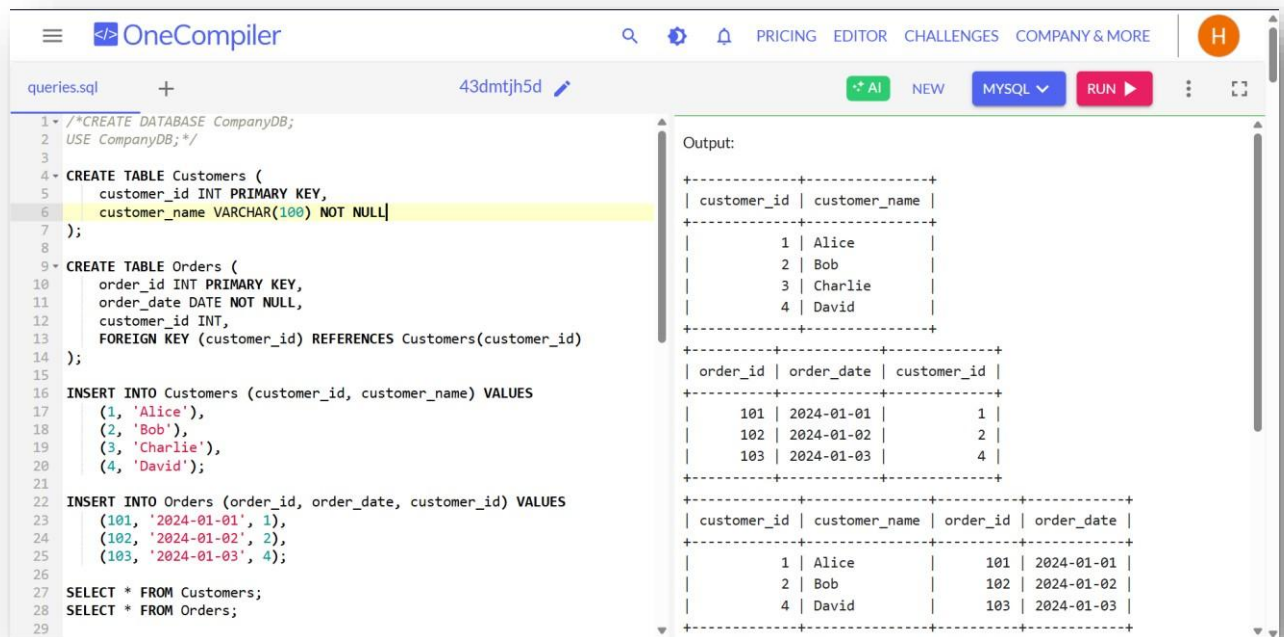
Orders o

ON

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



## Screenshot of Output:-



The screenshot shows the OneCompiler interface with a MySQL database. The left pane contains SQL queries, and the right pane shows the output of the executed queries.

**Queries:**

```
1 /*CREATE DATABASE CompanyDB;
2 USE CompanyDB;*/
3
4 CREATE TABLE Customers (
5     customer_id INT PRIMARY KEY,
6     customer_name VARCHAR(100) NOT NULL
7 );
8
9 CREATE TABLE Orders (
10     order_id INT PRIMARY KEY,
11     order_date DATE NOT NULL,
12     customer_id INT,
13     FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
14 );
15
16 INSERT INTO Customers (customer_id, customer_name) VALUES
17 (1, 'Alice'),
18 (2, 'Bob'),
19 (3, 'Charlie'),
20 (4, 'David');
21
22 INSERT INTO Orders (order_id, order_date, customer_id) VALUES
23 (101, '2024-01-01', 1),
24 (102, '2024-01-02', 2),
25 (103, '2024-01-03', 4);
26
27 SELECT * FROM Customers;
28 SELECT * FROM Orders;
```

**Output:**

customer\_id | customer\_name

1	Alice
2	Bob
3	Charlie
4	David

order\_id | order\_date | customer\_id

101	2024-01-01	1
102	2024-01-02	2
103	2024-01-03	4

customer\_id | customer\_name | order\_id | order\_date

1	Alice	101	2024-01-01
2	Bob	102	2024-01-02
4	David	103	2024-01-03



The screenshot shows the OneCompiler interface with a MySQL database. The left pane contains SQL queries, and the right pane shows the output of the executed queries.

**Queries:**

```
30 -- INNER JOIN: Customers who have placed orders
31 SELECT
32     c.customer_id,
33     c.customer_name,
34     o.order_id,
35     o.order_date
36 FROM
37     Customers c
38 INNER JOIN
39     Orders o
40 ON
41     c.customer_id = o.customer_id;
42
43 -- LEFT JOIN: ALL Customers with their Orders (if any)
44 SELECT
45     c.customer_id,
46     c.customer_name,
47     o.order_id,
48     o.order_date
49 FROM
50     Customers c
51 LEFT JOIN
52     Orders o
53 ON
54     c.customer_id = o.customer_id;
55
56 -- RIGHT JOIN: ALL Orders with Customer details
57 SELECT
58     c.customer_id
```

**Output:**

customer\_id | customer\_name | order\_id | order\_date

1	Alice	101	2024-01-01
2	Bob	102	2024-01-02
4	David	103	2024-01-03

customer\_id | customer\_name | order\_id | order\_date

1	Alice	101	2024-01-01
2	Bob	102	2024-01-02
3	Charlie	NULL	NULL
4	David	103	2024-01-03

customer\_id | customer\_name | order\_id | order\_date

1	Alice	101	2024-01-01
2	Bob	102	2024-01-02
4	David	103	2024-01-03

© 2006 The Authors  
Journal compilation © 2006 Blackwell Publishing Ltd

PRN:- 2124UCSM1027

Dept.: Cyber Security