

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:-

The screenshot shows the OneCompiler web interface. In the top navigation bar, there are links for AI, NEW, PRICING, EDITOR, CHALLENGES, COMPANY & MORE, and a user profile icon. Below the navigation, the code editor window is open with a file named 'queries.sql'. The code contains SQL commands to create a database, select from a table, insert data, and select again. The code is as follows:

```
1 /*create database company2;
2 use company2;*/
3 create table employees(
4 id int auto_increment primary key,
5 name varchar(50) not null,
6 position varchar(50) not null,
7 department varchar(50),
8 salary decimal(10,2),
9 join_date date
10 );
11 select * from employees;
12 insert into employees(name, position, department, salary, join_date)
13 values
14 ('Rushikesh','Student','Cyber',1000,'2025-01-15');
15 select * from employees;
```

On the right side of the interface, there are sections for 'STDIN' (Input for the program (Optional)) and 'Output'. The output section displays the results of the 'select' and 'insert' statements:

id	name	position	department	salary	join_date
1	Rushikesh	Student	Cyber	1000.00	2025-01-15

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:-

The screenshot shows the OneCompiler web interface. In the top navigation bar, there are links for PRICING, EDITOR, CHALLENGES, and COMPANY & MORE. On the right, there are buttons for AI, NEW, MYSQL (selected), RUN, and other options. The main area has tabs for queries.sql and 43dmtjh5d. The code editor contains the following MySQL script:

```
1 */*CREATE DATABASE Employee;
2 USE Employee;*/
3
4 CREATE TABLE Employee (
5     EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
6     Name VARCHAR(100) NOT NULL,
7     Salary DECIMAL(10,2) CHECK (Salary > 0),
8     JoiningDate DATE NOT NULL,
9     ActiveStatus BOOLEAN DEFAULT TRUE
10 );
11
12 SELECT * FROM Employee;
13
14 INSERT INTO Employee(Name, Salary, JoiningDate, ActiveStatus)
15 VALUES
16 ('Rushikesh', 100000.00, '2028-05-23', TRUE),
17 ('Ghanish', 50000.00, '2028-05-24', FALSE),
18 ('Mayank', 200000.00, '2028-05-25', TRUE),
19 ('Saif', 250000.00, '2028-05-26', FALSE);
20
21 SELECT * FROM Employee;
22
```

The output section shows the results of the last SELECT statement:

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Rushikesh	100000.00	2028-05-23	1
2	Ghanish	50000.00	2028-05-24	0
3	Mayank	200000.00	2028-05-25	1
4	Saif	250000.00	2028-05-26	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 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:-

The screenshot shows the OneCompiler web interface. In the top navigation bar, there are links for PRICING, EDITOR, CHALLENGES, COMPANY & MORE, and a user icon. Below the bar, there are buttons for AI, NEW, MySQL (selected), and RUN. The main area has tabs for 'queries.sql' and '+'. The code editor contains the following SQL queries:

```
1 /*CREATE DATABASE COMPANY;
2 USE COMPANY;*/
3 CREATE TABLE Employee(
4 EmployeeID INT PRIMARY KEY AUTO_INCREMENT,
5 Name VARCHAR(100) NOT NULL,
6 Salary DECIMAL(10,2) CHECK (Salary > 0),
7 JoiningDate DATE NOT NULL,
8 ActiveStatus BOOLEAN DEFAULT TRUE
9 );
10 select * from Employee;
11
12 INSERT INTO Employee(Name,Salary,JoiningDate,ActiveStatus)
13 VALUES
14 ('Rushikesh Patil',55000.00,'2023-06-15',TRUE),
15 ('Ghanish Patil',60000.00,'2022-06-15',TRUE),
16 ('mayank dhule',70000.00,'2021-06-15',FALSE),
17 ('Omkar Shelar',100000.00,'2020-06-15',TRUE);
18
19 select * from Employee;
20
21 SELECT EmployeeID, Name,Salary FROM Employee WHERE
22 ActiveStatus = TRUE;
23
24 select * from Employee;
25
26 UPDATE Employee SET Salary = Salary * 1.10 WHERE EmployeeID =
27 2;
28
```

The output section shows three tables of data. The first table lists all employees with their details. The second table shows employees where ActiveStatus is TRUE. The third table is identical to the first, likely a copy of the original data.

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	100000.00	2020-06-15	1

EmployeeID	Name	Salary
1	Rushikesh Patil	55000.00
2	Ghanish Patil	60000.00
4	Omkar Shelar	100000.00

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Rushikesh Patil	55000.00	2023-06-15	1
2	Ghanish Patil	60000.00	2022-06-15	1
4	Omkar Shelar	100000.00	2020-06-15	1

OneCompiler

queries.sql + 43dmtjh5d

PRICING EDITOR CHALLENGES COMPANY & MORE H

```
25 UPDATE Employee SET Salary = Salary * 1.10 WHERE EmployeeID =
26 2;
27
28 select * from Employee;
29
30
31 UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID =
32 4;
33
34 select * from Employee;
35
36 DELETE FROM Employee WHERE EmployeeID = 3;
37
38 select * from Employee;
39
40 SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
41
42 select * from Employee;
43
44 SELECT Name, Salary FROM Employee WHERE Salary > 60000;
45
46 select * from Employee;
47
48 SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS
49 LowestSalary FROM Employee;
50
51 select * from Employee;
52
53 SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
```

STDIN

Input for the program (Optional)

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	100000.00	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	100000.00	2020-06-15	1

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Rushikesh Patil	55000.00	2023-06-15	1

OneCompiler

queries.sql + 43dmtjh5d

PRICING EDITOR CHALLENGES COMPANY & MORE H

```
30
31 UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID =
32 4;
33
34 select * from Employee;
35
36 DELETE FROM Employee WHERE EmployeeID = 3;
37
38 select * from Employee;
39
40 SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
41
42 select * from Employee;
43
44 SELECT Name, Salary FROM Employee WHERE Salary > 60000;
45
46 select * from Employee;
47
48 SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS
49 LowestSalary FROM Employee;
50
51 select * from Employee;
52
53 SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
54
55 select * from Employee;
56
57
58
```

STDIN

Input for the program (Optional)

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	100000.00	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	100000.00	2020-06-15	0

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Rushikesh Patil	55000.00	2023-06-15	1

OneCompiler

queries.sql + 43dmtjh5d

```
30
31 UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID =
32 4;
33
34 select * from Employee;
35
36 DELETE FROM Employee WHERE EmployeeID = 3;
37
38 select * from Employee;
39
40 SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
41
42 select * from Employee;
43
44 SELECT Name, Salary FROM Employee WHERE Salary > 60000;
45
46 select * from Employee;
47
48 SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS
49 LowestSalary FROM Employee;
50
51 select * from Employee;
52
53 SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
54
55 select * from Employee;
56
57
```

STDIN

Input for the program (Optional)

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	100000.00	2020-06-15	0

Name	Salary
Ghanish Patil	66000.00
Omkar Shelar	100000.00

OneCompiler

queries.sql + 43dmtjh5d

```
30
31 UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID =
32 4;
33
34 select * from Employee;
35
36 DELETE FROM Employee WHERE EmployeeID = 3;
37
38 select * from Employee;
39
40 SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
41
42 select * from Employee;
43
44 SELECT Name, Salary FROM Employee WHERE Salary > 60000;
45
46 select * from Employee;
47
48 SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS
49 LowestSalary FROM Employee;
50
51 select * from Employee;
52
53 SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
54
55 select * from Employee;
56
57
```

STDIN

Input for the program (Optional)

Name	Salary
Ghanish Patil	66000.00
Omkar Shelar	100000.00

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	100000.00	2020-06-15	0

HighestSalary	LowestSalary
100000.00	55000.00

queries.sql

+

43dmtjh5d

+ AI NEW

MySQL

RUN ▶

```
30
31 UPDATE Employee SET ActiveStatus = FALSE WHERE EmployeeID =
32 4;
33
34 select * from Employee;
35
36 DELETE FROM Employee WHERE EmployeeID = 3;
37
38 select * from Employee;
39
40 SELECT * FROM Employee WHERE YEAR(JoiningDate) = 2023;
41
42 select * from Employee;
43
44 SELECT Name, Salary FROM Employee WHERE Salary > 60000;
45
46 select * from Employee;
47
48 SELECT MAX(Salary) AS HighestSalary, MIN(Salary) AS
49 LowestSalary FROM Employee;
50
51 select * from Employee;
52
53 SELECT * FROM Employee ORDER BY Salary DESC LIMIT 3;
54
55 select * from Employee;
56
57
58
```

STDIN

Input for the program (Optional)

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	100000.00	2020-06-15	0

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
4	Omkar Shelar	100000.00	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	100000.00	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:-

OneCompiler

queries.sql 43dmtjh5d

STDIN
Input for the program (Optional)

Output:

```
1 CREATE TABLE Department (
2 DeptID INT PRIMARY KEY,
3 DeptName VARCHAR(50)
4 );
5
6 CREATE TABLE Employee(
7 EmpID INT PRIMARY KEY,
8 Name VARCHAR(100) NOT NULL,
9 Email VARCHAR(100),
10 Salary DECIMAL(10,2) CHECK (Salary > 0),
11 DeptID INT REFERENCES Department(DeptID)
12 );
13
14 -- Insert Valid Data
15 INSERT INTO Department (DeptID, DeptName) VALUES (1,'HR');
16 INSERT INTO Department (DeptID, DeptName) VALUES (2,'IT');
17
18 INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (101,
19 'Alice', 'alice@example.com', 50000.00, 1);
20 INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (102,
21 'Bob', 'bob@example.com', 60000.00, 2);
22
23 select * from Employee;
24
25 /*INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (101,
26 'Charlie', 'charlie@example.com', 55000.00, 1);*/
27
28 select * from Employee;
```

OneCompiler

queries.sql 43dmtjh5d

STDIN
Input for the program (Optional)

Output:

```
13
14 -- Insert Valid Data
15 INSERT INTO Department (DeptID, DeptName) VALUES (1,'HR');
16 INSERT INTO Department (DeptID, DeptName) VALUES (2,'IT');
17
18 INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (101,
19 'Alice', 'alice@example.com', 50000.00, 1);
20 INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (102,
21 'Bob', 'bob@example.com', 60000.00, 2);
22
23 select * from Employee;
24
25 /*INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (101,
26 'Charlie', 'charlie@example.com', 55000.00, 1);*/
27
28 select * from Employee;
29
30 -- Duplicate Unique Email
31 INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (103,
32 'David', 'alice@example.com', 45000.00, 2);
33
34 select * from Employee;
35
36 -- Salary Check Constraint Violation
37 /*INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (105,
38 'Frank', 'frank@example.com', -40000.00, 1);*/
39
40 select * from Employee;
```

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:-

queries.sql

43dmtjh5d

STDRIN

Input for the program (Optional)

Output:

```
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, IsActive)
VALUES (1, 'John', 'Doe', 'john.doe@example.com', 1234567890, 1)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, IsActive)
VALUES (2, 'Jane', 'Smith', 'jane.smith@example.com', 0987654321, 1)
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, IsActive)
VALUES (3, NULL, 'Taylor', 'taylor@example.com', 5551234567, 1)
SELECT * FROM Customer;

-- Invalid data for CHECK constraint (Age Less than 18)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, IsActive)
VALUES (4, 'Alice', 'Johnson', 'alice.johnson@example.com', 1234567890, 1)
SELECT * FROM Customer;

-- Invalid data for UNIQUE constraint (Duplicate Email)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, IsActive)
VALUES (5, 'Bob', 'Brown', 'john.doe@example.com', 7771234567, 1)
SELECT * FROM Customer;
```

queries.sql

43dmtjh5d

STDRIN

Input for the program (Optional)

Output:

```
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, IsActive)
VALUES (1, 'John', 'Doe', 'john.doe@example.com', 1234567890, 1)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, IsActive)
VALUES (2, 'Jane', 'Smith', 'jane.smith@example.com', 0987654321, 1)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, IsActive)
VALUES (3, NULL, 'Taylor', 'taylor@example.com', 5551234567, 1)
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, IsActive)
VALUES (4, NULL, 'Taylor', 'taylor@example.com', 5551234567, 1)
SELECT * FROM Customer;

-- Invalid data for CHECK constraint (Age Less than 18)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, IsActive)
VALUES (5, 'Alice', 'Johnson', 'alice.johnson@example.com', 1234567890, 1)
SELECT * FROM Customer;

-- Invalid data for UNIQUE constraint (Duplicate Email)
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, IsActive)
VALUES (6, 'Bob', 'Brown', 'john.doe@example.com', 7771234567, 1)
SELECT * FROM Customer;
```

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:-

OneCompiler

queries.sql 43dmtjh5d

PRICING EDITOR CHALLENGES COMPANY & MORE

AI NEW MYSQL RUN

STDIN

Input for the program (Optional)

Output:

```
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 INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department,
11 Salary)
12 VALUES (1, 'John', 'Doe', 28, 'HR', 50000.00);
13 INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department,
14 Salary)
15 VALUES (2, 'Jane', 'Smith', 35, 'IT', 65000.00);
16 INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department,
17 Salary)
18 VALUES (3, 'Michael', 'Johnson', 40, 'Finance', 75000.00);
19
20 select * from Employees;
21
22 -- 1. Update a single column (e.g., update salary for EmployeeID 2)
23 UPDATE Employees
24 SET Salary = 70000.00
25 WHERE EmployeeID = 2;
26
27 select * from Employees;
28
29
```

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

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

OneCompiler

queries.sql 43dmtjh5d

PRICING EDITOR CHALLENGES COMPANY & MORE

AI NEW MYSQL RUN

STDIN

Input for the program (Optional)

Output:

```
28
29 -- 2. Update multiple columns for a specific row (e.g., update name and
30
31 /*SET FirstName = 'Janet', LastName = 'Williams', Salary = 75000.00
32 WHERE EmployeeID = 2;*/
33
34 select * from Employees;
35
36 -- 3. Update entire tuple (all columns for EmployeeID 3)
37 UPDATE Employees
38 SET FirstName = 'Michael', LastName = 'Brown', Age = 45, Department =
39 'Management', Salary = 80000.00
40 WHERE EmployeeID = 3;
41
42 select * from Employees;
43
44 -- 4. Update with a condition (e.g., increase salary by 10% for all empl
45 /*SET Salary = Salary * 1.10
46 WHERE Department = 'HR';*/
47
48 select * from Employees;
49
50 -- 5. Update with a subquery (e.g., increase salary for Employee with hi
51 /*UPDATE Employees
52 SET Salary = Salary + 5000
53 WHERE Salary = (SELECT MAX(Salary) FROM Employees);*/
54
55 select * from Employees;
56
```

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 |

| 3 | Michael | Brown | 45 | Management | 80000.00 |

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

OneCompiler

queries.sql + 43dmtjh5d

STDIN

Input for the program (Optional)

```
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;
```

OneCompiler

queries.sql + 43dmtjh5d

STDIN

Input for the program (Optional)

```
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;
```

OneCompiler

queries.sql 43dmtjh5d

PRICING EDITOR CHALLENGES COMPANY & MORE

STDIN

Input for the program (Optional)

```
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
3	Michael	Brown	45	Management	80000.00
2	Jane	Smith	35	IT	75600.00
3	Michael	Brown	45	Management	80000.00
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;
```

-- 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:-

queries.sql 43dmtjh5d

Output:

```
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

```

OneCompiler

queries.sql

43dmtjh5d

PRICING EDITOR CHALLENGES COMPANY & MORE

AI NEW MYSQL RUN

```
27 -- 3. Average price of products sold
28 SELECT AVG(Price) AS Average_Price FROM Sales;
29
30 -- 4. Minimum and Maximum price of a product sold
31 SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;
32
33 -- 5. Count the number of distinct products sold
34 SELECT COUNT(DISTINCT Product) AS Unique_Products FROM Sales;
35
36 -- 6. Count the number of sales per product
37 SELECT Product, COUNT(*) AS Sales_Count
38 FROM Sales
39 GROUP BY Product;
40
41 -- 7. Count the number of sales per day
42 SELECT SaleDate, COUNT(*) AS Sales_Per_Day
43 FROM Sales
44 GROUP BY SaleDate;
45
46 -- 8. Count the number of sales where more than 2 units were sold
47 SELECT COUNT(*) AS High_Quantity_Sales
48 FROM Sales
49 WHERE Quantity > 2;
50
51 -- 9. Count the number of sales in the current month
52 SELECT COUNT(*) AS Sales_This_Month
53 FROM Sales
54 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE);
55 AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
56
57 -- 10. Count the number of sales transactions where total sale value was
58 SELECT COUNT(*) AS High_Value_Sales
59 FROM Sales
60 WHERE (Quantity * Price) > 50000;
61
62 -- 11. Count the number of sales records for each product where total sa
63 SELECT Product, COUNT(*) AS High_Value_Transactions
64 FROM Sales
65 WHERE (Quantity * Price) > 40000
66 GROUP BY Product;
67
68 -- 12. Count the number of sales made after a specific date (e.g., Feb 3
69 SELECT COUNT(*) AS Sales_After_Date
70 FROM Sales
71 WHERE SaleDate > '2025-02-03';
72
73 -- SUM Queries
74 -- 1. Sum of total revenue generated
75 SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;
76
77 -- 2. Sum of total quantity of products sold
78 SELECT SUM(Quantity) AS Total_Quantity_Sold FROM Sales;
```

Average_Price
42833.33333

Min_Price Max_Price
20000.00 78000.00

Unique_Products
3

Product Sales_Count
Laptop 2
Mobile 2
Tablet 2

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

OneCompiler

queries.sql

43dmtjh5d

PRICING EDITOR CHALLENGES COMPANY & MORE

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;
```

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

queries.sql

+

43dmtjh5d

```

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
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

queries.sql

+

43dmtjh5d

```

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 transaction
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

queries.sql + 43dmtjh5d

128 -- 5. Average revenue per product
129 SELECT Product, AVG(Quantity * Price) AS Average_Revenue_Per_Product
130 FROM Sales
131 GROUP BY Product;
132
133 -- 6. Average quantity sold per product
134 SELECT Product, AVG(Quantity) AS Average_Quantity_Per_Product
135 FROM Sales
136 GROUP BY Product;
137
138 -- 7. Average revenue per day
139 SELECT SaleDate, AVG(Quantity * Price) AS Average_Revenue_Per_Day
140 FROM Sales
141 GROUP BY SaleDate;
142
143 -- 8. Average revenue in the current month
144 SELECT AVG(Quantity * Price) AS Average_Revenue_This_Month
145 FROM Sales
146 WHERE MONTH(SaleDate) = MONTH(CURRENT_DATE)
147 AND YEAR(SaleDate) = YEAR(CURRENT_DATE);
148
149 -- 9. Average price of products where more than 2 units were sold
150 SELECT AVG(Price) AS Avg_Price_High_Quantity_Sales
151 FROM Sales
152 WHERE Quantity > 2;
153
154 -- 10. Average revenue after a specific date (e.g., Feb 3, 2025)
155 SELECT AVG(Quantity * Price) AS Average_Revenue_After_Date
156 FROM Sales
157 WHERE SaleDate > '2025-02-03';
158
159 -- MIN/MAX Queries
160 -- 1. Minimum and Maximum price of a product sold
161 SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;
162
163 -- 2. Minimum and Maximum quantity of products sold in a single transaction
164 SELECT MIN(Quantity) AS Min_Quantity_Sold, MAX(Quantity) AS Max_Quantit
165
166 -- 3. Minimum and Maximum revenue generated from a single transaction
167 SELECT MIN(Quantity * Price) AS Min_Revenue, MAX(Quantity * Price) AS M
168
169 -- 4. Minimum and Maximum price per product
170 SELECT Product, MIN(Price) AS Min_Price_Per_Product, MAX(Price) AS Max_
171 FROM Sales
172 GROUP BY Product;
173
174 -- 5. Minimum and Maximum revenue per product
175 SELECT Product, MIN(Quantity * Price) AS Min_Revenue_Per_Product, MAX(Q
176 FROM Sales
177 GROUP BY Product;
178
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

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

queries.sql + 43dmtjh5d

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

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

queries.sql + 43dmtjh5d

179
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.

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

queries.sql + 43dmtjh5d

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;

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 | 1 |

OneCompiler

queries.sql + 43dmtjh5d

PRICING EDITOR CHALLENGES COMPANY & MORE H

```
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;
```

AI NEW MYSQL RUN

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

queries.sql + 43dmtjh5d

PRICING EDITOR CHALLENGES COMPANY & MORE H

```
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;
```

AI NEW MYSQL RUN

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

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(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:-

OneCompiler

queries.sql 43dmtjh5d

Output:

```
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;
```

OneCompiler

queries.sql 43dmtjh5d

Output:

```
-- 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,
```

OneCompiler

queries.sql 43dmtjh5d

AI NEW MYSQL RUN

```
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,
59   c.customer_name,
60   o.order_id,
61   o.order_date
62 FROM
63   Customers c
64 RIGHT JOIN
65   Orders o
66 ON
67   c.customer_id = o.customer_id;
```

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

Nmae: Vishnu Deepak Varma

PRN: 2124UCSM1067

Dept: Cyber security