

SANJIVANI UNIVERSITY

Department of Cyber Security



ACADEMIC YEAR: 2024- 2025

24UETBS103 – Database Management System Lab

Name: Ghanish Patil

Roll No: 2124UCSM1047

PRN NO: 2124UCSM1047

SEM/YEAR : SEM-2/ First Year

Exp No	List of experiments	Page No	Signature
1	Install and set up MySQL. Create a database and a table to store employee details. Perform basic operations like INSERT & DELETE	5	
2	Create a table for storing student information. Insert sample data and perform basic operations: INSERT, UPDATE, DELETE, and SELECT	7	
3	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.	9	
4	Create a table to store employee information with constraints like Primary Key, ForeignKey, and Unique. Insert valid and invalid data to test the constraints.	11	
5	Create a table for Customer details with various integrity constraints like NOTNULL, CHECK, and DEFAULT. Insert valid and invalid data to test these constraints and ensure data integrity.	13	
6	Use DDL commands to create tables and DML commands to insert, update, and delete data. Write SELECT queries to retrieve and verify data changes.	15	
7	Create a Sales table and use aggregate functions like COUNT, SUM, AVG, MIN, and MAX to summarize sales data and calculate statistics.	18	
8	Given Customers and Orders tables, write SQL queries to perform INNERJOIN, LEFTJOIN, and RIGHT JOIN to retrieve combined data for customer orders.	27	

Practical No: 1

Aim: Install and set up MySQL. Create a database and a table to store employee details. Perform basic operations like INSERT & DELETE

Code:

```
/* create database company;
```

```
use company;*/
```

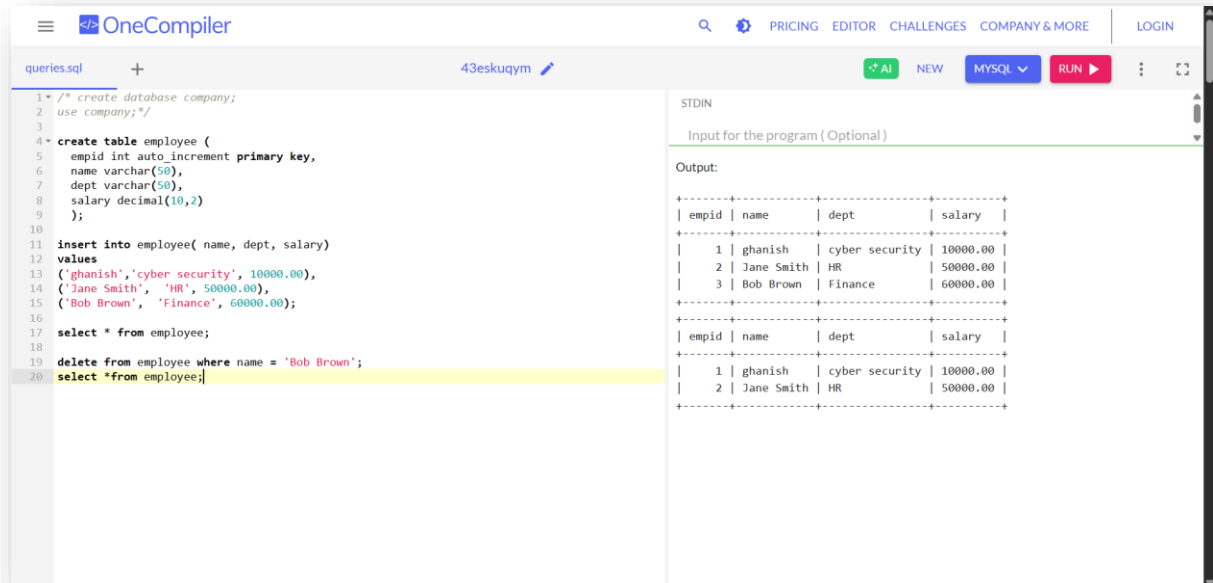
```
create table employee (  
    empid int auto_increment primary key,  
    name varchar(50),  
    dept varchar(50),  
    salary decimal(10,2)  
);
```

```
insert into employee( name, dept, salary)  
values  
('ghanish','cyber security', 10000.00),  
('Jane Smith', 'HR', 50000.00),  
('Bob Brown', 'Finance', 60000.00);
```

```
select * from employee;
```

```
delete from employee where name = 'Bob Brown';  
select *from employee;
```

Screenshot of Output:-



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

```
1 */* create database company;
2 use company;*/
3
4 create table employee (
5     empid int auto_increment primary key,
6     name varchar(50),
7     dept varchar(50),
8     salary decimal(10,2)
9 );
10
11 insert into employee( name, dept, salary)
12 values
13 ('ghanish','cyber security', 10000.00),
14 ('Jane Smith', 'HR', 50000.00),
15 ('Bob Brown', 'Finance', 60000.00);
16
17 select * from employee;
18
19 delete from employee where name = 'Bob Brown';
20 select *from employee;
```

The right pane shows the output of the queries. The first query (line 17) returns three rows. The second query (line 19) deletes the row for 'Bob Brown'. The third query (line 20) returns two rows.

STDIN

Input for the program (Optional)

Output:

empid	name	dept	salary
1	ghanish	cyber security	10000.00
2	Jane Smith	HR	50000.00
3	Bob Brown	Finance	60000.00

empid	name	dept	salary
1	ghanish	cyber security	10000.00
2	Jane Smith	HR	50000.00

Practical No: 2

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

Code:

```
create table students(  
    studentid int auto_increment primary key,  
    name varchar(50),  
    dept varchar (50),  
    year int  
);
```

```
select * from students;
```

```
insert into students (name, dept, year)  
values  
('ghanish', 'cyber security', 2),  
('Bob', 'IT', 3),  
('charlie', 'HR', 4);
```

```
select * from students;
```

```
update students  
set year = 4  
where year = 3;
```

```
select * from students;
```

```
delete from students  
where name = 'charlie';
```

```
select * from students;
```

select * from students where dept = 'cyber security';

Screenshot of output:

The screenshot shows the OneCompiler web interface for MySQL. The left pane contains a series of SQL queries, and the right pane shows the output of the final query. The queries performed are:

```
1- create table students(  
2  studentid int auto_increment primary key,  
3  name varchar(50),  
4  dept varchar(50),  
5  year int  
6 );  
7  
8 select * from students;  
9  
10 insert into students (name, dept, year)  
11 values  
12 ('ghanish', 'cyber security', 2),  
13 ('Bob', 'IT', 3),  
14 ('charlie', 'HR', 4);  
15  
16 select * from students;  
17  
18 update students  
19 set year = 4  
20 where year = 3;  
21  
22 select * from students;  
23  
24 delete from students  
25 where name = 'charlie';  
26  
27 select * from students;  
28  
29 select * from students where dept = 'cyber security';  
30  
31
```

The output of the final query is displayed in the right pane, showing a table with 4 columns: studentid, name, dept, and year. The data rows are:

studentid	name	dept	year
1	ghanish	cyber security	2
2	Bob	IT	4

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 TABLE employees (  
EmployeeID INT PRIMARY KEY AUTO_INCREMENT,  
Name VARCHAR(100),  
Salary DECIMAL(10,2),  
JoiningDate DATE,  
ActiveStatus BOOLEAN  
);  
  
INSERT INTO employees (Name, Salary, JoiningDate, ActiveStatus)  
VALUES  
( 'Ghanish ', 55000.50, '2022-01-15', TRUE),  
( 'Charlie', 60000.00, '2021-08-01', TRUE),  
( 'Bravo', 45000.75, '2020-05-10', FALSE),  
( 'Neha ', 70000.00, '2023-04-20', TRUE);  
  
select * from employees;  
  
update employees  
set Salary = 700000  
where EmployeeID = 1;  
  
select * from employees;  
  
update employees  
set ActiveStatus = FALSE  
where EmployeeID = 4;  
  
select * from employees;  
  
select * from employees where ActiveStatus = TRUE;
```

delete from employees

where name = 'Charlie';

select * from employees;

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 AUTO_INCREMENT,  
3 Name VARCHAR(100),  
4 Salary DECIMAL(10,2),  
5 JoiningDate DATE,  
6 ActiveStatus BOOLEAN  
7 );  
8  
9 INSERT INTO employees (Name, Salary, JoiningDate, ActiveStatus)  
10 VALUES  
11 ('Ghanish', 55000.50, '2022-01-15', TRUE),  
12 ('Charlie', 60000.00, '2021-08-01', TRUE),  
13 ('Bravo', 45000.75, '2020-05-10', FALSE),  
14 ('Neha', 70000.00, '2023-04-20', TRUE);  
15  
16 select * from employees;  
17  
18 update employees  
19 set Salary = 700000  
20 where EmployeeID = 1;  
21  
22 select * from employees;  
23  
24 update employees  
25 set ActiveStatus = FALSE  
26 where EmployeeID = 4;  
27 select * from employees;  
28  
29 select * from employees where ActiveStatus = TRUE;  
30  
31 delete from employees  
32 where name = 'Charlie';  
33  
34 select * from employees;
```

The right pane shows the output of the queries. The first query (line 16) returns 4 rows. The second query (line 22) returns 4 rows. The third query (line 27) returns 4 rows. The fourth query (line 29) returns 3 rows. The fifth query (line 34) returns 4 rows.

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Ghanish	55000.50	2022-01-15	1
2	Charlie	60000.00	2021-08-01	1
3	Bravo	45000.75	2020-05-10	0
4	Neha	70000.00	2023-04-20	1

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Ghanish	700000.00	2022-01-15	1
2	Charlie	60000.00	2021-08-01	1
3	Bravo	45000.75	2020-05-10	0
4	Neha	70000.00	2023-04-20	1

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Ghanish	700000.00	2022-01-15	1
2	Charlie	60000.00	2021-08-01	1
3	Bravo	45000.75	2020-05-10	0
4	Neha	70000.00	2023-04-20	0

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Ghanish	700000.00	2022-01-15	1
2	Charlie	60000.00	2021-08-01	1

EmployeeID	Name	Salary	JoiningDate	ActiveStatus
1	Ghanish	700000.00	2022-01-15	1
3	Bravo	45000.75	2020-05-10	0
4	Neha	70000.00	2023-04-20	0

Practical No: 4

Aim: Create a table to store employee information with constraints like Primary Key, ForeignKey, and Unique. Insert valid and invalid data to test the constraints.

Code:

```
CREATE TABLE departments (  
    DepartmentID INT PRIMARY KEY,  
    DepartmentName VARCHAR(50) UNIQUE  
);  
CREATE TABLE employees (  
    EmployeeID INT PRIMARY KEY,  
    Name VARCHAR(100) NOT NULL,  
    Email VARCHAR(100) UNIQUE,  
    Salary DECIMAL(10,2),  
    DepartmentID INT,  
    FOREIGN KEY (DepartmentID) REFERENCES departments(DepartmentID)  
);  
/* insert valid data */  
INSERT INTO departments (DepartmentID, DepartmentName)  
VALUES  
(1, 'HR'),  
(2, 'IT'),  
(3, 'Finance');  
  
INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
VALUES  
(101, 'Anjali Verma', 'anjali@company.com', 50000.00, 2),  
(102, 'Rohit Mehra', 'rohit@company.com', 60000.00, 1);  
  
select * from departments;  
select * from employees;  
  
/* insert invalid data */  
INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
VALUES  
(103, 'Simran Kaur', 'anjali@company.com', 55000.00, 2);  
  
INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
VALUES  
(104, 'Amit Roy', 'amit@company.com', 58000.00, 5);  
  
INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
VALUES  
(101, 'Neha Sharma', 'neha@company.com', 52000.00, 1);  
  
INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
VALUES  
(101, 'Neha Sharma', 'neha@company.com', 52000.00, 1);
```

Screenshot of output:

The screenshot shows the OneCompiler MySQL interface. The left pane contains SQL code for creating tables and inserting data. The right pane shows the output of the queries, including two tables and an error message.

```
1 CREATE TABLE departments (  
2   DepartmentID INT PRIMARY KEY,  
3   DepartmentName VARCHAR(50) UNIQUE  
4 );  
5 CREATE TABLE employees (  
6   EmployeeID INT PRIMARY KEY,  
7   Name VARCHAR(100) NOT NULL,  
8   Email VARCHAR(100) UNIQUE,  
9   Salary DECIMAL(10,2),  
10  DepartmentID INT,  
11  FOREIGN KEY (DepartmentID) REFERENCES departments(DepartmentID)  
12 );  
13 /* Insert valid data */  
14 INSERT INTO departments (DepartmentID, DepartmentName)  
15 VALUES  
16 (1, 'HR'),  
17 (2, 'IT'),  
18 (3, 'Finance');  
19  
20 INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
21 VALUES  
22 (101, 'Anjali Verma', 'anjali@company.com', 50000.00, 2),  
23 (102, 'Rohit Mehra', 'rohit@company.com', 60000.00, 1);  
24  
25 select * from departments;  
26 select * from employees;  
27  
28 /* Insert invalid data */  
29 INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
30 VALUES  
31 (103, 'Saran Kaun', 'anjali@company.com', 55000.00, 2);  
32  
33 INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
34 VALUES  
35 (104, 'Amit Roy', 'amit@company.com', 50000.00, 5);  
36  
37 INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
38 VALUES  
39 (105, 'Neha Sharma', 'neha@company.com', 52000.00, 1);  
40  
41 INSERT INTO employees (EmployeeID, Name, Email, Salary, DepartmentID)  
42 VALUES  
43 (101, 'Neha Sharma', 'neha@company.com', 52000.00, 1);  
44  
45
```

Output:

DepartmentID	DepartmentName
3	Finance
1	HR
2	IT

EmployeeID	Name	Email	Salary	DepartmentID
101	Anjali Verma	anjali@company.com	50000.00	2
102	Rohit Mehra	rohit@company.com	60000.00	1

ERROR 1062 (23000) at line 29: Duplicate entry 'anjali@company.com' for key 'employees.Email'

Practical No: 5

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) NOT NULL,  
    LastName VARCHAR(100) NOT NULL,  
    Email VARCHAR(100) UNIQUE,  
    Phone VARCHAR(15),  
    Age INT CHECK (Age >= 18),  
    IsActive BOOLEAN DEFAULT TRUE  
);  
  
INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age, IsActive)  
VALUES  
(1, 'John', 'Doe', 'john.doe@example.com', '1234567890', 25, TRUE),  
(2, 'Jane', 'Smith', 'jane.smith@example.com', '0987654321', 30, false);  
  
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);  
  
-- 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);
```

-- 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);

Screenshots of Ouput:

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

```
1 CREATE TABLE Customer (  
2   CustomerID INT PRIMARY KEY,  
3   FirstName VARCHAR(100) NOT NULL,  
4   LastName VARCHAR(100) NOT NULL,  
5   Email VARCHAR(100) UNIQUE,  
6   Phone VARCHAR(15),  
7   Age INT CHECK (Age >= 18),  
8   IsActive BOOLEAN DEFAULT TRUE  
9 );  
10  
11 INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age, IsActive)  
12 VALUES  
13 (1, 'John', 'Doe', 'john.doe@example.com', '1234567890', 25, TRUE),  
14 (2, 'Jane', 'Smith', 'jane.smith@example.com', '0987654321', 30, false);  
15  
16 select * from Customer;  
17  
18 -- Insert Invalid Data to Test Constraints  
19  
20 -- Invalid data for NOT NULL constraint (FirstName is NULL)  
21 INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)  
22 VALUES (3, NULL, 'Taylor', 'taylor@example.com', '5551234567', 20);  
23  
24 -- Invalid data for CHECK constraint (Age Less than 18)  
25 INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)  
26 VALUES (4, 'Alice', 'Johnson', 'alice.johnson@example.com', '6669876543', 16);  
27  
28 -- Invalid data for UNIQUE constraint (Duplicate Email)  
29 INSERT INTO Customer (CustomerID, FirstName, LastName, Email, Phone, Age)  
30 VALUES (5, 'Bob', 'Brown', 'john.doe@example.com', '7771234567', 28);  
31  
32
```

The right pane shows the output of the queries. The first query (SELECT) returns the following table:

Customer-ID	FirstName	LastName	Email	Phone	Age	IsActive
1	John	Doe	john.doe@example.com	1234567890	25	1
2	Jane	Smith	jane.smith@example.com	0987654321	30	0

The second query (INSERT) results in an error:

ERROR 1048 (23000) at line 21: Column 'FirstName' cannot be null

Practical No: 6

```
CREATE TABLE Employees (
```

```
    EmployeeID INT PRIMARY KEY,
```

```
    FirstName VARCHAR(50),
```

```
    LastName VARCHAR(50),
```

```
    Age INT,
```

```
    Department VARCHAR(50),
```

```
    Salary DECIMAL(10, 2)
```

```
);
```

```
-- (DML Command)
```

```
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);
```

```
-- Updates (DML Commands)
```

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

```
UPDATE Employees
```

```
SET Salary = 70000.00
```

```
WHERE EmployeeID = 2;
```

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

```
UPDATE Employees
```

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

```
WHERE EmployeeID = 2;
```

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

UPDATE Employees

SET FirstName = 'Michael', LastName = 'Brown', Age = 45, Department = 'Management', Salary = 80000.00

WHERE EmployeeID = 3;

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

UPDATE Employees

SET Salary = Salary * 1.10

WHERE Department = 'HR';

UPDATE Employees

SET Salary = CASE

WHEN Department = 'HR' THEN Salary * 1.05

WHEN Department = 'IT' THEN Salary * 1.08

WHEN Department = 'Finance' THEN Salary * 1.10

ELSE Salary

END;

-- Delete Data from the Table (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;

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:

Screenshots of Output:

The screenshot shows the OneCompiler SQL editor interface. The left pane contains a MySQL script with the following content:

```
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 -- (DML Command)  
10 INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)  
11 VALUES (1, 'John', 'Doe', 25, 'HR', 50000.00);  
12  
13 INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)  
14 VALUES (2, 'Jane', 'Smith', 35, 'IT', 65000.00);  
15  
16 INSERT INTO Employees (EmployeeID, FirstName, LastName, Age, Department, Salary)  
17 VALUES (3, 'Michael', 'Johnson', 40, 'Finance', 75000.00);  
18  
19 -- Updates (DML Commands)  
20  
21 -- 1. Update a single column (e.g., update salary for EmployeeID 2)  
22 UPDATE Employees  
23 SET Salary = 70000.00  
24 WHERE EmployeeID = 2;  
25  
26 -- 2. Update multiple columns for a specific row (e.g., update name and salary for EmployeeID 2)  
27 UPDATE Employees  
28 SET FirstName = 'Janet', LastName = 'Williams', Salary = 75000.00  
29 WHERE EmployeeID = 2;  
30  
31 -- 3. Update entire tuple (all columns for EmployeeID 3)  
32 UPDATE Employees  
33 SET FirstName = 'Michael', LastName = 'Brown', Age = 45, Department = 'Management', Salary = 80000.00  
34 WHERE EmployeeID = 3;  
35  
36 -- 4. Update with a condition (e.g., increase salary by 10% for all employees in HR)  
37 UPDATE Employees  
38 SET Salary = Salary * 1.10  
39 WHERE Department = 'HR';  
40  
41  
42 UPDATE Employees  
43 SET Salary = CASE  
44 WHEN Department = 'HR' THEN Salary * 1.05  
45 WHEN Department = 'IT' THEN Salary * 1.08  
46 WHEN Department = 'Finance' THEN Salary * 1.10  
47 ELSE Salary  
48 END;  
49  
50  
51 -- Delete Data from the Table (DML Command)  
52 DELETE FROM Employees  
53 WHERE EmployeeID = 1;  
54  
55 -- Select and Verify Data (SELECT Query)
```

The right pane shows the output of the script, which is a table with 6 columns: EmployeeID, FirstName, LastName, Age, Department, and Salary. The output is as follows:

EmployeeID	FirstName	LastName	Age	Department	Salary
2	Janet	Williams	35	IT	75000.00
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');  
  
-- Count the number of sales records  
SELECT COUNT(*) AS Total_Sales FROM Sales;  
  
-- Sum of total revenue generated  
SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;  
  
-- Average price of products sold  
SELECT AVG(Price) AS Average_Price FROM Sales;  
  
-- Minimum and Maximum price of a product sold  
SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;
```


-- COUNT

-- 1. Count the total number of sales records

```
SELECT COUNT(*) AS Total_Sales FROM Sales;
```

-- 2. Count the number of distinct products sold

```
SELECT COUNT(DISTINCT Product) AS Unique_Products FROM Sales;
```

-- 3. Count the number of sales per product

```
SELECT Product, COUNT(*) AS Sales_Count  
FROM Sales  
GROUP BY Product;
```

-- 4. Count the number of sales per day

```
SELECT SaleDate, COUNT(*) AS Sales_Per_Day  
FROM Sales  
GROUP BY SaleDate;
```

-- 5. Count the number of sales where more than 2 units were sold

```
SELECT COUNT(*) AS High_Quantity_Sales  
FROM Sales  
WHERE Quantity > 2;
```

-- 6. 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);
```

-- 7. 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;
```

-- 8. 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;
```

-- 9. 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

-- 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

-- 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-

-- 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';

Screenshot of output:

The screenshot shows the OneCompiler MySQL query editor. The SQL script on the left includes a table creation for 'Sales', an insert of 6 records, and several aggregate queries. The output on the right shows the results of these queries in a tabular format.

```
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) VALUES  
10 ('Laptop', 2, 75000.00, '2025-02-01'),  
11 ('Mobile', 5, 20000.00, '2025-02-02'),  
12 ('Tablet', 3, 30000.00, '2025-02-03'),  
13 ('Laptop', 1, 70000.00, '2025-02-04'),  
14 ('Mobile', 4, 22000.00, '2025-02-05'),  
15 ('Tablet', 2, 32000.00, '2025-02-06');  
16  
17 -- Count the number of sales records  
18 SELECT COUNT(*) AS Total_Sales FROM Sales;  
19  
20 -- Sum of total revenue generated  
21 SELECT SUM(Quantity * Price) AS Total_Revenue FROM Sales;  
22  
23 -- Average price of products sold  
24 SELECT AVG(Price) AS Average_Price FROM Sales;  
25  
26 -- Minimum and Maximum price of a product sold  
27 SELECT MIN(Price) AS Min_Price, MAX(Price) AS Max_Price FROM Sales;  
28  
29 -- COUNT  
30 -- 1. Count the total number of sales records  
31 SELECT COUNT(*) AS Total_Sales FROM Sales;  
32  
33 -- 2. Count the number of distinct products sold  
34 SELECT COUNT(DISTINCT Product) AS Unique_Products FROM Sales;  
35  
36 -- 3. Count the number of sales per product  
37 SELECT Product, COUNT(*) AS Sales_Count  
38 FROM Sales  
39 GROUP BY Product;  
40  
41 -- 4. Count the number of sales per day  
42 SELECT SaleDate, COUNT(*) AS Sales_Per_Day  
43 FROM Sales  
44 GROUP BY SaleDate;  
45  
46 -- 5. 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;
```

Output:

Total_Sales	6
Total_Revenue	570000.00
Average_Price	42833.333333
Min_Price	20000.00
Max_Price	78000.00
Product	Sales_Count
Laptop	2

queries.sql

43eskuqym

AI

NEW

MYSQL

RUN

```

157 WHERE SaleDate > '2025-02-01';
158
159 -- MIN, MAX-
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_Quantity_Sold FROM Sales;
165
166 -- 3. Minimum and Maximum revenue generated from a single transaction
167 SELECT MIN(Quantity * Price) AS Min_Revenue, MAX(Quantity * Price) AS Max_Revenue FROM Sales;
168
169 -- 4. Minimum and Maximum price per product
170 SELECT Product, MIN(Price) AS Min_Price_Per_Product, MAX(Price) AS Max_Price_Per_Product
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(Quantity * Price) AS Max_Revenue_Per_Product
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) AS Max_Quantity_Per_Product
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

```

STORM

Input for the program (Optional)

Average_Price		

	42833.333333	

Average_Quantity_Sold		

	2.8333	

Average_Revenue_Per_Transaction		

	95800.000000	

Product	Average_Price_Per_Product	

	Laptop	76500.000000
	Mobile	21000.000000
	Tablet	31000.000000

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

Practical No: 8

Aim: To Given Customers and Orders tables, write SQL queries to perform INNERJOIN, LEFTJOIN, and RIGHT JOIN to retrieve combined data for customer orders.

Code:

```
CREATE TABLE Customers (  
    CustomerID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    Email VARCHAR(100)  
);  
  
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY,  
    CustomerID INT,  
    Product VARCHAR(100),  
    OrderDate DATE,  
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
);  
  
INSERT INTO Customers (CustomerID, Name, Email)  
VALUES  
(1, 'Alice', 'alice@example.com'),  
(2, 'Bob', 'bob@example.com'),  
(3, 'Charlie', 'charlie@example.com'),  
(4, 'Diana', 'diana@example.com');  
  
INSERT INTO Orders (OrderID, CustomerID, Product, OrderDate)  
VALUES  
(101, 1, 'Laptop', '2024-12-01'),  
(102, 2, 'Keyboard', '2024-12-02'),  
(103, 1, 'Mouse', '2024-12-03'),  
(104, 3, 'Monitor', '2024-12-04');  
  
SELECT Customers.CustomerID, Customers.Name, Orders.Product, Orders.OrderDate  
FROM Customers
```

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

SELECT Customers.CustomerID, Customers.Name, Orders.Product, Orders.OrderDate

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

SELECT Customers.CustomerID, Customers.Name, Orders.Product, Orders.OrderDate

FROM Customers

RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;

Screenshot of output:

The screenshot shows the OneCompiler web interface. On the left, a SQL editor contains the following code:

```
1 CREATE TABLE Customers (  
2   CustomerID INT PRIMARY KEY,  
3   Name VARCHAR(100),  
4   Email VARCHAR(100)  
5 );  
6 CREATE TABLE Orders (  
7   OrderID INT PRIMARY KEY,  
8   CustomerID INT,  
9   Product VARCHAR(100),  
10  OrderDate DATE,  
11  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)  
12 );  
13 INSERT INTO Customers (CustomerID, Name, Email)  
14 VALUES  
15 (1, 'Alice', 'alice@example.com'),  
16 (2, 'Bob', 'bob@example.com'),  
17 (3, 'Charlie', 'charlie@example.com'),  
18 (4, 'Diana', 'diana@example.com');  
19  
20 INSERT INTO Orders (OrderID, CustomerID, Product, OrderDate)  
21 VALUES  
22 (101, 1, 'Laptop', '2024-12-01'),  
23 (102, 2, 'Keyboard', '2024-12-02'),  
24 (103, 1, 'Mouse', '2024-12-03'),  
25 (104, 3, 'Monitor', '2024-12-04');  
26  
27  
28 SELECT Customers.CustomerID, Customers.Name, Orders.Product, Orders.OrderDate  
29 FROM Customers  
30 INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;  
31  
32 SELECT Customers.CustomerID, Customers.Name, Orders.Product, Orders.OrderDate  
33 FROM Customers  
34 LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;  
35  
36 SELECT Customers.CustomerID, Customers.Name, Orders.Product, Orders.OrderDate  
37 FROM Customers  
38 RIGHT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;  
39  
40
```

On the right, the 'Output' section displays the results of the queries. The first query (INNER JOIN) shows 4 rows. The second query (LEFT JOIN) shows 4 rows. The third query (RIGHT JOIN) shows 3 rows.

CustomerID	Name	Product	OrderDate
1	Alice	Laptop	2024-12-01
1	Alice	Mouse	2024-12-03
2	Bob	Keyboard	2024-12-02
3	Charlie	Monitor	2024-12-04

CustomerID	Name	Product	OrderDate
1	Alice	Laptop	2024-12-01
1	Alice	Mouse	2024-12-03
2	Bob	Keyboard	2024-12-02
3	Charlie	Monitor	2024-12-04
4	Diana	NULL	NULL

CustomerID	Name	Product	OrderDate
1	Alice	Laptop	2024-12-01
2	Bob	Keyboard	2024-12-02
1	Alice	Mouse	2024-12-03
3	Charlie	Monitor	2024-12-04