PRACTICAL 1 TO 8

1. **Employee Table,Insertion,Deletion**

CREATE DATABASE company;

USE company;

CREATE TABLE employees (

Emp\_id INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(50) NOT NULL,

Age INT NOT NULL,

Department VARCHAR(50),

Salary DECIMAL(10,2)

);

INSERT INTO employees (Name, Age, Department, Salary) VALUES

('shubham paithankar', 18, 'CY', 80000.00),

('rohit sharma', 42, 'HR', 4500.00),

('surya yadav', 35, 'Finance', 60000.00),

('Hardik pandya', 18, 'CSE', 70000.00),

('Shantanu sonawne', 42, 'AIML', 90000.00),

('vaibhav kale', 25, 'AIDS', 50000.00);

SELECT \* FROM employees;

**DELETE FROM employees WHERE emp\_id = 5;**

SELECT \* FROM employees;

1. **Students Table, insertion , Update , Delete , Select**

CREATE TABLE students (

Student\_id INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(50),

Age INT,

Course VARCHAR(50),

Marks INT

);

INSERT INTO students (name, age, course, marks) VALUES

('Shubham Paithankar', 18, 'Cyber Security', 88),

('Suraj Chavhan', 21, 'Mechanical Engineering', 78);

SELECT \* FROM students;

**UPDATE students**

**SET Marks = 80**

**WHERE Student\_id = 1;**

**DELETE FROM students**

**WHERE Student\_id = 2;**

**SELECT \* FROM students;**

1. **table with columns for EmployeeID, Name, Salary, JoiningDate, and ActiveStatus perform queries to manipulate and retrieve data.**

CREATE DATABASE employee\_db;

USE employee\_db;

CREATE TABLE employees (

EmployeeID INT PRIMARY KEY AUTO\_INCREMENT,

Name VARCHAR(100) NOT NULL,

Salary DECIMAL(20,2),

JoiningDate DATE,

ActiveStatus BOOLEAN

);

INSERT INTO employees (Name, Salary, JoiningDate, ActiveStatus) VALUES

('Shubham Paithankar', 95000.00, '2022-06-23', TRUE),

('Rohit Kolhe', 70000.50, '2021-07-25', TRUE),

('Aarav Wable', 50000.75, '2023-01-02', True),

('Gita rajwade', 70000.25, '2019-11-05', TRUE),

('Tejas Gade', 70000.25, '2020-11-05', TRUE);

**SELECT \* FROM employees;**

**UPDATE employees**

**SET Salary = Round(Salary \* 1.15);**

**SELECT \* FROM employees;**

**SELECT \* FROM employees**

**ORDER BY Salary DESC;**

1. **Create a table to store employee information with constraints like Primary Key, Foreign Key, and Unique. Insert valid and invalid data to test the constraints.**

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);

-- 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);

-- Duplicate Unique Email

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (103,

'David', 'alice@example.com', 45000.00, 2);

-- Salary Check Constraint Violation

INSERT INTO Employee (EmpID, Name, Email, Salary, DeptID) VALUES (105,

'Frank', 'frank@example.com', -40000.00, 1);

5**) Create a table for Customer details with various integrity constraints like NOT NULL,**

**CHECK, and DEFAULT. Insert valid and invalid data to test these constraints and**

**ensure data integrity.**

CREATE TABLE Customer (

CustomerID INT PRIMARY KEY,

FirstName VARCHAR(100) NOT NULL,

LastName VARCHAR(100) NOT NULL,

Email VARCHAR(100) UNIQUE,

Phone VARCHAR(15),

Age INT CHECK (Age >= 18),

IsActive BOOLEAN DEFAULT TRUE

);

-- Insert Valid Data

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

IsActive)

VALUES

(1, '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);

-- 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);

**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.**

Code:

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';

-- 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);

-- 6. Update using a CASE statement (e.g., increase salary based on department)

UPDATE Employees

SET Salary = CASE

WHEN Department = 'HR' THEN Salary \* 1.05

WHEN Department = 'IT' THEN Salary \* 1.08

WHEN Department = 'Finance' THEN Salary \* 1.10

ELSE Salary

END;

-- Delete Data from the Table (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;

**7) Create a Sales table and use aggregate functions like COUNT, SUM, AVG, MIN, and**

**MAX to summarize sales data and calculate statistics.**

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 ᴃ50,000

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 ᴃ40,000

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 ᴃ40,000

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';

**8) Given Customers and Orders tables, write SQL queries to perform INNER JOIN, LEFT**

**JOIN, and RIGHT JOIN to retrieve combined data for customer orders.**

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

SELECT

c.customer\_id,

c.customer\_name,

o.order\_id,

o.order\_date

FROM

Customers c

INNER JOIN

Orders o

ON

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

LEFT JOIN - All Customers with their Orders (if any)

SELECT

c.customer\_id,

c.customer\_name,

o.order\_id,

o.order\_date

FROM

Customers c

LEFT JOIN

Orders o

ON

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

RIGHT JOIN - All Orders with Customer details (if any)

SELECT

c.customer\_id,

c.customer\_name,

o.order\_id,

o.order\_date

FROM

Customers c

RIGHT JOIN

Orders o

ON

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