Practical 1 (Company Database).:

```
CREATE DATABASE company;
USE company;

CREATE TABLE dept (
  deptno INT PRIMARY KEY,
  deptname VARCHAR(50),
  location VARCHAR(50)
);

CREATE TABLE emp (
  eno INT PRIMARY KEY,
  ename VARCHAR(50),
  job VARCHAR(30),
  hiredate DATE,
  salary DECIMAL(10,2),
  commission DECIMAL(10,2),
  deptno INT,
  FOREIGN KEY (deptno) REFERENCES dept(deptno)
);


INSERT INTO dept VALUES
(10, 'Dev', 'Pune'),
(20, 'Sales', 'Mumbai'),
(30, 'HR', 'Delhi');

INSERT INTO emp VALUES
(101, 'Isha', 'Salesman', '1981-02-20', 3000, 500, 20),
(102, 'Rahul', 'Manager', '1980-12-15', 5000, NULL, 10),
(103, 'Amit', 'Clerk', '1981-09-10', 1500, NULL, 10),
(104, 'Irfan', 'Salesman', '1982-10-25', 3500, 300, 20),
(105, 'Neha', 'Analyst', '1983-07-09', 4000, NULL, 30),
(106, 'Isha', 'Salesman', '1981-01-15', 3200, 400, 20);


SELECT MAX(salary) AS Max_Salary_Salesman
FROM emp
WHERE job = 'Salesman';

SELECT ename
FROM emp
WHERE ename LIKE 'I%';

SELECT *
FROM emp
WHERE hiredate < '1981-09-30';
```

```
SELECT *
FROM emp
ORDER BY salary DESC;

SELECT COUNT(*) AS Num_Employees, AVG(salary) AS Avg_Salary
FROM emp
WHERE deptno = 20;

SELECT hiredate, AVG(salary) AS Avg_Salary, MIN(salary) AS Min_Salary
FROM emp
WHERE deptno = 10
GROUP BY hiredate;

SELECT e.ename, d.deptname
FROM emp e
JOIN dept d ON e.deptno = d.deptno;

SELECT e.*
FROM emp e
JOIN dept d ON e.deptno = d.deptno
WHERE d.deptname = 'Dev';

SELECT e.*
FROM emp e
JOIN dept d ON e.deptno = d.deptno
WHERE d.deptname = 'Dev';

UPDATE emp
SET salary = salary * 1.05
WHERE deptno = 10;
```

```
/* ===============================================================
   PRACTICAL 2 – Employee–Company Database
   Database Name: p2
   =============================================================== */

-- STEP 1: Create Database
CREATE DATABASE p2;
USE p2;

-- STEP 2: Create Tables
CREATE TABLE employee (
  employee_name VARCHAR(50) PRIMARY KEY,
  street VARCHAR(50),
  city VARCHAR(50)
);

CREATE TABLE company (
  company_name VARCHAR(50) PRIMARY KEY,
  city VARCHAR(50)
);

CREATE TABLE works (
  employee_name VARCHAR(50),
  company_name VARCHAR(50),
  salary DECIMAL(10,2),
  FOREIGN KEY (employee_name) REFERENCES employee(employee_name),
  FOREIGN KEY (company_name) REFERENCES company(company_name)
);

CREATE TABLE manages (
  employee_name VARCHAR(50),
  manager_name VARCHAR(50),
  FOREIGN KEY (employee_name) REFERENCES employee(employee_name)
);

-- STEP 3: Insert Sample Data
INSERT INTO employee VALUES
('Anil', 'MG Road', 'Pune'),
('Sunita', 'FC Road', 'Mumbai'),
('Ravi', 'JM Road', 'Delhi'),
('Meena', 'Karve Road', 'Pune'),
('Suresh', 'Sinhgad Road', 'Chennai');

INSERT INTO company VALUES
('First Bank Corporation', 'Mumbai'),
('Small Bank Corporation', 'Delhi'),
('TechSoft Ltd', 'Pune');
```

```sql
INSERT INTO works VALUES
('Anil', 'First Bank Corporation', 12000),
('Sunita', 'Small Bank Corporation', 9000),
('Ravi', 'TechSoft Ltd', 15000),
('Meena', 'First Bank Corporation', 11000),
('Suresh', 'First Bank Corporation', 8000);

INSERT INTO manages VALUES
('Anil', 'Ravi'),
('Sunita', 'Meena'),
('Ravi', 'Suresh'),
('Meena', 'Ravi'),
('Suresh', 'Sunita');

-- ==========================================================
-- STEP 4: SIMPLE SQL QUERIES
-- ==========================================================

--1️⃣Names of all employees who work for First Bank Corporation
SELECT employee_name
FROM works
WHERE company_name = 'First Bank Corporation';

--2️⃣Employees who do not work for First Bank Corporation
SELECT employee_name
FROM works
WHERE company_name != 'First Bank Corporation';

--3️⃣Count of employees in each company
SELECT company_name, COUNT(employee_name) AS total_employees
FROM works
GROUP BY company_name;

--4️⃣List companies in the same city as Small Bank Corporation
SELECT company_name, city
FROM company
WHERE city IN (SELECT city FROM company WHERE company_name = 'Small Bank
Corporation');

--5️⃣Details of employees having salary greater than 10,000
SELECT e.employee_name, e.street, e.city, w.salary
FROM employee e, works w
WHERE e.employee_name = w.employee_name
AND w.salary > 10000;

--6️⃣Increase salary of all employees who work for First Bank Corporation by 10%
UPDATE works
SET salary = salary + (salary * 0.10)
```

```
WHERE company_name = 'First Bank Corporation';

--7️⃣ List employees and their managers
SELECT * FROM manages;

--8️⃣ Names, street, and cities of employees who work for First Bank Corporation and earn
more than 10,000
SELECT e.employee_name, e.street, e.city, w.salary
FROM employee e, works w
WHERE e.employee_name = w.employee_name
AND w.company_name = 'First Bank Corporation'
AND w.salary > 10000;

--9️⃣ Average salary for each company
SELECT company_name, AVG(salary) AS average_salary
FROM works
GROUP BY company_name;
```

```sql
/* ==========================================================
   PRACTICAL 3 – Hotel Management Database
   Database Name: p3
   ========================================================== */

-- STEP 1: Create Database
CREATE DATABASE p3;
USE p3;

-- STEP 2: Create Tables
CREATE TABLE Hotel (
  HotelNo INT PRIMARY KEY,
  Name VARCHAR(50),
  City VARCHAR(50)
);

CREATE TABLE Room (
  RoomNo INT,
  HotelNo INT,
  Type VARCHAR(20),
  Price DECIMAL(10,2),
  PRIMARY KEY (RoomNo, HotelNo),
  FOREIGN KEY (HotelNo) REFERENCES Hotel(HotelNo)
);

CREATE TABLE Guest (
  GuestNo INT PRIMARY KEY,
  GuestName VARCHAR(50),
  GuestAddress VARCHAR(100)
);

CREATE TABLE Booking (
  HotelNo INT,
  GuestNo INT,
  DateFrom DATE,
  DateTo DATE,
  RoomNo INT,
  PRIMARY KEY (HotelNo, GuestNo, DateFrom),
  FOREIGN KEY (HotelNo) REFERENCES Hotel(HotelNo),
  FOREIGN KEY (GuestNo) REFERENCES Guest(GuestNo)
);

-- STEP 3: Insert Sample Data
INSERT INTO Hotel VALUES
(1, 'Grosvenor Hotel', 'London'),
(2, 'Blue Moon', 'Pune'),
(3, 'Sunrise Inn', 'Mumbai');
```

```
INSERT INTO Room VALUES
(101, 1, 'Single', 45.00),
(102, 1, 'Double', 80.00),
(103, 1, 'Family', 120.00),
(201, 2, 'Single', 40.00),
(202, 2, 'Double', 75.00);

INSERT INTO Guest VALUES
(1, 'Ravi', 'Pune'),
(2, 'Isha', 'London'),
(3, 'Neha', 'Mumbai');

INSERT INTO Booking VALUES
(1, 1, '2024-09-01', '2024-09-05', 101),
(1, 2, '2024-09-10', '2024-09-15', 102),
(2, 3, '2024-09-12', '2024-09-14', 201);

-- STEP 4: Queries

--1️⃣List full details of all hotels
SELECT * FROM Hotel;

--2️⃣How many hotels are there?
SELECT COUNT(*) AS Total_Hotels FROM Hotel;

--3️⃣List price and type of all rooms at the Grosvenor Hotel
SELECT r.Type, r.Price
FROM Room r, Hotel h
WHERE r.HotelNo = h.HotelNo
AND h.Name = 'Grosvenor Hotel';

--4️⃣List number of rooms in each hotel
SELECT h.Name, COUNT(r.RoomNo) AS Total_Rooms
FROM Hotel h, Room r
WHERE h.HotelNo = r.HotelNo
GROUP BY h.Name;

--5️⃣Update the price of all rooms by 5%
UPDATE Room
SET Price = Price + (Price * 0.05);

--6️⃣List full details of all hotels in London
SELECT * FROM Hotel WHERE City = 'London';

--7️⃣Average price of a room
SELECT AVG(Price) AS Average_Price FROM Room;

--8️⃣List all guests currently staying at the Grosvenor Hotel
```

```
SELECT g.GuestName
FROM Guest g, Booking b, Hotel h
WHERE g.GuestNo = b.GuestNo
AND b.HotelNo = h.HotelNo
AND h.Name = 'Grosvenor Hotel';

--9️⃣List number of rooms in each hotel in London
SELECT h.Name, COUNT(r.RoomNo) AS No_Of_Rooms
FROM Hotel h, Room r
WHERE h.HotelNo = r.HotelNo AND h.City = 'London'
GROUP BY h.Name;

-- 🔟 Create one view and query it
CREATE VIEW LondonHotels AS
SELECT * FROM Hotel WHERE City = 'London';

SELECT * FROM LondonHotels;
```

```
/* ============================================================
   PRACTICAL 4 – Hotel Database (Advanced Queries)
   Database Name: p4
   ============================================================ */

CREATE DATABASE p4;
USE p4;

CREATE TABLE Hotel (
  HotelNo INT PRIMARY KEY,
  Name VARCHAR(50),
  City VARCHAR(50)
);

CREATE TABLE Room (
  RoomNo INT,
  HotelNo INT,
  Type VARCHAR(20),
  Price DECIMAL(10,2),
  PRIMARY KEY (RoomNo, HotelNo),
  FOREIGN KEY (HotelNo) REFERENCES Hotel(HotelNo)
);

CREATE TABLE Guest (
  GuestNo INT PRIMARY KEY,
  GuestName VARCHAR(50),
  GuestAddress VARCHAR(100)
);

CREATE TABLE Booking (
  HotelNo INT,
  GuestNo INT,
  DateFrom DATE,
  DateTo DATE,
  RoomNo INT,
  PRIMARY KEY (HotelNo, GuestNo, DateFrom),
  FOREIGN KEY (HotelNo) REFERENCES Hotel(HotelNo),
  FOREIGN KEY (GuestNo) REFERENCES Guest(GuestNo)
);

INSERT INTO Hotel VALUES
(1, 'Grosvenor Hotel', 'London'),
(2, 'Sunrise Inn', 'Pune');

INSERT INTO Room VALUES
(101, 1, 'Double', 80.00),
(102, 1, 'Family', 120.00),
(201, 2, 'Single', 40.00);
```

```sql
INSERT INTO Guest VALUES
(1, 'Ravi', 'London'),
(2, 'Meena', 'Pune');

INSERT INTO Booking VALUES
(1, 1, '2024-09-01', '2024-09-05', 101),
(2, 2, '2024-09-10', '2024-09-15', 201);

--1️⃣Total revenue per night from all double rooms
SELECT SUM(Price) AS Total_Revenue
FROM Room
WHERE Type = 'Double';

--2️⃣Details of all rooms at the Grosvenor Hotel including guest name
SELECT r.RoomNo, r.Type, r.Price, g.GuestName
FROM Room r
LEFT JOIN Booking b ON r.RoomNo = b.RoomNo
LEFT JOIN Guest g ON b.GuestNo = g.GuestNo
JOIN Hotel h ON h.HotelNo = r.HotelNo
WHERE h.Name = 'Grosvenor Hotel';

--3️⃣Average number of bookings for each hotel (example for month April)
SELECT h.Name, COUNT(b.GuestNo) AS Total_Bookings
FROM Hotel h
LEFT JOIN Booking b ON h.HotelNo = b.HotelNo
GROUP BY h.Name;

--4️⃣Create index on Room Type
CREATE INDEX idx_room_type ON Room(Type);

--5️⃣List full details of all hotels
SELECT * FROM Hotel;

--6️⃣List full details of all hotels in London
SELECT * FROM Hotel WHERE City = 'London';

--7️⃣Update the price of all rooms by 5%
UPDATE Room SET Price = Price * 1.05;

--8️⃣List number of rooms in each hotel in London
SELECT h.Name, COUNT(r.RoomNo) AS No_Of_Rooms
FROM Hotel h, Room r
WHERE h.HotelNo = r.HotelNo AND h.City = 'London'
GROUP BY h.Name;

--9️⃣List all double or family rooms with price below 40 in ascending order
SELECT * FROM Room
```

```
WHERE (Type = 'Double' OR Type = 'Family') AND Price < 40
ORDER BY Price ASC;


/* ============================================================
   PRACTICAL 5 – Hotel Database
   Database Name: p5
   ============================================================ */

CREATE DATABASE p5;
USE p5;

-- STEP 1: Create Tables
CREATE TABLE Hotel (
  HotelNo INT PRIMARY KEY,
  Name VARCHAR(50),
  City VARCHAR(50)
);

CREATE TABLE Room (
  RoomNo INT,
  HotelNo INT,
  Type VARCHAR(20),
  Price DECIMAL(10,2),
  PRIMARY KEY (RoomNo, HotelNo),
  FOREIGN KEY (HotelNo) REFERENCES Hotel(HotelNo)
);

CREATE TABLE Guest (
  GuestNo INT PRIMARY KEY,
  GuestName VARCHAR(50),
  GuestAddress VARCHAR(100)
);

CREATE TABLE Booking (
  HotelNo INT,
  GuestNo INT,
  DateFrom DATE,
  DateTo DATE,
  RoomNo INT,
  PRIMARY KEY (HotelNo, GuestNo, DateFrom),
  FOREIGN KEY (HotelNo) REFERENCES Hotel(HotelNo),
  FOREIGN KEY (GuestNo) REFERENCES Guest(GuestNo)
);

-- STEP 2: Insert Sample Data
INSERT INTO Hotel VALUES
(1, 'Grosvenor Hotel', 'London'),
```

```sql
(2, 'City Stay', 'Delhi');

INSERT INTO Room VALUES
(101, 1, 'Single', 40.00),
(102, 1, 'Double', 70.00),
(103, 1, 'Family', 100.00),
(201, 2, 'Single', 45.00),
(202, 2, 'Double', 80.00);

INSERT INTO Guest VALUES
(1, 'Amit', 'London'),
(2, 'Neha', 'Delhi'),
(3, 'Ravi', 'Pune');

INSERT INTO Booking VALUES
(1, 1, '2024-08-01', '2024-08-05', 101),
(1, 2, '2024-08-10', '2024-08-15', 102),
(2, 3, '2024-08-18', '2024-08-25', 202);

-- STEP 3: Queries

--1️⃣ List full details of all hotels
SELECT * FROM Hotel;

--2️⃣ How many hotels are there
SELECT COUNT(*) AS Total_Hotels FROM Hotel;

--3️⃣ List price and type of all rooms at Grosvenor Hotel
SELECT r.Type, r.Price
FROM Room r, Hotel h
WHERE r.HotelNo = h.HotelNo AND h.Name = 'Grosvenor Hotel';

--4️⃣ List number of rooms in each hotel
SELECT h.Name, COUNT(r.RoomNo) AS Total_Rooms
FROM Hotel h, Room r
WHERE h.HotelNo = r.HotelNo
GROUP BY h.Name;

--5️⃣ List all guests currently staying at Grosvenor Hotel
SELECT g.GuestName
FROM Guest g, Booking b, Hotel h
WHERE g.GuestNo = b.GuestNo AND b.HotelNo = h.HotelNo
AND h.Name = 'Grosvenor Hotel';

--6️⃣ List all double or family rooms with price below 40 ascending
SELECT * FROM Room
WHERE (Type = 'Double' OR Type = 'Family') AND Price < 40
ORDER BY Price ASC;
```

```
--7️⃣How many different guests have made bookings for August
SELECT COUNT(DISTINCT GuestNo) AS Guests_In_August
FROM Booking
WHERE MONTH(DateFrom) = 8;

--8️⃣Total income from bookings for Grosvenor Hotel today
SELECT SUM(r.Price) AS Total_Income
FROM Room r, Hotel h
WHERE r.HotelNo = h.HotelNo AND h.Name = 'Grosvenor Hotel';

--9️⃣Most commonly booked room type for each hotel in London
SELECT h.Name, r.Type
FROM Room r, Hotel h
WHERE h.City = 'London';

-- 🔟 Update price of all rooms by 5%
UPDATE Room SET Price = Price * 1.05;
```

```
/* ============================================================
   PRACTICAL 6 – Hotel Database (Index & Views)
   Database Name: p6
   ============================================================ */

CREATE DATABASE p6;
USE p6;

-- STEP 1: Create Tables
CREATE TABLE Hotel (
  HotelNo INT PRIMARY KEY,
  Name VARCHAR(50),
  City VARCHAR(50)
);

CREATE TABLE Room (
  RoomNo INT,
  HotelNo INT,
  Type VARCHAR(20),
  Price DECIMAL(10,2),
  PRIMARY KEY (RoomNo, HotelNo),
  FOREIGN KEY (HotelNo) REFERENCES Hotel(HotelNo)
);

CREATE TABLE Guest (
  GuestNo INT PRIMARY KEY,
  GuestName VARCHAR(50),
  GuestAddress VARCHAR(100)
);

CREATE TABLE Booking (
  HotelNo INT,
  GuestNo INT,
  DateFrom DATE,
  DateTo DATE,
  RoomNo INT,
  PRIMARY KEY (HotelNo, GuestNo, DateFrom),
  FOREIGN KEY (HotelNo) REFERENCES Hotel(HotelNo),
  FOREIGN KEY (GuestNo) REFERENCES Guest(GuestNo)
);

-- STEP 2: Sample Data
INSERT INTO Hotel VALUES
(1, 'Grosvenor Hotel', 'London'),
(2, 'Grand Palace', 'Pune');

INSERT INTO Room VALUES
(101, 1, 'Double', 80.00),
(102, 1, 'Family', 120.00),
(201, 2, 'Single', 50.00);

INSERT INTO Guest VALUES
(1, 'Ravi', 'London'),
(2, 'Isha', 'Pune');

INSERT INTO Booking VALUES
(1, 1, '2024-09-01', '2024-09-05', 101),
(2, 2, '2024-09-10', NULL, 201);

-- STEP 3: Queries

--1️⃣List all hotels
SELECT * FROM Hotel;

--2️⃣List all hotels in London
SELECT * FROM Hotel WHERE City = 'London';

--3️⃣List guests currently staying at Grosvenor Hotel
SELECT g.GuestName
FROM Guest g, Booking b, Hotel h
WHERE g.GuestNo = b.GuestNo AND b.HotelNo = h.HotelNo
AND h.Name = 'Grosvenor Hotel';

--4️⃣List names & addresses of all guests in London alphabetically
SELECT GuestName, GuestAddress
FROM Guest
WHERE GuestAddress LIKE '%London%'
ORDER BY GuestName ASC;

--5️⃣List bookings with no DateTo specified
SELECT * FROM Booking WHERE DateTo IS NULL;

--6️⃣How many hotels are there
SELECT COUNT(*) AS Total_Hotels FROM Hotel;

--7️⃣List rooms that are unoccupied at Grosvenor Hotel
SELECT r.RoomNo, r.Type
FROM Room r
WHERE r.RoomNo NOT IN (SELECT RoomNo FROM Booking)
AND r.HotelNo = 1;

--8️⃣Lost income from unoccupied rooms (today)
SELECT SUM(Price) AS Lost_Income
FROM Room
WHERE RoomNo NOT IN (SELECT RoomNo FROM Booking);
```

```
--9️⃣Create index on Room type
CREATE INDEX idx_type ON Room(Type);

-- 🔟 Create a view and query it
CREATE VIEW LondonHotels AS
SELECT * FROM Hotel WHERE City = 'London';

SELECT * FROM LondonHotels;
```

```
-- 💾 PRACTICAL NO. 7
-- DATABASE : p7
CREATE DATABASE p7;
USE p7;

-- 🧱 TABLE CREATION
-- 1. Project(project_id, proj_name, chief_arch)
CREATE TABLE Project (
    project_id VARCHAR(10) PRIMARY KEY,
    proj_name VARCHAR(50),
    chief_arch VARCHAR(50)
);

-- 2. Employee(Emp_id, Emp_name)
CREATE TABLE Employee (
    Emp_id INT PRIMARY KEY,
    Emp_name VARCHAR(50)
);

-- 3. Assigned_To(Project_id, Emp_id)
CREATE TABLE Assigned_To (
    Project_id VARCHAR(10),
    Emp_id INT,
    FOREIGN KEY (Project_id) REFERENCES Project(project_id),
    FOREIGN KEY (Emp_id) REFERENCES Employee(Emp_id)
);

-- 🧩 SAMPLE DATA
INSERT INTO Project VALUES
('C353', 'Database', 'Mr. A'),
('C354', 'AI System', 'Mr. B'),
('C453', 'Web Portal', 'Ms. C');

INSERT INTO Employee VALUES
(101, 'Ravi'),
(102, 'Sneha'),
(103, 'Ajay'),
(104, 'Neha'),
(105, 'Kiran');

INSERT INTO Assigned_To VALUES
('C353', 101),
('C353', 102),
('C354', 102),
('C354', 103),
('C453', 104);

-- 🧮 QUERIES
```

```sql
--1️⃣ Get the details of employees working on project C353
SELECT e.Emp_id, e.Emp_name
FROM Employee e
JOIN Assigned_To a ON e.Emp_id = a.Emp_id
WHERE a.Project_id = 'C353';

--2️⃣ Get number of employees working on project C353
SELECT COUNT(*) AS total_employees
FROM Assigned_To
WHERE Project_id = 'C353';

--3️⃣ Obtain details of employees working on 'Database' project
SELECT e.Emp_id, e.Emp_name, p.proj_name
FROM Employee e
JOIN Assigned_To a ON e.Emp_id = a.Emp_id
JOIN Project p ON p.project_id = a.Project_id
WHERE p.proj_name = 'Database';

--4️⃣ Get details of employees working on both C353 and C354
SELECT e.Emp_id, e.Emp_name
FROM Employee e
WHERE e.Emp_id IN (
   SELECT Emp_id FROM Assigned_To WHERE Project_id = 'C353'
)
AND e.Emp_id IN (
   SELECT Emp_id FROM Assigned_To WHERE Project_id = 'C354'
);

--5️⃣ Get employee numbers of employees who do NOT work on project C453
SELECT Emp_id, Emp_name
FROM Employee
WHERE Emp_id NOT IN (
   SELECT Emp_id FROM Assigned_To WHERE Project_id = 'C453'
);
```

```sql
/* ================================================================
   PRACTICAL 8 – Employee Duty Allocation Database
   Database Name: p8
   ================================================================ */

CREATE DATABASE p8;
USE p8;

-- STEP 1: Create Tables
CREATE TABLE Employee (
  emp_no INT PRIMARY KEY,
  name VARCHAR(50),
  skill VARCHAR(50),
  pay_rate DECIMAL(10,2)
);

CREATE TABLE Position (
  posting_no INT PRIMARY KEY,
  skill VARCHAR(50)
);

CREATE TABLE Duty_allocation (
  posting_no INT,
  emp_no INT,
  day DATE,
  shift VARCHAR(20),
  FOREIGN KEY (posting_no) REFERENCES Position(posting_no),
  FOREIGN KEY (emp_no) REFERENCES Employee(emp_no)
);

-- STEP 2: Sample Data
INSERT INTO Employee VALUES
(123459, 'Rahul', 'Chef', 450.00),
(123460, 'Amit', 'Waiter', 300.00),
(123461, 'XYZ', 'Chef', 500.00),
(123462, 'Neha', 'Cleaner', 250.00),
(123463, 'Priya', 'Chef', 600.00);

INSERT INTO Position VALUES
(201, 'Chef'),
(202, 'Waiter'),
(203, 'Cleaner');

INSERT INTO Duty_allocation VALUES
(201, 123461, '1986-04-02', 'Morning'),
(201, 123461, '1986-04-15', 'Evening'),
(202, 123460, '1986-04-10', 'Morning'),
(203, 123462, '1986-04-08', 'Evening'),
(201, 123463, '1986-04-05', 'Morning');

-- STEP 3: Queries

--1️⃣Get the duty allocation details for emp_no 123461 for the month of April 1986
SELECT *
FROM Duty_allocation
WHERE emp_no = 123461
AND MONTH(day) = 4
AND YEAR(day) = 1986;

--2️⃣Find the shift details for Employee 'xyz'
SELECT d.day, d.shift
FROM Duty_allocation d
JOIN Employee e ON e.emp_no = d.emp_no
WHERE LOWER(e.name) = 'xyz';

--3️⃣Get employees whose pay rate >= pay rate of employee 'xyz'
SELECT name, pay_rate
FROM Employee
WHERE pay_rate >= (SELECT pay_rate FROM Employee WHERE LOWER(name) = 'xyz');

--4️⃣Get names & pay rates of employees with emp_no < 123460 and pay_rate >
--    at least one employee with emp_no >= 123460
SELECT e1.name, e1.pay_rate
FROM Employee e1
WHERE e1.emp_no < 123460
AND e1.pay_rate > (
  SELECT MIN(e2.pay_rate) FROM Employee e2 WHERE e2.emp_no >= 123460
);

--5️⃣Find names of employees assigned to all positions requiring a Chef’s skill
SELECT DISTINCT e.name
FROM Employee e
WHERE e.skill = 'Chef'
AND e.emp_no IN (
  SELECT emp_no FROM Duty_allocation d
  JOIN Position p ON d.posting_no = p.posting_no
  WHERE p.skill = 'Chef'
);

--6️⃣Find the employee(s) with the lowest pay rate
SELECT *
FROM Employee
WHERE pay_rate = (SELECT MIN(pay_rate) FROM Employee);

--7️⃣Get employee numbers of all employees working on at least two dates
SELECT emp_no, COUNT(DISTINCT day) AS total_days
```

```
FROM Duty_allocation
GROUP BY emp_no
HAVING COUNT(DISTINCT day) >= 2;

--8️⃣Get list of names of employees with skill 'Chef' who are assigned a duty
SELECT DISTINCT e.name
FROM Employee e
JOIN Duty_allocation d ON e.emp_no = d.emp_no
WHERE e.skill = 'Chef';

--9️⃣Get list of employees not assigned any duty
SELECT e.name
FROM Employee e
WHERE e.emp_no NOT IN (SELECT emp_no FROM Duty_allocation);

-- 🔟 Get count of different employees on each shift
SELECT shift, COUNT(DISTINCT emp_no) AS total_employees
FROM Duty_allocation
GROUP BY shift;
```

```
/* ============================================================
   PRACTICAL 9 – Banking Database (Deposits, Borrow, Branch)
   Database Name: p9
   ============================================================ */

CREATE DATABASE p9;
USE p9;

-- STEP 1: Create Tables

CREATE TABLE Branch (
  bname VARCHAR(50) PRIMARY KEY,
  city VARCHAR(50)
);

CREATE TABLE Customers (
  cname VARCHAR(50) PRIMARY KEY,
  city VARCHAR(50)
);

CREATE TABLE Deposit (
  actno INT PRIMARY KEY,
  cname VARCHAR(50),
  bname VARCHAR(50),
  amount DECIMAL(10,2),
  adate DATE,
  FOREIGN KEY (cname) REFERENCES Customers(cname),
  FOREIGN KEY (bname) REFERENCES Branch(bname)
);

CREATE TABLE Borrow (
  loanno INT PRIMARY KEY,
  cname VARCHAR(50),
  bname VARCHAR(50),
  amount DECIMAL(10,2),
  FOREIGN KEY (cname) REFERENCES Customers(cname),
  FOREIGN KEY (bname) REFERENCES Branch(bname)
);

-- STEP 2: Sample Data

INSERT INTO Branch VALUES
('Perryridge', 'Nagpur'),
('Downtown', 'Pune'),
('Central', 'Mumbai');

INSERT INTO Customers VALUES
('Anil', 'Pune'),
('Sunil', 'Nagpur'),
('Ravi', 'Mumbai'),
('Isha', 'Nagpur');

INSERT INTO Deposit VALUES
(101, 'Anil', 'Perryridge', 5000.00, '1997-01-02'),
(102, 'Sunil', 'Downtown', 3000.00, '1996-12-15'),
(103, 'Ravi', 'Central', 8000.00, '1997-03-05'),
(104, 'Isha', 'Perryridge', 1500.00, '1996-12-25');

INSERT INTO Borrow VALUES
(201, 'Anil', 'Perryridge', 2000.00),
(202, 'Sunil', 'Downtown', 5000.00),
(203, 'Ravi', 'Central', 7000.00);

-- STEP 3: Queries

--1️⃣Display names of depositors having amount greater than 4000
SELECT cname, amount
FROM Deposit
WHERE amount > 4000;

--2️⃣Display account date of customer 'Anil'
SELECT cname, adate
FROM Deposit
WHERE cname = 'Anil';

--3️⃣Display account no. and deposit amount of customers having account
--    opened between 1-Dec-1996 and 1-May-1997
SELECT actno, amount, adate
FROM Deposit
WHERE adate BETWEEN '1996-12-01' AND '1997-05-01';

--4️⃣Find the average account balance at the Perryridge branch
SELECT AVG(amount) AS Avg_Balance
FROM Deposit
WHERE bname = 'Perryridge';

--5️⃣Find the names of all branches where the average account balance is > 1200
SELECT bname, AVG(amount) AS Avg_Balance
FROM Deposit
GROUP BY bname
HAVING AVG(amount) > 1200;

--6️⃣Delete depositors having deposit less than 5000
DELETE FROM Deposit
WHERE amount < 5000;
```

```
--7️⃣ Create a view on deposit table
CREATE VIEW DepositView AS
SELECT actno, cname, bname, amount, adate
FROM Deposit;

-- Query the view
SELECT * FROM DepositView;
```

```
/* ============================================================
   PRACTICAL 10 – Banking Database (Branch, Deposit & Borrow)
   Database Name: p10
   ============================================================ */

CREATE DATABASE p10;
USE p10;

-- STEP 1: Create Tables

CREATE TABLE Branch (
  bname VARCHAR(50) PRIMARY KEY,
  city VARCHAR(50)
);

CREATE TABLE Customers (
  cname VARCHAR(50) PRIMARY KEY,
  city VARCHAR(50)
);

CREATE TABLE Deposit (
  actno INT PRIMARY KEY,
  cname VARCHAR(50),
  bname VARCHAR(50),
  amount DECIMAL(10,2),
  adate DATE,
  FOREIGN KEY (cname) REFERENCES Customers(cname),
  FOREIGN KEY (bname) REFERENCES Branch(bname)
);

CREATE TABLE Borrow (
  loanno INT PRIMARY KEY,
  cname VARCHAR(50),
  bname VARCHAR(50),
  amount DECIMAL(10,2),
  FOREIGN KEY (cname) REFERENCES Customers(cname),
  FOREIGN KEY (bname) REFERENCES Branch(bname)
);

-- STEP 2: Sample Data

INSERT INTO Branch VALUES
('Perryridge', 'Nagpur'),
('Downtown', 'Bombay'),
('Central', 'Pune');

INSERT INTO Customers VALUES
```

```
('Anil', 'Pune'),
('Sunil', 'Mumbai'),
('Ravi', 'Bombay'),
('Isha', 'Nagpur');

INSERT INTO Deposit VALUES
(101, 'Anil', 'Central', 7000.00, '1997-01-05'),
(102, 'Sunil', 'Downtown', 4000.00, '1996-12-15'),
(103, 'Ravi', 'Perryridge', 9000.00, '1997-02-10'),
(104, 'Isha', 'Downtown', 2000.00, '1997-03-01');

INSERT INTO Borrow VALUES
(201, 'Anil', 'Central', 5000.00),
(202, 'Sunil', 'Downtown', 3000.00),
(203, 'Ravi', 'Perryridge', 6000.00);

-- STEP 3: Queries

--a Display names of all branches located in city Bombay
SELECT bname
FROM Branch
WHERE city = 'Bombay';

--b Display account no. and amount of depositors
SELECT actno, amount
FROM Deposit;

--c Update the city of customer 'Anil' from Pune to Mumbai
UPDATE Customers
SET city = 'Mumbai'
WHERE cname = 'Anil';

--d Find the number of depositors in the bank
SELECT COUNT(*) AS Total_Depositors
FROM Deposit;

--e Calculate Min and Max amount of customers
SELECT MIN(amount) AS Min_Amount, MAX(amount) AS Max_Amount
FROM Deposit;

--f Create an index on deposit table (amount field)
CREATE INDEX idx_amount ON Deposit(amount);

--g Create a view on Borrow table
CREATE VIEW BorrowView AS
SELECT loanno, cname, bname, amount
FROM Borrow;
```

```
-- Query the view
SELECT * FROM BorrowView;
```

```
/* ============================================================
   PRACTICAL 11 – Banking Database (Deposit, Branch, Borrow)
   Database Name: p11
   ============================================================ */

CREATE DATABASE p11;
USE p11;

-- STEP 1: Create Tables

CREATE TABLE Branch (
  bname VARCHAR(50) PRIMARY KEY,
  city VARCHAR(50)
);

CREATE TABLE Customers (
  cname VARCHAR(50) PRIMARY KEY,
  city VARCHAR(50)
);

CREATE TABLE Deposit (
  actno INT PRIMARY KEY,
  cname VARCHAR(50),
  bname VARCHAR(50),
  amount DECIMAL(10,2),
  adate DATE,
  FOREIGN KEY (cname) REFERENCES Customers(cname),
  FOREIGN KEY (bname) REFERENCES Branch(bname)
);

CREATE TABLE Borrow (
  loanno INT PRIMARY KEY,
  cname VARCHAR(50),
  bname VARCHAR(50),
  amount DECIMAL(10,2),
  FOREIGN KEY (cname) REFERENCES Customers(cname),
  FOREIGN KEY (bname) REFERENCES Branch(bname)
);

-- STEP 2: Sample Data

INSERT INTO Branch VALUES
('KAROLBAGH', 'Delhi'),
('Perryridge', 'Nagpur'),
('Downtown', 'Pune');

INSERT INTO Customers VALUES
('Anil', 'Pune'),
('Sunil', 'Nagpur'),
('Ravi', 'Mumbai'),
('Isha', 'Delhi');

INSERT INTO Deposit VALUES
(101, 'Anil', 'Downtown', 6000.00, '1997-01-02'),
(102, 'Sunil', 'Perryridge', 4000.00, '1997-02-15'),
(103, 'Ravi', 'KAROLBAGH', 9000.00, '1997-03-05'),
(104, 'Isha', 'Perryridge', 3000.00, '1997-04-01');

INSERT INTO Borrow VALUES
(201, 'Anil', 'Downtown', 2000.00),
(202, 'Sunil', 'Perryridge', 5000.00),
(203, 'Ravi', 'KAROLBAGH', 7000.00);

-- STEP 3: Queries

--a⃣ Display account date of customer 'Anil'
SELECT adate
FROM Deposit
WHERE cname = 'Anil';

--b⃣ Modify the size of attribute 'amount' in deposit table (example: to 12,2)
ALTER TABLE Deposit
MODIFY amount DECIMAL(12,2);

--c⃣ Display names of customers living in city Pune
SELECT cname, city
FROM Customers
WHERE city = 'Pune';

--d⃣ Display name of the city where branch KAROLBAGH is located
SELECT city
FROM Branch
WHERE bname = 'KAROLBAGH';

--e⃣ Find the number of tuples (records) in the customer relation
```

```
SELECT COUNT(*) AS Total_Customers
FROM Customers;

-- f Delete all records of customer 'Sunil'
DELETE FROM Customers
WHERE cname = 'Sunil';

-- g Create a view on Deposit table
CREATE VIEW DepositView AS
SELECT actno, cname, bname, amount, adate
FROM Deposit;

-- Query the view
SELECT * FROM DepositView;
```

```
/* ================================================================
   PRACTICAL 12 – Banking Database (Customer & Branch Relations)
   Database Name: p12
   ================================================================ */

CREATE DATABASE p12;
USE p12;

-- STEP 1: Create Tables

CREATE TABLE Branch (
  bname VARCHAR(50) PRIMARY KEY,
  city VARCHAR(50)
);

CREATE TABLE Customers (
  cname VARCHAR(50) PRIMARY KEY,
  city VARCHAR(50)
);

CREATE TABLE Deposit (
  actno INT PRIMARY KEY,
  cname VARCHAR(50),
  bname VARCHAR(50),
  amount DECIMAL(10,2),
  adate DATE,
  FOREIGN KEY (cname) REFERENCES Customers(cname),
  FOREIGN KEY (bname) REFERENCES Branch(bname)
);

CREATE TABLE Borrow (
  loanno INT PRIMARY KEY,
  cname VARCHAR(50),
  bname VARCHAR(50),
  amount DECIMAL(10,2),
  FOREIGN KEY (cname) REFERENCES Customers(cname),
  FOREIGN KEY (bname) REFERENCES Branch(bname)
);

-- STEP 2: Sample Data

INSERT INTO Branch VALUES
('NagpurBranch', 'Nagpur'),
('BombayBranch', 'Bombay'),
('PuneBranch', 'Pune');

INSERT INTO Customers VALUES
('Anil', 'Bombay'),
('Sunil', 'Nagpur'),
('Ravi', 'Pune'),
('Isha', 'Nagpur');

INSERT INTO Deposit VALUES
(101, 'Anil', 'BombayBranch', 2000.00, '1997-01-10'),
(102, 'Sunil', 'NagpurBranch', 5000.00, '1997-02-12'),
(103, 'Ravi', 'PuneBranch', 3000.00, '1997-03-15'),
(104, 'Isha', 'NagpurBranch', 8000.00, '1997-04-05');

INSERT INTO Borrow VALUES
(201, 'Anil', 'BombayBranch', 2500.00),
(202, 'Sunil', 'NagpurBranch', 4000.00),
(203, 'Ravi', 'PuneBranch', 5000.00),
(204, 'Isha', 'NagpurBranch', 3000.00);

-- STEP 3: Queries

--1️⃣Display customer names living in city 'Bombay' and branch city 'Nagpur'
SELECT d.cname
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
JOIN Branch b ON d.bname = b.bname
WHERE c.city = 'Bombay' AND b.city = 'Nagpur';

--2️⃣Display customer names having same living city as their branch city
SELECT d.cname
FROM Deposit d
JOIN Customers c ON d.cname = c.cname
JOIN Branch b ON d.bname = b.bname
WHERE c.city = b.city;

--3️⃣Display customer names who are borrowers as well as depositors and living in city
'Nagpur'
SELECT DISTINCT c.cname
FROM Customers c
WHERE c.city = 'Nagpur'
AND c.cname IN (SELECT cname FROM Deposit)
AND c.cname IN (SELECT cname FROM Borrow);

--4️⃣Display borrower names having deposit amount > 1000 AND loan amount > 2000
SELECT DISTINCT b.cname
FROM Borrow b
JOIN Deposit d ON b.cname = d.cname
WHERE d.amount > 1000 AND b.amount > 2000;

--5️⃣Display customer names living in the city where branch of depositor 'Sunil' is located
SELECT c.cname
```

```
FROM Customers c
WHERE c.city = (
  SELECT city FROM Branch
  WHERE bname = (
    SELECT bname FROM Deposit WHERE cname = 'Sunil'
  )
);

--6️⃣Create an index on Deposit table (amount field)
CREATE INDEX idx_amount ON Deposit(amount);

-- Query to verify data
SELECT * FROM Deposit;
```

```
/* ================================================================
   PRACTICAL 13 – Book Publication Database
   Database Name: p13
   ================================================================ */

CREATE DATABASE p13;
USE p13;

-- STEP 1: Create Tables

CREATE TABLE Publisher (
  PID INT PRIMARY KEY,
  PNAME VARCHAR(50),
  ADDRESS VARCHAR(100),
  STATE VARCHAR(50),
  PHONE VARCHAR(20),
  EMAILID VARCHAR(50)
);

CREATE TABLE Book (
  ISBN VARCHAR(20) PRIMARY KEY,
  BOOK_TITLE VARCHAR(100),
  CATEGORY VARCHAR(50),
  PRICE DECIMAL(10,2),
  COPYRIGHT_DATE DATE,
  YEAR INT,
  PAGE_COUNT INT,
  PID INT,
  FOREIGN KEY (PID) REFERENCES Publisher(PID)
);

CREATE TABLE Author (
  AID INT PRIMARY KEY,
  ANAME VARCHAR(50),
  STATE VARCHAR(50),
  CITY VARCHAR(50),
  ZIP VARCHAR(10),
  PHONE VARCHAR(20),
  URL VARCHAR(100)
);

CREATE TABLE Author_Book (
  AID INT,
  ISBN VARCHAR(20),
  FOREIGN KEY (AID) REFERENCES Author(AID),
  FOREIGN KEY (ISBN) REFERENCES Book(ISBN)
);
```

```
CREATE TABLE Review (
  RID INT PRIMARY KEY,
  ISBN VARCHAR(20),
  RATING INT,
  FOREIGN KEY (ISBN) REFERENCES Book(ISBN)
);

-- STEP 2: Sample Data

INSERT INTO Publisher VALUES
(1, 'MEHTA', 'MG Road', 'Maharashtra', '9876543210', 'mehta@pub.com'),
(2, 'TATA', 'FC Road', 'Delhi', '9998887770', 'tata@pub.com');

INSERT INTO Book VALUES
('B101', 'Database System Concepts', 'Database', 500.00, '2019-01-01', 2019, 890, 1),
('B102', 'AI Revolution', 'AI', 650.00, '2020-05-01', 2020, 450, 2),
('B103', 'Web Development Made Easy', 'Web', 300.00, '2018-07-01', 2018, 250, 1);

INSERT INTO Author VALUES
(11, 'CHETAN BHAGAT', 'Maharashtra', 'Pune', '411001', '8888000011', 'chetanbhagat.com'),
(12, 'KORTH', 'Delhi', 'Delhi', '110001', '9999000022', 'korthbooks.net'),
(13, 'RAHUL MEHTA', 'Maharashtra', 'Mumbai', '400001', '7777000033', 'rahulmehta.in');

INSERT INTO Author_Book VALUES
(11, 'B103'),
(12, 'B101'),
(13, 'B102');

INSERT INTO Review VALUES
(501, 'B101', 5),
(502, 'B102', 4),
(503, 'B103', 3);

-- STEP 3: Queries

--1️⃣Retrieve city, phone, and URL of author whose name is 'CHETAN BHAGAT'
SELECT CITY, PHONE, URL
FROM Author
WHERE ANAME = 'CHETAN BHAGAT';

--2️⃣Retrieve book title, review ID, and rating of all books
SELECT b.BOOK_TITLE, r.RID, r.RATING
FROM Book b
JOIN Review r ON b.ISBN = r.ISBN;

--3️⃣Retrieve book title, price, author name, and URL for publisher 'MEHTA'
SELECT b.BOOK_TITLE, b.PRICE, a.ANAME, a.URL
FROM Book b
```

```
JOIN Publisher p ON b.PID = p.PID
JOIN Author_Book ab ON ab.ISBN = b.ISBN
JOIN Author a ON ab.AID = a.AID
WHERE p.PNAME = 'MEHTA';

--4️⃣Update the phone number of publisher 'MEHTA' to 123456
UPDATE Publisher
SET PHONE = '123456'
WHERE PNAME = 'MEHTA';

--5️⃣Calculate and display average, maximum, and minimum price of each publisher
SELECT p.PNAME,
       AVG(b.PRICE) AS Avg_Price,
       MAX(b.PRICE) AS Max_Price,
       MIN(b.PRICE) AS Min_Price
FROM Publisher p
JOIN Book b ON p.PID = b.PID
GROUP BY p.PNAME;

--6️⃣Delete details of all books having a page count less than 100
DELETE FROM Book
WHERE PAGE_COUNT < 100;

--7️⃣Retrieve details of authors residing in city 'Pune' whose name begins with 'C'
SELECT *
FROM Author
WHERE CITY = 'Pune' AND ANAME LIKE 'C%';

--8️⃣Retrieve details of authors residing in same city as 'KORTH'
SELECT *
FROM Author
WHERE CITY = (SELECT CITY FROM Author WHERE ANAME = 'KORTH');

--9️⃣Create a procedure to update the value of page count of a book for given ISBN
DELIMITER //
CREATE PROCEDURE UpdatePageCount(IN book_isbn VARCHAR(20), IN new_pages
INT)
BEGIN
  UPDATE Book
  SET PAGE_COUNT = new_pages
  WHERE ISBN = book_isbn;
END //
DELIMITER ;

-- Example Execution:
-- CALL UpdatePageCount('B101', 920);

-- 🔟 Create a function that returns the price of a book with a given ISBN
```

```
DELIMITER //
CREATE FUNCTION GetBookPrice(book_isbn VARCHAR(20))
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
  DECLARE book_price DECIMAL(10,2);
  SELECT PRICE INTO book_price FROM Book WHERE ISBN = book_isbn;
  RETURN book_price;
END //
DELIMITER ;

-- Example Execution:
-- SELECT GetBookPrice('B102') AS Price;
```

```
/* ===================================================================
   PRACTICAL 14 – PL/SQL Blocks (Attendance & Banking Exception)
   Database Name: p14
   =================================================================== */

CREATE DATABASE p14;
USE p14;

-- ===================================================================
-- PART (a): Attendance Check – “Term Granted” or “Term Not Granted”
-- ===================================================================

-- STEP 1: Create Table
CREATE TABLE Stud (
  Roll INT PRIMARY KEY,
  Att DECIMAL(5,2),
  Status CHAR(2)
);

-- STEP 2: Insert Sample Data
INSERT INTO Stud VALUES
(1, 80.00, NULL),
(2, 60.00, NULL),
(3, 75.00, NULL);

-- STEP 3: PL/SQL Block for Attendance Verification
-- Requirement:
-- If attendance < 75% → Display “Term not granted”, set Status = 'D'
-- Else → Display “Term granted”, set Status = 'ND'

DELIMITER //
CREATE PROCEDURE CheckAttendance(IN roll_no INT)
BEGIN
  DECLARE att_percent DECIMAL(5,2);

  SELECT Att INTO att_percent FROM Stud WHERE Roll = roll_no;

  IF att_percent < 75 THEN
    UPDATE Stud SET Status = 'D' WHERE Roll = roll_no;
    SELECT CONCAT('Roll No ', roll_no, ': Term not granted') AS Message;
  ELSE
    UPDATE Stud SET Status = 'ND' WHERE Roll = roll_no;
    SELECT CONCAT('Roll No ', roll_no, ': Term granted') AS Message;
  END IF;
END //
DELIMITER ;

-- Example Execution:
```

```sql
-- CALL CheckAttendance(2);
-- CALL CheckAttendance(1);

-- ==========================================================
-- PART (b): Banking Exception – Withdrawal Handling
-- ==========================================================

-- STEP 1: Create Account Master Table
CREATE TABLE Account_Master (
  Acc_No INT PRIMARY KEY,
  Cust_Name VARCHAR(50),
  Balance DECIMAL(10,2)
);

-- STEP 2: Insert Sample Data
INSERT INTO Account_Master VALUES
(101, 'Anil', 10000.00),
(102, 'Sunil', 5000.00),
(103, 'Ravi', 2000.00);

-- STEP 3: PL/SQL Block with User-Defined Exception
-- Requirement:
-- If withdrawal > balance → Raise error "Insufficient Balance"
-- Else → Deduct amount and update balance

DELIMITER //
CREATE PROCEDURE WithdrawAmount(IN acc INT, IN amt DECIMAL(10,2))
BEGIN
  DECLARE curr_balance DECIMAL(10,2);
  DECLARE insufficient_balance CONDITION FOR SQLSTATE '45000';

  SELECT Balance INTO curr_balance FROM Account_Master WHERE Acc_No = acc;

  IF curr_balance < amt THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = '❌ Insufficient Balance!';
  ELSE
    UPDATE Account_Master SET Balance = Balance - amt WHERE Acc_No = acc;
    SELECT CONCAT('✅ Withdrawal successful! New Balance = ', Balance) AS Message
    FROM Account_Master WHERE Acc_No = acc;
  END IF;
END //
DELIMITER ;

-- Example Execution:
-- CALL WithdrawAmount(101, 3000);   -- ✅ Successful
-- CALL WithdrawAmount(103, 5000);   -- ❌ Insufficient Balance
```

```
/* ============================================================
   PRACTICAL 15 – PL/SQL (User Defined Exception & Fine Calculation)
   Database Name: p15
   ============================================================ */

CREATE DATABASE p15;
USE p15;

-- ============================================================
-- PART (a): User Defined Exception (Business Rule Violation)
-- ============================================================
-- Business Rule:
-- The 'bal_due' field in client_master must not be less than 0.
-- If violated → Raise a user-defined exception.

-- STEP 1: Create Table
CREATE TABLE Client_Master (
  Client_ID INT PRIMARY KEY,
  Client_Name VARCHAR(50),
  Bal_Due DECIMAL(10,2)
);

-- STEP 2: Insert Sample Data
INSERT INTO Client_Master VALUES
(1, 'Anil', 2000.00),
(2, 'Sunil', -500.00),
(3, 'Ravi', 1000.00);

-- STEP 3: PL/SQL Block for Checking Business Rule
DELIMITER //
CREATE PROCEDURE CheckBalance()
BEGIN
  DECLARE v_client INT;
  DECLARE v_bal DECIMAL(10,2);
  DECLARE done INT DEFAULT 0;
  DECLARE invalid_balance CONDITION FOR SQLSTATE '45000';
  DECLARE cur CURSOR FOR SELECT Client_ID, Bal_Due FROM Client_Master;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO v_client, v_bal;
    IF done THEN
      LEAVE read_loop;
    END IF;

    IF v_bal < 0 THEN
   SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = CONCAT('❌ Error: Negative
Balance for Client ID ', v_client);
    END IF;
  END LOOP;
  CLOSE cur;
END //
DELIMITER ;

-- Example Execution:
-- CALL CheckBalance();

-- =============================================================
-- PART (b): Fine Calculation Based on Book Return Delay
-- =============================================================

-- STEP 1: Create Tables
CREATE TABLE Borrow (
  Roll_No INT,
  Name VARCHAR(50),
  DateOfIssue DATE,
  NameOfBook VARCHAR(50),
  Status CHAR(1)
);

CREATE TABLE Fine (
  Roll_No INT,
  Date_Return DATE,
  Amt DECIMAL(10,2)
);

-- STEP 2: Insert Sample Data
INSERT INTO Borrow VALUES
(1, 'Ravi', '2025-09-01', 'DBMS Concepts', 'I'),
(2, 'Neha', '2025-08-15', 'AI Basics', 'I'),
(3, 'Kiran', '2025-08-28', 'Web Tech', 'I');

-- STEP 3: PL/SQL Block for Fine Calculation
-- Logic:
-- Days between Issue and Return:
--  0–14 days → No fine
-- 15–30 days → ₹5 per day
-- >30 days   → ₹50 per day
-- After submission → Status changes from 'I' to 'R'

DELIMITER //
CREATE PROCEDURE ReturnBook(IN roll INT, IN book_name VARCHAR(50), IN
return_date DATE)
BEGIN
```

```
  DECLARE issue_date DATE;
  DECLARE days_diff INT;
  DECLARE fine_amt DECIMAL(10,2);

  -- Get Issue Date
  SELECT DateOfIssue INTO issue_date
  FROM Borrow
  WHERE Roll_No = roll AND NameOfBook = book_name;

  -- Calculate Days Difference
  SET days_diff = DATEDIFF(return_date, issue_date);

  -- Fine Logic
  IF days_diff BETWEEN 15 AND 30 THEN
    SET fine_amt = days_diff * 5;
  ELSEIF days_diff > 30 THEN
    SET fine_amt = days_diff * 50;
  ELSE
    SET fine_amt = 0;
  END IF;

  -- Update Borrow Status
  UPDATE Borrow
  SET Status = 'R'
  WHERE Roll_No = roll AND NameOfBook = book_name;

  -- Insert Fine Record if applicable
  IF fine_amt > 0 THEN
    INSERT INTO Fine VALUES (roll, return_date, fine_amt);
    SELECT CONCAT('📚 Fine applied: ₹', fine_amt) AS Message;
  ELSE
    SELECT '✅ No fine, returned on time!' AS Message;
  END IF;
END //
DELIMITER ;

-- Example Executions:
-- CALL ReturnBook(1, 'DBMS Concepts', '2025-09-20');   -- 19 days → ₹95 fine
-- CALL ReturnBook(2, 'AI Basics', '2025-10-01');        -- 47 days → ₹2350 fine
-- CALL ReturnBook(3, 'Web Tech', '2025-09-10');         -- 13 days → No fine
```

```
/* ==========================================================
   PRACTICAL 16 – PL/SQL (Cursors: Implicit & Explicit)
   Database Name: p16
   ========================================================== */

CREATE DATABASE p16;
USE p16;

-- ==========================================================
-- PART (a): Implicit Cursor Example – Activate Dormant Accounts
-- ==========================================================
-- Requirement:
-- If account is inactive for over 365 days → activate it.
-- Display number of rows updated using implicit cursor attributes.

-- STEP 1: Create Table
CREATE TABLE Account (
  Acc_No INT PRIMARY KEY,
  Cust_Name VARCHAR(50),
  Last_Transaction DATE,
  Status VARCHAR(10)
);

-- STEP 2: Insert Sample Data
INSERT INTO Account VALUES
(101, 'Anil', '2023-09-01', 'Inactive'),
(102, 'Sunil', '2024-08-15', 'Active'),
(103, 'Ravi', '2023-06-20', 'Inactive'),
(104, 'Neha', '2022-10-10', 'Inactive');

-- STEP 3: Implicit Cursor Block
DELIMITER //
CREATE PROCEDURE ActivateAccounts()
BEGIN
  UPDATE Account
  SET Status = 'Active'
  WHERE DATEDIFF(CURDATE(), Last_Transaction) > 365;

  IF ROW_COUNT() > 0 THEN
    SELECT CONCAT('✅ ', ROW_COUNT(), ' accounts activated successfully!') AS
Message;
  ELSE
    SELECT 'ℹ️ No accounts required activation.' AS Message;
  END IF;
END //
DELIMITER ;

-- Example Execution:
```

```
-- CALL ActivateAccounts();

-- =============================================================
-- PART (b): Implicit Cursor Example – Salary Increment Below Average
-- =============================================================
-- Requirement:
-- Employees with salary < avg salary get +10% increment.
-- Each update is recorded in increment_salary table.

-- STEP 1: Create Tables
CREATE TABLE Employee (
  Emp_ID INT PRIMARY KEY,
  Emp_Name VARCHAR(50),
  Salary DECIMAL(10,2)
);

CREATE TABLE Increment_Salary (
  Emp_ID INT,
  Old_Salary DECIMAL(10,2),
  New_Salary DECIMAL(10,2),
  Increment_Date DATE
);

-- STEP 2: Insert Sample Data
INSERT INTO Employee VALUES
(1, 'Ravi', 40000.00),
(2, 'Neha', 55000.00),
(3, 'Kiran', 30000.00),
(4, 'Isha', 60000.00);

-- STEP 3: Procedure with Implicit Cursor
DELIMITER //
CREATE PROCEDURE UpdateLowSalary()
BEGIN
  DECLARE avg_sal DECIMAL(10,2);
  SET avg_sal = (SELECT AVG(Salary) FROM Employee);

  UPDATE Employee
  SET Salary = Salary + (Salary * 0.10)
  WHERE Salary < avg_sal;

  INSERT INTO Increment_Salary
  SELECT Emp_ID, Salary / 1.10, Salary, CURDATE()
  FROM Employee
  WHERE Salary / 1.10 < avg_sal;

  SELECT CONCAT('✅ Salary updated for ', ROW_COUNT(), ' employees below average.')
AS Message;
```

```
END //
DELIMITER ;

-- Example Execution:
-- CALL UpdateLowSalary();

-- ============================================================
-- PART (c): Explicit Cursor Example – Attendance-Based Detention
-- ============================================================
-- Requirement:
-- Students with attendance < 75% → Mark as ‘D’ (Detained)
-- Each such update is logged into D_Stud table.

-- STEP 1: Create Tables
CREATE TABLE Stud21 (
  Roll INT PRIMARY KEY,
  Att DECIMAL(5,2),
  Status CHAR(1)
);

CREATE TABLE D_Stud (
  Roll INT,
  Att DECIMAL(5,2),
  Detained_Date DATE
);

-- STEP 2: Insert Sample Data
INSERT INTO Stud21 VALUES
(1, 80.00, NULL),
(2, 60.00, NULL),
(3, 70.00, NULL),
(4, 85.00, NULL);

-- STEP 3: Explicit Cursor Block
DELIMITER //
CREATE PROCEDURE MarkDetained()
BEGIN
  DECLARE v_roll INT;
  DECLARE v_att DECIMAL(5,2);
  DECLARE done INT DEFAULT 0;

  DECLARE cur CURSOR FOR SELECT Roll, Att FROM Stud21;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO v_roll, v_att;
    IF done THEN
```

```
      LEAVE read_loop;
    END IF;

    IF v_att < 75 THEN
      UPDATE Stud21 SET Status = 'D' WHERE Roll = v_roll;
      INSERT INTO D_Stud VALUES (v_roll, v_att, CURDATE());
    ELSE
      UPDATE Stud21 SET Status = 'N' WHERE Roll = v_roll;
    END IF;
  END LOOP;
  CLOSE cur;

  SELECT '✅ Detained students marked and logged successfully.' AS Message;
END //
DELIMITER ;

-- Example Execution:
-- CALL MarkDetained();
-- SELECT * FROM D_Stud;
-- SELECT * FROM Stud21;
```

```
/* ==========================================================
   PRACTICAL 17 – PL/SQL (Parameterized Cursors)
   Database Name: p17
   ========================================================== */

CREATE DATABASE p17;
USE p17;

-- ==========================================================
-- PART (a): Implicit Cursor Example – Activate Dormant Accounts
-- ==========================================================
-- Requirement:
-- If account is inactive for over 365 days → activate it.
-- Display number of rows updated using implicit cursor attributes.

-- STEP 1: Create Table
CREATE TABLE Account (
  Acc_No INT PRIMARY KEY,
  Cust_Name VARCHAR(50),
  Last_Transaction DATE,
  Status VARCHAR(10)
);

-- STEP 2: Insert Sample Data
INSERT INTO Account VALUES
(201, 'Ravi', '2023-09-01', 'Inactive'),
(202, 'Anita', '2022-10-10', 'Inactive'),
(203, 'Nilesh', '2024-07-15', 'Active'),
(204, 'Seema', '2023-06-01', 'Inactive');

-- STEP 3: Implicit Cursor Block
DELIMITER //
CREATE PROCEDURE ActivateDormantAccounts()
BEGIN
  UPDATE Account
  SET Status = 'Active'
  WHERE DATEDIFF(CURDATE(), Last_Transaction) > 365;

  IF ROW_COUNT() > 0 THEN
    SELECT CONCAT('✅ ', ROW_COUNT(), ' dormant accounts activated!') AS Message;
  ELSE
    SELECT 'ℹ️ No dormant accounts found for activation.' AS Message;
  END IF;
END //
DELIMITER ;

-- Example Execution:
-- CALL ActivateDormantAccounts();
-- ============================================================
-- PART (b): Parameterized Cursor – Merge New & Old Roll Call
-- ============================================================
-- Requirement:
-- Merge data from N_RollCall into O_RollCall.
-- If record already exists → skip it.

-- STEP 1: Create Tables
CREATE TABLE O_RollCall (
  RollNo INT PRIMARY KEY,
  Name VARCHAR(50),
  Att_Per DECIMAL(5,2)
);

CREATE TABLE N_RollCall (
  RollNo INT PRIMARY KEY,
  Name VARCHAR(50),
  Att_Per DECIMAL(5,2)
);

-- STEP 2: Insert Sample Data
INSERT INTO O_RollCall VALUES
(1, 'Ravi', 90.00),
(2, 'Sneha', 85.00);

INSERT INTO N_RollCall VALUES
(2, 'Sneha', 88.00),
(3, 'Ajay', 92.00),
(4, 'Neha', 80.00);

-- STEP 3: Parameterized Cursor Procedure
DELIMITER //
CREATE PROCEDURE MergeRollCall()
BEGIN
  DECLARE v_roll INT;
  DECLARE v_name VARCHAR(50);
  DECLARE v_att DECIMAL(5,2);
  DECLARE done INT DEFAULT 0;

  DECLARE cur CURSOR FOR SELECT RollNo, Name, Att_Per FROM N_RollCall;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO v_roll, v_name, v_att;
    IF done THEN
      LEAVE read_loop;
    END IF;

    IF NOT EXISTS (SELECT * FROM O_RollCall WHERE RollNo = v_roll) THEN
      INSERT INTO O_RollCall VALUES (v_roll, v_name, v_att);
    END IF;
  END LOOP;
  CLOSE cur;

  SELECT '✅ RollCall merged successfully.' AS Message;
END //
DELIMITER ;

-- Example Execution:
-- CALL MergeRollCall();
-- SELECT * FROM O_RollCall;

-- ============================================================
-- PART (c): Parameterized Cursor – Dept Wise Average Salary
-- ============================================================
-- Requirement:
-- Department-wise average salary to be inserted into dept_salary table.

-- STEP 1: Create Tables
CREATE TABLE EMP (
  E_No INT PRIMARY KEY,
  D_No INT,
  Salary DECIMAL(10,2)
);

CREATE TABLE Dept_Salary (
  D_No INT,
  Avg_Salary DECIMAL(10,2)
);

-- STEP 2: Insert Sample Data
INSERT INTO EMP VALUES
(1, 10, 45000.00),
(2, 10, 50000.00),
(3, 20, 35000.00),
(4, 20, 30000.00),
(5, 30, 60000.00);

-- STEP 3: Parameterized Cursor Procedure
DELIMITER //
CREATE PROCEDURE DeptWiseAvgSalary()
BEGIN
  DECLARE v_dno INT;
  DECLARE v_avg DECIMAL(10,2);
  DECLARE done INT DEFAULT 0;

  DECLARE cur CURSOR FOR
    SELECT D_No, AVG(Salary) AS avg_sal FROM EMP GROUP BY D_No;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur;
  read_loop: LOOP
    FETCH cur INTO v_dno, v_avg;
    IF done THEN
      LEAVE read_loop;
    END IF;

    INSERT INTO Dept_Salary VALUES (v_dno, v_avg);
  END LOOP;
  CLOSE cur;

  SELECT '✅ Department-wise average salary inserted successfully.' AS Message;
END //
DELIMITER ;

-- Example Execution:
-- CALL DeptWiseAvgSalary();
-- SELECT * FROM Dept_Salary;
```

```
/* ============================================================
   PRACTICAL 18 – PL/SQL (Triggers)
   Database Name: p18
   ============================================================ */

CREATE DATABASE p18;
USE p18;

-- ============================================================
-- PART (a): Update & Delete Trigger – Audit Trail for Client Records
-- ============================================================
-- Requirement:
-- Whenever a record in clientmstr table is UPDATED or DELETED,
-- the old values must be stored in audit_trade table.

-- STEP 1: Create Tables
CREATE TABLE ClientMstr (
  Client_ID INT PRIMARY KEY,
  Client_Name VARCHAR(50),
  City VARCHAR(50),
  Bal_Due DECIMAL(10,2)
);

CREATE TABLE Audit_Trade (
  Action_Type VARCHAR(10),
  Client_ID INT,
  Client_Name VARCHAR(50),
  City VARCHAR(50),
  Bal_Due DECIMAL(10,2),
  Action_Date DATETIME
);

-- STEP 2: Insert Sample Data
INSERT INTO ClientMstr VALUES
(1, 'Ravi', 'Pune', 5000.00),
(2, 'Neha', 'Mumbai', 2500.00),
(3, 'Amit', 'Nashik', 7000.00);

-- STEP 3: ROW LEVEL TRIGGERS
DELIMITER //
CREATE TRIGGER trg_ClientMstr_Update
BEFORE UPDATE ON ClientMstr
FOR EACH ROW
BEGIN
  INSERT INTO Audit_Trade VALUES
  ('UPDATE', OLD.Client_ID, OLD.Client_Name, OLD.City, OLD.Bal_Due, NOW());
END //
DELIMITER ;
DELIMITER //
CREATE TRIGGER trg_ClientMstr_Delete
BEFORE DELETE ON ClientMstr
FOR EACH ROW
BEGIN
  INSERT INTO Audit_Trade VALUES
  ('DELETE', OLD.Client_ID, OLD.Client_Name, OLD.City, OLD.Bal_Due, NOW());
END //
DELIMITER ;

-- STEP 4: Statement-Level Demonstration (optional)
-- Updating and deleting records to trigger actions
UPDATE ClientMstr SET Bal_Due = 6000 WHERE Client_ID = 1;
DELETE FROM ClientMstr WHERE Client_ID = 2;

-- Example Verification:
-- SELECT * FROM ClientMstr;
-- SELECT * FROM Audit_Trade;

-- ============================================================
-- PART (b): Before Trigger – Salary Validation & Tracking
-- ============================================================
-- Requirement:
-- For table Emp, if salary < 50000 during INSERT or UPDATE,
-- the operation is rejected, and attempted values are stored in Tracking table.

-- STEP 1: Create Tables
CREATE TABLE Emp (
  E_No INT PRIMARY KEY,
  E_Name VARCHAR(50),
  Salary DECIMAL(10,2)
);

CREATE TABLE Tracking (
  E_No INT,
  Salary DECIMAL(10,2),
  Attempt_Date DATETIME
);

-- STEP 2: Triggers for Validation
DELIMITER //
CREATE TRIGGER trg_Emp_Insert
BEFORE INSERT ON Emp
FOR EACH ROW
BEGIN
  IF NEW.Salary < 50000 THEN
    INSERT INTO Tracking VALUES (NEW.E_No, NEW.Salary, NOW());
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = '❌ Insert Rejected: Salary cannot be less than 50000.';
  END IF;
END //
DELIMITER ;

DELIMITER //
CREATE TRIGGER trg_Emp_Update
BEFORE UPDATE ON Emp
FOR EACH ROW
BEGIN
  IF NEW.Salary < 50000 THEN
    INSERT INTO Tracking VALUES (NEW.E_No, NEW.Salary, NOW());
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = '❌ Update Rejected: Salary cannot be less than 50000.';
  END IF;
END //
DELIMITER ;

-- STEP 3: Insert Sample Data (Valid and Invalid)
INSERT INTO Emp VALUES (101, 'Anil', 55000.00); -- ✅ Valid
-- INSERT INTO Emp VALUES (102, 'Sneha', 45000.00); -- ❌ Will be rejected

-- STEP 4: Test Update Trigger
UPDATE Emp SET Salary = 40000 WHERE E_No = 101; -- ❌ Will be rejected
UPDATE Emp SET Salary = 60000 WHERE E_No = 101; -- ✅ Valid

-- Example Verification:
-- SELECT * FROM Emp;
-- SELECT * FROM Tracking;

-- ============================================================
-- END OF PRACTICAL 18
-- ============================================================
/* ==========================================================
   PRACTICAL 19 – MongoDB (Teachers & Students)
   Database Name: p19
   ========================================================== */

/* ==========================================================
   STEP 1: Create Database and Collections
   ========================================================== */
use p19;

db.createCollection("Teachers");
db.createCollection("Students");

/* ==========================================================
   STEP 2: Insert Sample Documents
   ========================================================== */
db.Teachers.insertMany([
  { Tname: "Praveen", dno: 101, dname: "COMP", experience: 5, salary: 15000,
date_of_joining: "2018-06-15" },
  { Tname: "Rajesh", dno: 102, dname: "IT", experience: 7, salary: 12000, date_of_joining:
"2017-08-10" },
  { Tname: "Smita", dno: 103, dname: "E&TC", experience: 10, salary: 20000,
date_of_joining: "2016-03-05" },
  { Tname: "Sonal", dno: 104, dname: "COMP", experience: 8, salary: 25000,
date_of_joining: "2015-02-22" },
  { Tname: "Amit", dno: 105, dname: "IT", experience: 4, salary: 9500, date_of_joining:
"2020-07-19" }
]);

db.Students.insertMany([
  { Sname: "Ravi", roll_no: 1, class: "TE COMP" },
  { Sname: "Neha", roll_no: 2, class: "SE IT" },
  { Sname: "Priya", roll_no: 3, class: "BE COMP" },
  { Sname: "Ankit", roll_no: 4, class: "TE E&TC" }
]);

/* ==========================================================
   STEP 3: Queries
   ========================================================== */

/* 1. Find the information about all teachers */
db.Teachers.find().pretty();

/* 2. Find the information about all teachers of computer department */
db.Teachers.find({ dname: "COMP" }).pretty();

/* 3. Find the information about all teachers of computer, IT, and E&TC department */
db.Teachers.find({ dname: { $in: ["COMP", "IT", "E&TC"] } }).pretty();
```

```
/* 4. Find the information about all teachers of computer, IT, and E&TC department
      having salary greater than or equal to 10000/- */
db.Teachers.find({
  dname: { $in: ["COMP", "IT", "E&TC"] },
  salary: { $gte: 10000 }
}).pretty();

/* 5. Find the student information having roll_no = 2 or Sname = 'xyz' */
db.Students.find({
  $or: [{ roll_no: 2 }, { Sname: "xyz" }]
}).pretty();

/* 6. Update the experience of teacher 'Praveen' to 10 years.
      If entry not available, insert new document (upsert). */
db.Teachers.updateOne(
  { Tname: "Praveen" },
  { $set: { experience: 10 } },
  { upsert: true }
);

/* 7. Update the department of all teachers working in IT department to COMP */
db.Teachers.updateMany(
  { dname: "IT" },
  { $set: { dname: "COMP" } }
);

/* 8. Find the teachers' name and their experience from teachers collection */
db.Teachers.find({}, { _id: 0, Tname: 1, experience: 1 }).pretty();

/* 9. Using save() method insert one entry in department collection */
db.createCollection("Department");
db.Department.save({
  dno: 201,
  dname: "MECH",
  location: "Pune"
});

/* 10. Using save() method change the dept of teacher 'Rajesh' to IT */
db.Teachers.save({
  Tname: "Rajesh",
  dno: 102,
  dname: "IT",
  experience: 7,
  salary: 12000,
  date_of_joining: "2017-08-10"
});
```

```
/* 11. Delete all the documents from teachers collection having IT dept */
db.Teachers.deleteMany({ dname: "IT" });

/* 12. Display with pretty() method, the first 3 documents in teachers collection in ascending
order */
db.Teachers.find().sort({ Tname: 1 }).limit(3).pretty();

/* ============================================================
   END OF PRACTICAL 19
   ============================================================ */
```

```
/* ================================================================
   PRACTICAL 20 – MongoDB (Teachers & Students)
   Database Name: p20
   ================================================================ */

/* ================================================================
   STEP 1: Create Database and Collections
   ================================================================ */
use p20;

db.createCollection("Teachers");
db.createCollection("Students");

/* ================================================================
   STEP 2: Insert Sample Documents
   ================================================================ */
db.Teachers.insertMany([
  { Tname: "Praveen", dno: 101, dname: "COMP", experience: 6, salary: 18000,
date_of_joining: "2019-05-10" },
  { Tname: "Rajesh", dno: 102, dname: "IT", experience: 8, salary: 27000, date_of_joining:
"2016-02-25" },
  { Tname: "Smita", dno: 103, dname: "E&TC", experience: 12, salary: 35000,
date_of_joining: "2013-11-12" },
  { Tname: "Sonal", dno: 104, dname: "COMP", experience: 10, salary: 40000,
date_of_joining: "2015-07-20" },
  { Tname: "Amit", dno: 105, dname: "IT", experience: 4, salary: 25000, date_of_joining:
"2020-08-19" }
]);

db.Students.insertMany([
  { Sname: "Ravi", roll_no: 21, class: "TE COMP" },
  { Sname: "Neha", roll_no: 22, class: "SE IT" },
  { Sname: "Priya", roll_no: 23, class: "BE COMP" },
  { Sname: "Ankit", roll_no: 25, class: "TE E&TC" },
  { Sname: "xyz", roll_no: 26, class: "FE COMP" }
]);

/* ================================================================
   STEP 3: Queries
   ================================================================ */

/* 1. Find the information about two teachers */
db.Teachers.find().limit(2).pretty();

/* 2. Find the information about all teachers of computer department */
db.Teachers.find({ dname: "COMP" }).pretty();

/* 3. Find the information about all teachers of computer, IT, and E&TC department */
```

```
db.Teachers.find({ dname: { $in: ["COMP", "IT", "E&TC"] } }).pretty();

/* 4. Find the information about all teachers of computer, IT, and E&TC department
      having salary greater than or equal to 25000/- */
db.Teachers.find({
  dname: { $in: ["COMP", "IT", "E&TC"] },
  salary: { $gte: 25000 }
}).pretty();

/* 5. Find the student information having roll_no = 25 or Sname = 'xyz' */
db.Students.find({
  $or: [{ roll_no: 25 }, { Sname: "xyz" }]
}).pretty();

/* 6. Update the experience of teacher 'Praveen' to 10 years.
      If entry not available, insert new document (upsert). */
db.Teachers.updateOne(
  { Tname: "Praveen" },
  { $set: { experience: 10 } },
  { upsert: true }
);

/* 7. Update the department of all teachers working in IT department to COMP */
db.Teachers.updateMany(
  { dname: "IT" },
  { $set: { dname: "COMP" } }
);

/* 8. Find the teachers' name and their experience from teachers collection */
db.Teachers.find({}, { _id: 0, Tname: 1, experience: 1 }).pretty();

/* 9. Using Save() method insert one entry in department collection */
db.createCollection("Department");
db.Department.save({
  dno: 201,
  dname: "MECH",
  location: "Mumbai"
});

/* 10. Using Save() method change the dept of teacher 'Rajesh' to IT */
db.Teachers.save({
  Tname: "Rajesh",
  dno: 102,
  dname: "IT",
  experience: 8,
  salary: 27000,
  date_of_joining: "2016-02-25"
});
```

```
/* 11. Delete all the documents from teachers collection having IT dept */
db.Teachers.deleteMany({ dname: "IT" });

/* 12. Display with pretty() method, the first 5 documents in teachers collection in ascending
order */
db.Teachers.find().sort({ Tname: 1 }).limit(5).pretty();

/* ============================================================
   END OF PRACTICAL 20
   ============================================================ */
```

```
/* ==============================================================
   PRACTICAL 21 – MongoDB (Aggregation Functions)
   Database Name: p21
   ============================================================== */

/* ==============================================================
   STEP 1: Create Database and Collections
   ============================================================== */
use p21;

db.createCollection("Teachers");
db.createCollection("Students");

/* ==============================================================
   STEP 2: Insert Sample Documents
   ============================================================== */
db.Teachers.insertMany([
  { Tname: "Praveen", dno: 101, dname: "COMP", experience: 6, salary: 20000,
date_of_joining: "2019-06-10" },
  { Tname: "Rajesh", dno: 102, dname: "IT", experience: 8, salary: 25000, date_of_joining:
"2018-02-25" },
  { Tname: "Smita", dno: 103, dname: "E&TC", experience: 10, salary: 32000,
date_of_joining: "2015-11-12" },
  { Tname: "Sonal", dno: 104, dname: "COMP", experience: 12, salary: 45000,
date_of_joining: "2012-07-20" },
  { Tname: "Amit", dno: 105, dname: "IT", experience: 4, salary: 18000, date_of_joining:
"2021-03-19" }
]);

db.Students.insertMany([
  { Sname: "Ravi", roll_no: 1, class: "TE COMP" },
  { Sname: "Neha", roll_no: 2, class: "SE IT" },
  { Sname: "Priya", roll_no: 3, class: "BE COMP" },
  { Sname: "Ankit", roll_no: 4, class: "TE E&TC" },
  { Sname: "xyz", roll_no: 5, class: "FE COMP" }
]);

/* ==============================================================
   STEP 3: Queries
   ============================================================== */

/* 1. Find the information about all teachers */
db.Teachers.find().pretty();

/* 2. Find the average salary of teachers in the computer department */
db.Teachers.aggregate([
  { $match: { dname: "COMP" } },
  { $group: { _id: "$dname", Average_Salary: { $avg: "$salary" } } }
]);

/* 3. Find the minimum and maximum salary of E&TC department teachers */
db.Teachers.aggregate([
  { $match: { dname: "E&TC" } },
  {
    $group: {
      _id: "$dname",
      Minimum_Salary: { $min: "$salary" },
      Maximum_Salary: { $max: "$salary" }
    }
  }
]);

/* 4. Find teachers of computer, IT, and E&TC departments having salary >= 10000 */
db.Teachers.find({
  dname: { $in: ["COMP", "IT", "E&TC"] },
  salary: { $gte: 10000 }
}).pretty();

/* 5. Find student info having roll_no = 2 or Sname = 'xyz' */
db.Students.find({
  $or: [{ roll_no: 2 }, { Sname: "xyz" }]
}).pretty();

/* 6. Update the experience of teacher 'Praveen' to 10 years (upsert if not found) */
db.Teachers.updateOne(
  { Tname: "Praveen" },
  { $set: { experience: 10 } },
  { upsert: true }
);

/* 7. Update the department of all teachers working in IT department to COMP */
db.Teachers.updateMany(
  { dname: "IT" },
  { $set: { dname: "COMP" } }
);

/* 8. Find teachers' names and their experience */
db.Teachers.find({}, { _id: 0, Tname: 1, experience: 1 }).pretty();

/* 9. Using save() method insert one entry in Department collection */
db.createCollection("Department");
db.Department.save({
  dno: 201,
  dname: "MECH",
  location: "Nashik"
});
```

```
/* 10. Find the total salary of all teachers */
db.Teachers.aggregate([
  { $group: { _id: null, Total_Salary: { $sum: "$salary" } } }
]);

/* ==========================================================
   END OF PRACTICAL 21
   ========================================================== */
```

```
/* ===============================================================
   PRACTICAL 22 – MongoDB (Aggregation, Indexing)
   Database Name: p22
   =============================================================== */

/* ===============================================================
   STEP 1: Create Database and Collections
   =============================================================== */
use p22;

db.createCollection("Teachers");
db.createCollection("Students");

/* ===============================================================
   STEP 2: Insert Sample Documents
   =============================================================== */
db.Teachers.insertMany([
  { Tname: "Praveen", dno: 101, dname: "COMP", experience: 6, salary: 20000,
date_of_joining: "2019-06-10" },
  { Tname: "Rajesh", dno: 102, dname: "IT", experience: 8, salary: 25000, date_of_joining:
"2018-02-25" },
  { Tname: "Smita", dno: 103, dname: "E&TC", experience: 10, salary: 32000,
date_of_joining: "2015-11-12" },
  { Tname: "Sonal", dno: 104, dname: "COMP", experience: 12, salary: 45000,
date_of_joining: "2012-07-20" },
  { Tname: "Amit", dno: 105, dname: "IT", experience: 4, salary: 18000, date_of_joining:
"2021-03-19" }
]);

db.Students.insertMany([
  { Sname: "Ravi", roll_no: 1, class: "TE COMP" },
  { Sname: "Neha", roll_no: 2, class: "SE IT" },
  { Sname: "Priya", roll_no: 3, class: "BE COMP" },
  { Sname: "Ankit", roll_no: 4, class: "TE E&TC" }
]);

/* ===============================================================
   STEP 3: Aggregation Queries
   =============================================================== */

/* 1. Display department-wise average salary */
db.Teachers.aggregate([
  { $group: { _id: "$dname", Average_Salary: { $avg: "$salary" } } }
]);

/* 2. Display number of employees working in each department */
db.Teachers.aggregate([
  { $group: { _id: "$dname", Total_Employees: { $sum: 1 } } }
]);

/* 3. Display department-wise total salary of departments having total salary >= 50000 */
db.Teachers.aggregate([
  { $group: { _id: "$dname", Total_Salary: { $sum: "$salary" } } },
  { $match: { Total_Salary: { $gte: 50000 } } }
]);

/* 4. Use operators like max, min, avg, etc. */
db.Teachers.aggregate([
  {
    $group: {
      _id: "$dname",
      Min_Salary: { $min: "$salary" },
      Max_Salary: { $max: "$salary" },
      Avg_Salary: { $avg: "$salary" }
    }
  }
]);

/* =============================================================
   STEP 4: Indexing Operations
   ============================================================= */

/* 5. Create unique index on any field (e.g., Tname) */
db.Teachers.createIndex({ Tname: 1 }, { unique: true });

/* 6. Create compound index on multiple fields (e.g., dname + salary) */
db.Teachers.createIndex({ dname: 1, salary: -1 });

/* 7. Show all indexes created in the database p22 */
db.getCollectionNames().forEach(function (col) {
  print("Indexes for collection:", col);
  printjson(db[col].getIndexes());
});

/* 8. Show all indexes created in Teachers collection */
db.Teachers.getIndexes();

/* =============================================================
   END OF PRACTICAL 22
   ============================================================= */
```

```
/* ==========================================================
   PRACTICAL 23 – MongoDB (Indexes and Performance)
   Database Name: p23
   ========================================================== */

/* ==========================================================
   STEP 1: Create Database and Import Data
   ========================================================== */
use p23;

/* Assume zip.json has been imported:
   mongoimport --db p23 --collection zip --file zip.json --jsonArray
*/

/* ==========================================================
   STEP 2: Create Indexes
   ========================================================== */

/* 1. Create a single-field index (on population) */
db.zip.createIndex({ pop: 1 });

/* 2. Create a composite index (on state and city) */
db.zip.createIndex({ state: 1, city: 1 });

/* 3. Create a multikey index (on location coordinates array if available) */
db.zip.createIndex({ loc: 1 });

/* ==========================================================
   STEP 3: Queries for Analysis
   ========================================================== */

/* 1. Display all cities having population above 1600 */
db.zip.find({ pop: { $gt: 1600 } }, { _id: 0, city: 1, pop: 1 }).pretty();

/* 2. Display all cities in state "KS" */
db.zip.find({ state: "KS" }, { _id: 0, city: 1, state: 1 }).pretty();

/* 3. Display location of city "TIMKEN" */
db.zip.find({ city: "TIMKEN" }, { _id: 0, city: 1, loc: 1 }).pretty();

/* ==========================================================
   STEP 4: Index Performance Check
   ==========================================================
   You can use `explain("executionStats")` to compare query performance
   before and after index creation.
   Example:
   db.zip.find({ state: "KS" }).explain("executionStats");
*/
```

```
/* ============================================================
   END OF PRACTICAL 23
   ============================================================ */
```

```
/* ===========================================================
   PRACTICAL 24 – MongoDB (Collections, Updates, Queries)
   Database Name: p24
   =========================================================== */

/* ===========================================================
   STEP 1: Create Database and Collection
   =========================================================== */
use p24;

db.createCollection("games");

/* ===========================================================
   STEP 2: Insert Sample Documents
   =========================================================== */
db.games.insertMany([
  { name: "Valorant", gametype: "Shooter", rating: 95 },
  { name: "Minecraft", gametype: "Sandbox", rating: 92 },
  { name: "Forza Horizon", gametype: "Racing", rating: 90 },
  { name: "CSGO", gametype: "Shooter", rating: 88 },
  { name: "GTA V", gametype: "Action", rating: 98 }
]);

/* ===========================================================
   STEP 3: Queries
   =========================================================== */

/* 1. Write a query that returns all the games */
db.games.find().pretty();

/* 2. Write a query that returns the 3 highest-rated games */
db.games.find().sort({ rating: -1 }).limit(3).pretty();

/* 3. Update two favourite games to have two achievements:
      'Game Master' and 'Speed Demon' */
db.games.updateOne(
  { name: "GTA V" },
  { $set: { achievements: ["Game Master", "Speed Demon"] } }
);

db.games.updateOne(
  { name: "Forza Horizon" },
  { $set: { achievements: ["Game Master", "Speed Demon"] } }
);

/* 4. Write a query that returns all the games that have both achievements */
db.games.find({
  achievements: { $all: ["Game Master", "Speed Demon"] }
}).pretty();

/* 5. Write a query that returns only games that have achievements */
db.games.find({
  achievements: { $exists: true }
}).pretty();

/* ============================================================
   END OF PRACTICAL 24
   ============================================================ */
```

```
/* ============================================================
   PRACTICAL 25 – MongoDB (MapReduce – Aggregation by Gender & Hobby)
   Database Name: p25
   ============================================================ */

/* ============================================================
   STEP 1: Create Database and Collection
   ============================================================ */
use p25;

db.createCollection("users");

/* ============================================================
   STEP 2: Insert Sample Documents
   ============================================================ */
db.users.insertMany([
  {
    id: 1,
    name: "Leanne Flinn",
    email: "leanne.flinn@unilogic.com",
    work: "Unilogic",
    age: 27,
    gender: "Female",
    Salary: 16660,
    hobbies: ["Acrobatics", "Photography", "Papier-Mache"]
  },
  {
    id: 2,
    name: "Jason Kent",
    email: "jason.kent@microtech.com",
    work: "Microtech",
    age: 31,
    gender: "Male",
    Salary: 22000,
    hobbies: ["Gaming", "Photography", "Reading"]
  },
  {
    id: 3,
    name: "Aditi Rao",
    email: "aditi.rao@unilogic.com",
    work: "Unilogic",
    age: 24,
    gender: "Female",
    Salary: 18000,
    hobbies: ["Sketching", "Reading", "Papier-Mache"]
  },
  {
    id: 4,
    name: "Rohan Mehta",
    email: "rohan.mehta@techhub.com",
    work: "TechHub",
    age: 29,
    gender: "Male",
    Salary: 25000,
    hobbies: ["Photography", "Acrobatics", "Gaming"]
  }
]);

/* ============================================================
   STEP 3: MapReduce Queries
   ============================================================ */

/* 1. Get the count of Males and Females */
var mapGender = function() {
  emit(this.gender, 1);
};

var reduceGender = function(key, values) {
  return Array.sum(values);
};

db.users.mapReduce(mapGender, reduceGender, {
  out: "gender_count"
});

print("Gender Count:");
db.gender_count.find().pretty();

/* ------------------------------------------------------------ */

/* 2. Count the number of users in each hobby */
var mapHobby = function() {
  for (var i = 0; i < this.hobbies.length; i++) {
    emit(this.hobbies[i], 1);
  }
};

var reduceHobby = function(key, values) {
  return Array.sum(values);
};

db.users.mapReduce(mapHobby, reduceHobby, {
  out: "hobby_count"
});
```

```
print("Hobby Count:");
db.hobby_count.find().pretty();

/* ==============================================================
   END OF PRACTICAL 25
   ============================================================== */
```

```
/* ==========================================================
   PRACTICAL 26 – DBMS: Aggregate Functions & GROUP BY
   Database Name: p26
   ========================================================== */

CREATE DATABASE p26;
USE p26;

-- ==========================================================
-- PART (a): Calculate Department-wise Average Salary
-- ==========================================================
-- Requirement:
-- Display each department’s average, minimum, and maximum salary.
-- Only include departments having more than 1 employee.

-- STEP 1: Create Table
CREATE TABLE Employee (
  Emp_ID INT PRIMARY KEY,
  Emp_Name VARCHAR(50),
  Department VARCHAR(30),
  Salary DECIMAL(10,2)
);

-- STEP 2: Insert Sample Data
INSERT INTO Employee VALUES
(1, 'Ravi', 'HR', 40000.00),
(2, 'Neha', 'HR', 45000.00),
(3, 'Anil', 'IT', 60000.00),
(4, 'Sunita', 'IT', 65000.00),
(5, 'Kiran', 'Sales', 30000.00);

-- STEP 3: Query – Use Aggregate Functions with GROUP BY
SELECT
  Department,
  AVG(Salary) AS Avg_Salary,
  MIN(Salary) AS Min_Salary,
  MAX(Salary) AS Max_Salary
FROM Employee
GROUP BY Department
HAVING COUNT(*) > 1;

-- ==========================================================
-- PART (b): Find Employees Earning Between a Salary Range
-- ==========================================================
-- Requirement:
-- Display employees whose salary is between 35,000 and 60,000.

SELECT Emp_ID, Emp_Name, Department, Salary
```

```sql
FROM Employee
WHERE Salary BETWEEN 35000 AND 60000;

-- ============================================================
-- PART (c): Find Employees with 'a' in Their Name
-- ============================================================
-- Requirement:
-- Use LIKE operator to list employees having 'a' in their name.

SELECT Emp_ID, Emp_Name, Department
FROM Employee
WHERE Emp_Name LIKE '%a%';
```

```
/* ==============================================================
   PRACTICAL 27 – DBMS: JOINS & SUBQUERIES
   Database Name: p27
   ============================================================== */

CREATE DATABASE p27;
USE p27;

-- ==============================================================
-- PART (a): INNER JOIN – Display Employee Details with Department
-- ==============================================================
-- Requirement:
-- Show employee name, department name, and location using INNER JOIN.

-- STEP 1: Create Tables
CREATE TABLE Department (
  Dept_ID INT PRIMARY KEY,
  Dept_Name VARCHAR(30),
  Location VARCHAR(30)
);

CREATE TABLE Employee (
  Emp_ID INT PRIMARY KEY,
  Emp_Name VARCHAR(50),
  Dept_ID INT,
  Salary DECIMAL(10,2),
  FOREIGN KEY (Dept_ID) REFERENCES Department(Dept_ID)
);

-- STEP 2: Insert Sample Data
INSERT INTO Department VALUES
(1, 'HR', 'Pune'),
(2, 'IT', 'Mumbai'),
(3, 'Sales', 'Delhi');

INSERT INTO Employee VALUES
(101, 'Ravi', 1, 40000.00),
(102, 'Neha', 2, 60000.00),
(103, 'Amit', 3, 35000.00),
(104, 'Isha', 2, 70000.00);

-- STEP 3: Query – INNER JOIN
SELECT
  E.Emp_ID,
  E.Emp_Name,
  D.Dept_Name,
  D.Location,
  E.Salary
```

```
FROM Employee E
INNER JOIN Department D ON E.Dept_ID = D.Dept_ID;

-- ============================================================
-- PART (b): LEFT JOIN – Employees Without Departments
-- ============================================================
-- Requirement:
-- Display all employees, even if they don’t belong to any department.

-- Insert one record without Dept_ID
INSERT INTO Employee VALUES (105, 'Kiran', NULL, 30000.00);

SELECT
  E.Emp_Name,
  D.Dept_Name,
  D.Location
FROM Employee E
LEFT JOIN Department D ON E.Dept_ID = D.Dept_ID;

-- ============================================================
-- PART (c): SUBQUERY – Employees Earning Above Average Salary
-- ============================================================
-- Requirement:
-- Display employee names and salaries above the company average.

SELECT Emp_Name, Salary
FROM Employee
WHERE Salary > (SELECT AVG(Salary) FROM Employee);

-- ============================================================
-- PART (d): SUBQUERY – Employees Working in Same Department as ‘Neha’
-- ============================================================
-- Requirement:
-- Display names of employees working in same department as Neha.

SELECT Emp_Name, Salary
FROM Employee
WHERE Dept_ID = (
  SELECT Dept_ID FROM Employee WHERE Emp_Name = 'Neha'
);
```