Lab 7

Step 1: Create the Database

```
-- Create Database
CREATE DATABASE CompanyFlightDB;
USE CompanyFlightDB;
```

Step 2: Create Tables

Employee Schema

```sql
-- Employee table
CREATE TABLE Employee (
    employee_id INT PRIMARY KEY,
    name VARCHAR(50),
    birthdate DATE,
    salary DECIMAL(10, 2),
    department_id INT
);

-- Department table
CREATE TABLE Department (
    department_id INT PRIMARY KEY,
    name VARCHAR(50),
    location VARCHAR(50)
);

-- Project table
CREATE TABLE Project (
    project_id INT PRIMARY KEY,
    project_name VARCHAR(50),
    department_id INT,
    location VARCHAR(50),
    FOREIGN KEY (department_id) REFERENCES Department(department_id)
);

-- Dependent table
CREATE TABLE Dependent (
    dependent_id INT PRIMARY KEY,
    employee_id INT,
    name VARCHAR(50),
    birthdate DATE,
    FOREIGN KEY (employee_id) REFERENCES Employee(employee_id)
);

-- Works table (Employees working on projects)
CREATE TABLE Works (
    employee_id INT,
    project_id INT,
    PRIMARY KEY (employee_id, project_id),
    FOREIGN KEY (employee_id) REFERENCES Employee(employee_id),
    FOREIGN KEY (project_id) REFERENCES Project(project_id)
);
```

Flight Schema

```sql
-- Pilot table
CREATE TABLE Pilot (
    employee_id INT PRIMARY KEY,
    name VARCHAR(50),
    salary DECIMAL(10, 2)
);

-- Certified table (Certifications for employees on specific aircraft)
CREATE TABLE Certified (
    employee_id INT,
    aircraft_id INT,
    certification_date DATE,
    PRIMARY KEY (employee_id, aircraft_id),
    FOREIGN KEY (employee_id) REFERENCES Employee(employee_id)
);

-- Aircraft table
CREATE TABLE Aircraft (
    aircraft_id INT PRIMARY KEY,
    cruising_range INT
);

-- Flight table
CREATE TABLE Flight (
    flight_id INT PRIMARY KEY,
    aircraft_id INT,
    distance INT,
    FOREIGN KEY (aircraft_id) REFERENCES Aircraft(aircraft_id)
);
```

Step 3: Insert Sample Data

Now, insert values into the tables that adhere to the constraints and triggers.

Employee Data

```sql
-- Insert employees
INSERT INTO Employee (employee_id, name, birthdate, salary, department_id)
VALUES
(1, 'John Doe', '1980-01-01', 60000, 1),
(2, 'Jane Smith', '1990-02-02', 55000, 1),
(3, 'Mark Johnson', '1975-03-03', 50000, 2);

-- Insert departments
INSERT INTO Department (department_id, name, location) VALUES
(1, 'HR', 'New York'),
(2, 'Finance', 'Los Angeles');

-- Insert projects
INSERT INTO Project (project_id, project_name, department_id, location) VALUES
(1, 'Project Alpha', 1, 'New York'),
(2, 'Project Beta', 2, 'Los Angeles');

-- Insert dependents
INSERT INTO Dependent (dependent_id, employee_id, name, birthdate) VALUES
(1, 1, 'Dependent 1', '2010-05-05'),
(2, 2, 'Dependent 2', '2015-06-06'),
(3, 3, 'Dependent 3', '2008-07-07');
```

Works Data (Employee-Project Association)

```sql
-- Employees working on projects
INSERT INTO Works (employee_id, project_id) VALUES
(1, 1),
(2, 1),
(3, 2);
```

Pilot Data

```sql
-- Insert pilots
INSERT INTO Pilot (employee_id, name, salary) VALUES
(4, 'Pilot A', 70000),
(5, 'Pilot B', 80000);

-- Insert certifications
INSERT INTO Certified (employee_id, aircraft_id, certification_date) VALUES
(4, 1, '2022-01-01'),
(5, 2, '2023-02-02');
```

Aircraft and Flight Data

```
-- Insert aircraft
INSERT INTO Aircraft (aircraft_id, cruising_range) VALUES
(1, 5000),
(2, 6000);

-- Insert flights
INSERT INTO Flight (flight_id, aircraft_id, distance) VALUES
(1, 1, 4500),
(2, 2, 5500);
```

1. Employee Schema Triggers

a. Assure that deleting details of an employee deletes his dependent records also.

```
CREATE TRIGGER delete_employee_cascade
AFTER DELETE ON Employee
FOR EACH ROW
BEGIN
  DELETE FROM Dependent WHERE employee_id = OLD.employee_id;
END;
```

b. When a department with exactly one project is shifted to a new location, ensure that the project is also shifted to the new location.

```
CREATE TRIGGER shift_project_location
AFTER UPDATE OF location ON Department
FOR EACH ROW
WHEN (SELECT COUNT(*) FROM Project WHERE department_id = OLD.department_id) = 1
BEGIN
  UPDATE Project SET location = NEW.location WHERE department_id =
OLD.department_id;
END;
```

c. Assure at all times that there are no departments with more than 3 projects.

```
CREATE TRIGGER limit_department_projects
BEFORE INSERT OR UPDATE ON Project
FOR EACH ROW
WHEN (SELECT COUNT(*) FROM Project WHERE department_id = NEW.department_id) >=
3
BEGIN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A department cannot have more
than 3 projects';
END;
```

d. Assure that no employees work for more than one department.

```
CREATE TRIGGER unique_department_per_employee
BEFORE INSERT OR UPDATE ON Works
FOR EACH ROW
BEGIN
  IF (SELECT COUNT(*) FROM Works WHERE employee_id = NEW.employee_id) > 1 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'An employee cannot work for
more than one department';
  END IF;
END;
```

e. When a project is dropped, dissociate all the employees from that particular project.

```
CREATE TRIGGER drop_project
AFTER DELETE ON Project
FOR EACH ROW
BEGIN
  DELETE FROM Works WHERE project_id = OLD.project_id;
END;
```

f. When a new department is inaugurated, ensure that it is not co-located with any other departments.

```
CREATE TRIGGER check_department_location
BEFORE INSERT ON Department
FOR EACH ROW
BEGIN
  IF (SELECT COUNT(*) FROM Department WHERE location = NEW.location) > 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'A department cannot be co-
located with any other department';
  END IF;
END;
```

g. For every employee, ensure that his dependent's birthdate is less than his own.

```
CREATE TRIGGER check_dependent_birthdate
BEFORE INSERT OR UPDATE ON Dependent
FOR EACH ROW
BEGIN
  IF (SELECT birthdate FROM Employee WHERE employee_id = NEW.employee_id) <=
NEW.birthdate THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Dependent birthdate must be
less than employee birthdate';
  END IF;
END;
```

h. Increment 1000 rupees to the salary if any of his/her dependents expire.

```
CREATE TRIGGER increment_salary_on_dependent_death
AFTER DELETE ON Dependent
FOR EACH ROW
BEGIN
  UPDATE Employee SET salary = salary + 1000 WHERE employee_id =
OLD.employee_id;
END;
```

---

2. Flight Schema Triggers

i. Create a trigger that handles an update command to find the total salary of all pilots.

```
CREATE TRIGGER check_total_pilot_salary
BEFORE UPDATE ON Pilot
FOR EACH ROW
BEGIN
  IF NEW.salary IS NULL OR NEW.salary < 50000 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Pilot salary must be greater
than 50,000';
  END IF;
END;
```

j. Create a trigger to set salary as 30,000 if NULL is present. Ensure the salary of a pilot is greater than a non-pilot.

```
CREATE TRIGGER set_default_salary_if_null
BEFORE INSERT OR UPDATE ON Employee
FOR EACH ROW
BEGIN
  IF NEW.salary IS NULL THEN
    SET NEW.salary = 30000;
  END IF;

  IF (NEW.role = 'Pilot' AND NEW.salary <= (SELECT MAX(salary) FROM Employee
WHERE role != 'Pilot')) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Pilot salary must be greater
than non-pilot salary';
  END IF;
END;
```

k. Create a trigger to foil any attempt to lower the salary of an employee.

```
CREATE TRIGGER prevent_salary_decrease
BEFORE UPDATE ON Employee
FOR EACH ROW
BEGIN
  IF NEW.salary < OLD.salary THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'salary cannot be decreased';
  END IF;
END;
```

l. When inserting a new certification for an employee, check that the aircraft ID exists in the Aircraft.

```
CREATE TRIGGER check_aircraft_exists
BEFORE INSERT ON Certified
FOR EACH ROW
BEGIN
  IF (SELECT COUNT(*) FROM Aircraft WHERE aircraft_id = NEW.aircraft_id) = 0
THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Aircraft does not exist';
  END IF;
END;
```

m. When making modifications to the Aircraft table, check that the cruising range is greater than or equal to the distance of flights.

```
CREATE TRIGGER check_cruising_range
BEFORE UPDATE ON Aircraft
FOR EACH ROW
BEGIN
  IF NEW.cruising_range < (SELECT MAX(distance) FROM Flight WHERE aircraft_id =
OLD.aircraft_id) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Cruising range must be greater
than or equal to flight distance';
  END IF;
END;
```

n. When a new certification is inserted into Certified, also insert an employee with the ID and a NULL salary.

```
CREATE TRIGGER insert_employee_on_certification
AFTER INSERT ON Certified
FOR EACH ROW
BEGIN
  INSERT INTO Employee (employee_id, salary) VALUES (NEW.employee_id, NULL);
END;
```

o. Terminate pilots and their certification when the pilot retires.

```
CREATE TRIGGER retire_pilot
AFTER DELETE ON Pilot
FOR EACH ROW
BEGIN
  DELETE FROM Certified WHERE employee_id = OLD.employee_id;
END;
```

p. Prevent the average salary of employees from dropping below Rs. 50,000.

```
CREATE TRIGGER prevent_avg_salary_drop
BEFORE INSERT OR UPDATE OR DELETE ON Employee
FOR EACH ROW
BEGIN
  IF (SELECT AVG(salary) FROM Employee) < 50000 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Average salary cannot drop
below Rs. 50,000';
  END IF;
END;
```

Here’s the implementation of stored procedures using **Cursors** and **Exception Handling** for the corresponding queries on both **Employee Schema** and **Flight Schema**.

1. Employee Schema Stored Procedures

q. Stored procedure to insert a new attribute ‘address’ in `DEPENDENT` and update the same as that of the employee’s address.

```
-- Add address column to Dependent table if not exists
ALTER TABLE Dependent ADD address VARCHAR(100);

-- Stored procedure to update the dependent's address to the employee's address
DELIMITER $
CREATE PROCEDURE UpdateDependentAddress()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE emp_id INT;
    DECLARE emp_address VARCHAR(100);

    -- Cursor to loop through employee records
    DECLARE cur CURSOR FOR SELECT employee_id, address FROM Employee;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO emp_id, emp_address;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Update dependent's address with employee's address
        UPDATE Dependent SET address = emp_address WHERE employee_id = emp_id;
    END LOOP;
    CLOSE cur;
END$
DELIMITER ;
```

r. Stored procedure to display the first name, SSN, salary, and grade of an employee based on their salary.

```
DELIMITER $
CREATE PROCEDURE GetEmployeeGrade()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE emp_name VARCHAR(50);
    DECLARE emp_ssn INT;
    DECLARE emp_salary DECIMAL(10,2);
    DECLARE emp_grade VARCHAR(10);

    -- Cursor to loop through employees
    DECLARE cur CURSOR FOR SELECT name, employee_id, salary FROM Employee;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO emp_name, emp_ssn, emp_salary;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Assign grade based on salary
        IF emp_salary BETWEEN 1 AND 10000 THEN
            SET emp_grade = 'Grade 3';
        ELSEIF emp_salary BETWEEN 10001 AND 50000 THEN
            SET emp_grade = 'Grade 2';
        ELSEIF emp_salary > 50000 THEN
            SET emp_grade = 'Grade 1';
        ELSE
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid salary range';
        END IF;

        -- Display the employee details
        SELECT emp_name, emp_ssn, emp_salary, emp_grade;
    END LOOP;
    CLOSE cur;
END$
DELIMITER ;
```

s. Stored procedure to display department number, average salary, and number of employees in each department.

```
DELIMITER $
CREATE PROCEDURE GetDepartmentSummary()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE dept_id INT;
    DECLARE avg_salary DECIMAL(10,2);
    DECLARE num_employees INT;

    -- Cursor to loop through departments
    DECLARE cur CURSOR FOR SELECT department_id FROM Department;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO dept_id;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Calculate average salary and employee count
        SELECT AVG(salary), COUNT(*) INTO avg_salary, num_employees FROM
Employee WHERE department_id = dept_id;

        -- Handle exception if invalid department
        IF num_employees = 0 THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid department
number';
        END IF;

        -- Display department summary
        SELECT dept_id, avg_salary, num_employees;
    END LOOP;
    CLOSE cur;
END$
DELIMITER ;
```

---

2. Flight Schema Stored Procedures

t. Stored procedure to update an employee record given the employee id, with exception handling for an invalid employee id.

```
DELIMITER $
CREATE PROCEDURE UpdateEmployeeRecord(IN emp_id INT, IN new_salary
DECIMAL(10,2), IN new_name VARCHAR(50))
BEGIN
    DECLARE emp_exists INT;

    -- Check if employee exists
    SELECT COUNT(*) INTO emp_exists FROM Employee WHERE employee_id = emp_id;

    IF emp_exists = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid employee ID';
    ELSE
        -- Update the employee record
        UPDATE Employee SET salary = new_salary, name = new_name WHERE
employee_id = emp_id;
        -- Print success message
        SELECT CONCAT('Employee ID ', emp_id, ' updated successfully') AS
message;
    END IF;
END$
DELIMITER ;
```

u. Stored procedure to display the name and salary of each employee, and rank them as Grade A or B based on salary.

```
DELIMITER $
CREATE PROCEDURE RankEmployees()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE emp_name VARCHAR(50);
    DECLARE emp_salary DECIMAL(10,2);
    DECLARE emp_grade CHAR(1);

    -- Cursor to loop through employees
    DECLARE cur CURSOR FOR SELECT name, salary FROM Employee;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO emp_name, emp_salary;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Rank employee based on salary
        IF emp_salary > 50000 THEN
            SET emp_grade = 'A';
        ELSE
            SET emp_grade = 'B';
        END IF;

        -- Display employee details
        SELECT emp_name, emp_salary, emp_grade;
    END LOOP;
    CLOSE cur;
END$
DELIMITER ;
```

v. Stored procedure to build a name list of employees certified for a Boeing aircraft, with exception handling.

```
DELIMITER $
CREATE PROCEDURE GetBoeingCertifiedEmployees()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE emp_name VARCHAR(50);

    -- Cursor to loop through employees certified for Boeing aircraft (assume
Boeing aircraft_id is 1)
    DECLARE cur CURSOR FOR
        SELECT E.name FROM Employee E
        JOIN Certified C ON E.employee_id = C.employee_id
        WHERE C.aircraft_id IN (SELECT aircraft_id FROM Aircraft WHERE
aircraft_id = 1);

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN cur;
    read_loop: LOOP
        FETCH cur INTO emp_name;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Display employee name
        SELECT emp_name;
    END LOOP;

    -- Handle exception if no employees found
    IF done THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No employees certified for
Boeing aircraft';
    END IF;

    CLOSE cur;
END$
DELIMITER ;
```

3

Certainly! Below are the SQL queries separated into individual code blocks for each question:

```
--a
-- Select employees with odd Ssn
SELECT *
FROM Employee
WHERE MOD(CAST(Ssn AS UNSIGNED), 2) = 1;
```

```
ERROR 1054 (42S22): Unknown column 'id' in 'where clause'
mysql> SELECT *  FROM Employee WHERE MOD(Ssn, 2) = 1;
+-----------+---------+-------+------+---------------+----------+-----------+------+
| Ssn       | Fname   | Lname | Sex  | Address       | Salary   | Super_ssn | Dno  |
+-----------+---------+-------+------+---------------+----------+-----------+------+
| 123456789 | John    | Smith | M    | 123 Elm St    | 55000.00 | NULL      |    1 |
| 345678901 | Jim     | Brown | M    | 345 Maple St  | 50000.00 | 123456789 |    3 |
| 567890123 | Jill    | Black | F    | 567 Birch St  | 80000.00 | 345678901 |    5 |
| 789012345 | Tejaswi | Kumar | F    | 789 Walnut St | 75000.00 | 567890123 |   10 |
| 901234567 | Tim     | Blue  | M    | 901 Fir St    | 40000.00 | 789012345 |    2 |
+-----------+---------+-------+------+---------------+----------+-----------+------+
5 rows in set (0.00 sec)
```

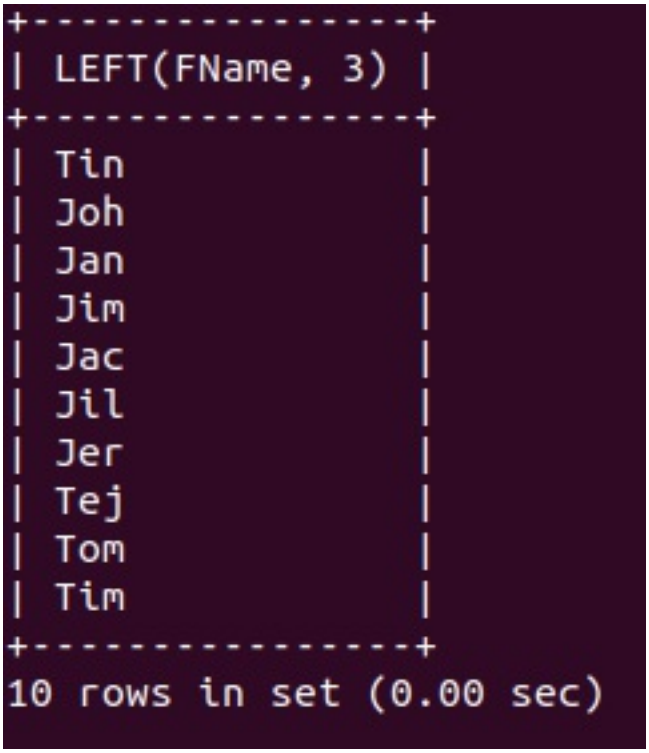
```sql
--b
-- Select employees with even Ssn
SELECT *
FROM Employee
WHERE MOD(CAST(Ssn AS UNSIGNED), 2) = 0;
```

```
+-----------+-------+-------+------+--------------+----------+-----------+------+
| Ssn       | Fname | Lname | Sex  | Address      | Salary   | Super_ssn | Dno  |
+-----------+-------+-------+------+--------------+----------+-----------+------+
| 012345678 | Tina  | Red   | F    | 012 Elm St   | 35000.00 | 890123456 |    3 |
| 234567890 | Jane  | Doe   | F    | 234 Oak St   | 60000.00 | 123456789 |    2 |
| 456789012 | Jack  | White | M    | 456 Pine St  | 70000.00 | 234567890 |    4 |
| 678901234 | Jerry | Green | M    | 678 Cedar St | 90000.00 | 456789012 |    6 |
| 890123456 | Tom   | Gray  | M    | 890 Ash St   | 30000.00 | 678901234 |    1 |
+-----------+-------+-------+------+--------------+----------+-----------+------+
5 rows in set (0.00 sec)
```

```sql
--c
-- Extract the year from BirthDate
SELECT SUBSTRING(BirthDate, 1, 4) AS Year
FROM Employee;
```

```
mysql> SELECT SUBSTRING(BirthDate, 1, 4) AS Year
    -> FROM Employee;
+------+
| Year |
+------+
| 1980 |
| 1990 |
| 1975 |
+------+
3 rows in set (0.00 sec)
```

```sql
--d
-- Extract the first 3 characters of FName
SELECT LEFT(FName, 3)
FROM Employee;
```



```
+-----------------+
| LEFT(FName, 3) |
+-----------------+
| Tin             |
| Joh             |
| Jan             |
| Jim             |
| Jac             |
| Jil             |
| Jer             |
| Tej             |
| Tom             |
| Tim             |
+-----------------+
10 rows in set (0.00 sec)
```

```sql
--e
-- Find duplicate FNames
SELECT Fname, COUNT(*)
FROM Employee
GROUP BY Fname
HAVING COUNT(*) > 1;
```

```
mysql> SELECT Fname, COUNT(*)  FROM Employee GROUP BY Fname HAVING COUNT(*) > 1;
+-------+----------+
| Fname | COUNT(*) |
+-------+----------+
| Tim   |        2 |
+-------+----------+
1 row in set (0.00 sec)
```

```sql
--f
-- Remove duplicate entries based on FName, keeping the one with the minimum
Ssn
CREATE TEMPORARY TABLE Temp AS
SELECT MIN(Ssn) as Ssn FROM Employee GROUP BY FName;
DELETE FROM Employee WHERE Ssn NOT IN (SELECT Ssn FROM Temp);
```

```
mysql> select * from Temp;
+-----------+
| Ssn       |
+-----------+
| 012345678 |
| 123456780 |
| 123456789 |
| 234567890 |
| 345678901 |
| 456789012 |
| 567890123 |
| 678901234 |
| 789012345 |
| 890123456 |
+-----------+
10 rows in set (0.00 sec)
```

```sql
--g
-- Remove duplicate entries based on FName, keeping the one with the minimum Ssn
CREATE TEMPORARY TABLE Temp AS
SELECT MIN(Ssn) as Ssn FROM Employee GROUP BY FName;
DELETE FROM Employee WHERE Ssn NOT IN (SELECT Ssn FROM Temp);
```

```sql
--h
-- Find the 3rd highest unique salary
SELECT DISTINCT salary
FROM Employee
ORDER BY salary DESC
LIMIT 1 OFFSET 2;
```

```
mysql> SELECT DISTINCT Salary
    -> FROM Employee
    -> ORDER BY Salary DESC
    -> LIMIT 1 OFFSET 2;
+----------+
| Salary   |
+----------+
| 75000.00 |
+----------+
1 row in set (0.00 sec)
```

```sql
-- For nth max salary:
SELECT DISTINCT salary
FROM Employee
ORDER BY salary DESC
LIMIT 1 OFFSET n-1; -- Replace `n` with the desired position
```

```sql
--i
-- Find the top 3 unique salaries
SELECT DISTINCT salary
FROM Employee
ORDER BY salary DESC
LIMIT 3;
```

```
mysql> SELECT DISTINCT Salary
    -> FROM Employee
    -> ORDER BY Salary DESC
    -> LIMIT 3;
+----------+
| Salary   |
+----------+
| 90000.00 |
| 80000.00 |
| 75000.00 |
+----------+
3 rows in set (0.00 sec)
```

```sql
-- For the top n max salaries:
SELECT DISTINCT salary
FROM Employee
ORDER BY salary DESC
LIMIT n; -- Replace `n` with the desired number of top salaries
```

```sql
--j
-- Extract the year, month, and day from BirthDate
SELECT YEAR(BirthDate) AS Year, MONTH(BirthDate) AS Month, DAY(BirthDate) AS
Day
FROM Employee;
```

```
mysql> SELECT YEAR(BirthDate) AS Year, MONTH(BirthDate) AS Month, DAY(BirthDate) AS Day
    -> FROM Employee;
+------+-------+------+
| Year | Month | Day  |
+------+-------+------+
| 1980 |     1 |    1 |
| 1990 |     2 |    2 |
| 1975 |     3 |    3 |
+------+-------+------+
3 rows in set (0.00 sec)
```

```sql
--k
-- Extract date part from BirthDate
SELECT DATE(BirthDate)
FROM Employee;
```

```
mysql> SELECT DATE(BirthDate)
    -> FROM Employee;
+------------------+
| DATE(BirthDate) |
+------------------+
| 1980-01-01       |
| 1990-02-02       |
| 1975-03-03       |
+------------------+
3 rows in set (0.00 sec)
```

```sql
--l
-- Find the position of 'a' in 'Sundar Pitchai'
SELECT LOCATE('a', 'Sundar Pitchai');
```

```
mysql> SELECT LOCATE('a', 'Sundar Pitchai');
+-------------------------------+
| LOCATE('a', 'Sundar Pitchai') |
+-------------------------------+
|                             5 |
+-------------------------------+
1 row in set (0.00 sec)
```

```sql
--m
-- Remove leading spaces from FName
SELECT LTRIM(FName)
FROM Employee;
```

```
mysql> SELECT LTRIM(FName)  FROM Employee;
+--------------+
| LTRIM(FName) |
+--------------+
| Tina         |
| Tim          |
| John         |
| Jane         |
| Jim          |
| Jack         |
| Jill         |
| Jerry        |
| Tejaswi      |
| Tom          |
| Tim          |
+--------------+
11 rows in set (0.00 sec)
```

```
--n
-- Get the length of FName
SELECT LENGTH(FName)
FROM Employee;
```

```
mysql> SELECT LENGTH(FName)
    -> FROM Employee;
+---------------+
| LENGTH(FName) |
+---------------+
|             4 |
|             3 |
|             4 |
|             4 |
|             3 |
|             4 |
|             4 |
|             5 |
|             7 |
|             3 |
|             3 |
+---------------+
11 rows in set (0.00 sec)
```

```sql
--o
-- Replace 'o' with '*' in FName
SELECT REPLACE(FName, 'o', '*')
FROM Employee;
```

```
mysql> SELECT REPLACE(FName, 'o', '*')
    -> FROM Employee;
+--------------------------+
| REPLACE(FName, 'o', '*') |
+--------------------------+
| Tina                     |
| Tim                      |
| J*hn                     |
| Jane                     |
| Jim                      |
| Jack                     |
| Jill                     |
| Jerry                    |
| Tejaswi                  |
| T*m                      |
| Tim                      |
+--------------------------+
11 rows in set (0.00 sec)
```

```sql
--p
-- Concatenate FName and LName with an underscore
SELECT CONCAT(FName, '_', LName)
FROM Employee;
```

```
mysql> SELECT CONCAT(FName, '_', LName)
    -> FROM Employee;
+-----------------------------+
| CONCAT(FName, '_', LName) |
+-----------------------------+
| Tina_Red                    |
| Tim_Blue                    |
| John_Smith                  |
| Jane_Doe                    |
| Jim_Brown                   |
| Jack_White                  |
| Jill_Black                  |
| Jerry_Green                 |
| Tejaswi_Kumar               |
| Tom_Gray                    |
| Tim_Blue                    |
+-----------------------------+
11 rows in set (0.00 sec)
```

```sql
--q
-- Find employees whose FName contains 'jai' (case-insensitive)
SELECT *
FROM Employee
WHERE LOWER(FName) LIKE '%jai%';
```

```
mysql> SELECT *  FROM Employee  WHERE LOWER(FName) LIKE '%jim%';
+-----------+-------+-------+------+--------------+----------+-----------+------+
| Ssn       | Fname | Lname | Sex  | Address      | Salary   | Super_ssn | Dno  |
+-----------+-------+-------+------+--------------+----------+-----------+------+
| 345678901 | Jim   | Brown | M    | 345 Maple St | 50000.00 | 123456789 |    3 |
+-----------+-------+-------+------+--------------+----------+-----------+------+
1 row in set (0.00 sec)
```

```sql
--r
-- Count employees by gender with birth dates between 1980-05-01 and 2024-12-31
SELECT Gender, COUNT(*)
FROM Employee
WHERE BirthDate BETWEEN '1980-05-01' AND '2024-12-31'
GROUP BY Gender;
```

```
mysql> SELECT Gender, COUNT(*)
    -> FROM Employee
    -> WHERE BirthDate BETWEEN '1980-05-01' AND '2024-12-31'
    -> GROUP BY Gender;
+--------+----------+
| Gender | COUNT(*) |
+--------+----------+
| F      |        1 |
+--------+----------+
1 row in set (0.00 sec)
```

```
--s
-- Retrieve user and authentication_string from the mysql.user table
SELECT user, authentication_string
FROM mysql.user;
```

```
mysql> SELECT user, authentication_string
    -> FROM mysql.user;
+------------------+-------------------------------------------------------------------------+
| user             | authentication_string                                                   |
+------------------+-------------------------------------------------------------------------+
| 106122119        | $A$005$◆A?#Bg Lm), /nQn{C]RcV14NwYy5c7h0qG7Z0mZ1h6FutU7MjzWI.MjLxwSn7 |
| debian-sys-maint | $A$005$@{mVsqc%63m;]G~0MVUQRgUC0Yf6rWm969lIGDjIV5aojIlKgzhJ90MVS3 |
| mysql.infoschema | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| mysql.session    | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| mysql.sys        | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| root             |                                                                         |
| testuser         | $A$005$~73pyFLPh~;lN
                                    JI%6KhRQfL45pkx5Zrn9P1506tEU0MmMDkwcQc7WO3espo4 |
| testuser1        | $A$005$z*WR<P      W[R"d#67DjeBP13HUPzrjOSuQpQJ7vLGe03bgVMYw6We6eQfC/A |
+------------------+-------------------------------------------------------------------------+
8 rows in set (0.00 sec)
```

Each code block is now clearly separated for easy reference and execution.