### **(DDL)**

### 

### **Prompt :**

Generate a set of DDL (Data Definition Language) queries in PostgreSQL to manage the structure of a company's employee database. The database should include the following:

1. A **table** named employees with the following columns:
   * id (integer, primary key, auto-incremented)
   * name (string, up to 100 characters)
   * position (string, up to 50 characters)
   * hire\_date (date)
   * salary (decimal, 10 digits, 2 decimal places)
2. An **index** on the name column of the employees table for faster lookups.
3. A **constraint** to ensure that the salary column has values greater than zero.
4. A **table** named departments with the following columns:
   * dept\_id (integer, primary key, auto-incremented)
   * dept\_name (string, up to 100 characters)
   * location (string, up to 100 characters)
5. An **ALTER** query to add a department\_id column to the employees table, which will reference the dept\_id in the departments table as a foreign key.
6. A **query** to drop the departments table.

### **DDL Queries:**

* **Create the employees Table**:

CREATE TABLE employees (

id SERIAL PRIMARY KEY,

name VARCHAR(100),

position VARCHAR(50),

hire\_date DATE,

salary DECIMAL(10, 2),

CHECK (salary > 0) -- Ensure salary is greater than 0

);

* **Create an Index on the name Column**:  
    
  CREATE INDEX idx\_employees\_name ON employees (name);
* **Create the departments Table**:

CREATE TABLE departments (

dept\_id SERIAL PRIMARY KEY,

dept\_name VARCHAR(100),

location VARCHAR(100)

);

* **Add a Foreign Key Constraint to employees Table**:

ALTER TABLE employees

ADD COLUMN department\_id INT,

ADD CONSTRAINT fk\_department

FOREIGN KEY (department\_id) REFERENCES departments(dept\_id);

* **Drop the departments Table**:

DROP TABLE departments;

### **Explanation:**

* **Create Table**: Defines the structure of the employees and departments tables, including columns and constraints (like primary keys and check constraints).
* **Create Index**: Creates an index on the name column of the employees table to speed up searches by name.
* **Alter Table**: Adds a new column (department\_id) to the employees table and establishes a foreign key relationship between employees and departments.
* **Drop Table**: Deletes the departments table from the database.

2. **DML**

### **Prompt:**

Create DML queries in PostgreSQL for the following scenario:

1. Insert sample data into the employees table, including employee names, positions, hire dates, and salaries (ensure the salary is greater than zero).
2. Insert sample data into the departments table with department names and locations.
3. Update the salary of a specific employee (e.g., id = 3) to a new value.
4. Delete an employee from the employees table with a specific id (e.g., id = 5).
5. Select all employees who belong to a specific department (e.g., department\_id = 2).
6. Select all employees who have a salary greater than a specific amount (e.g., salary > 50000).
7. Update the department of an employee (e.g., employee id = 2) to a new department (e.g., department\_id = 3).

### **DML Queries for the Given Data:**

* **Insert Sample Data into the employees Table**:

INSERT INTO employees (name, position, hire\_date, salary)

VALUES

('John Doe', 'Software Engineer', '2020-06-01', 75000),

('Jane Smith', 'Product Manager', '2019-08-15', 95000),

('Alice Brown', 'HR Specialist', '2021-01-10', 55000),

('Bob White', 'Database Administrator', '2018-04-25', 65000);

* **Insert Sample Data into the departments Table**:

INSERT INTO departments (dept\_name, location)

VALUES

('Engineering', 'New York'),

('Human Resources', 'Chicago'),

('Marketing', 'San Francisco');

* **Update the Salary of an Employee (e.g., id = 3)**:

UPDATE employees

SET salary = 58000

WHERE id = 3;

* **Delete an Employee with a Specific id (e.g., id = 5)**:

DELETE FROM employees

WHERE id = 5;

* **Select All Employees Who Belong to a Specific Department (e.g., department\_id = 2)**:

SELECT e.id, e.name, e.position, e.hire\_date, e.salary

FROM employees e

WHERE e.department\_id = 2;

* **Select All Employees with a Salary Greater than a Specific Amount (e.g., salary > 50000)**:

SELECT e.id, e.name, e.position, e.salary

FROM employees e

WHERE e.salary > 50000;

* **Update the Department of an Employee (e.g., id = 2)**:

UPDATE employees

SET department\_id = 3

WHERE id = 2;

### **Explanation of DML Queries:**

* **Insert**: Inserts new data into the employees and departments tables. Sample values like employee names, positions, and department details are inserted.
* **Update**: Modifies an existing record in the employees table. In this case, it changes the salary of an employee with id = 3.
* **Delete**: Removes an employee from the employees table based on a given condition (employee id = 5).
* **Select**: Retrieves specific data from the employees table based on certain criteria, such as selecting employees from a specific department or those with a salary higher than a certain value.

### DCL

### **Prompt:**

Create DCL queries in PostgreSQL for the following scenario:

1. Grant permission to a user john\_user to **SELECT** from the employees and departments tables.
2. Grant **UPDATE** permission to a user jane\_user on the salary column of the employees table.
3. Revoke **SELECT** permission from john\_user on the employees table.
4. Grant **ALL** privileges on the departments table to a user admin\_user.
5. Revoke **ALL** privileges from the user john\_user on the departments table.
6. Create a new role hr\_manager and assign the SELECT permission on the employees table to the hr\_manager role.
7. Grant **EXECUTE** permission on a function calculate\_salary() to the user admin\_user.

### **DCL Queries for the Given Data:**

* **Grant SELECT Permission to john\_user**:

GRANT SELECT ON employees, departments TO john\_user;

* **Grant UPDATE Permission to jane\_user on the salary Column of the employees Table**:

GRANT UPDATE(salary) ON employees TO jane\_user;

* **Revoke SELECT Permission from john\_user on the employees Table**:

REVOKE SELECT ON employees FROM john\_user;

* **Grant ALL Privileges on the departments Table to admin\_user**:

GRANT ALL PRIVILEGES ON departments TO admin\_user;

* **Revoke ALL Privileges from john\_user on the departments Table**:

REVOKE ALL PRIVILEGES ON departments FROM john\_user;

* **Create a Role hr\_manager and Assign the SELECT Permission on the employees Table**:

CREATE ROLE hr\_manager;

GRANT SELECT ON employees TO hr\_manager;

* **Grant EXECUTE Permission on the Function calculate\_salary() to admin\_user**:

GRANT EXECUTE ON FUNCTION calculate\_salary() TO admin\_user;

### **Explanation of DCL Queries:**

* **GRANT**: Used to assign permissions to a user or role. For example, granting the SELECT privilege on the employees table to john\_user or granting ALL privileges on the departments table to admin\_user.
* **REVOKE**: Used to remove permissions that were previously granted. For example, revoking SELECT permission from john\_user on the employees table.
* **CREATE ROLE**: Used to create a new role in the database (e.g., hr\_manager), which can then have permissions granted to it.
* **EXECUTE**: Grants permission to run a specific function in the database, such as the function calculate\_salary().