**PostgreSQL**

**Post Test**

**Q1**. Create a PostgreSQL database named company\_db that contains tables for storing employee information (name, age, department, salary) and department information (department name, department head).

create database company\_db;

create table emp\_info(

name varchar(30),

dept varchar(23),

salary int

)

create table dept\_info(

dept\_name varchar(23),

dept\_head varchar(23)

)

**Q2.**  Create a table called employees with the following columns:

* + emp\_id (integer, primary key)
  + emp\_name (varchar, not null)
  + emp\_age (integer, not null)
  + emp\_department (varchar)
  + emp\_salary (numeric)

create table employees(

emp\_id int primary key,

emp\_name varchar(23) not null,

emp\_age int not null,

emp\_department varchar(23),

emp\_salary numeric

)

--------

create table department(

emp\_id int references employees(emp\_id) ,

dept\_id int,

dept\_salary int

)

select \* from department

insert into department values

(102,01,50000),(105,04,40000),(104,09,600000),

(102,06,30000)

**Q3.** Insert 5 records into the employees table with realistic data for employee name, age, department, and salary.

insert into employees values(101,'srikanth',23,'finance','300000'),

(102,'Naveen',24,'HR',400000),

(103,'sai krian',23,'IT',500000),

(104,'mohan',24,'QE','450000'),

(105,'satya Narayna',23,'Engineering',200000),

(106,'krishna',25,'HR',20000)

**Q4.** Write a SQL query to fetch the names and ages of employees who work in the "Engineering" department and have a salary greater than 50,000.

select emp\_name,emp\_age from employees

where emp\_department ='Engineering' and emp\_salary>50000

**Q5.** Write an SQL query to increase the salary by 10% for all employees working in the "HR" department.

update employees

set emp\_salary = emp\_salary+(emp\_salary\*0.10)

where emp\_department = 'HR'

**Q6.** Write a SQL query to delete all employees whose salary is less than 30,000.

delete from employees where emp\_salary<30000

**Q7.** You have two tables: employees and departments. Write a query to join these tables and fetch the employee name, department name, and salary for each employee.

select e.emp\_name,e.emp\_department,e.emp\_salary

from employees e join department d

on e.emp\_id=d.emp\_id

**Q8.** Write a query that groups employees by department and calculates the average salary for each department.

select emp\_department,avg(emp\_salary) as dep\_avg\_salry

from employees

group by emp\_department

**Q9.** Write a query that fetches the employee name(s) who have the highest salary in the "Engineering" department.

select emp\_name from employees where emp\_salary=

(select max(emp\_salary) from employees

where emp\_department ='Engineering')

**Q10.** Alter the employees table to add a constraint that ensures no employee can have a duplicate email address.

alter table employees add email\_address varchar(25) unique

**Q11. Stored Procedure Problem:**

**Problem Statement:** Create a **stored procedure** named update\_employee\_salary that takes an employee's employee\_id and a salary\_increment as input parameters. The procedure should update the salary of the employee in the employees table by adding the salary\_increment to the existing salary.

The employees table has the following structure:

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY,

employee\_name VARCHAR(100),

salary NUMERIC

);

**Requirements:**

* The procedure should raise an exception if the employee does not exist in the table.
* After updating the salary, the procedure should log the change in the salary\_changes table.
* The salary\_changes table has the following structure:

sql

Copy code

CREATE TABLE salary\_changes (

change\_id SERIAL PRIMARY KEY,

employee\_id INT,

old\_salary NUMERIC,

new\_salary NUMERIC,

change\_date TIMESTAMP

);

create or replace procedure update\_employee\_salary(emps\_id int,salary\_increment int)

language plpgsql

as $$

declare

old\_sal numeric;

new\_sal numeric;

begin

select emp\_salary into old\_sal from employees where emp\_id=emps\_id;

new\_sal:=emp\_salary+salary\_increment;

update employees set emp\_salary=new\_sal where emp\_id=emps\_id;

insert into salary\_changes(employee\_id,old\_salary,new\_salary,change\_date) values (emp\_id,old\_sal,new\_sal,now());

raise notice 'salary updated for the employee id %',emp\_id;

end;$$;

**Q12. Function Problem:**

**Problem Statement:** Write a **function** named get\_employee\_bonus that calculates the bonus of an employee based on their salary. The bonus is calculated as 10% of the salary if the salary is greater than 50000, and 5% if the salary is less than or equal to 50000.

The function should return the calculated bonus as a numeric value. The input to the function should be the employee\_id.

The employees table is the same as the one mentioned in the stored procedure problem.

**Requirements:**

* The function should return NULL if the employee does not exist.

create or replace function get\_employee\_bonus(emp\_id int) returns int

as $$

declare

emp\_salary int;

bonus int;

begin

select emp\_salary from employees where emp\_id= emp\_id;

if emp\_salary is null then

return null;

end if;

if emp\_salary>50000 then

bonus:=emp\_salary\*0.1;

else

bonus:=emp\_salary\*0.5;

end if;

return bonus;

end;

$$ language plpgsql;

**Q13. Trigger Problem:**

**Problem Statement:** Write a **trigger** that automatically inserts a record into the audit\_log table whenever an update is made to an employee's salary in the employees table. The trigger should log the employee\_id, old salary, new salary, and the current timestamp.

The audit\_log table has the following structure:

CREATE TABLE audit\_log (

log\_id SERIAL PRIMARY KEY,

employee\_id INT,

old\_salary NUMERIC,

new\_salary NUMERIC,

change\_time TIMESTAMP

);

**Requirements:**

* The trigger should fire before any update is made to the employees table.
* The trigger should store the old and new salary values in the audit\_log table.

create or replace function log\_salary\_change() returns Trigger

as $$

begin

insert into audit\_log(employee\_id,old\_salary,new\_salary,change\_time)

values(OLD.emp\_id,OLD.emp\_salary,NEW.emp\_salary,current\_timestamp);

return new;

end;

$$ language plpgsql;

create trigger salary\_change\_trigger

before update on employees

for each row

when(OLD.emp\_salary is distinct from NEW.emp\_salary)

execute procedure

log\_salary\_change();

select \* from audit\_log;

select \* from employees;

update employees set emp\_salary=220000 where emp\_id=4;

**Q14. Stored Procedure Problem (with Input/Output Parameters):**

**Problem Statement:** Create a **stored procedure** named transfer\_employee that transfers an employee to a new department. The procedure should take employee\_id and new\_department\_id as input parameters. The procedure should update the employee's department in the employees table.

The employees table is as described in the first problem, and assume there is a departments table with the following structure:

CREATE TABLE departments (

department\_id SERIAL PRIMARY KEY,

department\_name VARCHAR(100)

);

**Requirements:**

* The procedure should check if the new\_department\_id exists in the departments table before making the update.
* If the employee\_id does not exist, the procedure should raise an exception.

**Q15. Function Problem (with Aggregation):**

**Problem Statement:** Write a **function** called average\_department\_salary that takes department\_id as input and returns the average salary of employees in that department. The employees table has a column department\_id to indicate the department of each employee.

**Requirements:**

* The function should return NULL if the department does not have any employees.
* If the department exists but has no employees, the function should return 0.

The employees table structure:

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY,

employee\_name VARCHAR(100),

salary NUMERIC,

department\_id INT

);

create or replace function avg\_dept\_sal(deptid int) returns int

as $$

declare

avg\_salary int;

begin

select avg(salary) from employees where deptid=deptid;

if avg\_salary is null then

return 0;

end if;

return avg\_salary;

end;

$$ language plpgsql;