**SQL Final Project Submission :**

**Name** : Snehal Yerkuntwar (snehalyr)

**LINK to onecompiler** : <https://onecompiler.com/postgresql/43tz3nwuh>

**NOTE** : In onecompiler , The Data Creation and Data Entry Part are common for all sub-tasks. The SQL codes and their explanation for all the (Questions and its Answer) tasks are in the same file. You need to scroll down to arrive at every question and its SQL query along with its explanation. The main query or code is Inactivated by “- -”.

So whenever you have to check the query, just remove “- -” before ‘SELECT /INSERT’ command and verify it on Output prompt, and then reapply “- -” so that it does not overlap on other queries.

You will find comment “ Task Begins from Here” and from there apply as mentioned above.

**Task 2 Project: Employee Payroll Management System (PostgreSQL)**

**Objective:** Design and implement an employee payroll system to store, manage, and analyze salary data.

The project will include the following tasks :

**Database Setup:**

Create a database named payroll\_database. Create a table employees with columns: EMPLOYEE\_ID (integer), NAME (text), DEPARTMENT (text), EMAIL (text), PHONE\_NO (numeric), JOINING\_DATE (date), SALARY (numeric), BONUS (numeric), TAX\_PERCENTAGE (numeric).

**Data Entry:**

Insert 10 sample employee records

Payroll Queries:

a) Retrieve the list of employees sorted by salary in descending order.

b) Find employees with a total compensation (SALARY + BONUS) greater than $100,000.

c) Update the bonus for employees in the ‘Sales’ department by 10%.

d) Calculate the net salary after deducting tax for all employees.

e) Retrieve the average, minimum, and maximum salary per department.

**Advanced Queries:**

a) Retrieve employees who joined in the last 6 months.

b) Group employees by department and count how many employees each has.

c) Find the department with the highest average salary.

d) Identify employees who have the same salary as at least one other employe

**Main Code: Task wise begins from here**

--- Connecting to PostgreSQL Server and create the database:

**CREATE DATABASE payroll\_database;**

--- Connect to the newly created database and create the employees table:

**\c payroll\_database; *-- Connect to the database***

---- Creating Employee Table

CREATE TABLE employees (

EMPLOYEE\_ID INT PRIMARY KEY,

NAME VARCHAR(255),

DEPARTMENT VARCHAR(100),

EMAIL VARCHAR(255),

PHONE\_NO VARCHAR(20),

JOINING\_DATE DATE,

SALARY DECIMAL(10, 2),

BONUS DECIMAL(10, 2),

TAX\_PERCENTAGE DECIMAL(5, 2)

);

1. **Data Entry**

**---- Inserting sample data to the table**

INSERT INTO employees (EMPLOYEE\_ID, NAME, DEPARTMENT, EMAIL, PHONE\_NO, JOINING\_DATE, SALARY, BONUS, TAX\_PERCENTAGE)

VALUES

(1, 'Alice Smith', 'Sales', 'alice.s@example.com', '123-456-7890', '2022-03-15', 70000.00, 5000.00, 10.00),

(2, 'Bob Johnson', 'Marketing', 'bob.j@example.com', '098-765-4321', '2021-08-20', 65000.00, 3000.00, 12.00),

(3, 'Charlie Brown', 'Sales', 'charlie.b@example.com', '111-222-3333', '2023-01-10', 80000.00, 7000.00, 15.00),

(4, 'Diana Miller', 'IT', 'diana.m@example.com', '444-555-6666', '2022-11-01', 90000.00, 8000.00, 18.00),

(5, 'Eve Davis', 'HR', 'eve.d@example.com', '777-888-9999', '2023-04-01', 55000.00, 2000.00, 8.00),

(6, 'Frank White', 'Finance', 'frank.w@example.com', '222-333-4444', '2022-06-20', 100000.00, 10000.00, 20.00),

(7, 'Grace Lee', 'Sales', 'grace.l@example.com', '555-666-7777', '2023-02-14', 72000.00, 6000.00, 13.00),

(8, 'Harry Wilson', 'IT', 'harry.w@example.com', '888-999-0000', '2021-09-01', 95000.00, 9000.00, 19.00),

(9, 'Ivy Taylor', 'Marketing', 'ivy.t@example.com', '333-444-5555', '2022-07-25', 68000.00, 4000.00, 11.00),

(10, 'Jack Green', 'Finance', 'jack.g@example.com', '666-777-8888', '2023-05-01', 110000.00, 12000.00, 22.00);

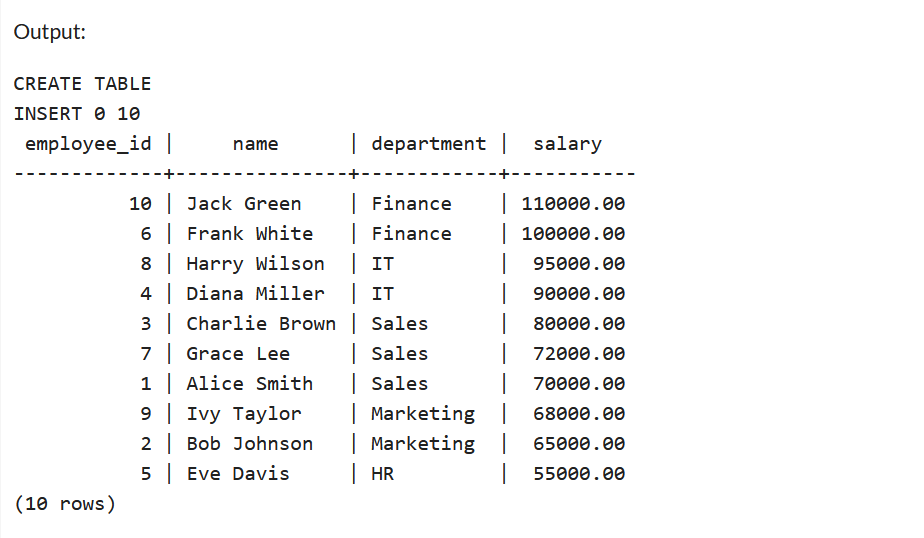
**B. Payroll Queries:**

**a) Retrieve the list of employees sorted by salary in descending order.**

SELECT \* FROM employees

ORDER BY SALARY DESC;

---- This query selects all columns from the employees table and orders the results by the SALARY column in descending order.

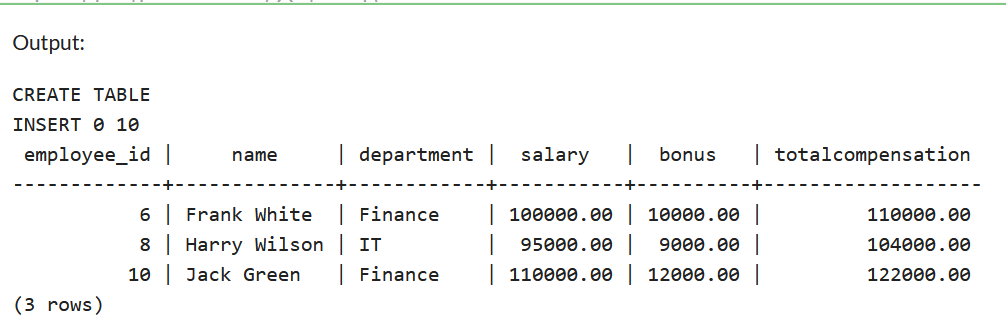


b**) Find employees with a total compensation (SALARY + BONUS) greater than $100,000.**

SELECT \* FROM employees

WHERE (SALARY + BONUS) > 100000;

---- This query filters the results to only include employees whose combined SALARY and BONUS is greater than 100,000.



**c) Update the bonus for employees in the 'Sales' department by 10%**

UPDATE employees

SET SALARY = SALARY + (SALARY \* 10/100)

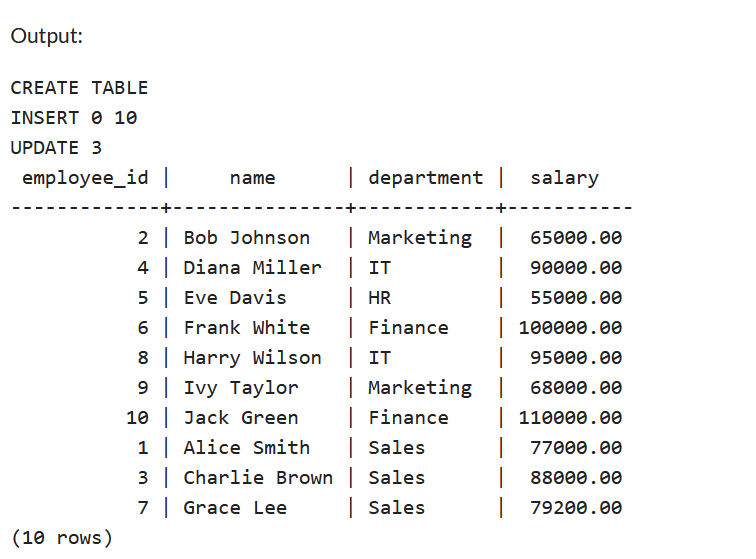
WHERE DEPARTMENT = 'Sales';

SELECT EMPLOYEE\_ID, NAME, DEPARTMENT, SALARY

FROM employees;

---- This query updates the BONUS column for all employees whose DEPARTMENT is 'Sales', increasing their bonus by 10%.

---- In the Output we see that Salaries of Employees with Department as Sales have 10% additional salaries now



d) **Calculate the net salary after deducting tax for all employees**

SELECT

NAME,

SALARY,

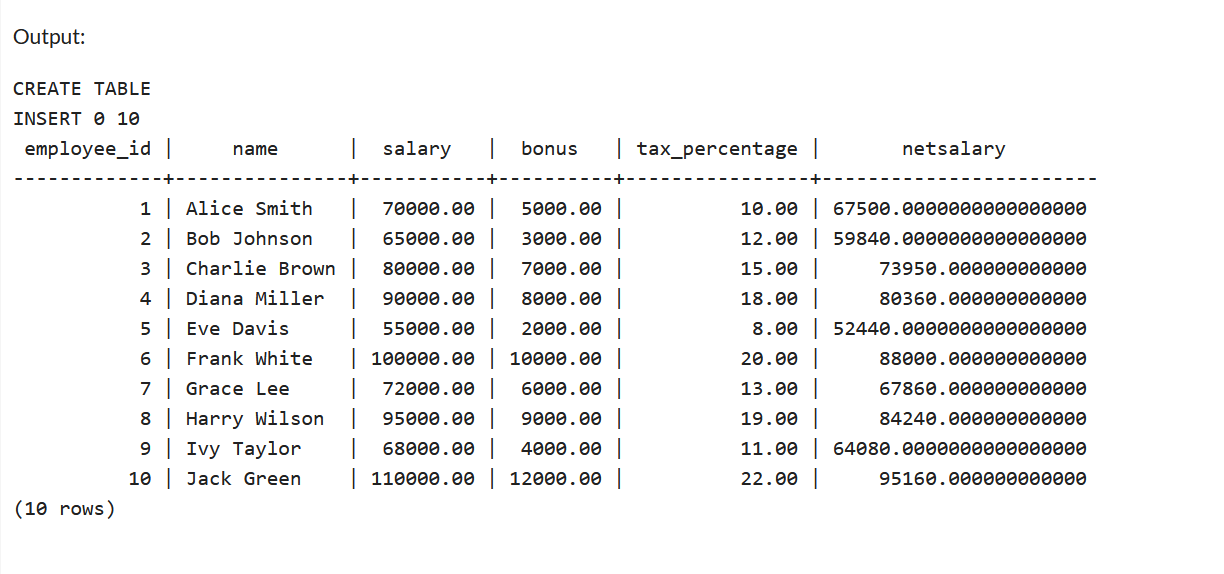
BONUS,

TAX\_PERCENTAGE,

(SALARY + BONUS) - ((SALARY + BONUS) \* (TAX\_PERCENTAGE / 100)) AS NET\_SALARY

FROM employees;

---- This query calculates the net salary by subtracting the calculated tax amount from the total compensation (salary plus bonus)



e**) Retrieve the average, minimum, and maximum salary per department**

SELECT

DEPARTMENT,

AVG(SALARY) AS AVERAGE\_SALARY,

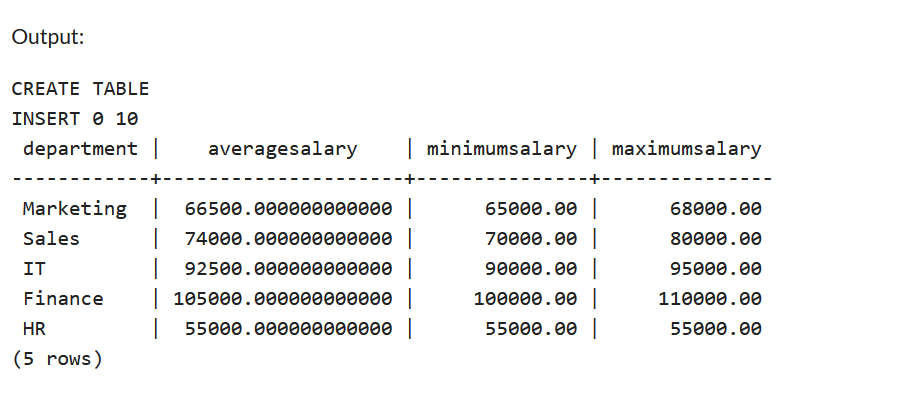
MIN(SALARY) AS MINIMUM\_SALARY,

MAX(SALARY) AS MAXIMUM\_SALARY

FROM employees

GROUP BY DEPARTMENT;

---- This query groups the results by DEPARTMENT and then calculates the average, minimum, and maximum salary for each department using the aggregate functions AVG, MIN, and MAX respectively.



**C. Advanced Queries:**

**a) Retrieve employees who joined in the last 6 months**

---- Since the sample data was with employee joining in year 2022, 2023 etc, it wouldn’t have provided correct result for this specific query

---- So I manipulated employee table by using UPDATE

UPDATE employees

SET JOINING\_DATE = CURRENT\_DATE - INTERVAL '2 months'

WHERE EMPLOYEE\_ID = 3;

UPDATE employees

SET JOINING\_DATE = CURRENT\_DATE - INTERVAL '1 month'

WHERE EMPLOYEE\_ID = 5;

UPDATE employees

SET JOINING\_DATE = CURRENT\_DATE - INTERVAL '5 months'

WHERE EMPLOYEE\_ID = 7;

SELECT

EMPLOYEE\_ID,

NAME,

DEPARTMENT,

JOINING\_DATE

FROM

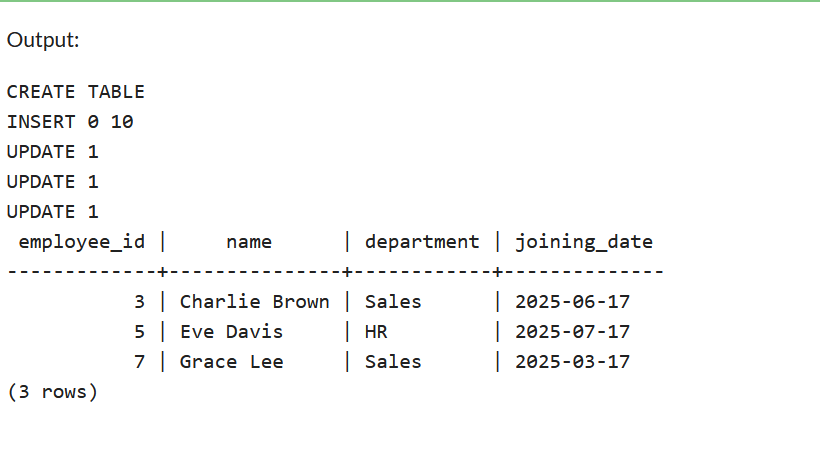
employees

WHERE

EXTRACT(YEAR FROM JOINING\_DATE) \* 12 + EXTRACT(MONTH FROM JOINING\_DATE) >=

EXTRACT(YEAR FROM CURRENT\_DATE - INTERVAL '6 months') \* 12 + EXTRACT(MONTH FROM CURRENT\_DATE - INTERVAL '6 months');

---- This query selects employees whose JOINING\_DATE is within the last 6 months from the current date.



b) **Group employees by department and count how many employees each has**

SELECT

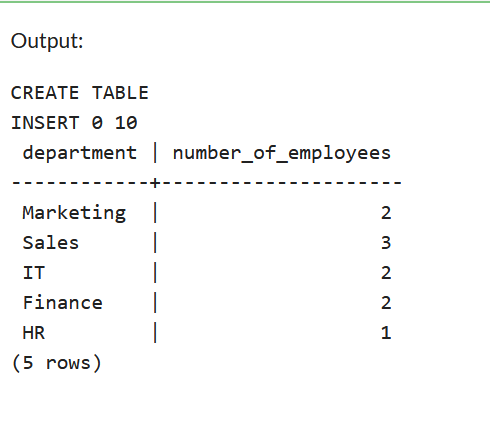
DEPARTMENT,

COUNT(EMPLOYEE\_ID) AS NUMBER\_OF\_EMPLOYEES

FROM employees

GROUP BY DEPARTMENT;

---- This query groups the results by DEPARTMENT and counts the number of employees in each department using the COUNT aggregate function.



c**) Find the department with the highest average salary**

SELECT

DEPARTMENT,

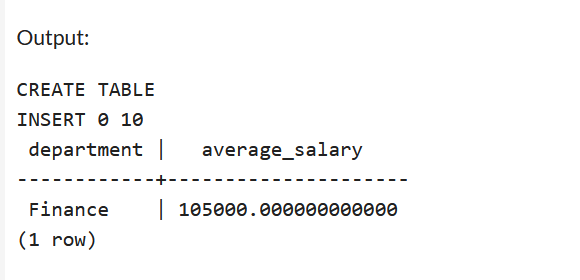
AVG(SALARY) AS AVERAGE\_SALARY

FROM employees

GROUP BY DEPARTMENT

ORDER BY AVERAGE\_SALARY DESC LIMIT 1;

---- This query calculates the average salary per department, orders them in descending order, and then retrieves the top 1, representing the department with the highest average salary.



d) **Identify employees who have the same salary as at least one other employee**

-- Manipulating Data in employees table by adding new employees with similar salaries

INSERT INTO employees (EMPLOYEE\_ID, NAME, DEPARTMENT, EMAIL, PHONE\_NO, JOINING\_DATE, SALARY, BONUS, TAX\_PERCENTAGE)

VALUES (11, 'Sam Peterson', 'Sales', 'alice.s@example.com', '123-456-7890', '2022-03-15', 70000.00, 5000.00, 10.00),

(12, 'Bruce Wayne', 'Marketing', 'bob.j@example.com', '098-765-4321', '2021-08-20', 65000.00, 3000.00, 12.00),

(13, 'Patric Jane', 'Sales', 'charlie.b@example.com', '111-222-3333', '2023-01-10', 80000.00, 7000.00, 15.00);

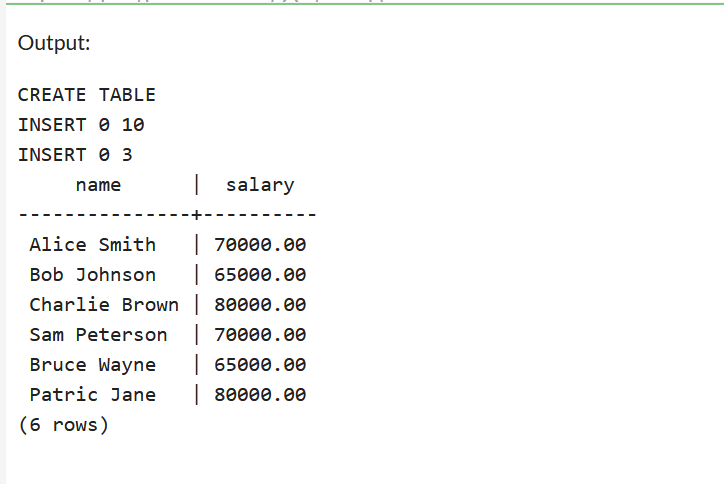
SELECT e1.NAME, e1.SALARY

FROM employees e1

JOIN employees e2 ON e1.SALARY = e2.SALARY

AND e1.EMPLOYEE\_ID <> e2.EMPLOYEE\_ID;

---- This query uses a self-join to compare employee salaries within the same table. It joins the employees table with itself (e1 and e2 aliases) where the salaries are equal but the employee IDs are different, thus identifying employees with matching salaries.



**END OF TASK**