```
--> Write down the SQL subquery considering below schema of Apple store database
Customer (Customer_id, first_name, Last_name)
Payment (customer_id, Product_id, Payment_id, amount, Payment_date)
Product (Product_id, Product_name, Product_type, Color)
i. Give the name of customers who have made the payment in the middle of 1st Aug
2023 and 10th Aug 2023.
ii. Give the list of product name whose color is red and type is iPhone.
iii. Give all the product names and product types which were bought by Shyam Patel
iv. Give the first name of customers whose total payment is greater than 2lack
rupees.

1. SELECT first_name, Last_name from Customer WHERE Customer_id IN (SELECT
customer_id FROM Payment WHERE Payment_date BETWEEN '2023-08-01' AND '2023-08-10' );

2.  SELECT Product_name FROM Product WHERE Color = 'red' AND Product_type =
'iPhone';

3.  SELECT p.Product_name, p.Product_type FROM Product p INNER JOIN Payment py ON
p.Product_id = py.Product_id
    INNER JOIN Customer c ON py.customer_id = c.Customer_id WHERE c.first_name =
'Shyam' AND c.Last_name = 'Patel';

4.  SELECT first_name FROM Customer WHERE Customer_id IN (SELECT customer_id FROM
Payment GROUP BY customer_id HAVING SUM(amount) > 200000);

------------------------------------------------------------------------------------
-------------------------------------------------------------

--> Write a SQL query considering below schema of database
Manager (Mid, Eid, Mname)
Department (Did,Mid, Dname, location)
Employee (Eid, Ename, mobile, salary, joining_date, Mid)
i.Give the name and salary of employees whose salary are greater than each and every
employees working under manager id 14.
ii.Give the name of employees who have not assigned any department.
iii.Give the employees’ name whose location is Kota.
iv.Give the name of manager along with count of employees assigned to him/her in
descending order.

1.  SELECT E.Ename, E.salary FROM Employee E WHERE E.salary > ALL (SELECT E2.salary
FROM Employee E2 WHERE E2.Mid = 14 );

2.  SELECT E.Ename FROM Employee E WHERE E.Eid NOT IN (SELECT D.Eid FROM Department
D );

3.  SELECT E.Ename FROM Employee E INNER JOIN Department D ON E.Mid = D.Mid WHERE
D.location = 'Kota';

4.  SELECT M.Mname AS Manager_Name, COUNT(E.Eid) AS Employee_Count FROM Manager M
LEFT JOIN Employee E ON M.Mid = E.Mid
```

```
    GROUP BY M.Mname ORDER BY Employee_Count DESC;

-------------------------------------------------------------------------------------
-----------------------------------------------------------

--> Write SQL query considering below schema of database
Film(film_id, title, legth, rental_rate)
Actor(film_id, actor_id, first_name, last_name)
Category(film_id, rating, language, release_year)
i.Give the name of actors whose actor id is 23
ii.Give the title of films whose id is between 25 and 40
iii.Give the name of actor whose last name contains Kapoor
iv.Give the title of film which was released in 2022
v.Give the name of actors played who have role in film title ‘Chhello divas’
vi.Give the rating and count of film in each rating.
vii.Give the titles of film in which Ranveer played the role as an actor
viii.Give the total count of film for each language available in database

1.  SELECT first_name, last_name FROM Actor WHERE actor_id = 23;

2.  SELECT title FROM Film WHERE film_id BETWEEN 25 AND 40;

3.  SELECT first_name, last_name FROM Actor WHERE last_name LIKE '%Kapoor%';

4.  SELECT title FROM Film INNER JOIN Category on Film.film_id = Category.film_id
WHERE YEAR(release_year) = 2022;

5.  SELECT A.first_name, A.last_name FROM Actor A JOIN Film F ON A.film_id =
F.film_id WHERE F.title = 'Chhello divas';

6.  SELECT rating, COUNT(*) AS film_count FROM Category GROUP BY rating;

7.  SELECT F.title FROM Film F JOIN Actor A ON F.film_id = A.film_id WHERE
A.first_name = 'Ranveer';

8.  SELECT language, COUNT(*) AS film_count FROM Category GROUP BY language;

-------------------------------------------------------------------------------------
-----------------------------------------------------

--> Consider following schema and write query statement for given statement
emp (eid,ename,city,dname,salary)
Project(eid,pid,pname,location)
(1) Display ename whose dname is computer.
(2) Display eid whose ename start from J.
(3) Display all details of emp whose salary is from 10000 to 20000.
(4) Display ename who is having maximum salary
(5) Display ename whose salary is higher than average salary of the employee
(6) Display ename whose pid is 3 and location is Mumbai
```

```
1.   SELECT ename FROM emp WHERE dname = 'computer';

2.   SELECT eid FROM emp WHERE ename LIKE 'J%';

3.   SELECT * FROM emp WHERE salary BETWEEN 10000 AND 20000;

4.   SELECT ename FROM emp WHERE salary = (SELECT MAX(salary) FROM emp);

5.   SELECT ename FROM emp WHERE salary > (SELECT AVG(salary) FROM emp);

6.   SELECT e.ename FROM emp e INNER JOIN Project p ON e.eid = p.eid WHERE p.pid = 3
AND p.location = 'Mumbai';

--------------------------------------------------------------------------------------
-------------------------------------------------

--> We have following relation orders(order_id,customer_id,order_date,amount)
1) Find out the number of orders for each customer by customer_id.
2) Find out the total amount by order_id and order_date.
3) Find out the number of orders for each customer by customer_id. Show only
customer_id with number of orders above 5.

1.   SELECT customer_id, COUNT(order_id) AS order_count FROM orders GROUP BY
customer_id;

2.   SELECT order_id, order_date, SUM(amount) AS total_amount FROM orders GROUP BY
order_id, order_date;

3.   SELECT customer_id, COUNT(order_id) AS order_count FROM orders GROUP BY
customer_id HAVING COUNT(order_id) > 5;

--------------------------------------------------------------------------------------
-------------------------------------------------

--> Write a query for the following.
employee(id,name,salary,address) department(d_id,d_name,id)
(1) Create a view department_details of department table.
(2) To join two existing tables using inner join.
(3) To drop a view.

1.   CREATE VIEW department_details AS SELECT d.*, e.name AS employee_name, e.salary
AS employee_salary, e.address AS employee_address
     FROM department d INNER JOIN employee e ON d.id = e.id;

2.   SELECT d.*, e.name AS employee_name, e.salary AS employee_salary, e.address AS
employee_address FROM department d
     INNER JOIN employee e ON d.id = e.id;

3.   DROP VIEW department_details;
```

--> A) Write pl pgSQL function to increment the employee’s salary by 10% if his/her department is ‘HR’ for given(inputted) employee_id
Schema Employee(Employee_id, first_name, last_name, department, Salary)
B) Write pl pgSQL function to find the number of Sunday between given dates.

```
1.   CREATE OR REPLACE FUNCTION increase_salary(employee_id INT)
RETURNS INTEGER
AS $$
DECLARE
  dept_name VARCHAR;
  emp_salary NUMERIC;
begin
  -- Get the department name and employee salary
  SELECT department, salary INTO dept_name, emp_salary
  FROM Employee
  WHERE Employee_id = employee_id;

  -- Check if the department is 'HR' and increment salary by 10%
  IF dept_name = 'HR' THEN
    UPDATE Employee
    SET salary = emp_salary * 1.10
    WHERE Employee_id = employee_id;
  END IF;
END;
$$
LANGUAGE plpgsql;
```

```
2.  CREATE OR REPLACE FUNCTION count_sundays(start_date
DATE, end_date DATE) RETURNS INTEGER AS $$
DECLARE
num_sundays INTEGER := 0;
curr_date DATE := start_date;
BEGIN
WHILE curr_date <= end_date LOOP
IF EXTRACT(DOW FROM curr_date) = 0 THEN
-- 0 represents Sunday
num_sundays := num_sundays + 1;
END IF;
curr_date := curr_date + INTERVAL '1 day';
END LOOP;
RETURN num_sundays;
END;
$$ LANGUAGE plpgsql;
SELECT count_sundays('2023-08-01', '2023-08-31');
```

--> Write pl pgSQL block using Explicit cursor to insert the whole tuple from film table to film_pay table if amount is greater than 5$
Schema
Film (film_id, title, length, amount,rating)
Film_pay(film_id, title, length, amount,rating)

```
DO $$
DECLARE
    film_record Film%ROWTYPE;
BEGIN
    -- Open a cursor to fetch records from the Film table
    FOR film_record IN (SELECT * FROM Film) LOOP
        -- Check if the amount is greater than $5
        IF film_record.amount > 5 THEN
            -- Insert the record into the Film_pay table
            INSERT INTO Film_pay (film_id, title, length, amount, rating)
            VALUES (film_record.film_id, film_record.title, film_record.length,
film_record.amount, film_record.rating);
        END IF;
    END LOOP;
END $$;
```

--> Write pl pg sql using trigger for insertion of first_name, last_name, amount and payment_id into customer_backup table when deletion happens from payment table considering below schema
Customer_backup (first_name, last_name, amount, payment_id)
Payment (Payment_id, Customer_id, amount)
Customer (Customer_id, first_name, last_name)

```
 CREATE OR REPLACE FUNCTION payment_delete_trigger()
RETURNS TRIGGER AS $$
BEGIN
    -- Insert the deleted data into the customer_backup table
    INSERT INTO customer_backup (first_name, last_name, amount, payment_id)
    SELECT C.first_name, C.last_name, OLD.amount, OLD.payment_id
    FROM Customer C
    JOIN OLD Payment ON C.Customer_id = OLD.Customer_id;

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

-- Create a trigger that fires on DELETE from the Payment table
CREATE TRIGGER payment_delete
AFTER DELETE ON Payment
FOR EACH ROW
```

```
EXECUTE FUNCTION payment_delete_trigger();
```

```
------------------------------------------------------------------------------------
---------------------------------------
```

--> We are running an art gallery. We have a database with four tables paintings, artists, collectors, and sales as:
Paintings (p_id, name, a_id, listed_price)
Artists (a_id, first_name, last_name)
Customer (c_id, first_name, last_name)
Sales (s_id, date, p_id, a_id, c_id, sales_price).
Write subqueries to retrieve the following data.
1) Find the paintings that were sold in first thirteen days of month August,2023.
2) Display painting name and its listed price with its sales_price which is sold out at a price higher than the listed price to any customer.
3) Retrieve details of paintings that were sold at a price is equal the listed price to all customer.
4) Display the name of customer who has bought at least one painting.
5) Finds the total sales value for each artist.

```
1.  SELECT name FROM Paintings WHERE p_id IN (SELECT DISTINCT s.p_id FROM Sales s
    WHERE EXTRACT(MONTH FROM s.date) = 8 AND EXTRACT(DAY FROM s.date) <= 13 AND
EXTRACT(YEAR FROM s.date) = 2023);

2.  SELECT p.name, p.listed_price, s.sales_price FROM Paintings p INNER JOIN Sales s
ON p.p_id = s.p_id WHERE s.sales_price > p.listed_price;

3.  SELECT * FROM Paintings WHERE p_id IN (SELECT DISTINCT s.p_id FROM Sales s WHERE
s.sales_price = Paintings.listed_price);

4.  SELECT DISTINCT c.first_name, c.last_name FROM Customer c INNER JOIN Sales s ON
c.c_id = s.c_id;

5.  SELECT a.first_name, a.last_name, SUM(s.sales_price) AS total_sales_value FROM
Artists a LEFT JOIN Sales s ON a.a_id = s.a_id GROUP BY a.first_name,a.last_name;
```

```
------------------------------------------------------------------------------------
--------------------------------------
```

--> Using the following schema, write SQL statement to fetch the correct data.
Insurance_Company (c_id, c_name, city, state, policy_no, policy_name, premium, id, a_id)
Agent (a_id, a_name, address, a_city, a_state, c_name, salary, incentive)
Customer (id, name, age cust_address, cust_city, cust_state, a_id, policy_no)
1) Display the details of customer who have taken policy from ‘Ahmedabad Insurance Company’.
2) Display agent name who have not sold even a single policy.
3) Display the name of Insurance company who has maximum number of customers.

```
1.   SELECT * FROM Customer JOIN Insurance_Company ON Customer.policy_no =
Insurance_Company.policy_no
     WHERE Insurance_Company.c_name = 'Ahmedabad Insurance Company';

2.   SELECT DISTINCT a_name FROM Agent LEFT JOIN Customer ON Agent.a_id =
Customer.a_id WHERE Customer.a_id IS NULL;

3.   SELECT IC.c_name FROM Insurance_Company IC JOIN Customer C ON IC.policy_no =
C.policy_no
     GROUP BY IC.c_name ORDER BY COUNT(C.id) DESC LIMIT 1;

-------------------------------------------------------------------------------------
--------------------------------------------

--> Consider the following student relation:
Student (name, rollno, marks, percentage, address, dob)
1)Create a view of ‘Student_info’ from relation Student with fields name, rollno,
percentage.
2) Rename the view created, as ‘Student_Academics’

1.   CREATE VIEW Student_info AS SELECT name, rollno, percentage FROM Student;

2.   ALTER VIEW Student_info RENAME TO Student_Academics;

-------------------------------------------------------------------------------------
----------------------------------------------

--> Consider following schema and write SQL statements for given queries without
using subqueries.
Faculty (f_id, f_name, address, city, state, age, salary, d_id)
Department (dep_id, d_name, course, location, sub_code, subject)
Student (enrl_no, s_name, s_address, s_city, s_state, s_age, course, f_id, hobby)
1)Display name of faculty and student who are not living in the same city.
2)Find out how much amount is spent on faculties as their salary who are working in
‘IT Engineering Department’.
3)Display department name, course, subject code and subject of person who have
enrolled as a student.
4)Display id and name of faculty who are working in ‘Computer Engineering’
department and have a salary more than Rs. 60000.

1.   SELECT DISTINCT F.f_name AS faculty_name, S.s_name AS student_name FROM Faculty
F, Student S WHERE F.city <> S.s_city;

2.   SELECT SUM(F.salary) AS total_salary FROM Faculty F INNER JOIN Department D ON
F.d_id = D.dep_id
     WHERE D.d_name = 'IT Engineering Department';

3.   SELECT D.d_name AS department_name, S.course, D.sub_code, D.subject FROM
Student S INNER JOIN Department D ON S.course = D.course;

4.   SELECT F.f_id, F.f_name FROM Faculty F INNER JOIN Department D ON F.d_id =
```

D.dep_id WHERE D.d_name = 'Computer Engineering' AND F.salary > 60000;

--> Prepare a PL/pgSQL block that simulates an ATM transaction system.
•If the withdrawal amount is less than or equal to 0, the block should display an "Invalid withdrawal amount" message.
•If the withdrawal amount is greater than the account balance, the block should display an "Insufficient funds" message.
•If the withdrawal amount is up to Rs. 20000, the block should perform a withdrawal and deduct the amount from the account balance.
•If the withdrawal amount is up to Rs. 40000, the block should perform a withdrawal, deduct the amount, and apply a transaction fee 5%.
•For withdrawal amounts greater than Rs. 40000, the block should perform a withdrawal, deduct the amount, and apply a higher transaction fee 18%.
The block should also display the current account balance after each scenario.

```
DO $$
DECLARE
    account_balance NUMERIC := 100000; -- Initial account balance
    withdrawal_amount NUMERIC := 25000; -- Specify the withdrawal amount here

BEGIN
    IF withdrawal_amount <= 0 THEN
        RAISE EXCEPTION 'Invalid withdrawal amount';
    ELSIF withdrawal_amount > account_balance THEN
        RAISE EXCEPTION 'Insufficient funds';
    ELSE
        IF withdrawal_amount <= 20000 THEN
            account_balance := account_balance - withdrawal_amount;
        ELSIF withdrawal_amount <= 40000 THEN
            account_balance := account_balance - withdrawal_amount -
(withdrawal_amount * 0.05); -- 5% transaction fee
        ELSE
            account_balance := account_balance - withdrawal_amount -
(withdrawal_amount * 0.18); -- 18% transaction fee
        END IF;

        -- Display the current account balance
        RAISE NOTICE 'Transaction successful! Current account balance: Rs. %',
account_balance;
    END IF;
END;
$$;
```

--> Calculate the total price (numeric) of all products with a price greater than

```
Rs. 500 using a cursor
    from following schema: Products (p_id int, p_name varchar (100), price numeric).

    Display all product name and price with total price of products.

DO $$
DECLARE
    total_price numeric := 0;
    product_name varchar(100);
    product_price numeric;

    -- Declare a cursor to fetch products with price > 500
    product_cursor CURSOR FOR
        SELECT p_name, price
        FROM Products
        WHERE price > 500;
BEGIN
    -- Open the cursor
    OPEN product_cursor;

    -- Loop through the cursor and calculate the total price
    LOOP
        FETCH product_cursor INTO product_name, product_price;

        -- Exit the loop when no more rows are found
        EXIT WHEN NOT FOUND;

        -- Calculate total price
        total_price := total_price + product_price;

        -- Display product name and price
        RAISE NOTICE 'Product: %, Price: Rs. %', product_name, product_price;
    END LOOP;

    -- Close the cursor
    CLOSE product_cursor;

    -- Display the total price of products
    RAISE NOTICE 'Total Price of Products: Rs. %', total_price;
END;
$$;
```

--> Create PL/pgSQL procedure for the increment of employees where in salary less than 35000 will get hike of 15% in their previous salary and other will get 10% hike in their previous salary. Using following schema, Employees (id, name, department, salary) call the procedure by id and print employee’s name with their updated salary.

```
CREATE OR REPLACE FUNCTION increment_salary(employee_id INT)
RETURNS VOID AS $$
DECLARE
    emp_name VARCHAR(255);
    emp_salary NUMERIC(10, 2);
BEGIN
    -- Get the employee's name and salary
    SELECT name, salary INTO emp_name, emp_salary
    FROM Employees
    WHERE id = employee_id;

    -- Check if the employee's salary is less than 35000
    IF emp_salary < 35000 THEN
        -- Increment salary by 15%
        emp_salary := emp_salary + (emp_salary * 0.15);
    ELSE
        -- Increment salary by 10%
        emp_salary := emp_salary + (emp_salary * 0.10);
    END IF;

    -- Update the employee's salary in the Employees table
    UPDATE Employees
    SET salary = emp_salary
    WHERE id = employee_id;

    -- Print the employee's name and updated salary
    RAISE NOTICE 'Employee %: Updated Salary: $%', emp_name, emp_salary;

    -- Commit the transaction
    COMMIT;
END;
$$ LANGUAGE plpgsql;
```

--> You are designing a PostgreSQL database for a library management system. Each book in the library has multiple copies, and you want to implement a feature that allows users to check the availability of a specific book by its ISBN (International Standard Book Number).Write Pl/pgsql block using stored function. If book is available then it must return1else return 0. Consider relation:Books(books_isbn,book_name) Now what you will write to check availability of book having isbn no-9780451524935

```
CREATE OR REPLACE FUNCTION check_book_availability(isbn TEXT)
RETURNS INTEGER AS $$
DECLARE
    available_count INTEGER;
BEGIN
```

```
    -- Initialize the available_count to 0
    available_count := 0;

    -- Check the number of available copies of the book
    SELECT COUNT(*)
    INTO available_count
    FROM Books
    WHERE books_isbn = isbn;

    -- Return 1 if at least one copy is available, else return 0
    IF available_count > 0 THEN
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

--> You are responsible for maintaining a PostgreSQL database for a shipping company.Write a Pl/pgsql Stored Procedure to insert data in status_logs.Let name of Procedure be ‘update_shipment_status’ Consider relation: status_logs(shipment_id, new_status, update_time) Insert below data with the help of Procedure you created shipment_id=123,new_status=delivered,update_time=2023-09-15 10:30:00.

```
CREATE OR REPLACE PROCEDURE update_shipment_status(
    shipment_id INT,
    new_status VARCHAR(255),
    update_time TIMESTAMP
)
LANGUAGE plpgsql AS $$
BEGIN
    INSERT INTO status_logs(shipment_id, new_status, update_time)
    VALUES (shipment_id, new_status, update_time);

    COMMIT;
END;
$$;

CALL update_shipment_status(123, 'delivered', '2023-09-15 10:30:00');
```

--> Write PL/pgSQL program that calculates the factorial of a number=5 using a loop.

```
CREATE OR REPLACE FUNCTION calculate_factorial(num INT)
```

```
RETURNS INT
LANGUAGE plpgsql
AS $$
DECLARE
    factorial INT := 1;
    i INT := 1;
BEGIN
    -- Check if the input number is less than 0
    IF num < 0 THEN
        RAISE EXCEPTION 'Input number must be non-negative';
    END IF;

    -- Calculate the factorial using a loop
    WHILE i <= num LOOP
        factorial := factorial * i;
        i := i + 1;
    END LOOP;

    -- Return the calculated factorial
    RETURN factorial;
END;
$$;

SELECT calculate_factorial(5);
```

--> Write PL/pgSQL program to calculate the sum of first 100 even numbers

```
DO $$
DECLARE
    even_sum INT := 0;
    current_number INT := 2;
    count_even INT := 0;
BEGIN
    WHILE count_even < 100 LOOP
        even_sum := even_sum + current_number;
        current_number := current_number + 2;
        count_even := count_even + 1;
    END LOOP;

    RAISE NOTICE 'The sum of the first 100 even numbers is %', even_sum;
END;
$$ LANGUAGE plpgsql;
```

--> Write a PL/SQL block using an explicit cursor that will transfer the record of

account no, customer name and balance from the “account” table to the “branch_surat” table if the branch name is surat in the “account” table. Furthermore, delete the record from the “account” table whichever record transfers to the “branch_surat” table.
Use the following tables:
account (ano, customer_name, balance, branch_name)
branch_surat (ano, customer_name, balance)

```
DO $$
DECLARE
    -- Declare cursor for selecting records from the account table
    CURSOR account_cursor IS
        SELECT ano, customer_name, balance
        FROM account
        WHERE branch_name = 'surat';

    -- Declare variables to store account information
    account_ano account.ano%TYPE;
    account_customer_name account.customer_name%TYPE;
    account_balance account.balance%TYPE;
BEGIN
    -- Open the cursor
    OPEN account_cursor;

    -- Loop through the cursor results
    LOOP
        -- Fetch a record from the cursor
        FETCH account_cursor INTO account_ano, account_customer_name,
account_balance;

        -- Exit the loop if there are no more records
        EXIT WHEN account_cursor%NOTFOUND;

        -- Insert the fetched record into the branch_surat table
        INSERT INTO branch_surat (ano, customer_name, balance)
        VALUES (account_ano, account_customer_name, account_balance);

        -- Delete the record from the account table
        DELETE FROM account WHERE ano = account_ano;
    END LOOP;

    -- Close the cursor
    CLOSE account_cursor;

    -- Commit the transaction to save changes
    COMMIT;
END;
$$
```

--> In PostgreSQL, create a PL/SQL block that defines a function named calculate_salary_bonus which takes two parameters: employee_id and bonus_percentage.

This function should calculate the bonus amount for the employee with the given employee_id based on their current salary and the provided bonus percentage. Assume “employee” table with fields – eid, ename and salary.

```
CREATE OR REPLACE FUNCTION calculate_salary_bonus(employee_id INT, bonus_percentage
DECIMAL)
RETURNS DECIMAL AS
$$
DECLARE
    employee_salary DECIMAL;
    bonus DECIMAL;
BEGIN
    -- Get the salary of the employee with the provided employee_id
    SELECT salary INTO employee_salary FROM employee WHERE eid = employee_id;

    -- Calculate the bonus amount
    bonus := (employee_salary * bonus_percentage) / 100.0;

    RETURN bonus;
END;
$$
LANGUAGE plpgsql;
```


--> In PostgreSQL, create a PL/SQL block that defines a procedure named update_employee_salary which takes two parameters: employee_id and new_salary. This procedure should update the salary of the employee with the given employee_id to the new salary value. Assume “employee” table with fields – eid, ename and salary.

```
CREATE OR REPLACE PROCEDURE update_employee_salary(
    IN employee_id INT,
    IN new_salary DECIMAL
)
LANGUAGE plpgsql
AS $$
BEGIN
    -- Update the salary of the employee with the provided employee_id
    UPDATE employee
    SET salary = new_salary
    WHERE eid = employee_id;

    -- Commit the transaction to save the changes
    COMMIT;
```

```
END;
$$;

CALL update_employee_salary(1, 50000); -- Update the salary of employee with eid = 1
to $50,000

-------------------------------------------------------------------------------------
----------------------------

--> Consider following schema and write SQL for given statements.
title (id,designation,DOJ)
bonus(id,bonus_date,amount)
1)Retrieve the employees who haven't received any bonuses
2)Retrieve the total bonus amount received by each employee:
3)Retrieve the highest bonus amount received:
4)List out id's whose bonus amount at most 4000 and designation is admin

SELECT t.id, t.designation, t.DOJ
FROM title t
WHERE t.id NOT IN (
    SELECT DISTINCT b.id
    FROM bonus b
);


SELECT t.id, t.designation, t.DOJ, COALESCE(SUM(b.amount), 0) AS total_bonus_amount
FROM title t
LEFT JOIN bonus b ON t.id = b.id
GROUP BY t.id, t.designation, t.DOJ;


SELECT MAX(b.amount) AS highest_bonus_amount
FROM bonus b;


SELECT t.id
FROM title t
INNER JOIN bonus b ON t.id = b.id
WHERE b.amount <= 4000 AND t.designation = 'admin';

-------------------------------------------------------------------------------------
-----------------------

--> Consider following schema and write SQL for given statements.
Student (RollNo, Name, DeptCode, City)
Department (DeptCode, DeptName)
Result (RollNo, Semester, SPI)
1)Retrieve all students' names and their respective department names:
2)Retrieve the average SPI (Semester Performance Index) for each student:
3)Retrieve the students who have the highest SPI in a “desired_semester”
```

4)Retrieve the students who belong to a “specific_city” and their department names:

```
1.  SELECT s.Name AS StudentName, d.DeptName
FROM Student s
INNER JOIN Department d ON s.DeptCode = d.DeptCode;

2.  SELECT RollNo, AVG(SPI) AS AverageSPI
FROM Result
GROUP BY RollNo;

3.  SELECT s.RollNo, s.Name AS StudentName, MAX(r.SPI) AS HighestSPI
FROM Student s
INNER JOIN Result r ON s.RollNo = r.RollNo
WHERE r.Semester = 'desired_semester'
GROUP BY s.RollNo, s.Name HAVING MAX(r.SPI) = (SELECT MAX(SPI) FROM Result WHERE
Semester = 'desired_semester');

4.   SELECT s.Name AS StudentName, d.DeptName
FROM Student s
INNER JOIN Department d ON s.DeptCode = d.DeptCode
WHERE s.City = 'specific_city';
```


--> we have following relations:
Supplier(S#,sname,status,city)
Parts(P#,pname,color,weight,city)
SP(S#,P#,quantity)
Answer the following queries.
(1) Find s# of supplier who supplies ‘GREEN’ part.
(2) Count number of supplier who supplies ‘red’ part.
(3) Sort the supplier table by sname?
(4) List suppliers who supply parts to more than one city

```
SELECT DISTINCT SP.S#
FROM Parts P
JOIN SP ON P.P# = SP.P#
WHERE P.color = 'GREEN';


SELECT COUNT(DISTINCT SP.S#)
FROM Parts P
JOIN SP ON P.P# = SP.P#
WHERE P.color = 'RED';


SELECT *
FROM Supplier
ORDER BY sname;
```

```
SELECT S#
FROM SP
GROUP BY S#
HAVING COUNT(DISTINCT city) > 1;

------------------------------------------------------------------------------------
------------------------------

--> A stored function is created to perform the acct_no check operation. f_ChkAcctNo
() is the name of function
    which accepts a variable acct_no from the user and returns value 0 if acct_no
does not exist or 1 if acct_no exists.
    Assume account table with fields: account_number, name, type and balance.


CREATE OR REPLACE FUNCTION f_ChkAcctNo(acct_no VARCHAR2) RETURN NUMBER IS
   acct_count NUMBER;
BEGIN
   -- Check if the account number exists in the "account" table
   SELECT COUNT(*) INTO acct_count
   FROM account
   WHERE account_number = acct_no;

   -- Return 1 if the account number exists, otherwise return 0
   IF acct_count > 0 THEN
      RETURN 1;
   ELSE
      RETURN 0;
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      -- Handle exceptions as needed (e.g., return -1 for errors)
      RETURN -1;
END f_ChkAcctNo;


DECLARE
   account_number VARCHAR2(20) := '12345'; -- Replace with the account number to
check
   result NUMBER;
BEGIN
   result := f_ChkAcctNo(account_number);
   IF result = 1 THEN
      DBMS_OUTPUT.PUT_LINE('Account exists.');
   ELSE
      DBMS_OUTPUT.PUT_LINE('Account does not exist.');
   END IF;
END;
```

```
-----------------------------------------------------------------------------------
---------------------------------

--> Write a Stored procedure to Insert Data in Departments (dept_id, dept_name, 
balance) and
    transactions (transaction_id, from_dept_id, to_dept_id, amount, 
transaction_date) table.

CREATE OR REPLACE PROCEDURE insert_department_and_transaction(
    p_dept_id INT,
    p_dept_name VARCHAR(255),
    p_balance NUMERIC,
    p_transaction_id INT,
    p_from_dept_id INT,
    p_to_dept_id INT,
    p_amount NUMERIC,
    p_transaction_date TIMESTAMP
)
AS
BEGIN
    -- Insert data into the Departments table
    INSERT INTO Departments(dept_id, dept_name, balance)
    VALUES (p_dept_id, p_dept_name, p_balance);

    -- Insert data into the Transactions table
    INSERT INTO Transactions(transaction_id, from_dept_id, to_dept_id, amount, 
transaction_date)
    VALUES (p_transaction_id, p_from_dept_id, p_to_dept_id, p_amount, 
p_transaction_date);

    -- Commit the transaction
    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Data inserted into Departments and Transactions tables.');
EXCEPTION
    WHEN OTHERS THEN
        -- Handle exceptions if needed
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
/

-----------------------------------------------------------------------------------
---------------------------

-->  An expression below creates a procedure emp_infor that accept two parameters, 
employeeno and job_title, then pass them to explicit cursor named emp_list to 
retrieve
the firstname, lastname, hiredate, job_id and salary, of all employees earning a 
salary less
```

that 11000 and have an “O” as the second letter in the lastname. Create an anonymous

block that will call the procedure to display the details of employee as per entry of
job_title and employeeno parameters. Use this table Employee (employeeno, firstname,

lastname, hiredate, job_id ,salary, job_title)

```
CREATE OR REPLACE PROCEDURE emp_info(employeeno INT, job_title VARCHAR)
LANGUAGE plpgsql
AS $$
DECLARE
    emp_record Employee%ROWTYPE;
    emp_cursor CURSOR(emp_no INT, job_title VARCHAR) IS
        SELECT firstname, lastname, hiredate, job_id, salary
        FROM Employee
        WHERE salary < 11000 AND substring(lastname FROM 2 FOR 1) = 'O'
        AND (employeeno IS NULL OR employeeno = emp_no)
        AND (job_title IS NULL OR job_title = job_id);
BEGIN
    OPEN emp_cursor(employeeno, job_title);
    FETCH NEXT FROM emp_cursor INTO emp_record;

    IF NOT FOUND THEN
        RAISE NOTICE 'No employees found matching the criteria.';
    ELSE
        WHILE FOUND LOOP
            RAISE NOTICE 'Employee: % %, Hire Date: %, Job ID: %, Salary: %',
                         emp_record.firstname, emp_record.lastname,
                         emp_record.hiredate, emp_record.job_id, emp_record.salary;
            FETCH NEXT FROM emp_cursor INTO emp_record;
        END LOOP;
    END IF;

    CLOSE emp_cursor;
END;
$$;

DO $$
DECLARE
    employeeno INT := 123; -- Replace with the desired employeeno
    job_title VARCHAR := 'JOB_TITLE'; -- Replace with the desired job_title
BEGIN
    CALL emp_info(employeeno, job_title);
END;
$$;
```