**CRPostgreSQL Practice Task Sheet**

### **📃 Sample Schemas**

-- Employees and Departments

CREATE TABLE departments (

department\_id SERIAL PRIMARY KEY,

department\_name TEXT

);

CREATE TABLE employees (

employee\_id SERIAL PRIMARY KEY,

first\_name TEXT,

last\_name TEXT,

salary NUMERIC,

department\_id INTEGER REFERENCES departments(department\_id),

manager\_id INTEGER,

hire\_date DATE

);

-- Customers and Orders

CREATE TABLE customers (

customer\_id SERIAL PRIMARY KEY,

customer\_name TEXT,

email TEXT

);

CREATE TABLE orders (

order\_id SERIAL PRIMARY KEY,

customer\_id INTEGER REFERENCES customers(customer\_id),

order\_date DATE,

amount NUMERIC

);

-- Students Table

CREATE TABLE students (

student\_id SERIAL PRIMARY KEY,

student\_name TEXT,

marks NUMERIC

);

-- Users Table

CREATE TABLE users (

user\_id SERIAL PRIMARY KEY,

name TEXT,

email TEXT

);

### **📘 Task Sections**

#### **🔹 Section 1: Views (5 Tasks)**

1. **Create a View:** Create a view employee\_summary showing first\_name, last\_name, department\_name, and salary.

CREATE VIEW employee\_summary AS

SELECT

e.first\_name,

e.last\_name,

d.department\_name,

e.salary

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id;

1. **View with Filtered Join:** Create a view high\_earners that includes employees with salary > 80,000 and their department names.

CREATE VIEW high\_earners AS

SELECT

e.first\_name,

e.last\_name,

d.department\_name,

e.salary

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

WHERE e.salary > 80000;

1. **Aggregated View:** Create a view department\_avg\_salary that displays department name and average salary.

CREATE VIEW department\_avg\_salary AS

SELECT

d.department\_name,

AVG(e.salary) AS avg\_salary

FROM employees e

JOIN departments d ON e.department\_id = d.department\_id

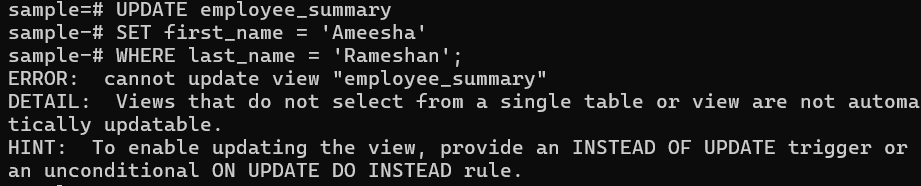
GROUP BY d.department\_name;

1. **Update Through a View:** Try updating an employee's name through the employee\_summary view. What do you observe?

UPDATE employee\_summary

SET first\_name = 'Ameesha'

WHERE last\_name = 'Rameshan';



1. **Drop and Recreate:** Drop and recreate the employee\_summary view to also include the hire\_date.  
   DROP VIEW IF EXISTS employee\_summary;  
   CREATE VIEW employee\_summary AS  
   SELECT   
    e.first\_name,  
    e.last\_name,  
    d.department\_name,  
    e.salary,  
    e.hire\_date  
   FROM employees e  
   JOIN departments d ON e.department\_id = d.department\_id;

#### **🔹 Section 2: Common Table Expressions (CTEs) (5 Tasks)**

1. **Simple CTE:** Use a CTE to list all employees earning more than 50,000 and filter those in the IT department.  
   WITH high\_paid AS (  
    SELECT \*  
    FROM employees  
    WHERE salary > 50000  
   )  
   SELECT \*  
   FROM high\_paid  
   JOIN departments d ON high\_paid.department\_id = d.department\_id  
   WHERE d.department\_name = 'IT';
2. **Recursive CTE (Hierarchy):** Using the manager\_id field in employees, create a recursive CTE that returns the hierarchy of employees starting from employee\_id = 1.

WITH RECURSIVE emp\_hierarchy AS (

SELECT employee\_id, first\_name, last\_name, manager\_id, 1 AS level

FROM employees

WHERE employee\_id = 1

UNION ALL

SELECT e.employee\_id, e.first\_name, e.last\_name, e.manager\_id, h.level + 1

FROM employees e

JOIN emp\_hierarchy h ON e.manager\_id = h.employee\_id

)

SELECT \* FROM emp\_hierarchy;

1. **CTE with Aggregation:** Use a CTE to calculate total order amount per customer and show only those whose total is over 5000.

WITH customer\_totals AS (

SELECT customer\_id, SUM(amount) AS total\_amount

FROM orders

GROUP BY customer\_id

)

SELECT c.customer\_name, ct.total\_amount

FROM customer\_totals ct

JOIN customers c ON ct.customer\_id = c.customer\_id

WHERE ct.total\_amount > 5000;

1. **CTE with Join:** Use a CTE to display all orders with their respective customer names where the amount > 1000.

WITH big\_orders AS (

SELECT \*

FROM orders

WHERE amount > 1000

)

SELECT bo.order\_id, c.customer\_name, bo.amount

FROM big\_orders bo

JOIN customers c ON bo.customer\_id = c.customer\_id;

1. **Multiple CTEs:** First CTE: top 5 highest paid employees. Second CTE: their department names. Join and display.  
   WITH top5\_employees AS (  
    SELECT \*  
    FROM employees  
    ORDER BY salary DESC  
    LIMIT 5  
   ),  
   departments\_info AS (  
    SELECT e.employee\_id, e.first\_name, e.salary, d.department\_name  
    FROM top5\_employees e  
    JOIN departments d ON e.department\_id = d.department\_id  
   )  
   SELECT \* FROM departments\_info;

#### **🔹 Section 3: Window Functions (5 Tasks)**

1. **ROW\_NUMBER():** Display employee name and ROW\_NUMBER() partitioned by department and ordered by salary descending.  
   SELECT   
    first\_name,  
    department\_id,  
    salary,  
    ROW\_NUMBER() OVER (PARTITION BY department\_id ORDER BY salary DESC) AS row\_num  
   FROM employees;
2. **Running Total:** For each customer, show a running total of their amount spent (orders table).

SELECT

customer\_id,

order\_date,

amount,

SUM(amount) OVER (PARTITION BY customer\_id ORDER BY order\_date) AS running\_total

FROM orders;

1. **Rank vs Dense Rank:** Use both RANK() and DENSE\_RANK() to rank employees within departments by salary. Note differences.

SELECT

first\_name,

department\_id,

salary,

RANK() OVER (PARTITION BY department\_id ORDER BY salary DESC) AS rank,

DENSE\_RANK() OVER (PARTITION BY department\_id ORDER BY salary DESC) AS dense\_rank

FROM employees;

1. **LEAD and LAG:** For each customer, display current, previous, and next order amount using LAG() and LEAD().

SELECT

customer\_id,

order\_date,

amount,

LAG(amount) OVER (PARTITION BY customer\_id ORDER BY order\_date) AS prev\_amount,

LEAD(amount) OVER (PARTITION BY customer\_id ORDER BY order\_date) AS next\_amount

FROM orders;

1. **PERCENT\_RANK():** Use PERCENT\_RANK() to rank students by their marks.  
   SELECT   
    student\_name,  
    marks,  
    PERCENT\_RANK() OVER (ORDER BY marks DESC) AS percent\_rank  
   FROM students;

#### **🔹 Section 4: Indexes (5 Tasks)**

1. **Basic Index:** Create an index on the email column of the users table.  
   CREATE INDEX idx\_users\_email ON users(email);
2. **Expression Index:** Create an index on LOWER(email) of users for case-insensitive searches.  
   CREATE INDEX idx\_users\_lower\_email ON users(LOWER(email));
3. **Composite Index:** Create a multi-column index on last\_name and first\_name of employees.  
   CREATE INDEX idx\_emp\_name ON employees(last\_name, first\_name);
4. **EXPLAIN ANALYZE:** Use EXPLAIN ANALYZE on a query with WHERE email = 'abc@example.com' to see index usage.  
   EXPLAIN ANALYZE

SELECT \* FROM users WHERE email = 'abc@example.com';

1. **Drop Index:** Drop the index created on LOWER(email).  
   DROP INDEX IF EXISTS idx\_users\_lower\_email;