

SQL Assignment

Intern: Sumit Dhar

Project: SQL using PostgreSQL

Date: 10th October, 2025

1. Introduction

In this session, we explored a wide range of SQL problems using **PostgreSQL**. Our goal was to deepen our understanding of both **basic** and **intermediate** SQL concepts, focusing on how to structure queries efficiently and clearly. We worked through realistic, interview-style problems involving employees, customers, transactions, user activity and more.

We aimed to not just *solve* the problems, but also to understand why each solution works, explore alternate approaches, and write queries that are both correct and optimized.

Key Objectives:

- Understand how to **create tables** with the correct schema and constraints
 - Practice using JOINS, GROUP BY, and HAVING to filter and aggregate data
 - Use CTEs (WITH clauses) to structure complex queries
 - Apply **window functions** like ROW_NUMBER() and COUNT() OVER(...) for advanced row-wise logic
 - Handle **date filtering**, including ranges like “last 90 days”
 - Simulate real-world use-cases such as leaderboards, rankings, and product tracking
 - Think in terms of **optimization** and clarity — choosing the best tools for each scenario
-

2. Pre-requisites

Before start, the following tools were installed and configured:

1. PostgreSQL (local or cloud SQL environment)
2. Basic understanding of:
 - Table creation and constraints (PRIMARY KEY, FOREIGN KEY)
 - Writing SELECT queries
 - Aggregation and filtering
3. No prior advanced SQL knowledge was required, but we progressively introduced:
 - Window functions
 - Common Table Expressions (CTEs)
 - Subqueries
 - Conditional logic with CASE and COALESCE

Questions

1. Write an SQL query to report the managers with at least five direct reports.

Return the result table in any order. The query result format is in the following example.

Input:

Employee table:

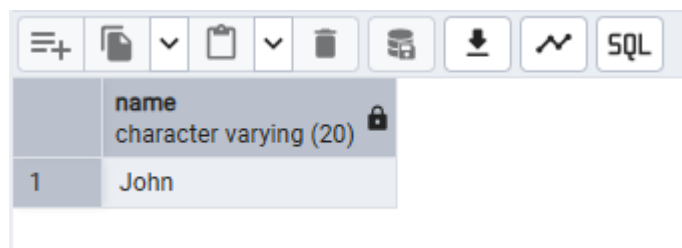
id	name	department	managerId
101	John	A	None
102	Dan	A	101
103	James	A	101
104	Amy	A	101
105	Anne	A	101
106	Ron	B	101

Program Code:

```
SELECT m.name
FROM Employee e
JOIN Employee m ON m.id = e.managerId
GROUP BY m.id, m.name
HAVING COUNT(*) >= 5;
```

Output:

name
John



The screenshot shows a database interface with a toolbar at the top containing icons for menu, file, dropdown, clipboard, trash, database, download, refresh, and SQL. Below the toolbar, a table is displayed with the following structure:

	name character varying (20) 🔒
1	John

2. Write an SQL query to report the nth highest salary from the Employee table. If there is no nth highest salary, the query should report null.

The query result format is in the following example.

Input:

Employee table:

id	salary
1	100
2	200
3	300

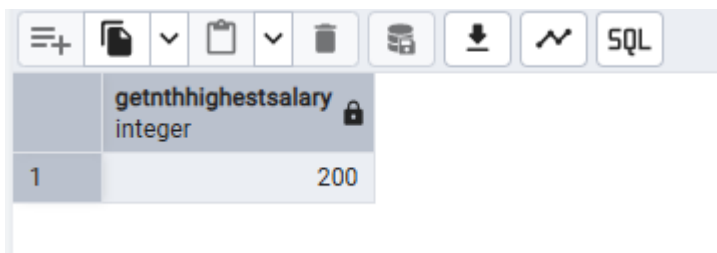
n = 2

Program Code:

```
WITH cte AS (  
    SELECT id,  
           salary,  
           ROW_NUMBER() OVER(ORDER BY salary DESC) as rn  
    FROM Employee  
)  
SELECT salary AS getNthHighestSalary  
FROM cte WHERE rn = 2
```

Output:

getNthHighestSalary(2)
200



getnthhighestsalary integer
1 200

3. Write an SQL query to find the people who have the most friends and the most friends number. The test cases are generated so that only one person has the most friends.

The query result format is in the following example.

Input:

RequestAccepted table:

requester_id	accepter_id	accept_date
1	2	2016/06/03
1	3	2016/06/08
2	3	2016/06/08
3	4	2016/06/09

Program Code:

```
WITH all_friends AS(  
    SELECT requester_id AS id FROM RequestAccepted  
    UNION ALL  
    SELECT accepter_id FROM RequestAccepted  
)  
,  
total_friends AS (  
    SELECT id, COUNT(*) AS num  
    FROM all_friends  
    GROUP BY id  
)  
SELECT * FROM total_friends  
ORDER BY num DESC  
LIMIT 1
```

Output:

id	num
3	3



	id integer	num bigint
1	3	3

4. Write an SQL query to swap the seat id of every two consecutive students. If the number of students is odd, the id of the last student is not swapped.

Column Name	Type
id	int
name	varchar

id is the primary key column for this table. Each row of this table indicates the name and the ID of a student.

id is a continuous increment. Return the result table ordered by id in ascending order.

The query result format is in the following example.

Input:

Seat table:

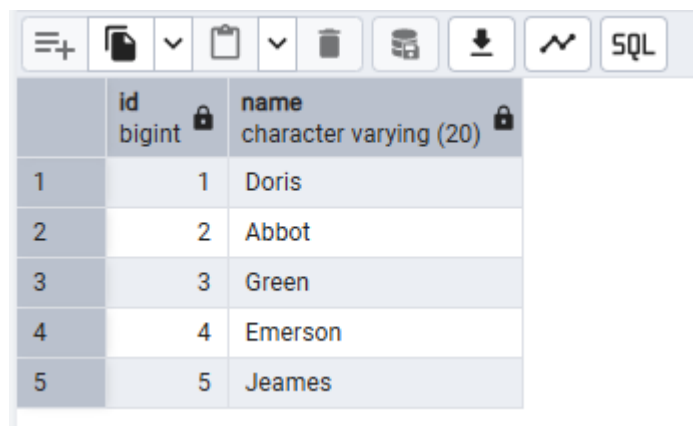
id	student
1	Abbot
2	Doris
3	Emerson
4	Green
5	Jeames

Program Code:

```
WITH grp1 AS (  
    SELECT *, FLOOR((ROW_NUMBER() OVER(ORDER BY id) - 1)/2) AS rn  
    FROM SEAT  
)  
grp2 AS (  
    SELECT *,  
    ROW_NUMBER() OVER(PARTITION BY rn ORDER BY id DESC) AS pos  
    FROM grp1  
)  
SELECT ROW_NUMBER() OVER() AS id, name  
FROM grp2  
ORDER BY rn, pos
```

Output:

id	student
1	Doris
2	Abbot
3	Green
4	Emerson
5	Jeames



	id bigint	name character varying (20)
1	1	Doris
2	2	Abbot
3	3	Green
4	4	Emerson
5	5	Jeames

5. Write an SQL query to report the customer ids from the Customer table that bought all the products in the Product table.

Table: Customer

Column Name	Type
customer_id	int
product_key	int

There is no primary key for this table. It may contain duplicates. product_key is a foreign key to Product table.

Table: Product

Column Name	Type
product_key	int

product_key is the primary key column for this table. Return the result table in any order. The query result format is in the following example.

Input:

Customer table:

customer_id	product_key
1	5
2	6
3	5
3	6
1	6

Product table:

product_key
5
6

Program Code:

```
SELECT customer_id
FROM Customer
GROUP BY customer_id
HAVING COUNT(DISTINCT product_key) = (SELECT COUNT(DISTINCT product_key) FROM Product)
```

Output:

customer_id
1
3



customer_id	integer
1	1
2	3

6. Write an SQL query to find for each user, the join date and the number of orders they made as a buyer in 2019.

Table: Users

Column Name	Type
user_id	int
join_date	date
favorite_brand	varchar

user_id is the primary key of this table.

This table has the info of the users of an online shopping website where users can sell and buy items.

Table: Orders

Column Name	Type
item_id	int
Item_brand	varchar

item_id is the primary key of this table.

Return the result table in any order. The query result format is in the following example.

Input:

Users table:

user_id	join_date	favorite_brand
1	2018-01-01	Lenovo
2	2018-02-09	Samsung
3	2018-01-19	LG
4	2018-05-21	HP

Orders table:

order_id	order_date	item_id	buyer_id	seller_id
1	2019-08-01	4	1	2
2	2018-08-02	2	1	3
3	2019-08-03	3	2	3
4	2018-08-04	1	4	2
5	2018-08-04	1	3	4
6	2019-08-05	2	2	4

Items table:

item_id	item_brand
1	Samsung
2	Lenovo
3	LG
4	HP

Program Code:

```
WITH Buyer AS (  
    SELECT buyer_id, COUNT(*) AS orders_in_2019  
    FROM Users u  
    JOIN ORDERS o ON u.user_id = o.buyer_id  
    WHERE EXTRACT(YEAR FROM order_date) = 2019  
    GROUP BY buyer_id  
)  
SELECT user_id AS buyer_id, join_date, COALESCE(orders_in_2019, 0) AS orders_in_2019  
FROM Users u LEFT JOIN Buyer b ON u.user_id = b.buyer_id
```

Output:

buyer_id	join_date	orders_in_2019
1	2018-01-01	1
2	2018-02-09	2
3	2018-01-19	0
4	2018-05-21	0

	user_id [PK] integer	join_date date	coalesce bigint
1	1	2018-01-01	1
2	2	2018-02-09	2
3	3	2018-01-19	0
4	4	2018-05-21	0

7. Write an SQL query to reports for every date within at most 90 days from today, the number of users that logged in for the first time on that date. Assume today is 2019-06-30.

Table: Traffic

Column Name	Type
user_id	int
activity	enum
activity_date	date

There is no primary key for this table, it may have duplicate rows. The activity column is an ENUM type of ('login', 'logout', 'jobs', 'groups', 'homepage').

Return the result table in any order. The query result format is in the following example.

Input:

Traffic table:

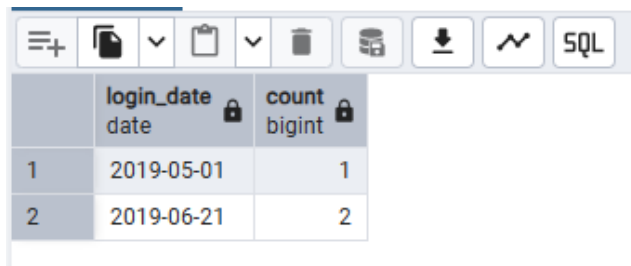
user_id	activity	activity_date
1	login	2019-05-01
1	homepage	2019-05-01
1	logout	2019-05-01
2	login	2019-06-21
2	logout	2019-06-21
3	login	2019-01-01
3	jobs	2019-01-01
3	logout	2019-01-01
4	login	2019-06-21
4	groups	2019-06-21
4	logout	2019-06-21
5	login	2019-03-01
5	logout	2019-03-01
5	login	2019-06-21
5	logout	2019-06-21

Program Code:

```
WITH users_ids AS (  
    SELECT user_id, MIN(activity_date) AS login_date  
    FROM Traffic  
    WHERE activity = 'login'  
    GROUP BY user_id  
)  
SELECT login_date, COUNT(*) from users_ids  
WHERE login_date BETWEEN DATE '2019-06-30' - INTERVAL '90 DAY' AND DATE('2019-06-30')  
GROUP BY login_date
```

Output:

login_date	user_count
2019-05-01	1
2019-06-21	2



	login_date date	count bigint
1	2019-05-01	1
2	2019-06-21	2

8. Write an SQL query to find the prices of all products on 2019-08-16. Assume the price of all products before any change is 10.

Column Name	Type
product_id	int
new_price	int
change_date	int

(product_id, change_date) is the primary key of this table. Each row of this table indicates that the price of some product was changed to a new price at some date.

Return the result table in any order. The query result format is in the following example.

Input:

Products table:

product_id	new_price	change_date
1	20	2019-08-14
2	50	2019-08-14
1	30	2019-08-15
1	35	2019-08-16
2	65	2019-08-17
3	20	2019-08-18

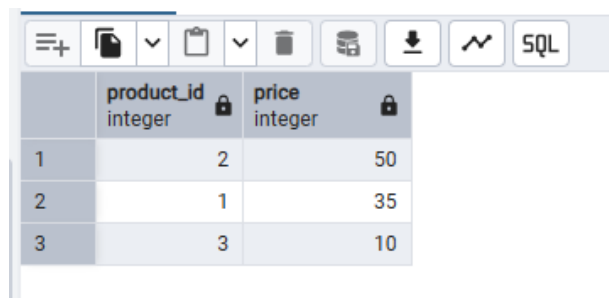
Output:

product_id	price
2	50
1	35
3	10

Program Code:

```
WITH change_price AS (  
    SELECT product_id, new_price,  
           ROW_NUMBER() OVER(PARTITION BY product_id ORDER BY change_date DESC) as row_num  
    FROM Products  
    WHERE change_date <= '2019-08-16'  
)  
all_products AS (  
    SELECT DISTINCT product_id FROM Products  
)  
SELECT a.product_id, COALESCE(cp.new_price,10) AS price  
FROM all_products a LEFT JOIN change_price cp  
ON a.product_id = cp.product_id AND cp.row_num = 1  
ORDER BY price DESC
```

Output:



	product_id integer	price integer
1	2	50
2	1	35
3	3	10

9. Write an SQL query to find for each month and country: the number of approved transactions and their total amount, the number of chargebacks, and their total amount.

Table: Transactions

Column Name	Type
id	int
country	varchar
state	enum
amount	int
trans_date	date

id is the primary key of this table. The table has information about incoming transactions. The state column is an enum of type ["approved", "declined"].

Table: Chargebacks

Column Name	Type
trans_id	int
trans_date	date

Chargebacks contains basic information regarding incoming chargebacks from some transactions placed in Transactions table. trans_id is a foreign key to the id column of Transactions table. Each chargeback corresponds to a transaction made previously even if they were not approved.

Note: In your query, given the month and country, ignore rows with all zeros.

Return the result table in any order. The query result format is in the following example.

Input:

Transactions table:

id	country	state	amount	trans_date
101	US	approved	1000	2019-05-18
102	US	declined	2000	2019-05-19
103	US	approved	3000	2019-06-10
104	US	declined	4000	2019-06-13
105	US	approved	5000	2019-06-15

Chargebacks table:

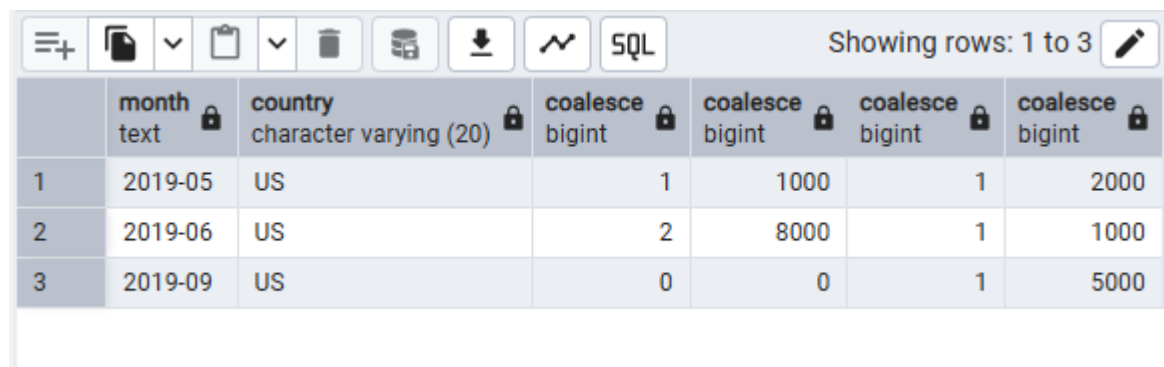
id	trans_date
102	2019-05-29
101	2019-06-30
105	2019-09-18

Program Code:

```
WITH g1 AS(
    SELECT TO_CHAR(trans_date, 'YYYY-MM') AS month,
           country,
           COUNT(*) as approved_count,
           SUM(amount) as approved_amount
    FROM Transactions
    WHERE state = 'approved'
    GROUP BY month, country
),
g2 AS(
    SELECT TO_CHAR(c.trans_date, 'YYYY-MM') AS month, country,
           COUNT(*) as chargeback_count,
           SUM(amount) as chargeback_amount
    FROM Transactions t JOIN Chargebacks c ON t.id = c.trans_id
    GROUP BY month, country
)
SELECT
    COALESCE(g1.month, g2.month) AS month,
    COALESCE(g1.country, g2.country) AS country,
    COALESCE(approved_count, 0),
    COALESCE(approved_amount, 0),
    COALESCE(chargeback_count, 0),
    COALESCE(chargeback_amount, 0)
FROM g2 LEFT JOIN g1 ON g2.month = g1.month
ORDER BY month
```

Output:

month	country	approved_count	approved_amount	chargeback_count	chargeback_amount
2019-05	US	1	1000	1	2000
2019-06	US	2	8000	1	1000
2019-09	US	0	0	1	5000



	month text	country character varying (20)	coalesce bigint	coalesce bigint	coalesce bigint	coalesce bigint
1	2019-05	US	1	1000	1	2000
2	2019-06	US	2	8000	1	1000
3	2019-09	US	0	0	1	5000

10. Write an SQL query that selects the team_id, team_name and num_points of each team in the tournament after all described matches.

Table: Teams

Column Name	Type
team_id	int
team_name	varchar

team_id is the primary key of this table. Each row of this table represents a single football team.

Table: Matches

Column Name	Type
match_id	int
host_team	int
guest_team	int
host_goals	int
guest_goals	int

match_id is the primary key of this table. Each row is a record of a finished match between two different teams. Teams host_team and guest_team are represented by their IDs in the Teams table (team_id), and they scored host_goals and guest_goals goals, respectively.

You would like to compute the scores of all teams after all matches. Points are awarded as follows: A team receives three points if they win a match (i.e., Scored more goals than the opponent team). A team receives one point if they draw a match (i.e., Scored the same number of goals as the opponent team). A team receives no points if they lose a match (i.e., Scored fewer goals than the opponent team).

Return the result table ordered by num_points in decreasing order. In case of a tie, order the records by team_id in increasing order.

The query result format is in the following example.

Input:

Teams table:

team_id	team_name
10	Leetcode FC
20	NewYork FC
30	Atlanta FC
40	Chicago FC
50	Toronto FC

Matches table:

match_id	host_team	guest_team	host_goals	guest_goals
1	10	20	3	0
2	30	10	2	2
3	10	50	5	1
4	20	30	1	0
5	50	30	1	0

Program Code:

```
WITH all_teams AS(
    SELECT host_team AS team_id,
           CASE
             WHEN host_goals > guest_goals THEN 3
             WHEN host_goals = guest_goals THEN 1
             ELSE 0
           END AS host_points
    FROM Matches
    UNION ALL
    SELECT guest_team AS team_id,
           CASE
             WHEN guest_goals > host_goals THEN 3
             WHEN host_goals = guest_goals THEN 1
             ELSE 0
           END AS guset_points
    FROM Matches
)
SELECT t.team_id, t.team_name, COALESCE(SUM(host_points), 0) AS num_points
FROM Teams t LEFT JOIN all_teams a
ON t.team_id = a.team_id
GROUP BY t.team_id, team_name
ORDER BY num_points DESC, t.team_id
```

Output:

team_id	team_name	num_points
10	Leetcode FC	7
20	NewYork FC	3
50	Toronto FC	3
30	Atlanta FC	1
40	Chicago FC	0

	team_id [PK] integer	team_name character varying (20)	num_points bigint
1	10	Leetcode FC	7
2	20	NewYork FC	3
3	50	Toronto FC	3
4	30	Atlanta FC	1
5	40	Chicago FC	0

Summary

We practiced real-world SQL problems using **PostgreSQL**, covering:

- Table creation with keys and constraints
- **JOIN, GROUP BY, HAVING, CASE, and COALESCE**
- **CTEs, subqueries and window functions** for advanced queries
- Date filtering, ranking, and conditional logic

Topics included employee hierarchies, salaries, login activity, product prices, transactions, and sports scores.