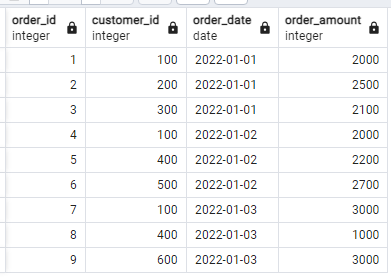
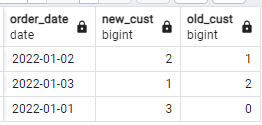
**Q1: -** Find out first\_visit count and repeat\_visit count of customers.

**Input: -**



**Output: -**



with first\_visit as (

select customer\_id, min(order\_date) as first\_date

from customer\_orders group by customer\_id)

select c.order\_date,

sum (case when order\_date=first\_date then 1 else 0 end) as new\_cust,

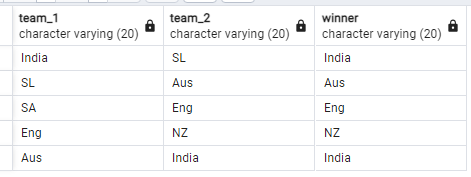
sum (case when order\_date!=first\_date then 1 else 0 end) as old\_cust

from customer\_orders c join first\_visit f on c.customer\_id=f.customer\_id

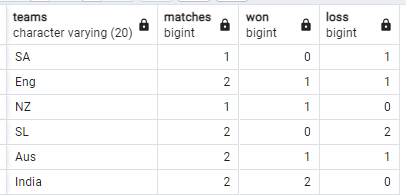
group by order\_date;

**Q2: -** Find number of wins and losses

**Input: -**



**Output:-**



select teams, count (1) as matches, sum(win) as won, count (1) - sum(win) as loss

from

(select team\_1 as teams, case when team\_1 = winner then 1 else 0 end as win from icc\_world\_cup

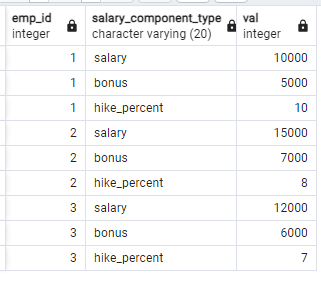
union all

select team\_2 as teams, case when team\_2 = winner then 1 else 0 end as win from icc\_world\_cup) a

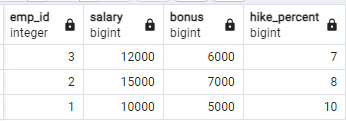
group by teams;

**Q3: -** Pivot without using function

**Input: -**

****

**Output:-**

****

select emp\_id,

sum (Case when salary\_component\_type = 'salary' then val end) as Salary,

sum (Case when salary\_component\_type = 'bonus' then val end) as bonus,

sum (Case when salary\_component\_type = 'hike\_percent' then val end) as hike\_percent

from emp\_compensation

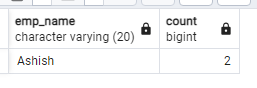
group by emp\_id;

**Q4: -** Find duplicate row in a table

**Input: -**

****

**Output:-**

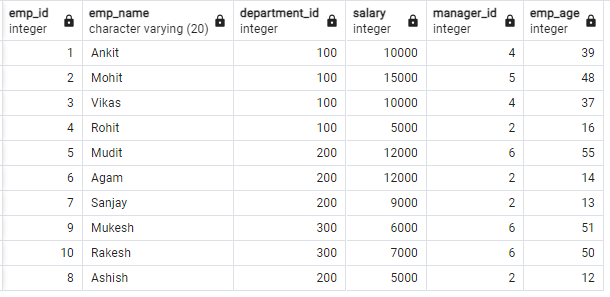


select emp\_name, count (1) from emp1 group by emp\_name having count (1)>1;

**Q5: -** Delete Duplicate records

**Input: -** Same as above

**Output:-**

****

WITH cte AS (

SELECT emp\_id, ROW\_NUMBER () OVER (PARTITION BY emp\_name, department\_id ORDER BY emp\_id) AS row\_num FROM emp1

)

DELETE FROM emp1 WHERE emp\_id IN (

SELECT emp\_id FROM cte WHERE row\_num > 1

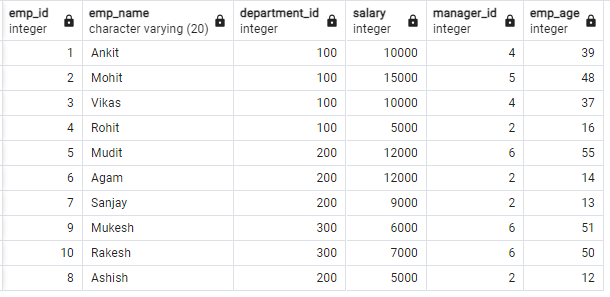
);

**Q6: -** Find the employees whose department does not match with departments table department.

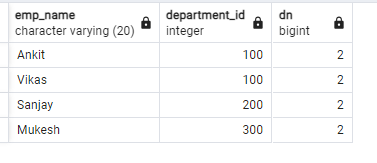
select \* from employees e left join departments d on e.department = d.department where d.department is NULL;

**Q7: -** Find second highest salary from each department

**Input: -**

****

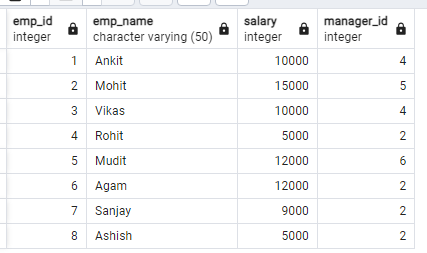
**Output: -**



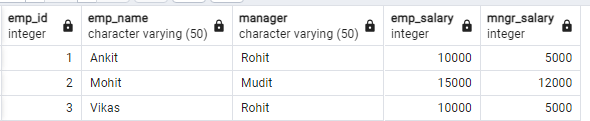
select \* from (select emp\_name,department\_id, dense\_rank() over(partition by department\_id order by salary desc) as dn from emp1) e where dn =2;

**Q8:-**Find the emp whose salary is greater than it’s manager (self-join)

**Input: -**

****

**Output:-**

****

select e.emp\_id, e.emp\_name, m.emp\_name as manager, e.salary as emp\_salary, m.salary as mngr\_salary

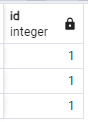
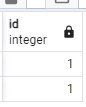
from emp\_manager e inner join emp\_manager m on m.emp\_id = e.manager\_id

where e.salary >m.salary;

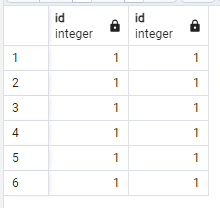
**Q9: -** Tricky question on Joins. Find inner join of below tables

**Input: -**

Tbl\_1: Tbl2:



**Output:-**

****

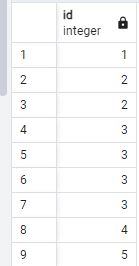
select \* from tbl\_1 t1 inner join tbl\_2 t2 on t1.id = t2.id;

Same output for inner, left, right and full outer joins

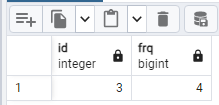
NULL! = NULL in SQL

**Q10: - Find** mode of given column**.**

**Input: -**

****

**Output:-**

****

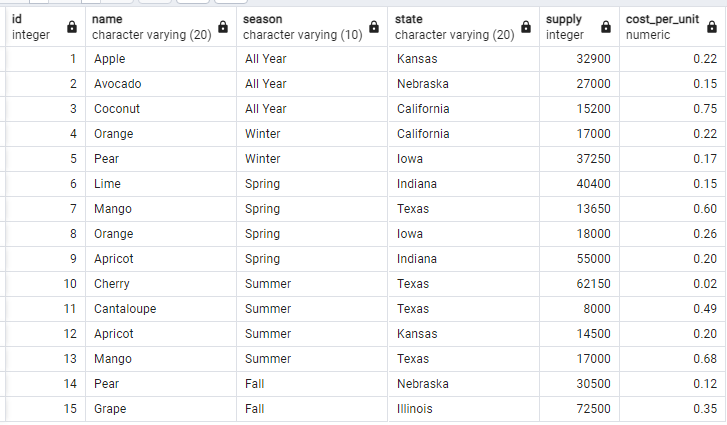
with cte as

(select id, count (1) frq from mode group by id)

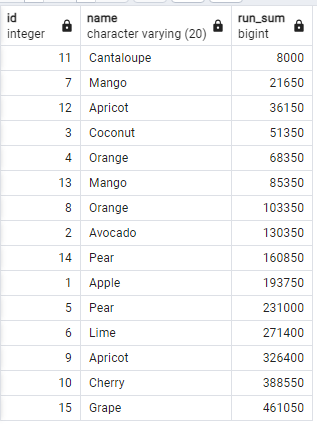
select \* from cte where frq = (select max(frq) from cte);

**Q11: -** Running Sum

**Input: -**

****

**Output: -**

****

select id, name, sum(supply)

over (order by supply rows between unbounded preceding and current row) as run\_sum

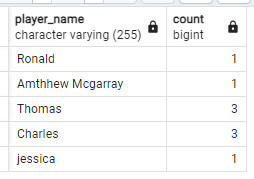
from fruit\_imports;

**Q12: -** Find Player who won gold but not silver and bronze.

**Input: -**

****

**Output: -**

****

with cte as (

select gold as player\_name from events where gold not in (

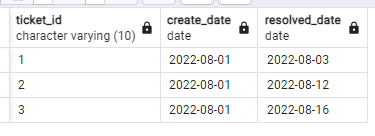
select silver from events union all

select bronze from events))

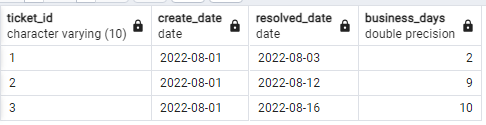
select player\_name, count(1) from cte group by player\_name;

**Q13: -** Find business days between created date and resolved date of tickets by excluding weekends and public holidays.

**Input: -**



**Output: -**

****

SELECT ticket\_id, create\_date, resolved\_date

, (resolved\_date - create\_date) - 2\*round ((resolved\_date - create\_date)/7) - count(holiday\_date) as business\_days

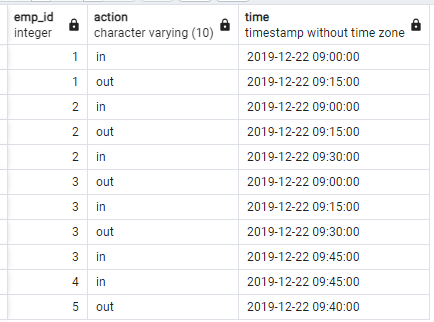
FROM tickets left join holidays on holiday\_date between create\_date and resolved\_date

and trim (TO\_CHAR (holiday\_date, 'Day'))! = 'Sunday' and trim (TO\_CHAR (holiday\_date, 'Day'))! = 'Saturday'

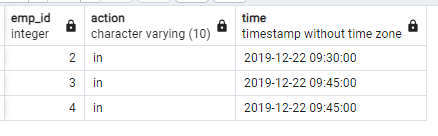
group by ticket\_id, create\_date, resolved\_date order by ticket\_id;

**Q14: -** Find emp who is present inside the hospital.

**Input: -**

****

**Output: -**

****

WITH CTE AS (

SELECT \*,

RANK() OVER (PARTITION BY emp\_id ORDER BY time DESC) rnk

FROM hospital

)

SELECT emp\_id,action,time

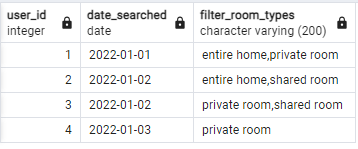
FROM CTE

WHERE rnk=1

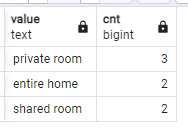
AND action='in';

**Q15: -** find most searched room type and count.

**Input: -**

****

**Output: -**

****

with room as

(select sum (case when filter\_room\_types like '%entire%' then 1 else 0 end) as en,

sum (case when filter\_room\_types like '%private%' then 1 else 0 end) as pr,

sum (case when filter\_room\_types like '%shared%' then 1 else 0 end) as sh from airbnb\_searches)

select 'entire home' as value,en cnt from room

union all select 'private room' as value,pr cnt from room

union all select 'shared room' as value,sh cnt from room order by cnt desc;

**or with String\_Split(SSMS): -**

select value as room\_type, count (1) as no\_of\_seraches from airbnb\_searches

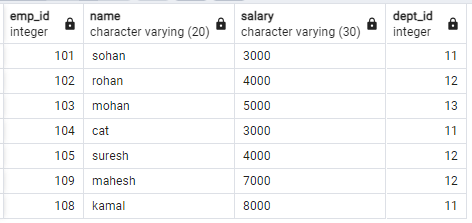
cross apply string\_split(filter\_rooms\_type,’,’)

group by value

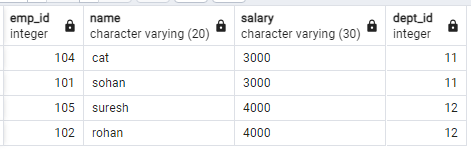
order by no\_of\_seraches desc;

**Q16: -** Find all emp whose salary is same in same dept.

**Input: -**

****

**Output: -**

****

select a.emp\_id,a.name,a.salary,a.dept\_id from emp\_salary a

inner join emp\_salary b

on a.dept\_id=b.dept\_id and a.emp\_id<>b.emp\_id

where a.salary=b.salary

order by a.dept\_id;

**OR**

WITH cte AS (

SELECT \*,

COUNT(\*) OVER (PARTITION BY dept\_id, salary) num\_same\_sal

FROM emp\_salary

)

SELECT \*

FROM cte

WHERE num\_same\_sal > 1

**Q17: -** Find unmatching records like below  
  


SELECT COALESCE([a.id](javascript:void(0);), b.id) AS id,

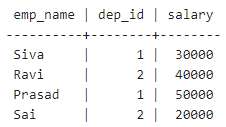
CASE WHEN [a.name](javascript:void(0);) IS NULL THEN 'New in target'

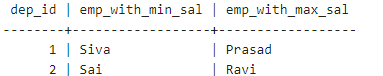
WHEN [b.name](javascript:void(0);) IS NULL THEN 'New in Source' ELSE 'Mismatch' END AS comment

FROM source AS a FULL OUTER JOIN target AS b ON [a.id](javascript:void(0);) = [b.id](javascript:void(0);)

WHERE [a.Name](javascript:void(0);) <> [b.Name](javascript:void(0);) OR [a.id](javascript:void(0);) IS NULL or [b.id](javascript:void(0);) IS NULL;

**Q18: -** Find min and max salary from each department.

**Input: **

**Output: **

with cte as (

select \*,

ROW\_NUMBER()over(partition by dep\_id order by salary asc) as min\_Sal,

ROW\_NUMBER()over(partition by dep\_id order by salary desc) as max\_Sal

from employee)

select dep\_id,

max(case when min\_Sal =1 then emp\_name else null end )as emp\_with\_min\_sal

,max(case when max\_Sal =1 then emp\_name else null end) as emp\_with\_max\_sal

from cte

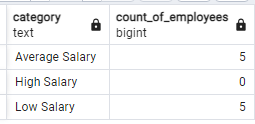
group by dep\_id

**Q19:** Categories the low, avg and high salary including 0 count.

**Input:**



**Output:**



WITH SalaryCategories AS (

SELECT

CASE

WHEN Salary < 10000 THEN 'Low Salary'

WHEN Salary >= 10000 AND Salary < 30000 THEN 'Average Salary'

WHEN Salary >= 30000 THEN 'High Salary'

END AS Category

FROM emp1

),

CategoryList AS (

SELECT 'Low Salary' AS Category

UNION ALL

SELECT 'Average Salary' AS Category

UNION ALL

SELECT 'High Salary' AS Category

)

SELECT

cl.Category,

COUNT(sc.Category) AS Count\_of\_employees

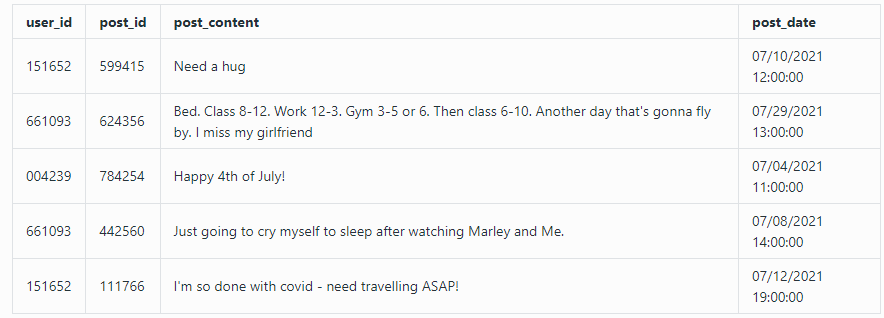
FROM CategoryList cl

LEFT JOIN SalaryCategories sc ON cl.Category = sc.Category

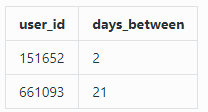
GROUP BY cl.Category;

**Q20: -** find the number of days between each user’s first post of the year and last post of the year in the year 2021

**Input:**

****

**Output:**

****

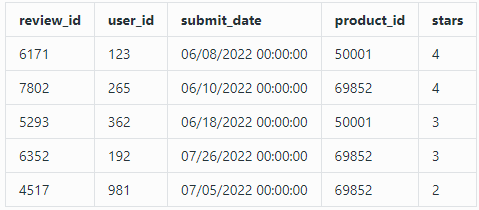
select user\_id, max(post\_date::DATE) - min(post\_date::DATE)

from posts where date\_part('year', post\_date)='2021'

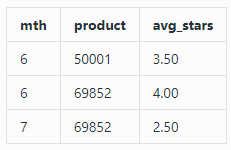
group by user\_id having count(1)>=2;

**Q21: -** Write a query to retrieve the average star rating for each product, grouped by month. The output should display the month as a numerical value, product ID, and average star rating rounded to two decimal places. Sort the output first by month and then by product ID.

**Input:**

****

**Output:**

****

SELECT

EXTRACT(MONTH FROM submit\_date) AS mth,

product\_id,

ROUND(AVG(stars), 2) AS avg\_stars

FROM reviews

GROUP BY

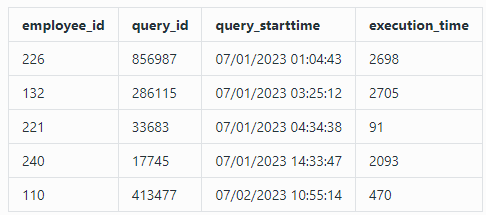
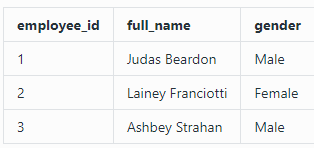
EXTRACT(MONTH FROM submit\_date),

product\_id

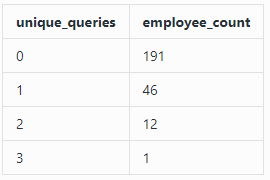
ORDER BY mth, product\_id;

Q22:- Find unique queries histogram including 0.

**Input:**

**Output: -**

****

WITH query\_count AS (

SELECT

e.employee\_id,

COALESCE(COUNT(DISTINCT q.query\_id),0) as unique\_queries

FROM employees e

LEFT JOIN queries q

ON e.employee\_id = q.employee\_id

AND EXTRACT(MONTH FROM q.query\_starttime) >= 7

AND EXTRACT(MONTH FROM q.query\_starttime) < 10

GROUP BY e.employee\_id

)

SELECT

unique\_queries,

COUNT(employee\_id) as employee\_count

FROM query\_count

GROUP BY unique\_queries

ORDER BY unique\_queries

**Q23: -**

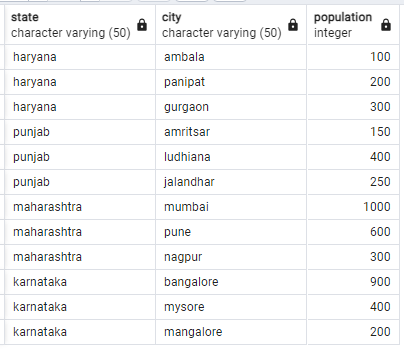
select first\_name,length(first\_name) as count\_char,

LOWER(SUBSTRING(first\_name FROM 1 FOR 4)) || TO\_CHAR(hire\_date, 'DDMM') AS password

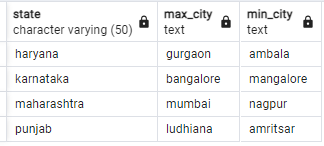
from employees;

**Q24: -** Find max, min populated cities in every state.

**Input:**

****

**Output:**

****

with cte as (

select state

,case when population = max(population) over(partition by state) then city else null end as max\_city

,case when population = min(population) over(partition by state) then city else null end as min\_city

from city\_population

)

select state

,max(max\_city) as max\_city

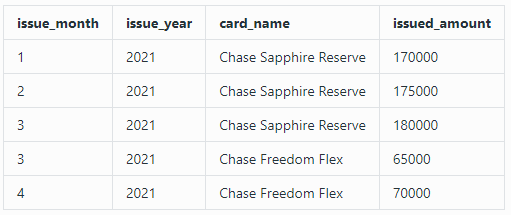
,min(min\_city) as min\_city

from cte

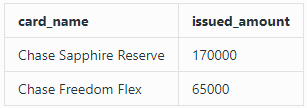
group by state;

**Q25:** - Find name of the credit card, and how many cards were issued in its launch month. Order the results starting from the biggest issued amount.

**Input:**

****

**Output:**

****

with cte as

(SELECT card\_name,issued\_amount, issue\_month || '-' || issue\_year as issue\_date,

min(issue\_month || '-' || issue\_year) over(partition by card\_name) as launch\_date

from monthly\_cards\_issued)

select card\_name, issued\_amount

from cte where issue\_date = launch\_date

order by issued\_amount desc;

**Q26: - Explain Index**

In SQL, an index is a database object that improves the speed of data retrieval operations on a table by creating a data structure that allows for faster searching, sorting, and filtering. It works like an index in a book, enabling quick access to the desired data without scanning the entire table. However, indexes can slow down data modification operations like INSERT, UPDATE, and DELETE because the index also needs to be updated.

CREATE INDEX idx\_customer\_name ON customers (last\_name, first\_name);

DROP INDEX idx\_customer\_name ON customers;

**Type:**

 Unique Index

 Composite Index

 Full-Text Index

 Clustered Index

 Non-Clustered Index

 Bitmap Index

 Filtered Index

 Hash Index

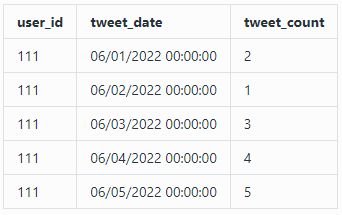
 Spatial Index

A **clustered index** organizes the actual data rows in the table to match the order of the indexed column(s). Think of it like a library where books (rows) are physically arranged on the shelves in alphabetical order by title (indexed column). Because of this, there can only be one clustered index per table, as you can’t physically arrange the books in more than one way.

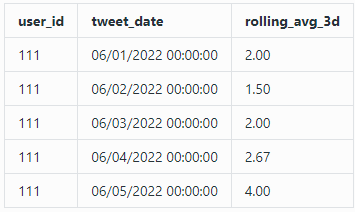
In contrast, a **non-clustered index** is like having an index card in a catalog that lists the book title (indexed column) and points you to the specific shelf and spot where the book is stored. The books (data rows) remain in their original order on the shelves, separate from the order defined by the non-clustered index.

**Q27: -** Calculate the 3-day rolling average of tweets for each user. Output the user ID, tweet date, and rolling averages rounded to 2 decimal places.

**Input: -**

****

**Output: -**

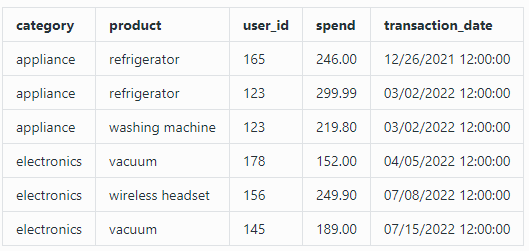
****

SELECT user\_id, tweet\_date, round(AVG(tweet\_count)

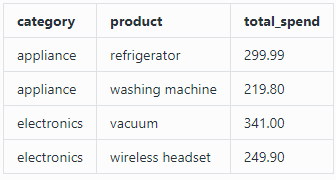
over(partition by user\_id order by tweet\_date rows BETWEEN 2 PRECEDING and current row),2) as rolling\_avg\_3d from tweets;

**Q28: -** Write a query to identify the top two highest-grossing products within each category in the year 2022.

**Input: -**

****

**Output: -**

****

WITH ranked\_spending\_cte AS (

SELECT category, product, SUM(spend) AS total\_spend, RANK() OVER ( PARTITION BY category ORDER BY SUM(spend) DESC) AS ranking

FROM product\_spend

WHERE EXTRACT(YEAR FROM transaction\_date) = 2022

GROUP BY category, product)

SELECT category, product, total\_spend

FROM ranked\_spending\_cte

WHERE ranking <= 2

ORDER BY category, ranking;