**SQL Window Function**

# Window Function

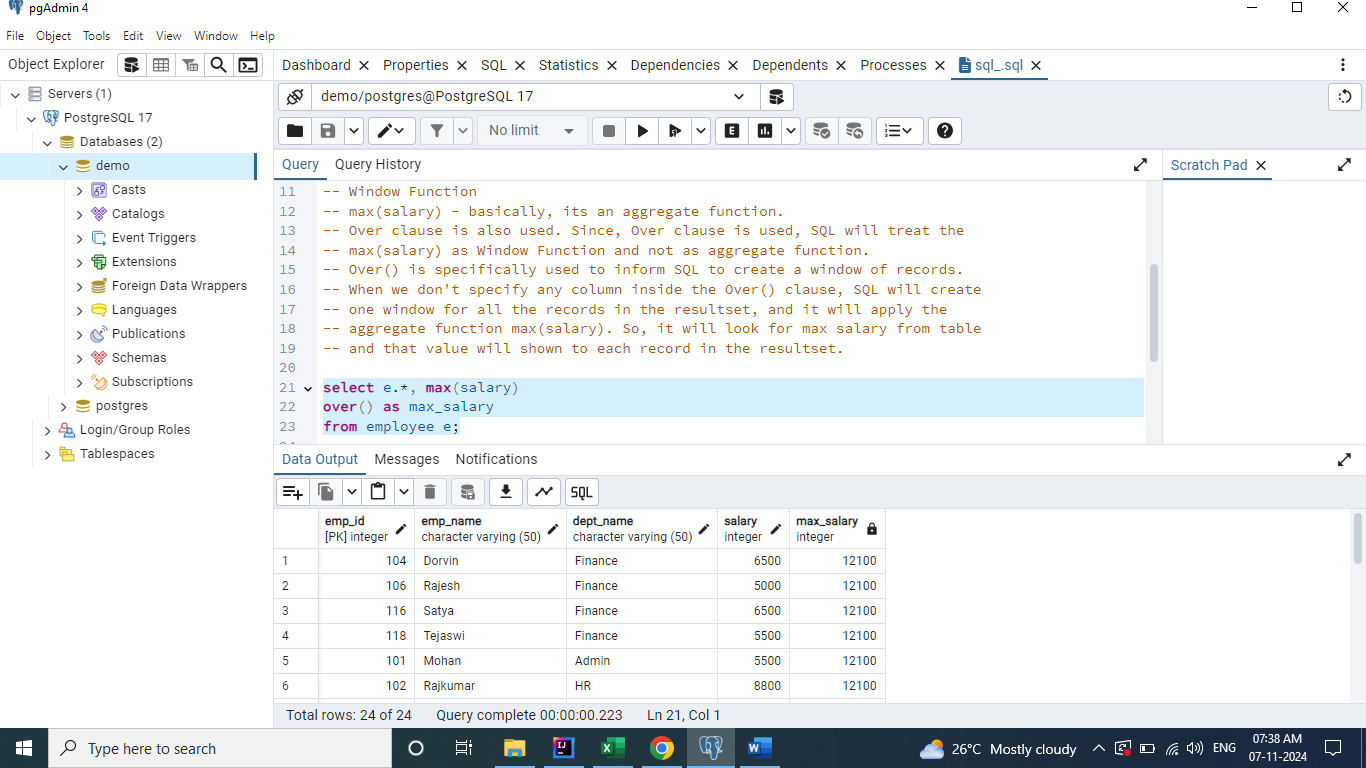
max(salary) - basically, it’s an aggregate function. Over clause is also used. Since, over clause is used, SQL will treat the max(salary) as Window Function and not as aggregate function.

Over() is specifically used to inform SQL to create a window of records. When we don't specify any column inside the Over() clause, SQL will create one window for all the records in the resultset, and it will apply the aggregate function max(salary). So, it will look for max salary from table and that value will show to each record in the resultset.

**select e.\*, max(salary)**

**over() as max\_salary**

**from employee e;**

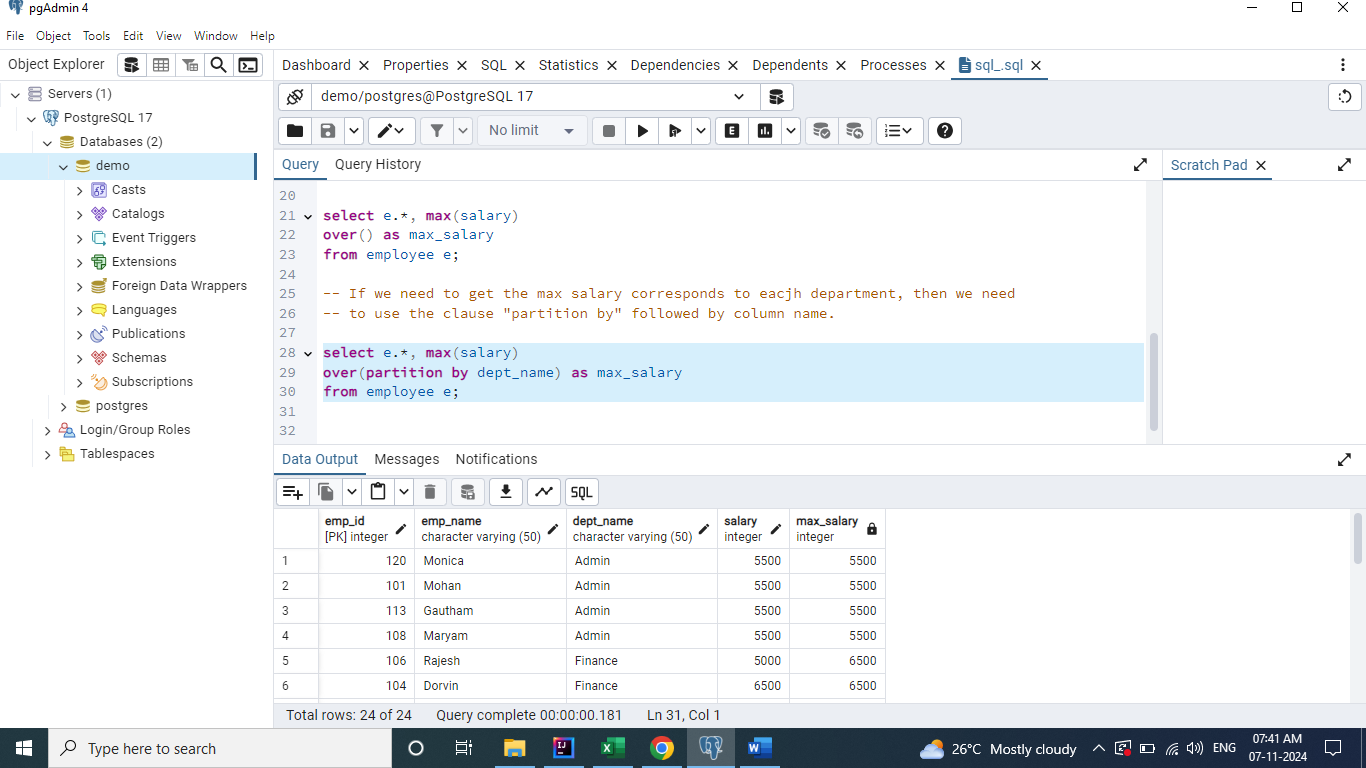


If we need to get the max salary corresponds to each department, then we need to use the clause "partition by" followed by column name.

**select e.\*, max(salary)**

**over(partition by dept\_name) as max\_salary**

**from employee e;**



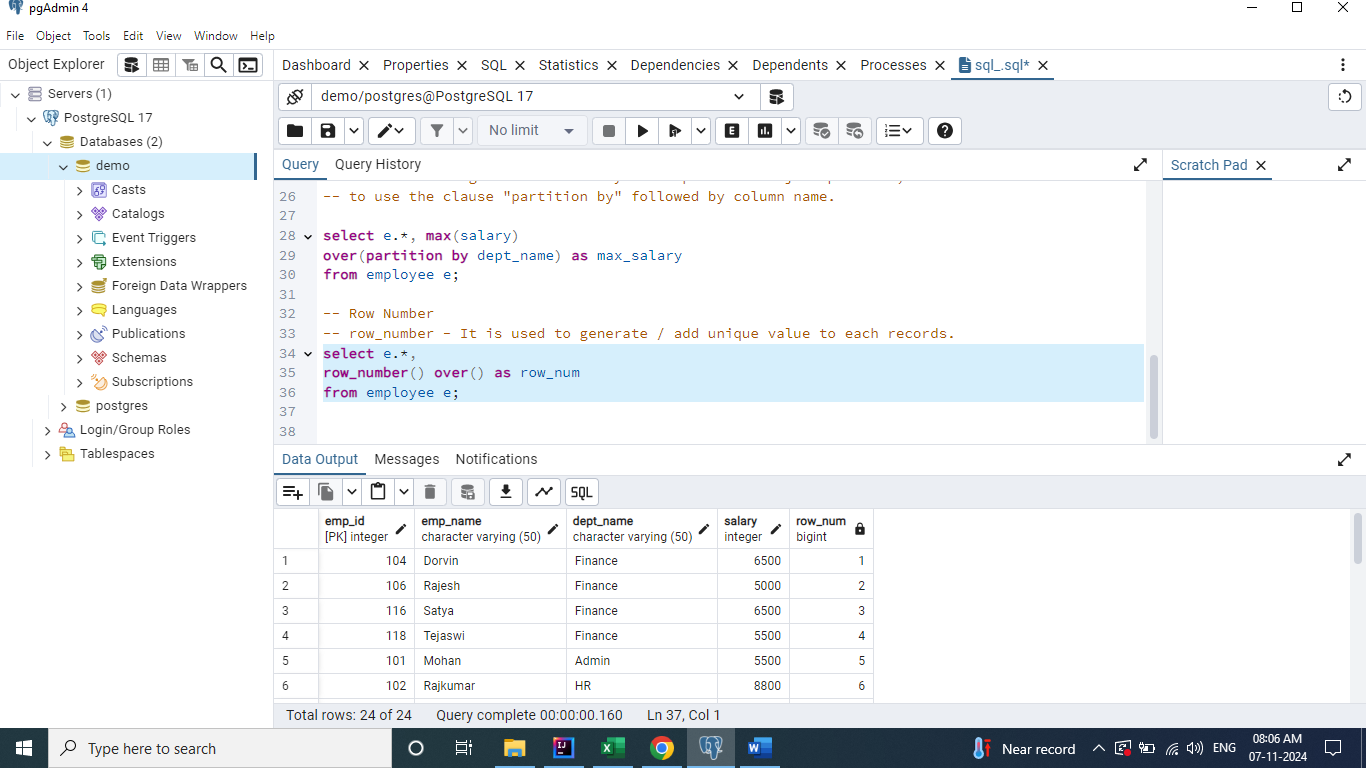
# Row Number

**It is used to generate / add unique value to each record.**

**select e.\*,**

**row\_number() over() as row\_num**

**from employee e;**

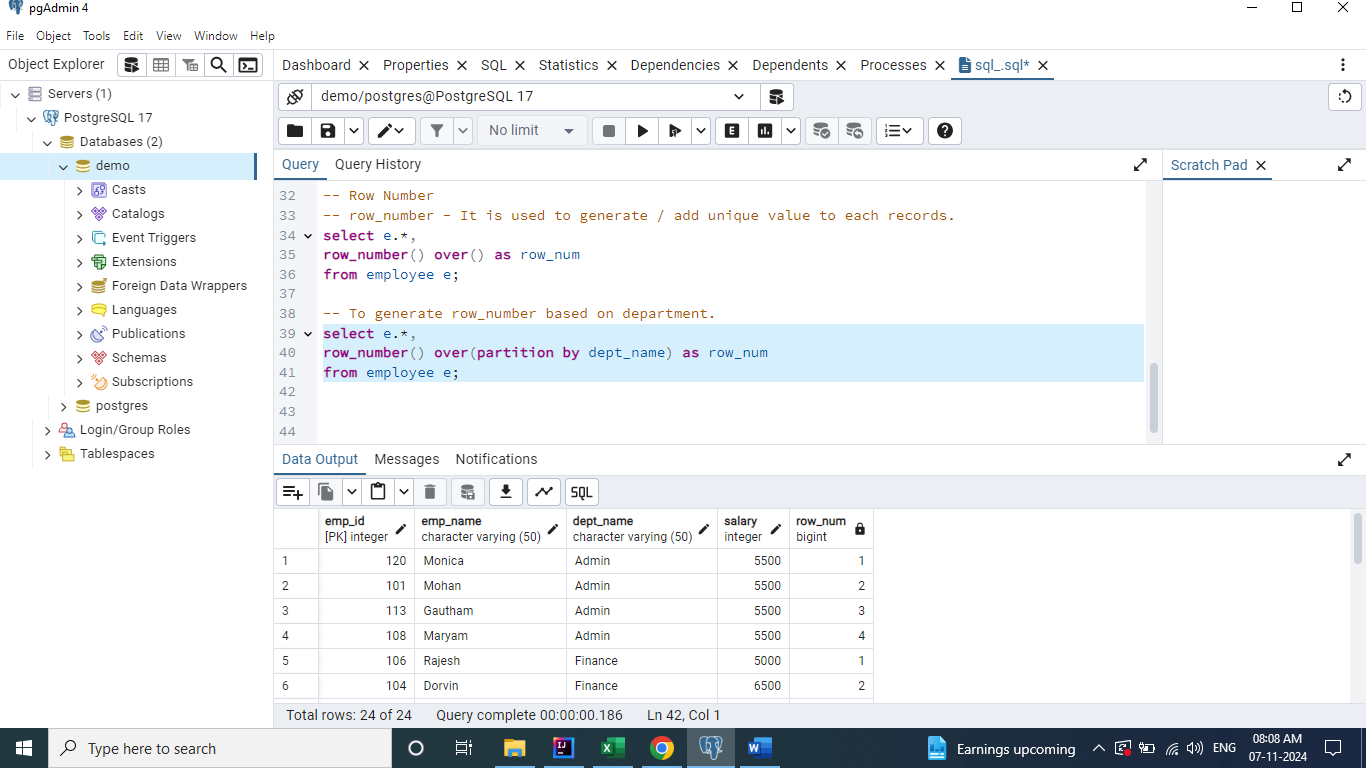


**To generate row\_number based on department.**

**select e.\*,**

**row\_number() over(partition by dept\_name) as row\_num**

**from employee e;**



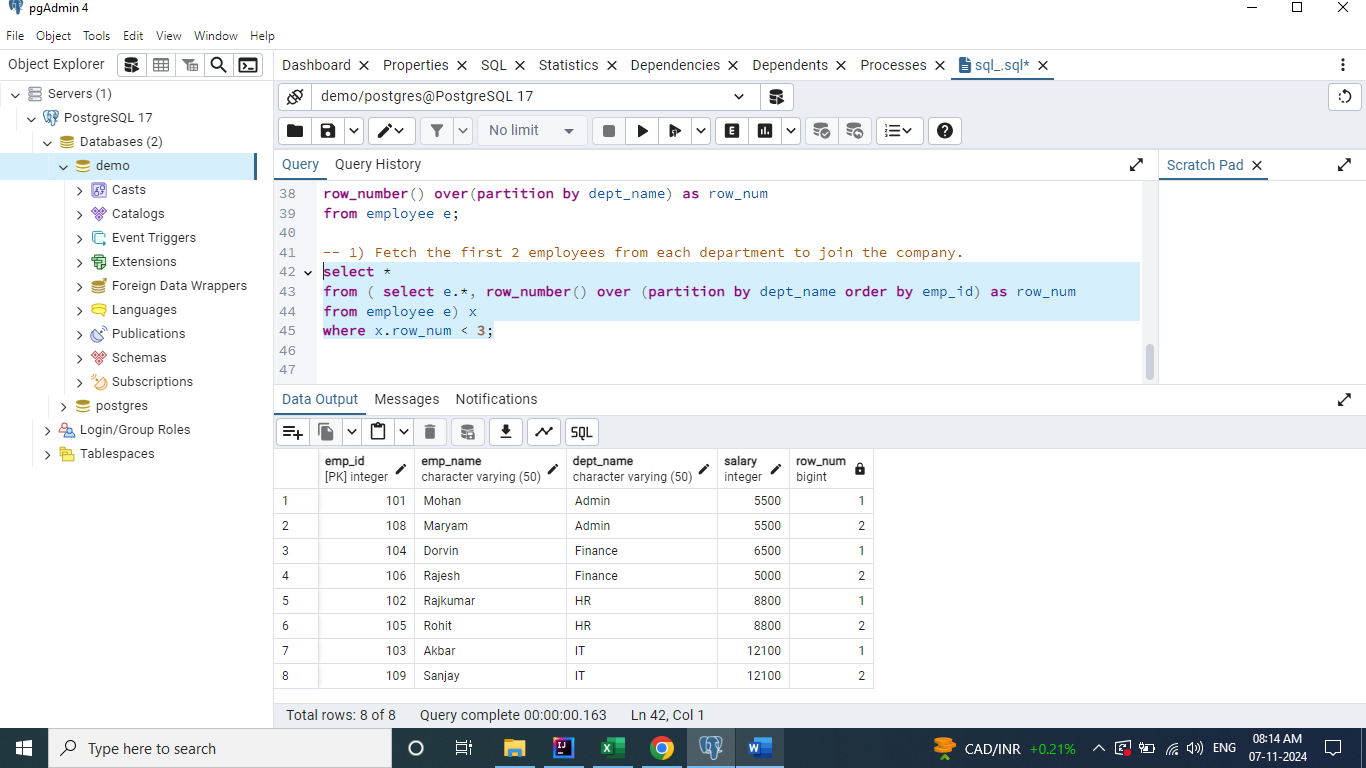
## 1) Fetch the first 2 employees from each department to join the company.

**select \***

**from ( select e.\*, row\_number() over (partition by dept\_name order by emp\_id) as row\_num**

**from employee e) x**

**where x.row\_num < 3;**



# Rank

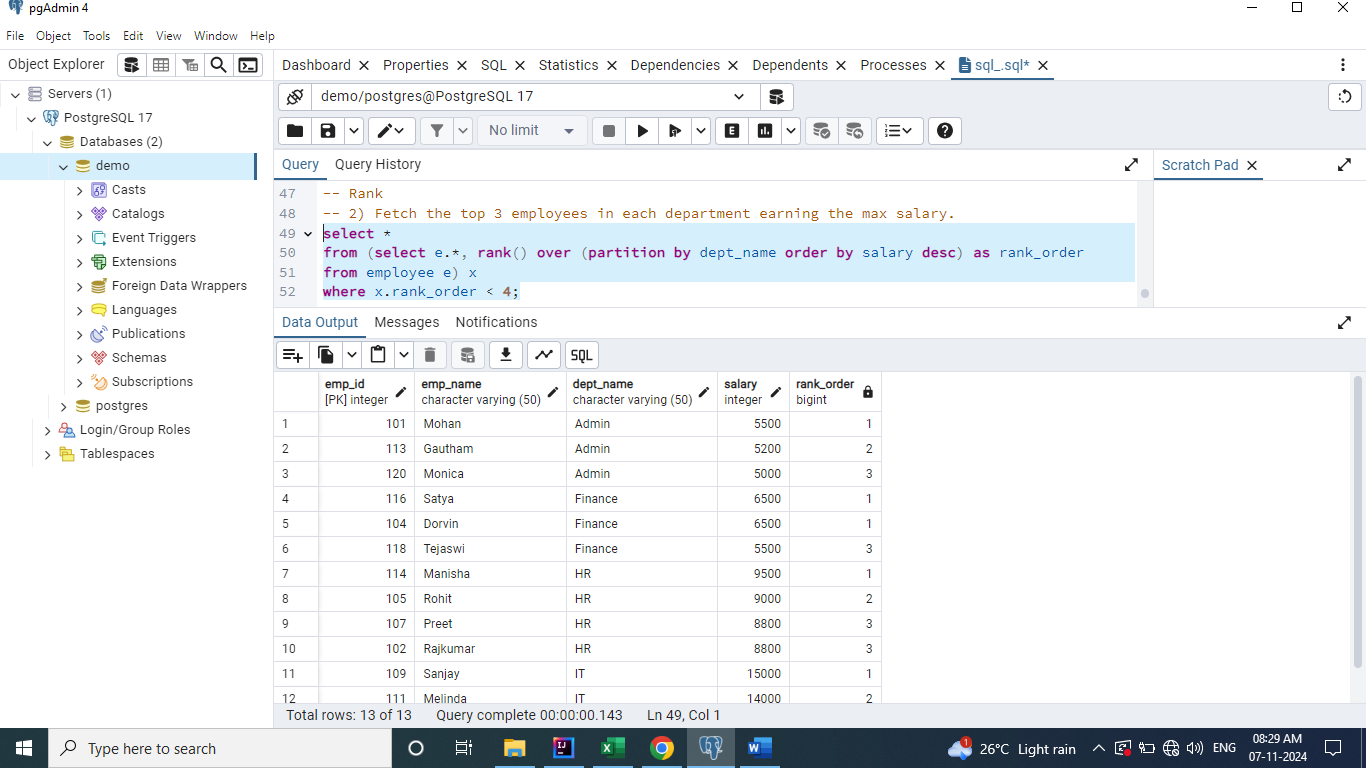
## 2) Fetch the top 3 employees in each department earning the max salary.

**select \***

**from (select e.\*, rank() over (partition by dept\_name order by salary desc) as rank\_order**

**from employee e) x**

**where x.rank\_order < 4;**



# Dense Rank

**Rank will skip a value for every duplicate record whereas Dense Rank will not skip a value.**

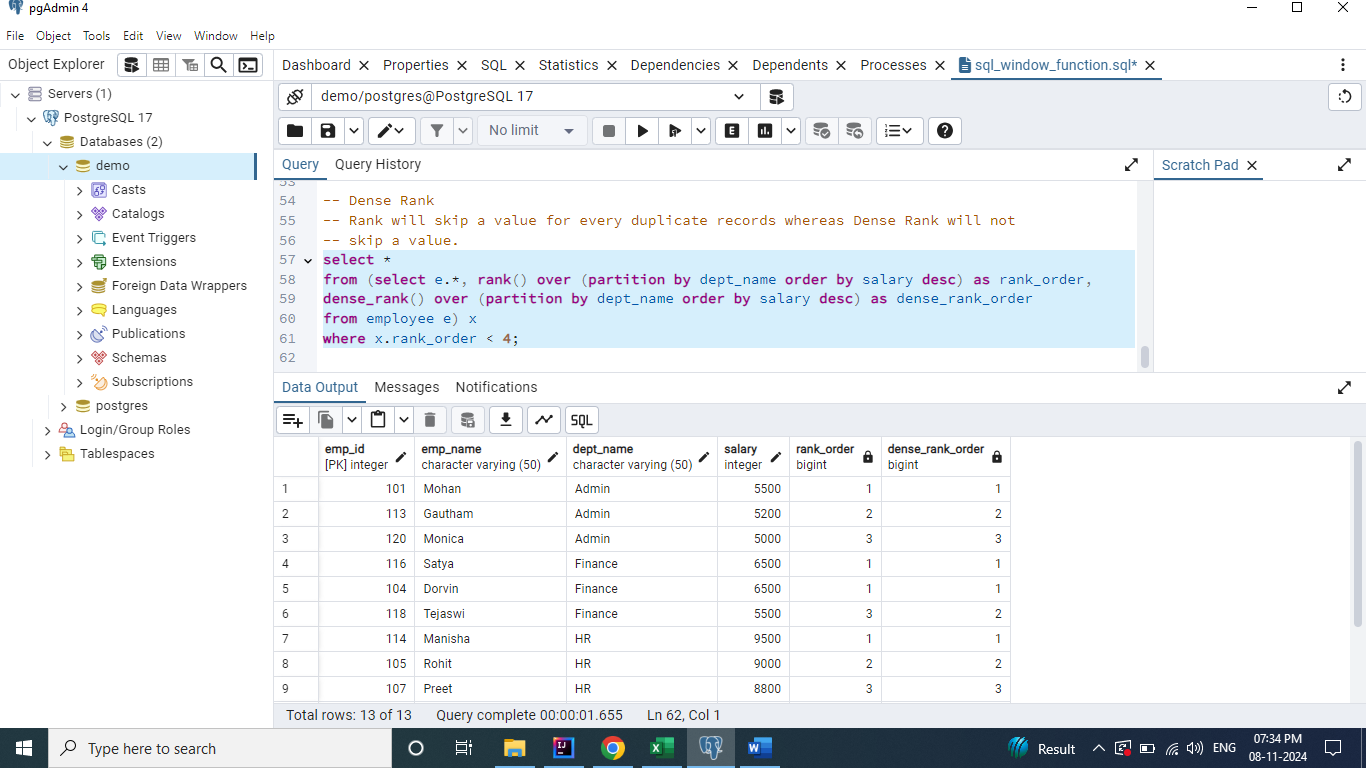
**select \***

**from (select e.\*, rank() over (partition by dept\_name order by salary desc) as rank\_order,**

**dense\_rank() over (partition by dept\_name order by salary desc) as dense\_rank\_order**

**from employee e) x**

**where x.rank\_order < 4;**



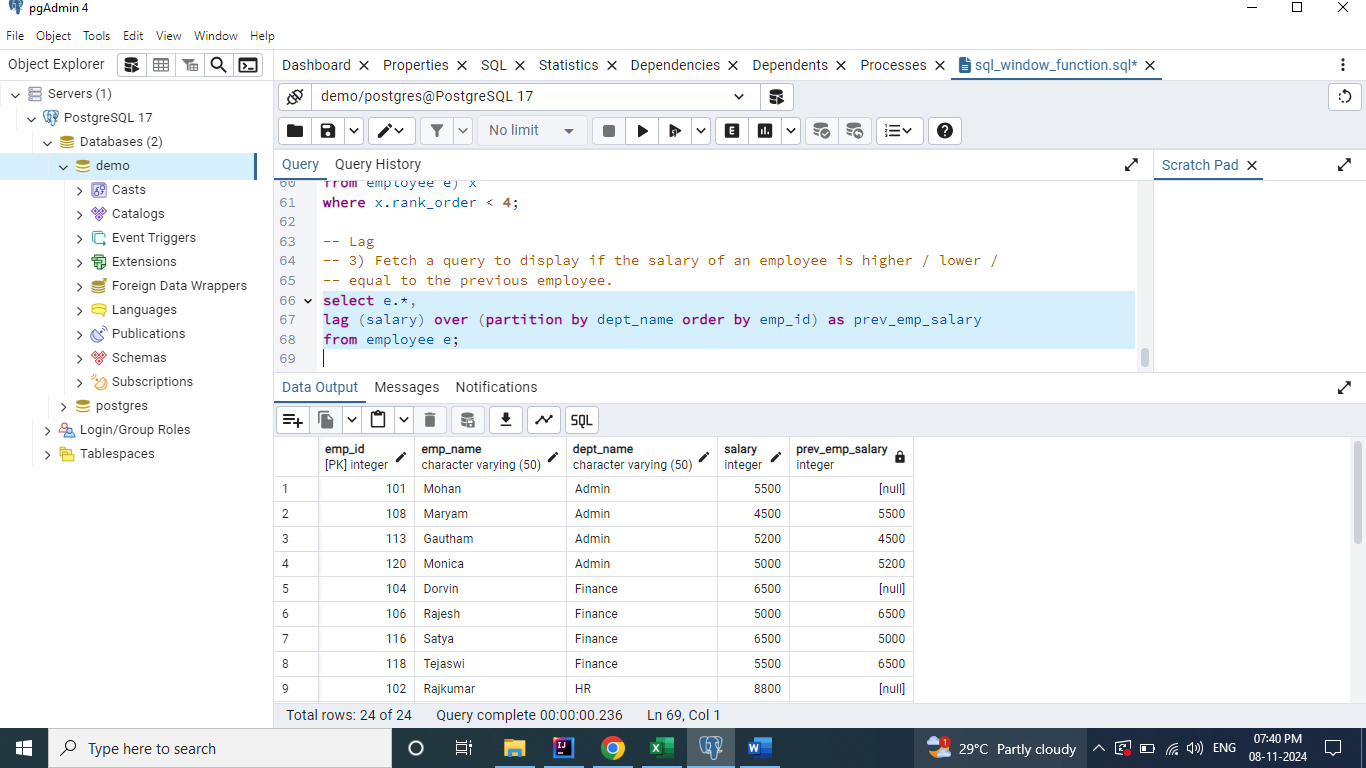
# Lag

## 3) Fetch a query to display if the salary of an employee is higher / lower / equal to the previous employee.

**select e.\*,**

**lag (salary) over (partition by dept\_name order by emp\_id) as prev\_emp\_salary**

**from employee e;**



**lag(salary, 2, 0)**

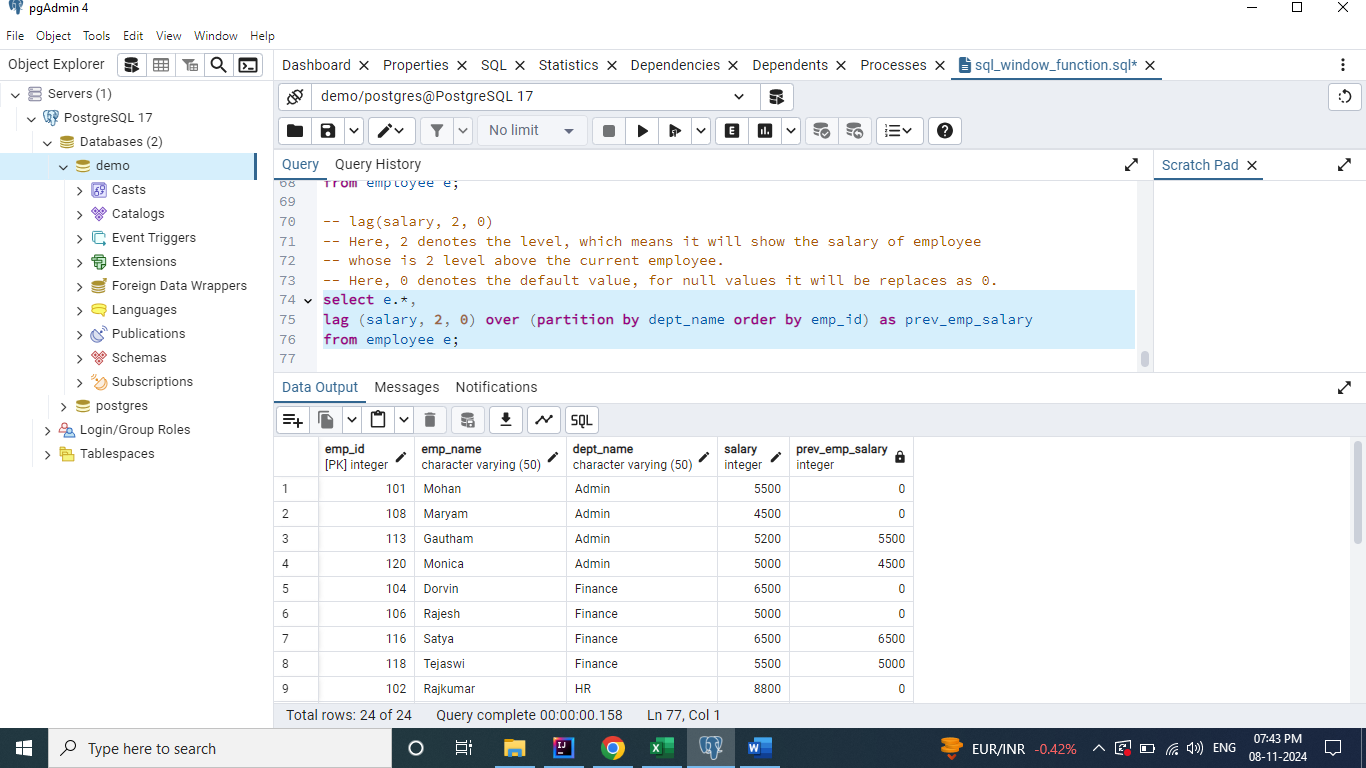
**Here, 2 denotes the level, which means it will show the salary of employee whose is 2 level above the current employee.**

**Here, 0 denotes the default value, for null values it will be replaces as 0.**

**select e.\*,**

**lag (salary, 2, 0) over (partition by dept\_name order by emp\_id) as prev\_emp\_salary**

**from employee e;**

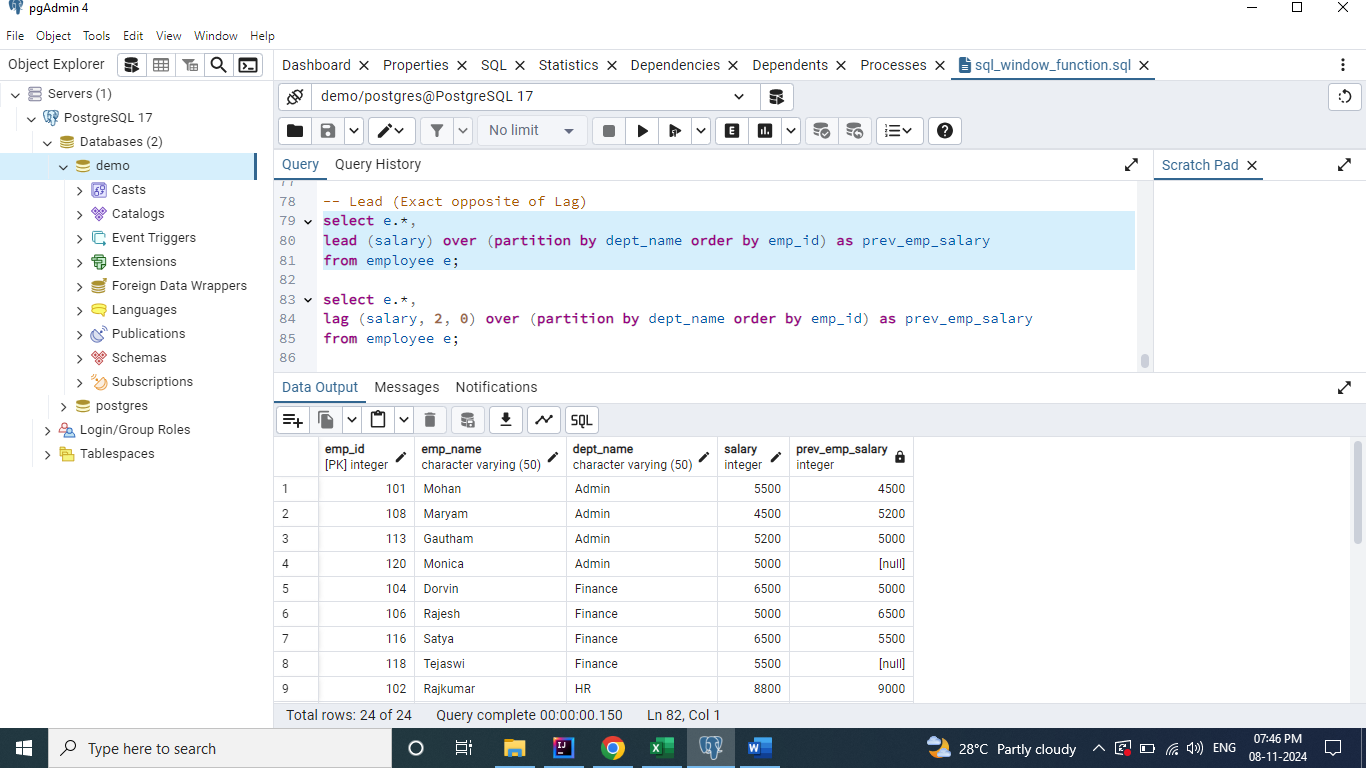


# Lead (Exact opposite of Lag)

**select e.\*,**

**lead (salary) over (partition by dept\_name order by emp\_id) as next\_emp\_salary**

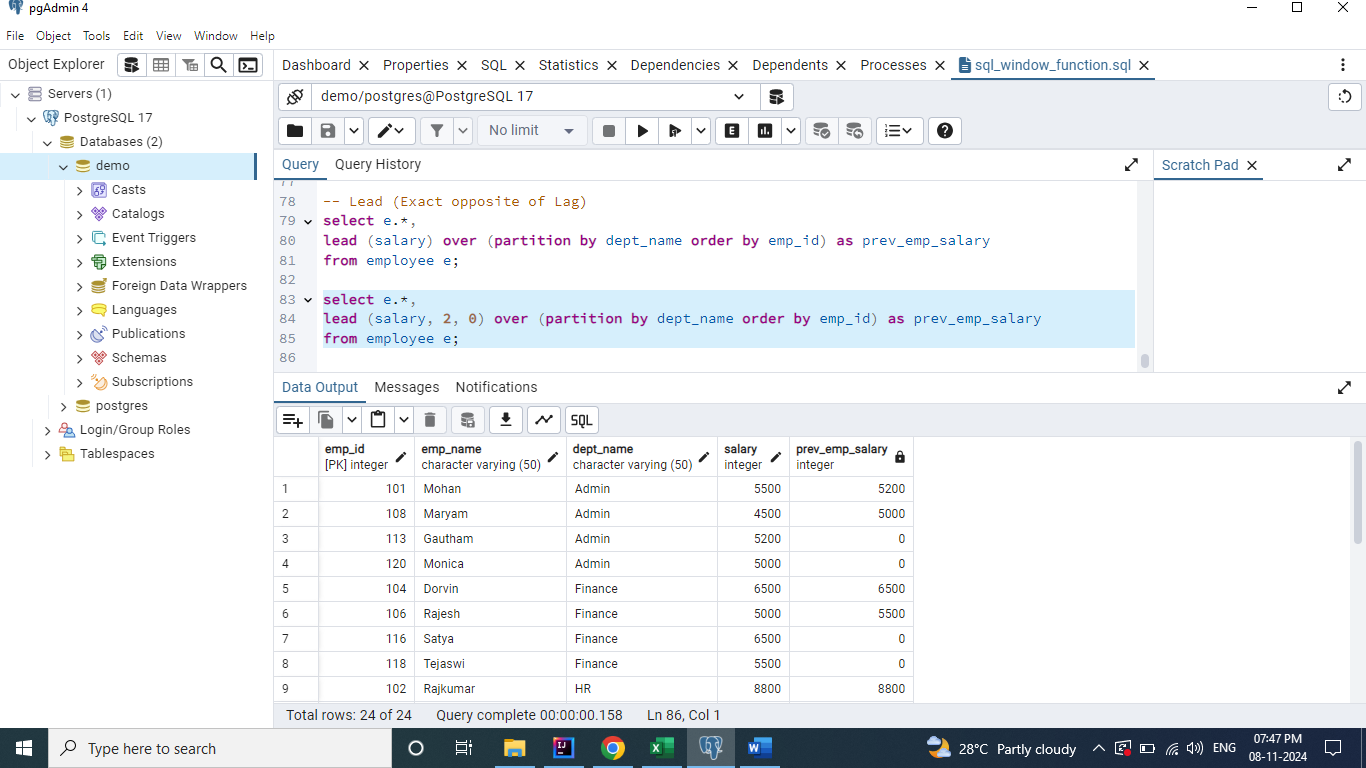
**from employee e;**



**select e.\*,**

**lead (salary, 2, 0) over (partition by dept\_name order by emp\_id) as next\_emp\_salary**

**from employee e;**



## Use Case

**select e.\*,**

**lag (salary) over (partition by dept\_name order by emp\_id) as prev\_emp\_salary,**

**case**

**when e.salary > lag(salary) over (partition by dept\_name order by emp\_id)**

**then 'Salary Higher than previous employee'**

**when e.salary < lag(salary) over (partition by dept\_name order by emp\_id)**

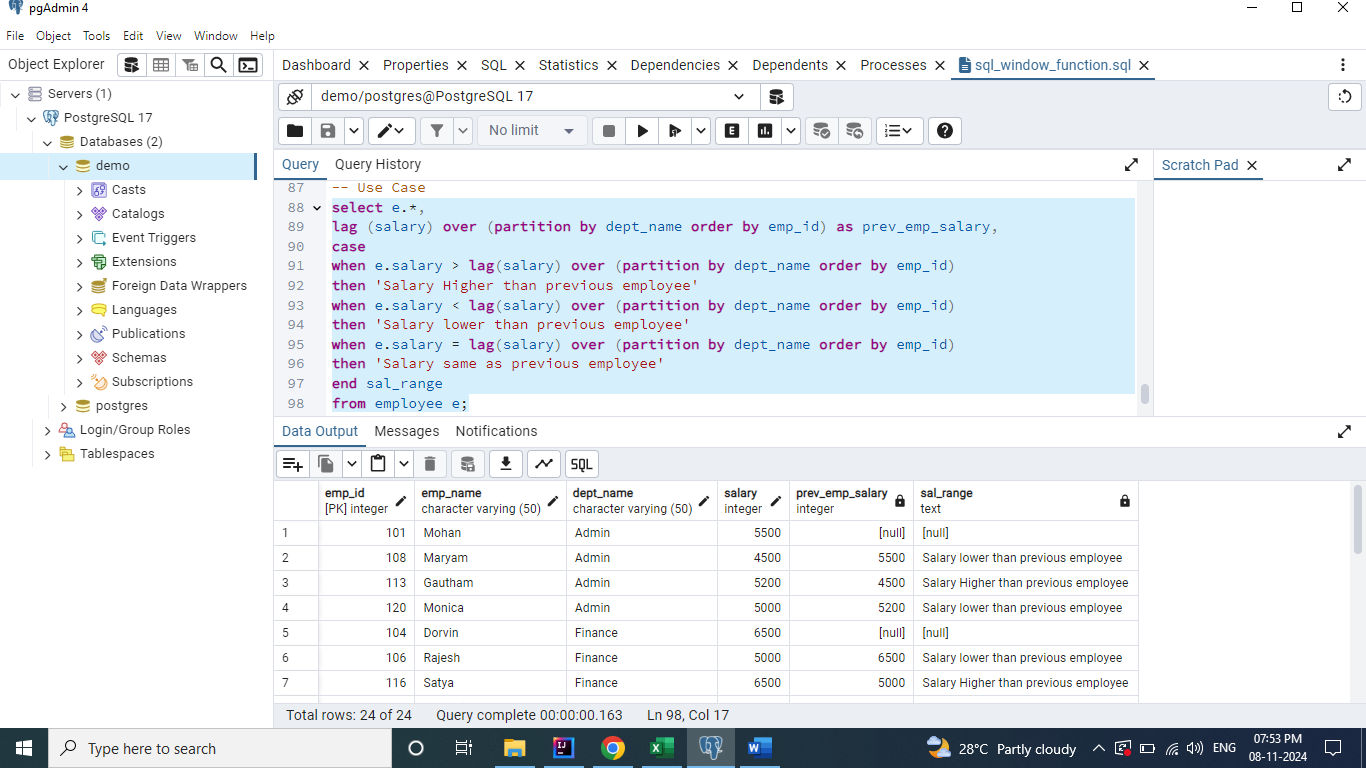
**then 'Salary lower than previous employee'**

**when e.salary = lag(salary) over (partition by dept\_name order by emp\_id)**

**then 'Salary same as previous employee'**

**end sal\_range**

**from employee e;**



# First Value

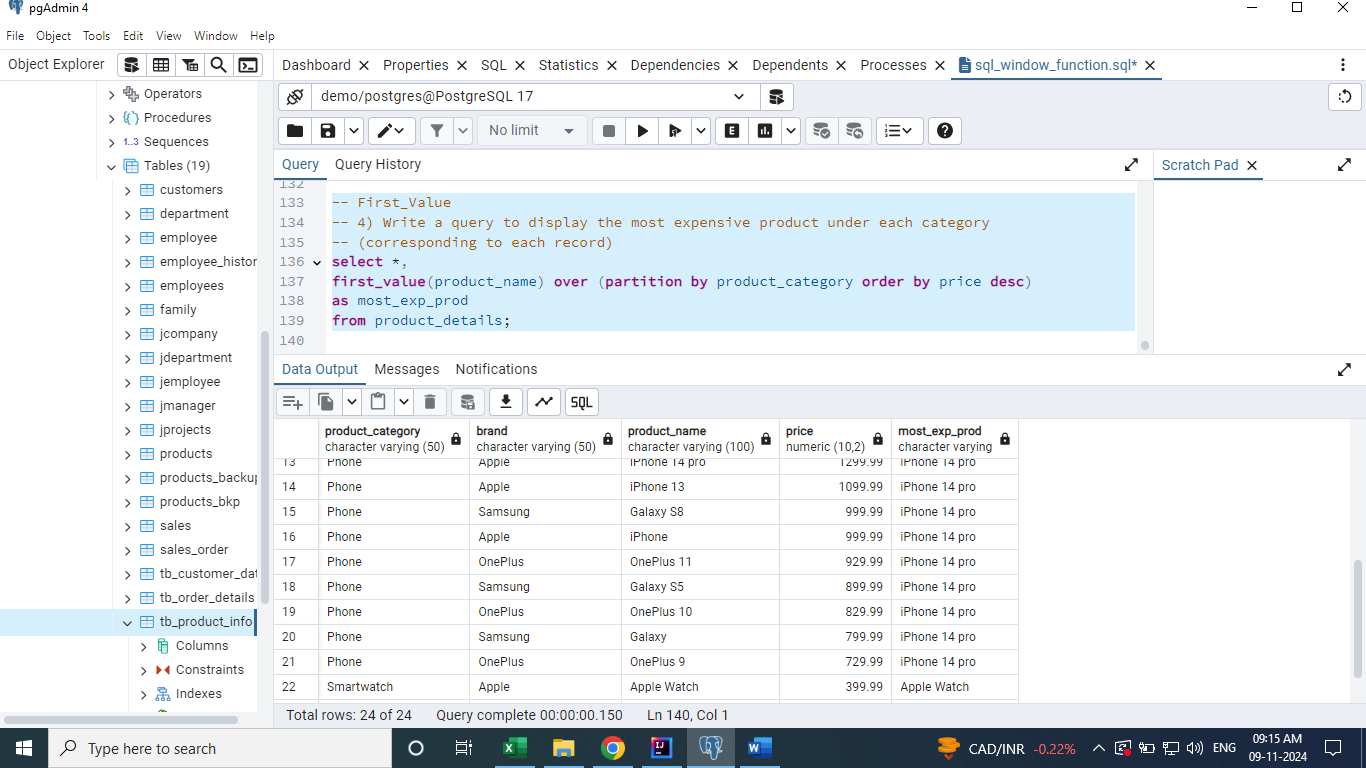
## 4) Write a query to display the most expensive product under each category (corresponding to each record)

**select \*,**

**first\_value(product\_name) over (partition by product\_category order by price desc)**

**as most\_exp\_prod**

**from product\_details;**



# Last Value

## 5) Write a query to display the least expensive product under each category (corresponding to each record)

**select \*,**

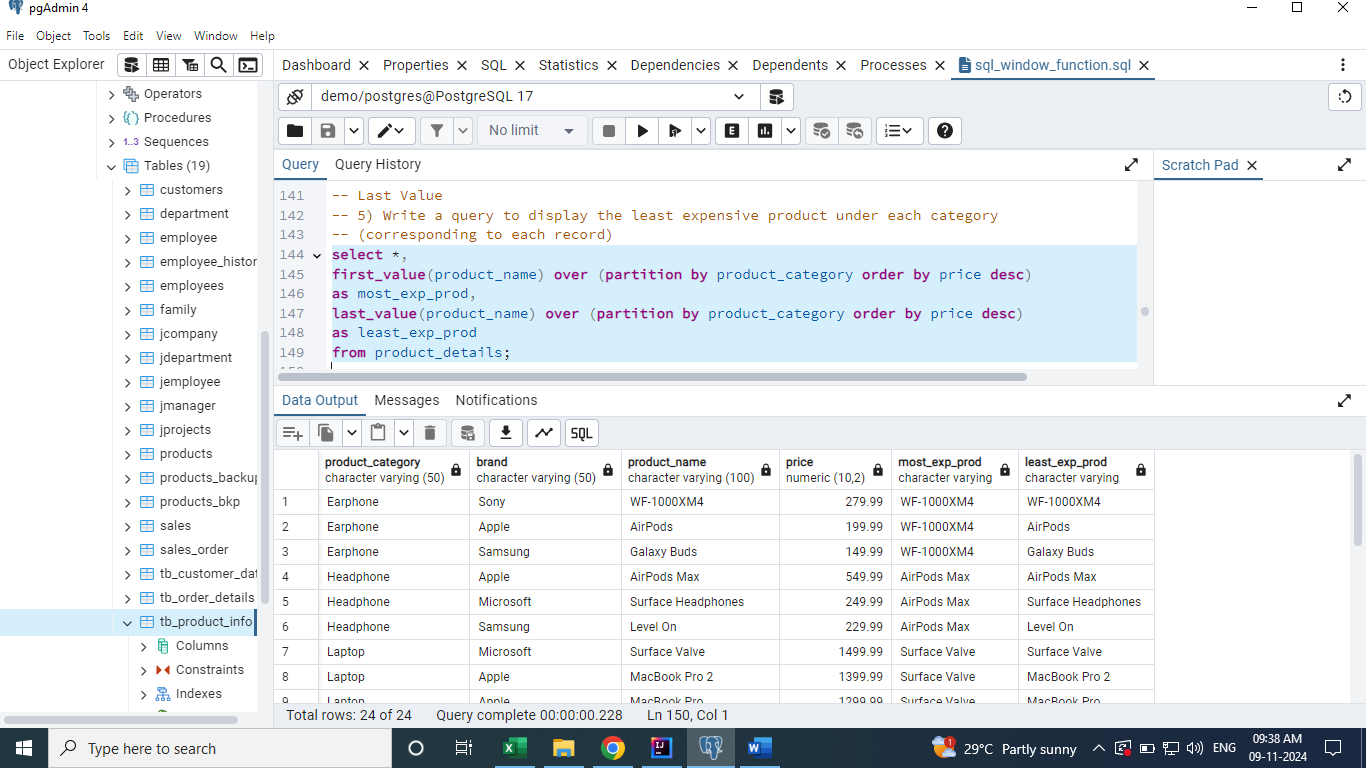
**first\_value(product\_name) over (partition by product\_category order by price desc)**

**as most\_exp\_prod,**

**last\_value(product\_name) over (partition by product\_category order by price desc)**

**as least\_exp\_prod**

**from product\_details;**



# Frame Clause

**Whenever we are using a window function, it creates a window / partition and it applies that window functions to each of the partitions. Inside each of these partitions again we create a subset record called as Frames. So, basically a frame is a subset of a partition.**

**select \*,**

**first\_value(product\_name) over (partition by product\_category order by price desc)**

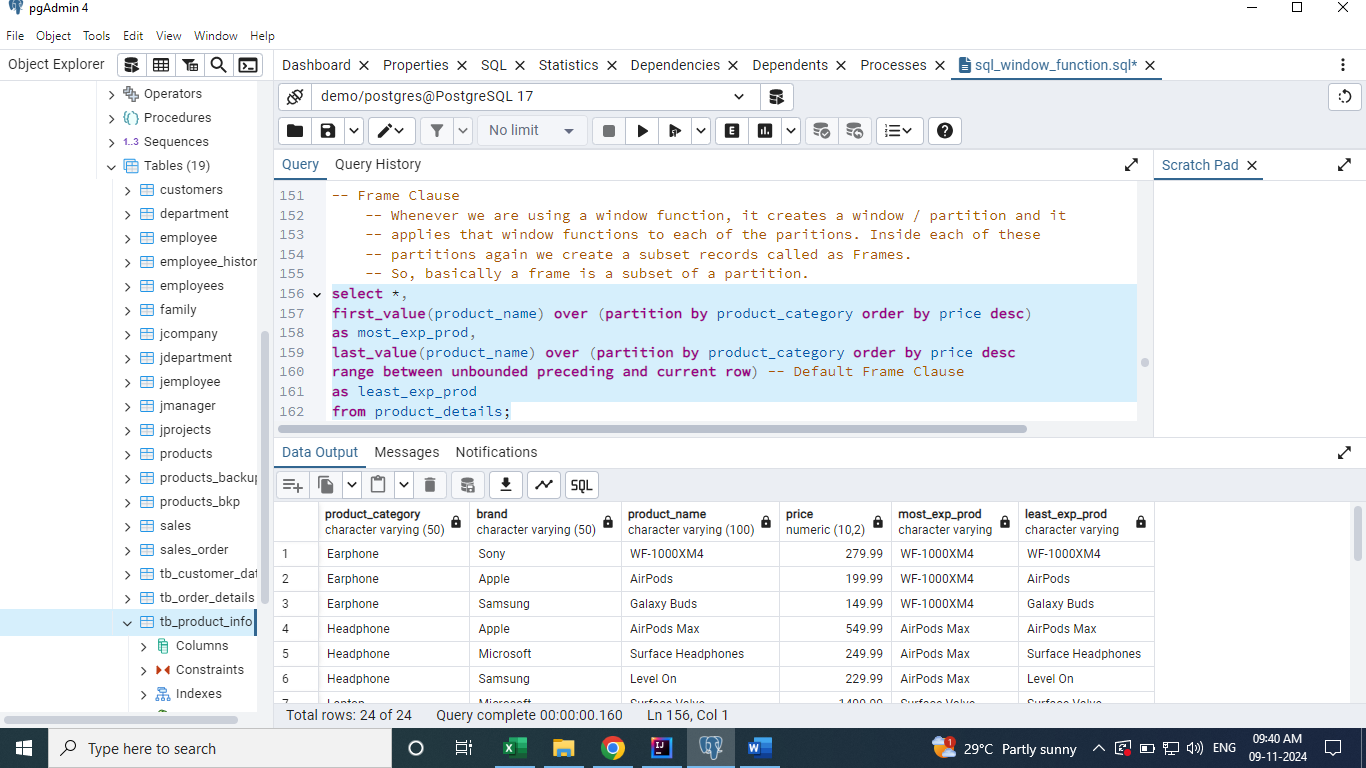
**as most\_exp\_prod,**

**last\_value(product\_name) over (partition by product\_category order by price desc**

**range between unbounded preceding and current row) -- Default Frame Clause**

**as least\_exp\_prod**

**from product\_details;**



# Default Frame Clause

**Though we mention this 'range between unbounded preceding and current row' or not.**

**Sql will consider it by default. Range - tells what is the range of records this last value window function to consider.**

**unbounded preceding - means it basically the rows preceding to the current row.**

**unbounded basically means the very first row of the partition.**

**select \*,**

**first\_value(product\_name) over (partition by product\_category order by price desc)**

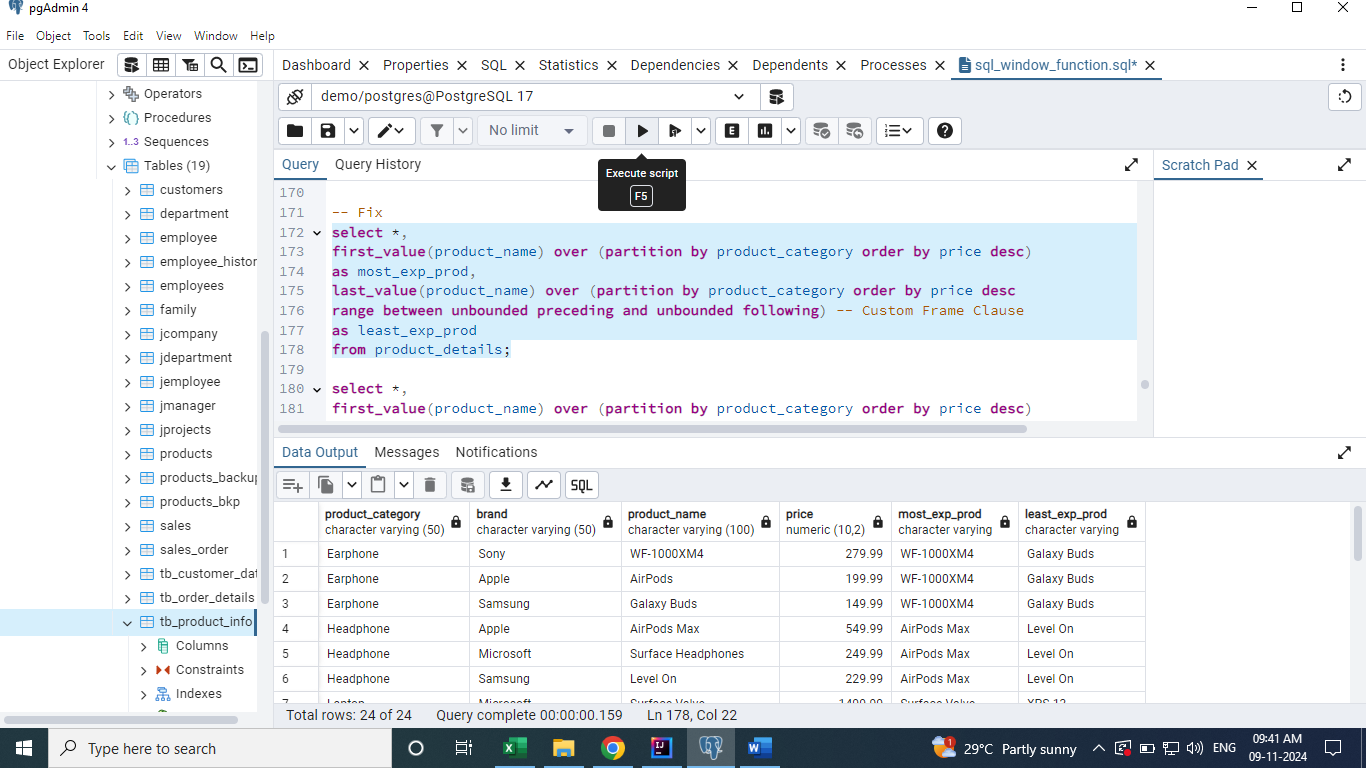
**as most\_exp\_prod,**

**last\_value(product\_name) over (partition by product\_category order by price desc**

**range between unbounded preceding and unbounded following) -- Custom Frame Clause**

**as least\_exp\_prod**

**from product\_details;**



**select \*,**

**first\_value(product\_name) over (partition by product\_category order by price desc)**

**as most\_exp\_prod,**

**last\_value(product\_name) over (partition by product\_category order by price desc**

**rows between unbounded preceding and unbounded following)**

**as least\_exp\_prod**

**from product\_details**

**where product\_category = 'Phone';**

**select \*,**

**first\_value(product\_name) over (partition by product\_category order by price desc)**

**as most\_exp\_prod,**

**last\_value(product\_name) over (partition by product\_category order by price desc**

**rows between unbounded preceding and current row)**

**as least\_exp\_prod**

**from product\_details**

**where product\_category = 'Phone';**

**select \*,**

**first\_value(product\_name) over (partition by product\_category order by price desc)**

**as most\_exp\_prod,**

**last\_value(product\_name) over (partition by product\_category order by price desc**

**range between 2 preceding and 2 following)**

**as least\_exp\_prod**

**from product\_details**

**where product\_category = 'Phone';**

**Alternate way to write a SQL query using Window Functions.**

**select \*,**

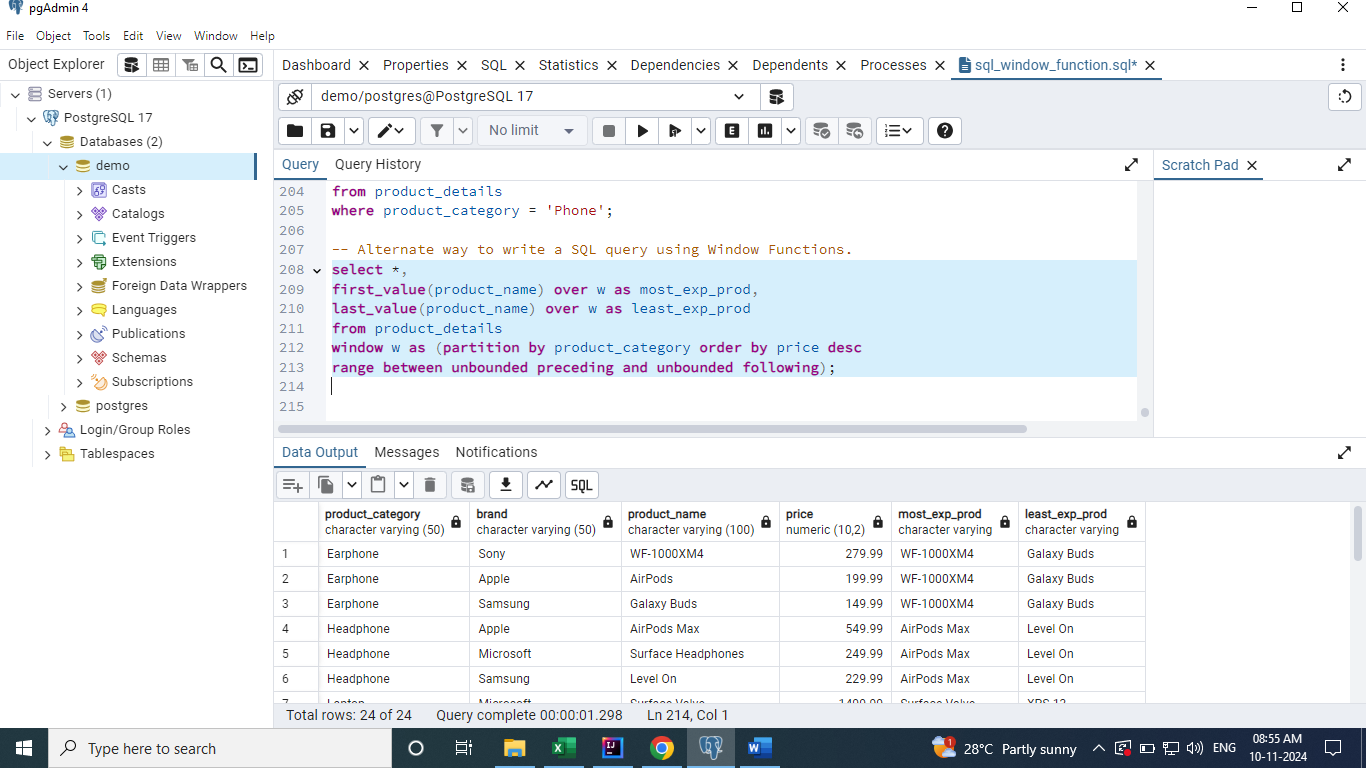
**first\_value(product\_name) over w as most\_exp\_prod,**

**last\_value(product\_name) over w as least\_exp\_prod**

**from product\_details**

**window w as (partition by product\_category order by price desc**

**range between unbounded preceding and unbounded following);**



# Nth Value

## 6) Write a query to display the second most expensive product under each category.

**select \*,**

**first\_value(product\_name) over w as most\_exp\_prod,**

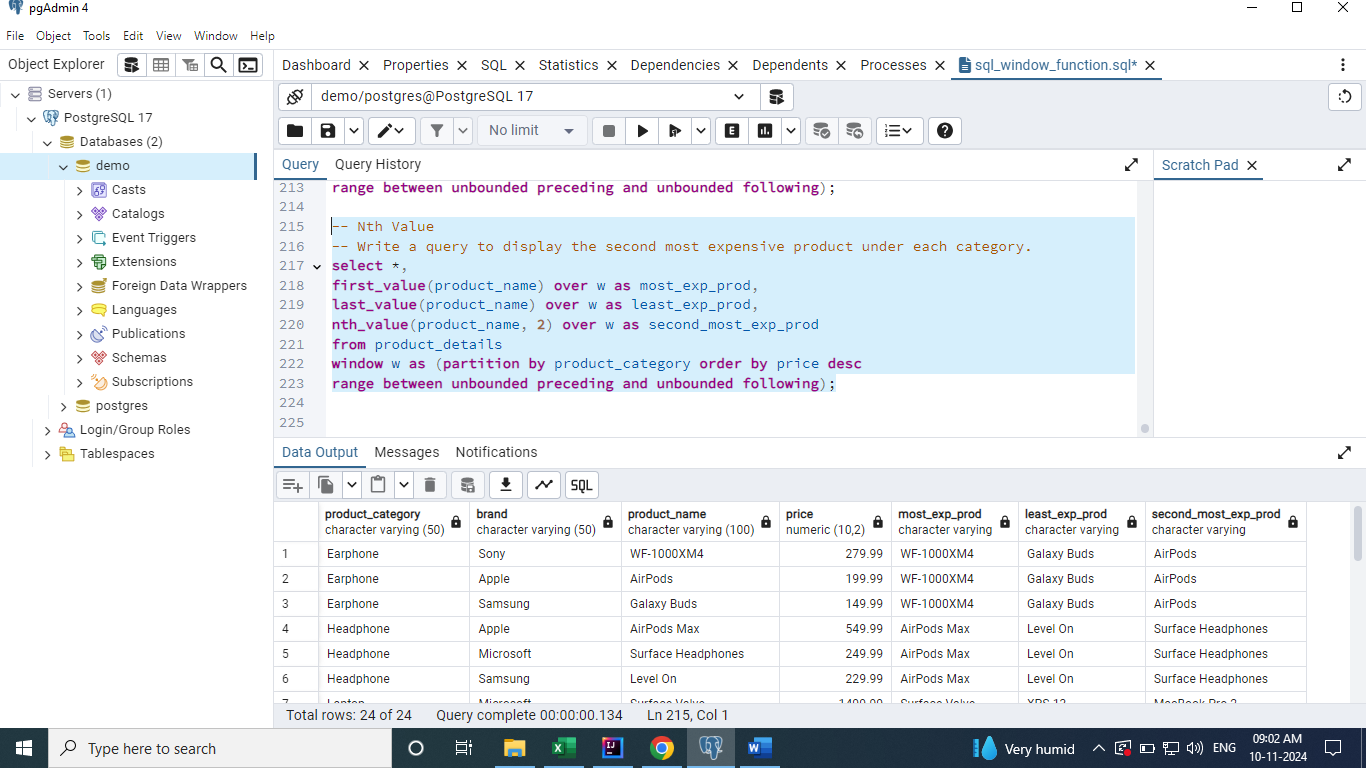
**last\_value(product\_name) over w as least\_exp\_prod,**

**nth\_value(product\_name, 2) over w as second\_most\_exp\_prod**

**from product\_details**

**window w as (partition by product\_category order by price desc**

**range between unbounded preceding and unbounded following);**



# NTile

**Whenever we want to group together a few records into some buckets.**

## 7) Write a query to segregate all the expensive phones, mid range phones and cheaper phones.

**select product\_name,**

**case**

**when x.buckets = 1 then 'Expensive Phones'**

**when x.buckets = 2 then 'Medium Phones'**

**when x.buckets = 3 then 'Cheap Phones'**

**end phone\_category**

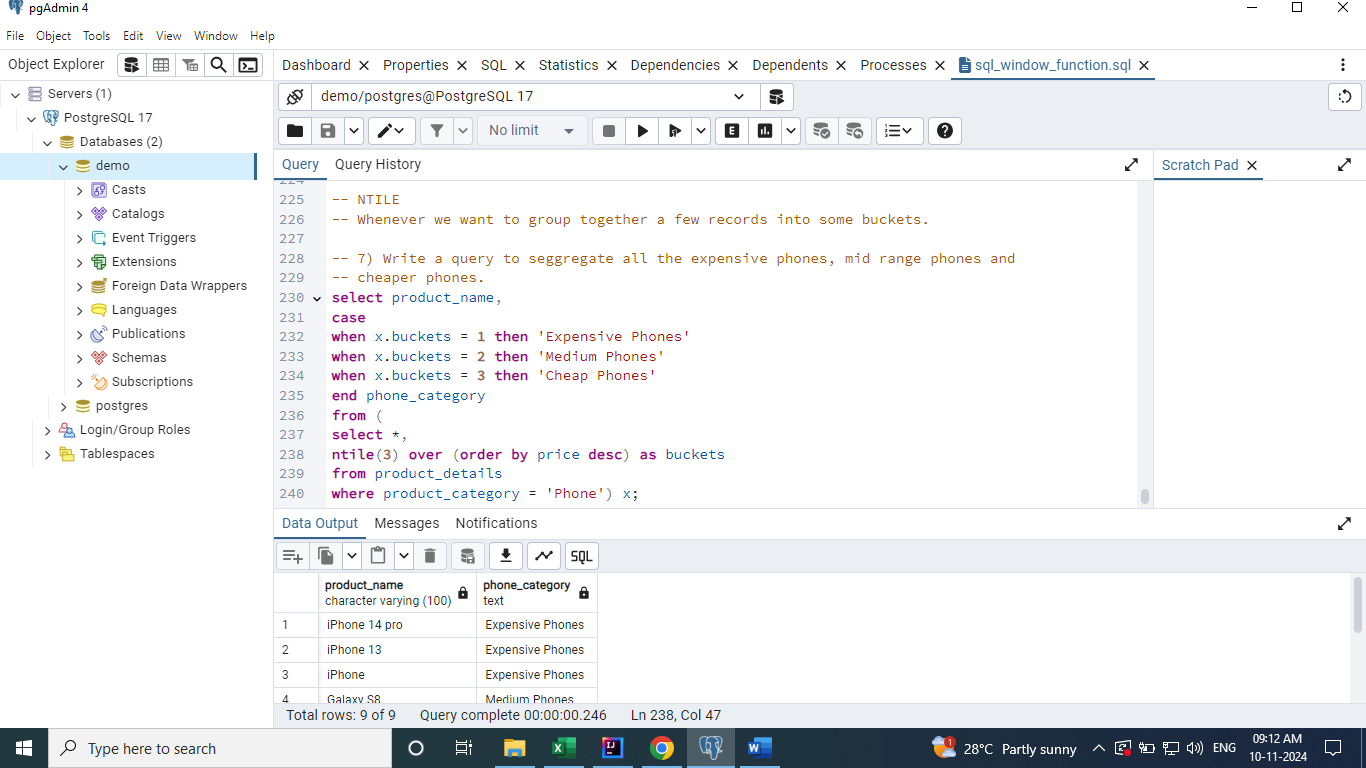
**from (**

**select \*,**

**ntile(3) over (order by price desc) as buckets**

**from product\_details**

**where product\_category = 'Phone') x;**



# Cume\_Dist [Cumulative Distribution]

**Value -> 1 <= CUME\_DIST > 0**

**Formula = Current Row No (or Row No with value same as current row) / Total no of rows**

## 8) Query to fetch all products which are constituting the first 30% of the data in product details table based on price.

**select product\_name, (cume\_dist\_percent||'%') as cume\_dist\_percent**

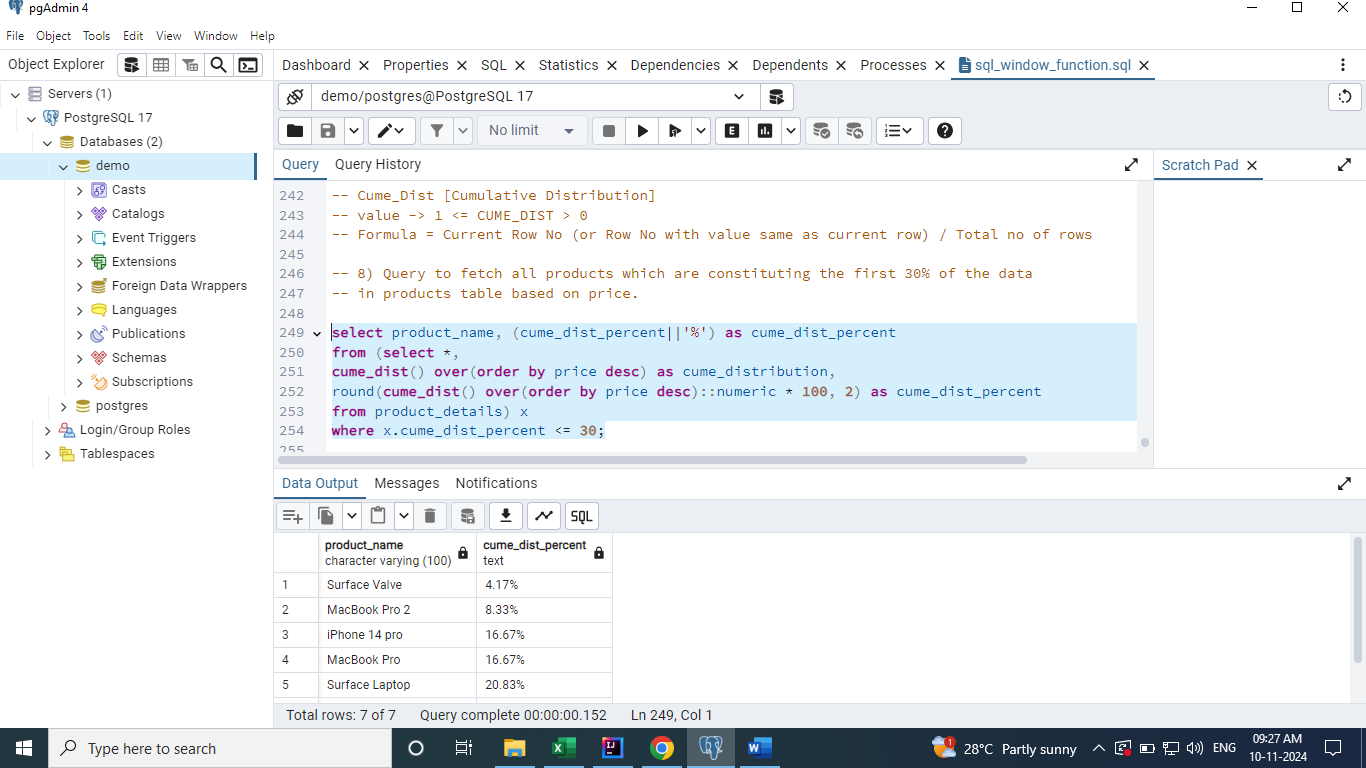
**from (select \*,**

**cume\_dist() over(order by price desc) as cume\_distribution,**

**round(cume\_dist() over(order by price desc)::numeric \* 100, 2) as cume\_dist\_percent**

**from product\_details) x**

**where x.cume\_dist\_percent <= 30;**



# Percent Rank

**[Relative Rank of the current row / Percentage Ranking]**

**Value 1 <= PERCENT\_RANK > 0**

**Formula = Current row no - 1 / Total no of rows - 1**

## 9) Query to identify how much percentage more expensive is "Galaxy S5" when compared to all products.

**select product\_name, perc\_rank**

**from**

**( select \*,**

**percent\_rank() over(order by price desc) as percentage\_rank,**

**round(percent\_rank() over(order by price desc)::numeric \* 100, 2) as perc\_rank**

**from product\_details ) x**

**where x.product\_name = 'Galaxy S5';**

