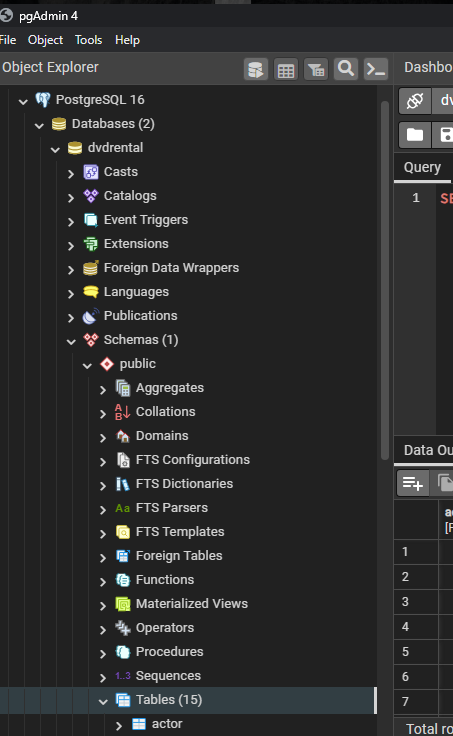
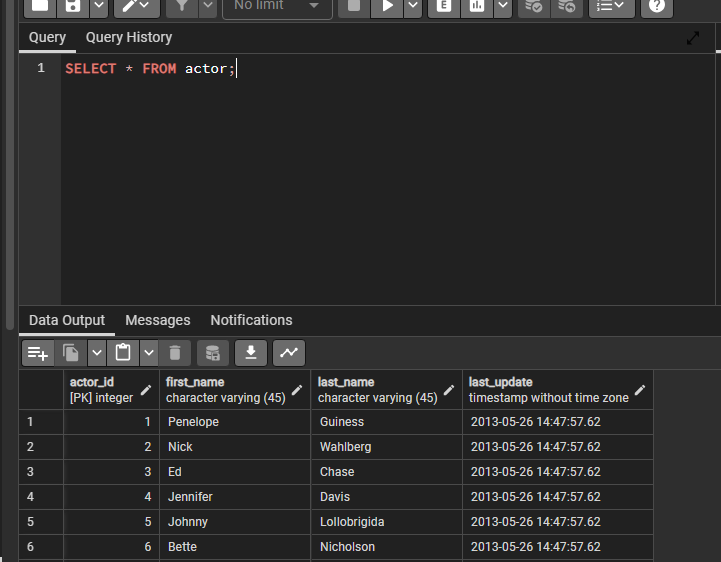
# SQL Chapter 1

To View the tables in a DB,



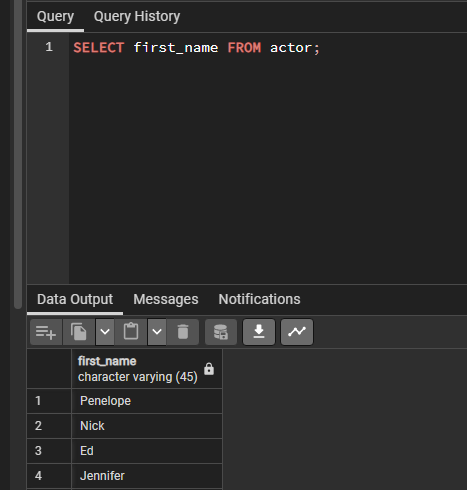
Viewing all contents of a table

SELECT \* FROM table\_name;



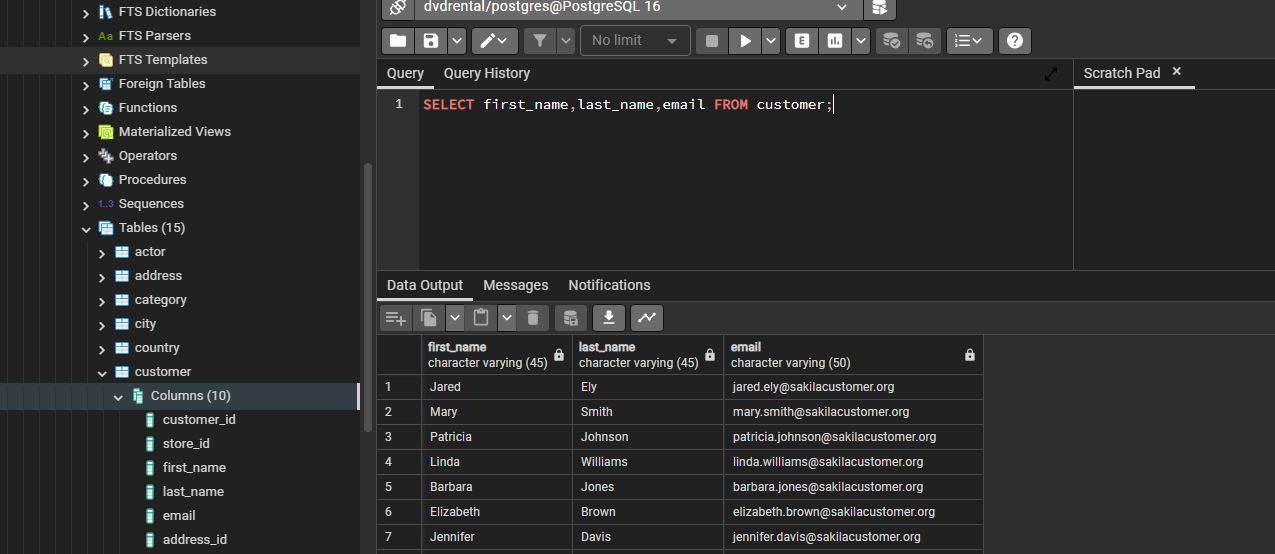
If only one column is required

SELECT first\_name FROM actor;



Get multiple data

SELECT first\_name,last\_name,email FROM customer;

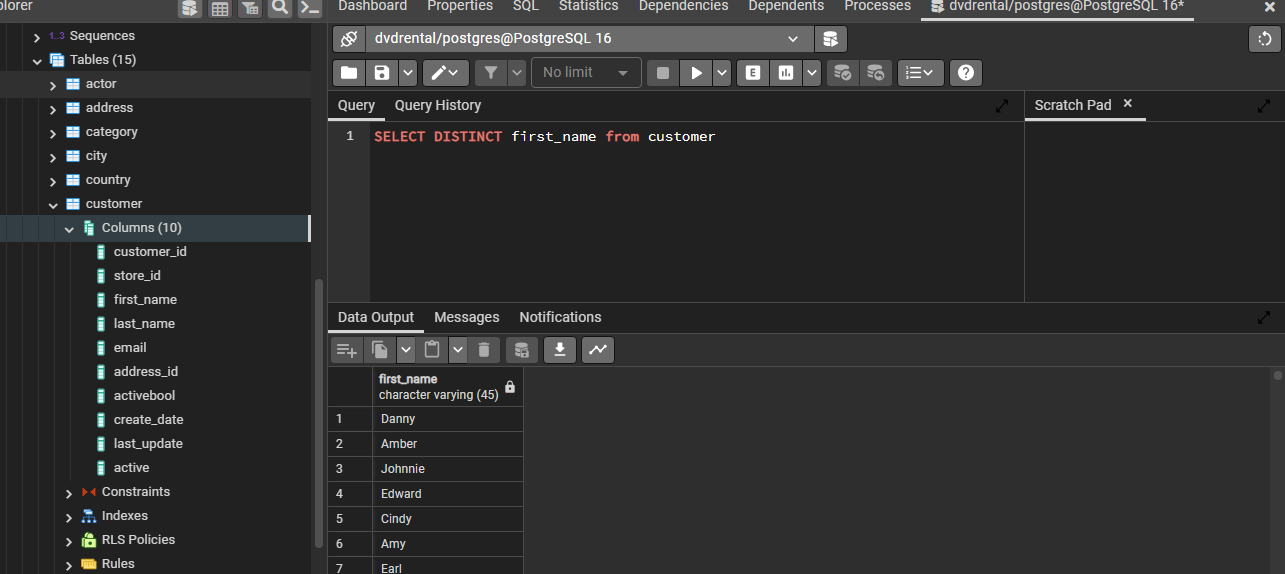


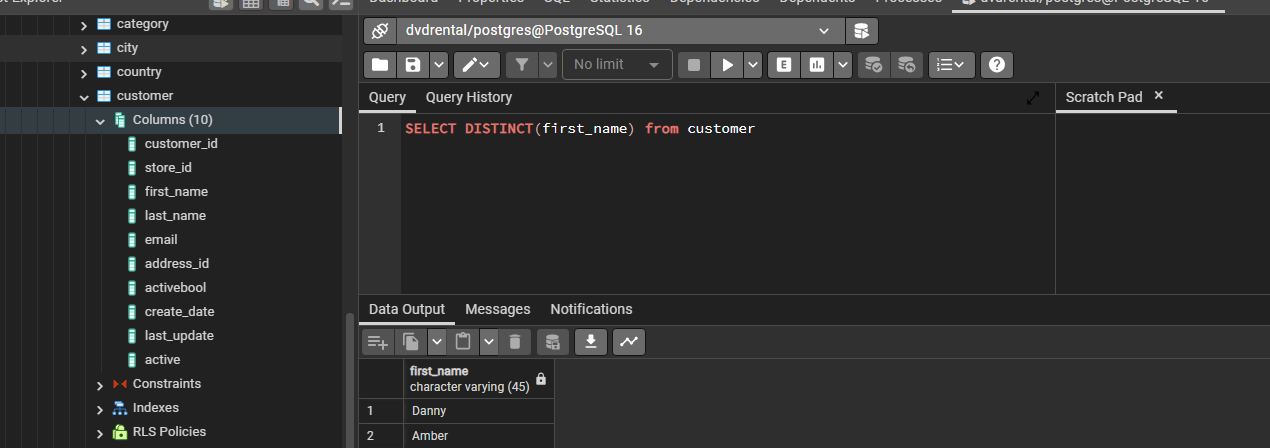
DISTINCT KEYWORD

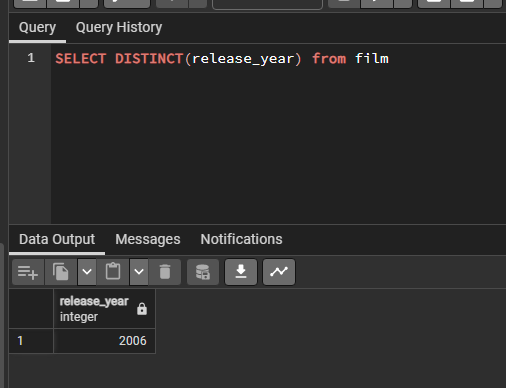
SELECT DISTINCT column FROM table

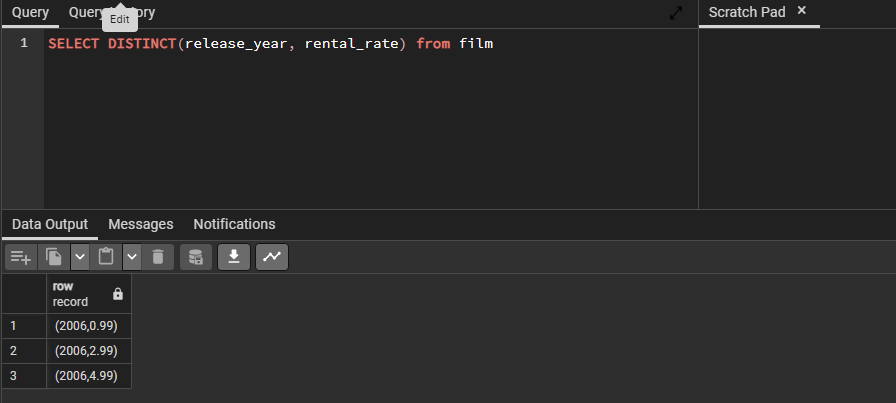
OR

SELECT DISTINCT(col) FROM table



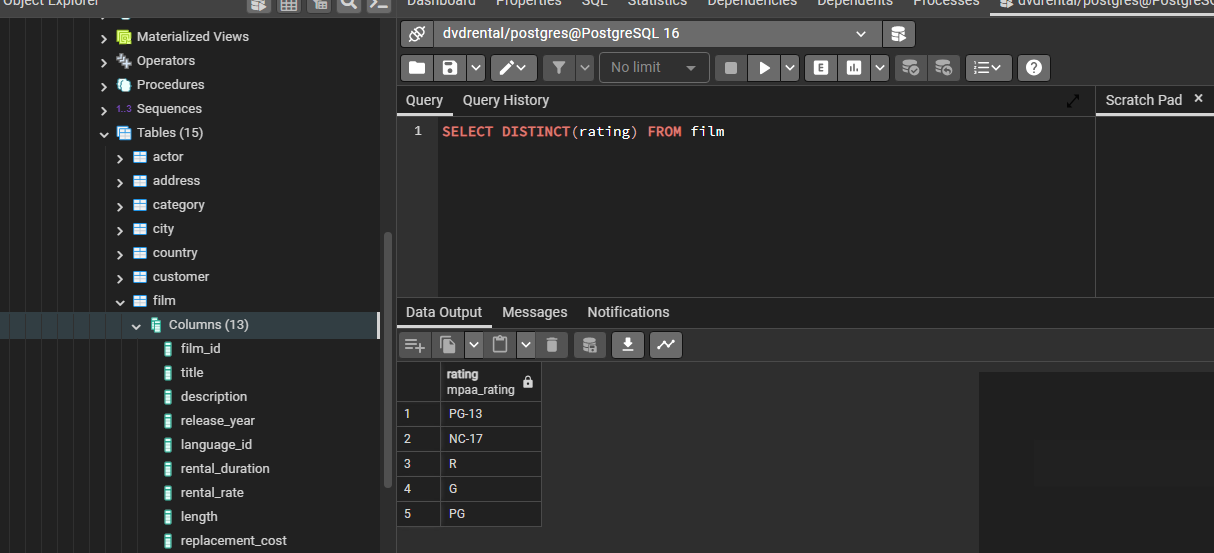


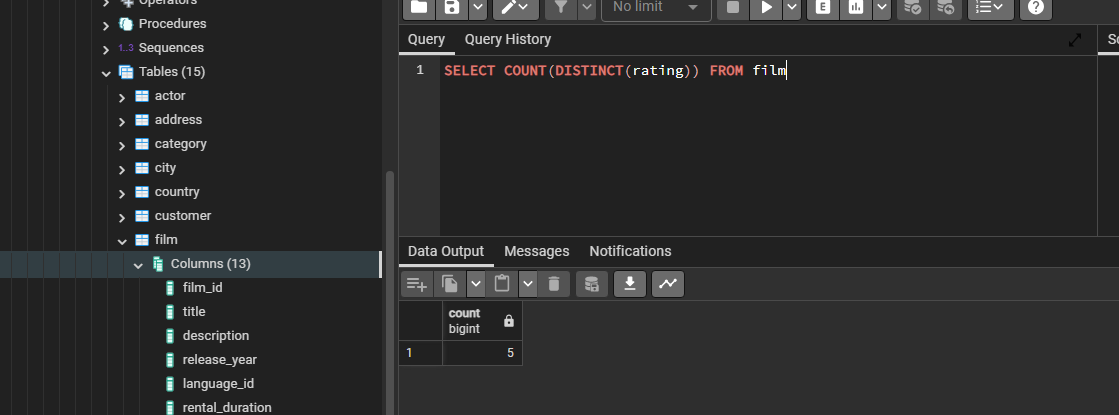




COUNT Keyword

SELECT COUNT(DISTINCT(rating)) FROM film

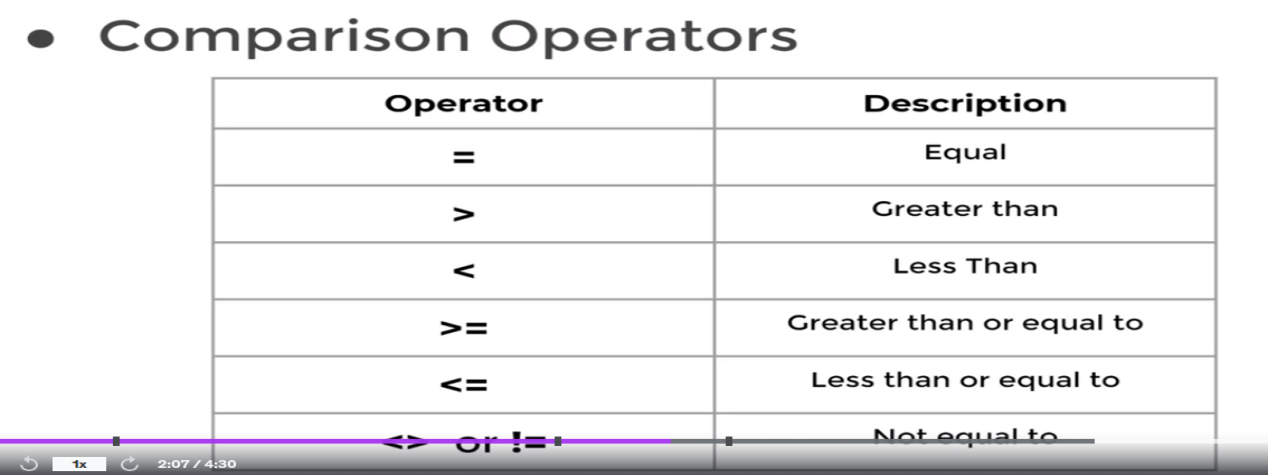




#### SELECT WHERE KEYWORD

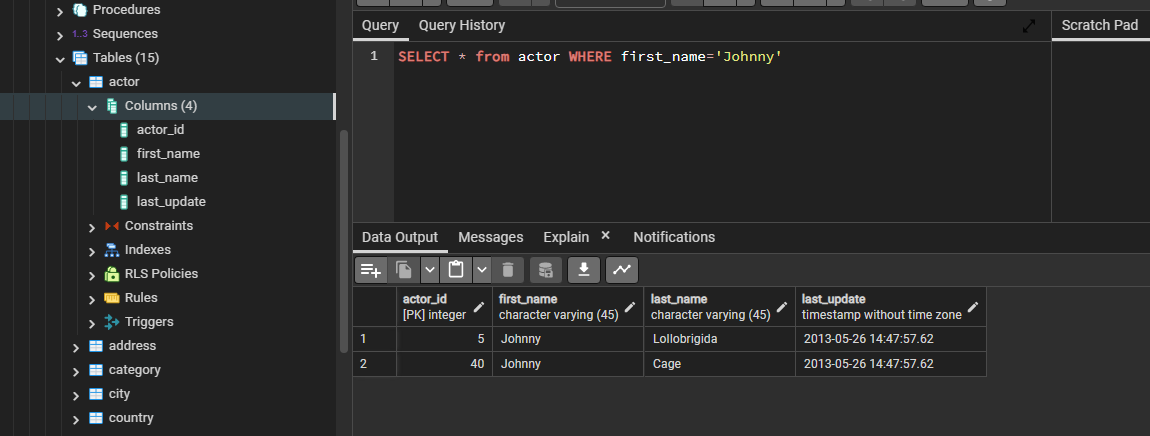
SELECT col1, col2 FROM table WHERE condition

Conditions can be made using comparison operators :

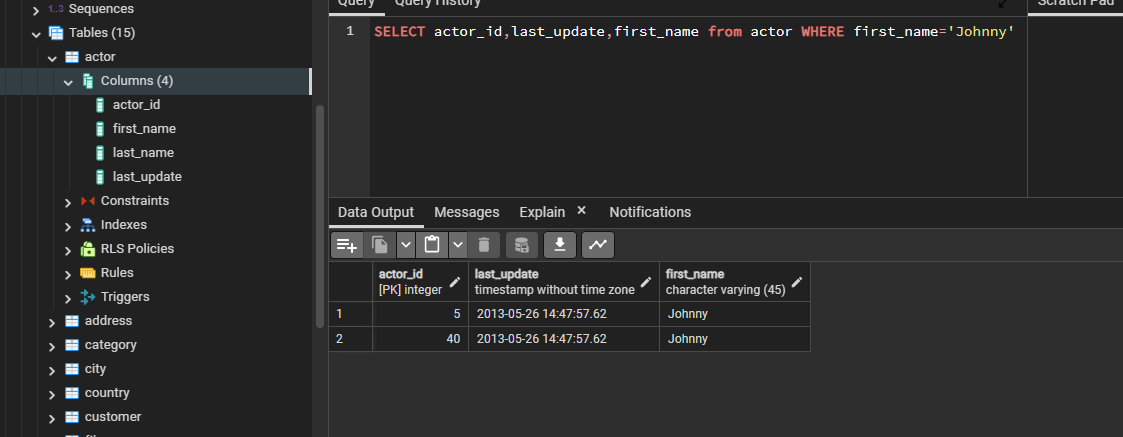


Logical Operators : AND, OR, NOT

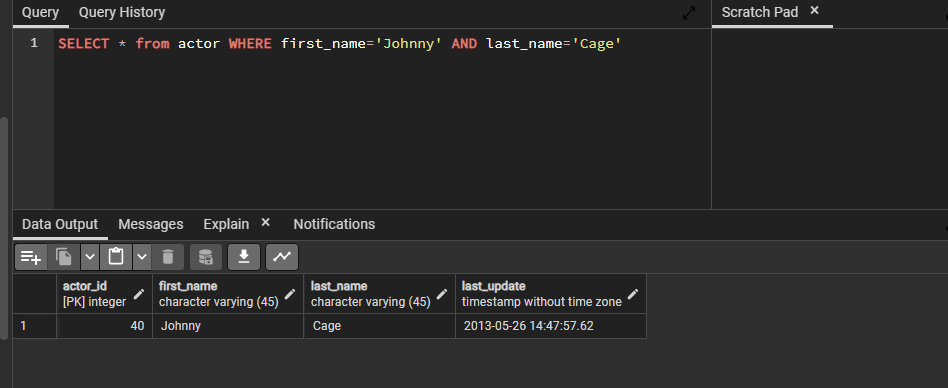
SELECT \* from actor WHERE first\_name='Johnny'



SELECT actor\_id,last\_update,first\_name from actor WHERE first\_name='Johnny'

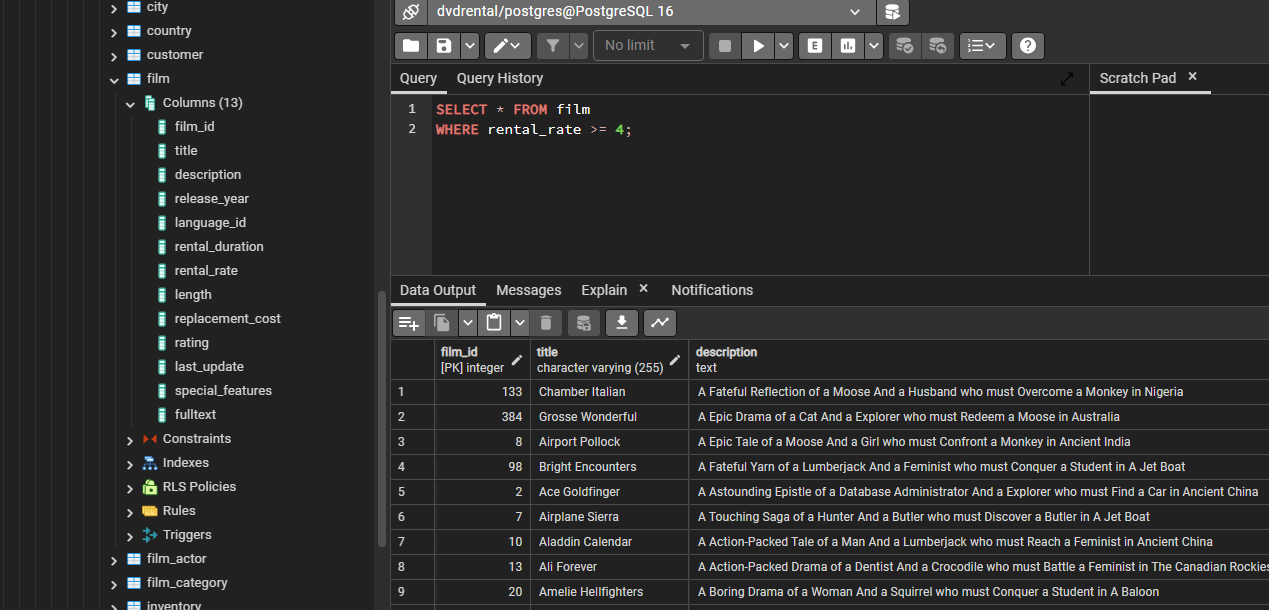


SELECT \* from actor WHERE first\_name='Johnny' AND last\_name='Cage';



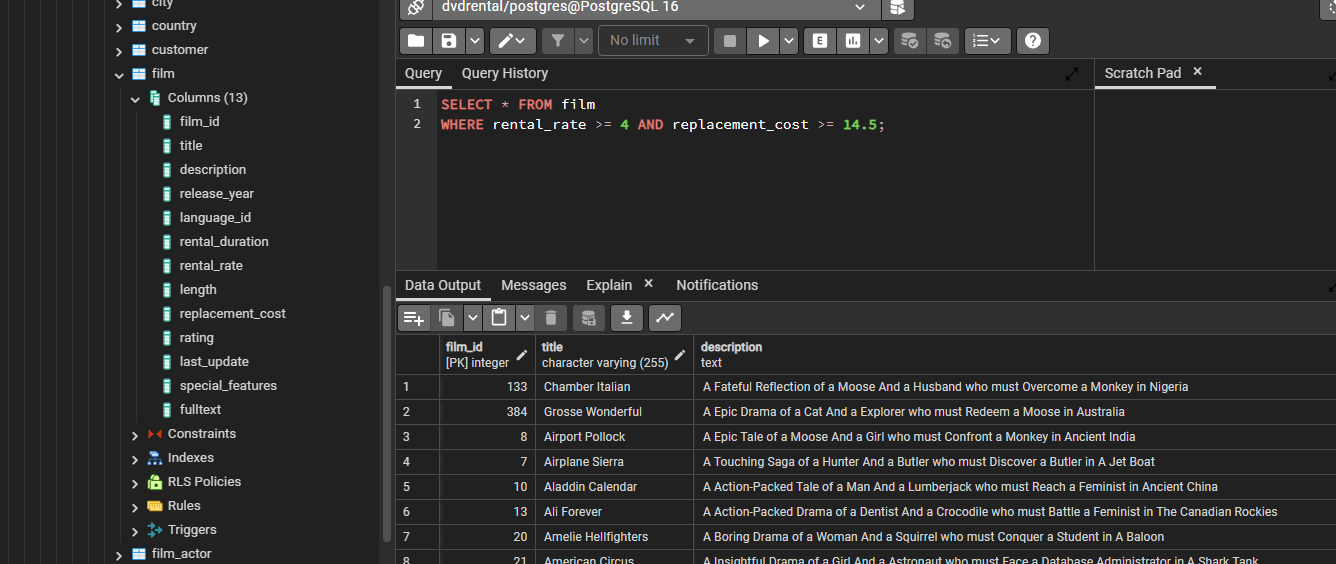
SELECT \* FROM film

WHERE rental\_rate >= 4;



SELECT \* FROM film

WHERE rental\_rate >= 4 AND replacement\_cost >= 14.5;



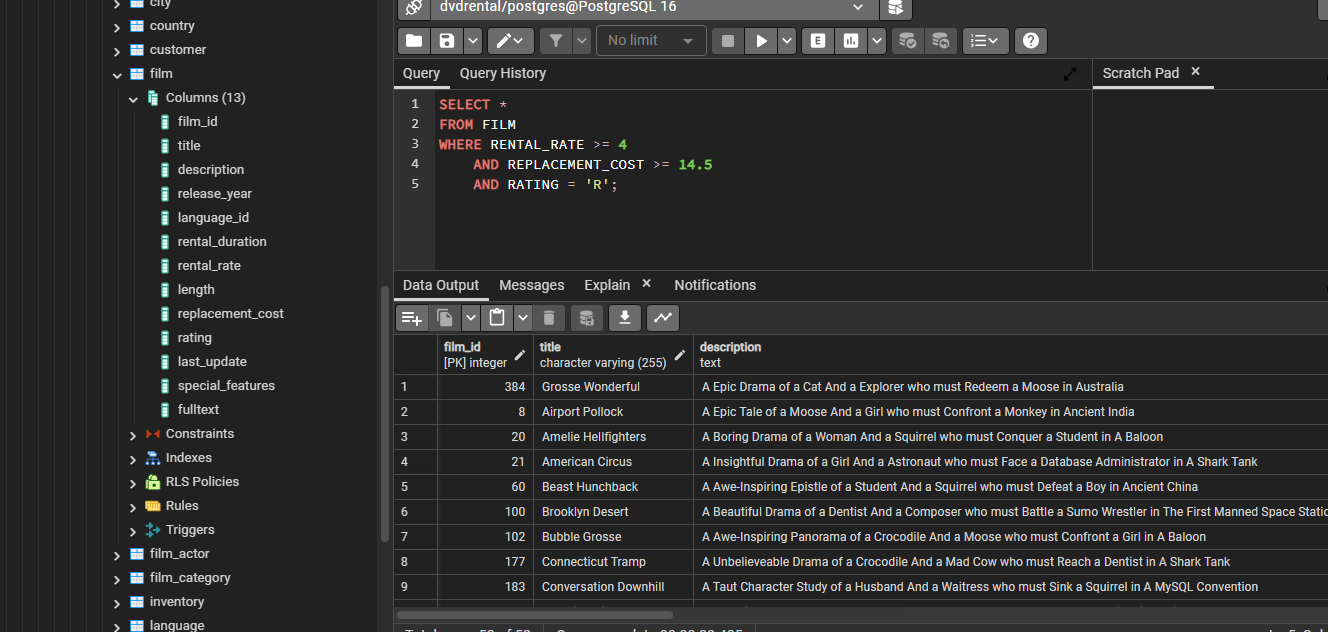
SELECT \*

FROM FILM

WHERE RENTAL\_RATE >= 4

AND REPLACEMENT\_COST >= 14.5

AND RATING = 'R';



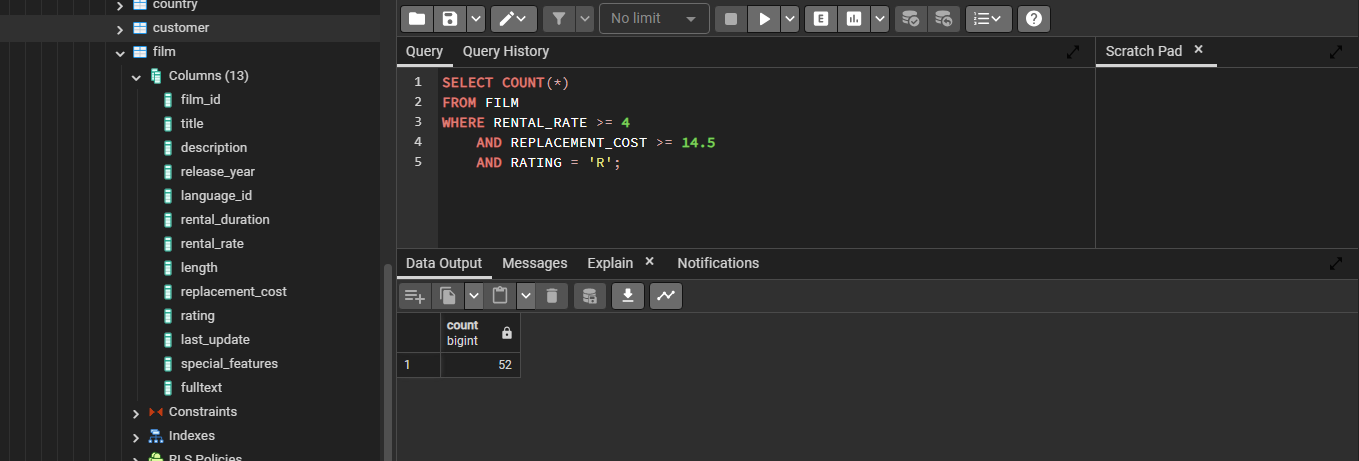
SELECT COUNT(\*)

FROM FILM

WHERE RENTAL\_RATE >= 4

AND REPLACEMENT\_COST >= 14.5

AND RATING = 'R';



SELECT COUNT(\*)

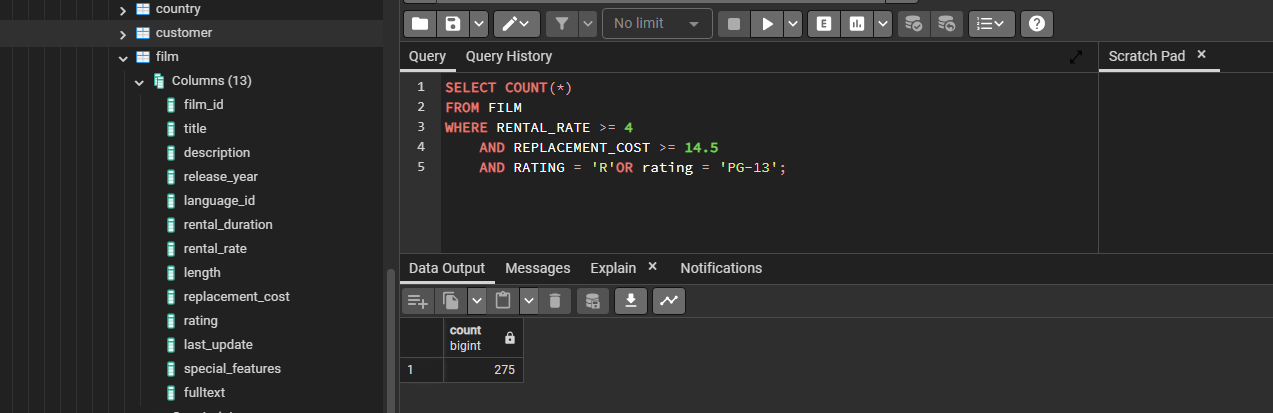
FROM FILM

WHERE RENTAL\_RATE >= 4

AND REPLACEMENT\_COST >= 14.5

AND RATING = 'R'

OR RATING = 'PG-13';

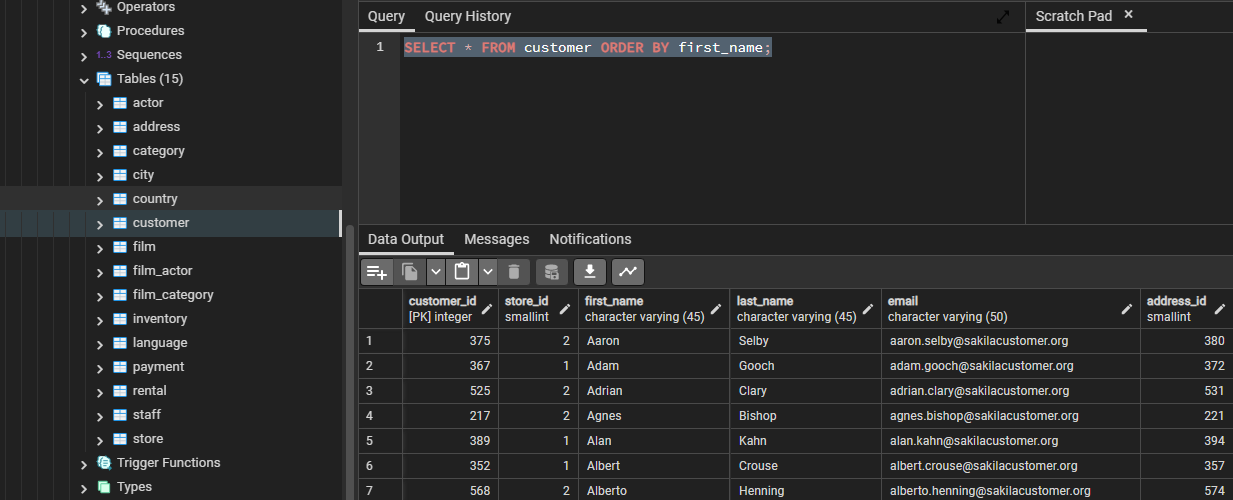


## ORDER BY

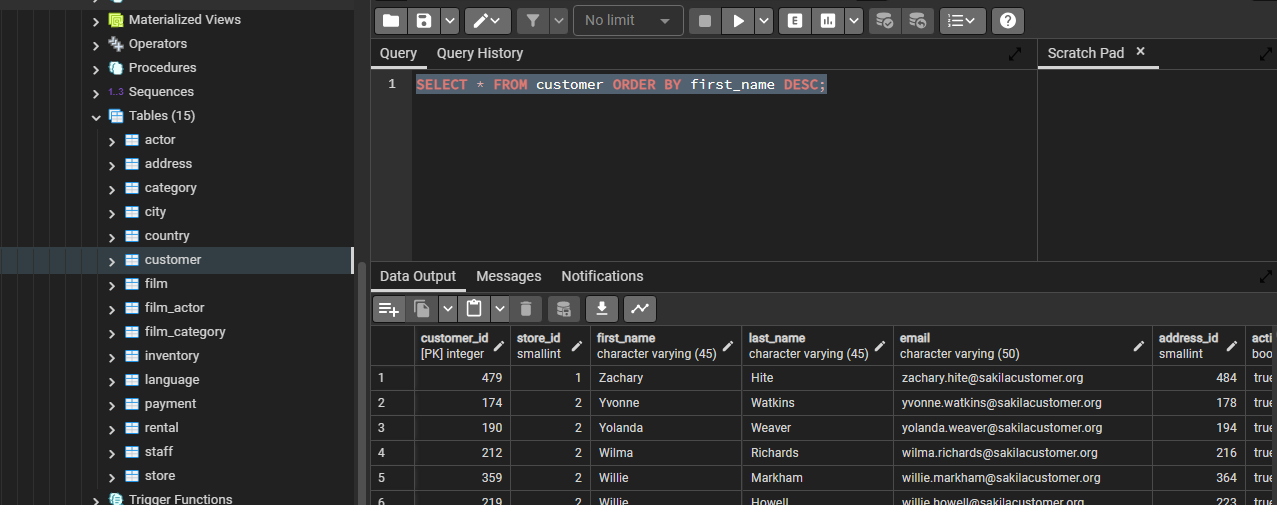
SELECT col1,col2 FROM table ORDER BY col1 ASC/DESC

If we leave it blank, ASC by default

SELECT \* FROM customer ORDER BY first\_name;



SELECT \* FROM customer ORDER BY first\_name DESC;



SELECT STORE\_ID,

FIRST\_NAME,

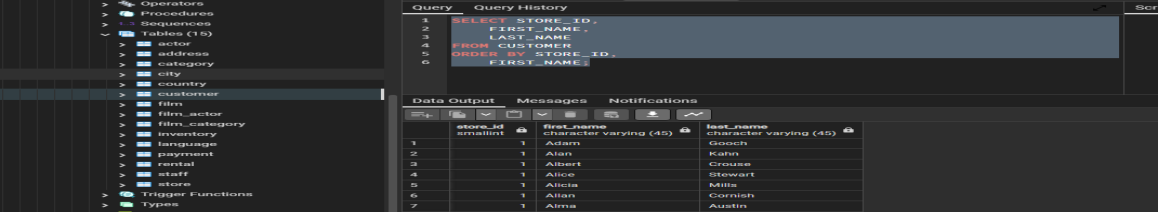
LAST\_NAME

FROM CUSTOMER

ORDER BY STORE\_ID,

FIRST\_NAME;

Here first sort by store\_id, then sort by firstname



SELECT STORE\_ID,

FIRST\_NAME,

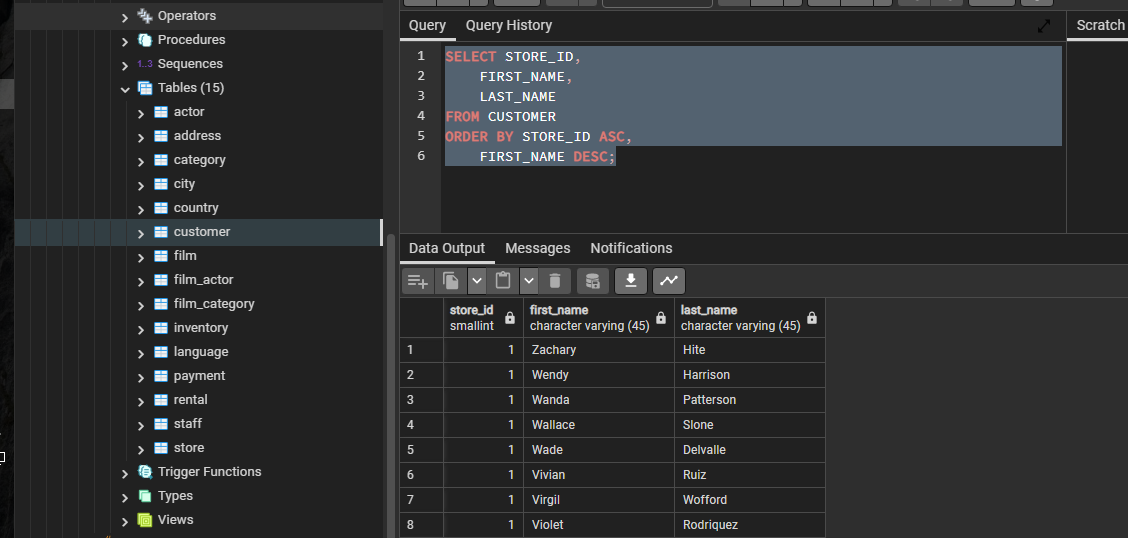
LAST\_NAME

FROM CUSTOMER

ORDER BY STORE\_ID ASC,

FIRST\_NAME DESC;

Sorts first in ascending store\_id and then descending firstname



### LIMIT Keyword

Limit the number of rows

SELECT STORE\_ID,

FIRST\_NAME,

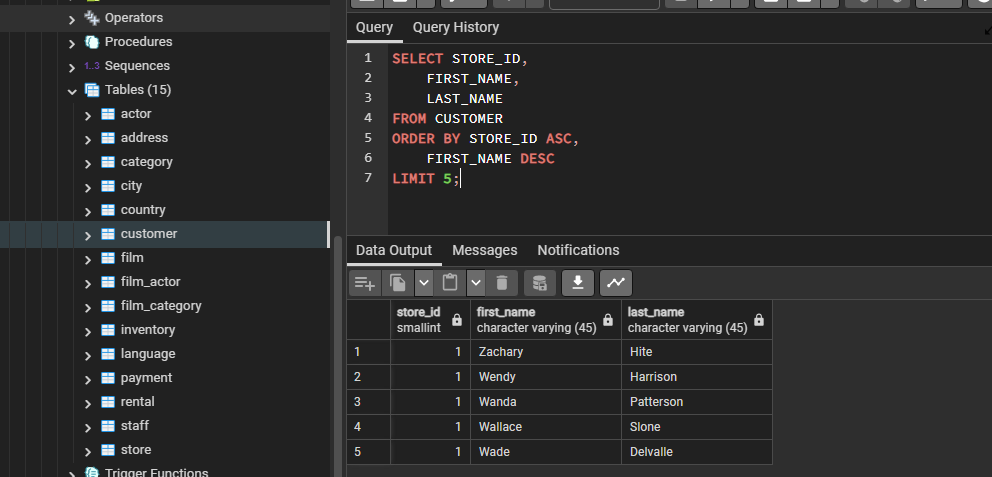
LAST\_NAME

FROM CUSTOMER

ORDER BY STORE\_ID ASC,

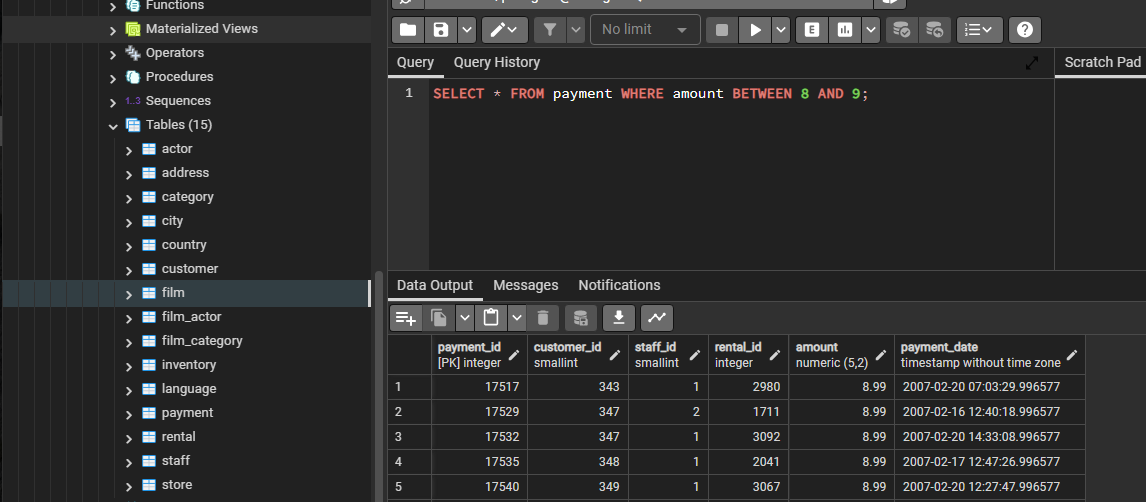
FIRST\_NAME DESC

LIMIT 5;

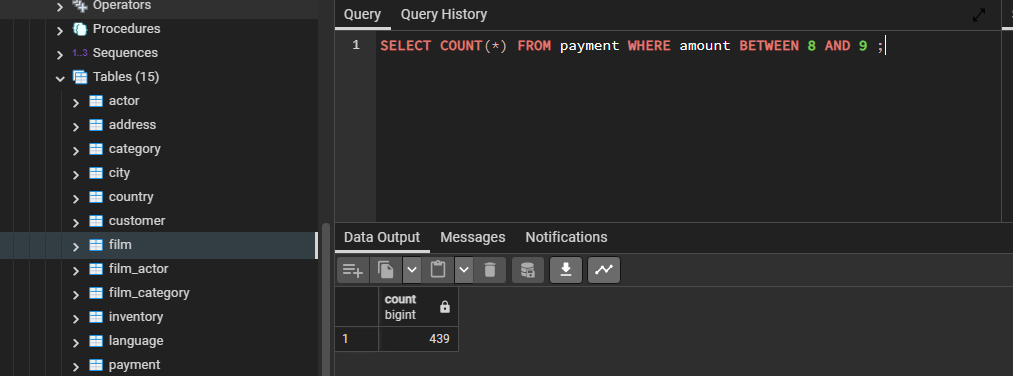


### BETWEEN Keyword

SELECT \* FROM payment WHERE amount BETWEEN 8 AND 9;



SELECT COUNT(\*) FROM payment WHERE amount BETWEEN 8 AND 9 ;



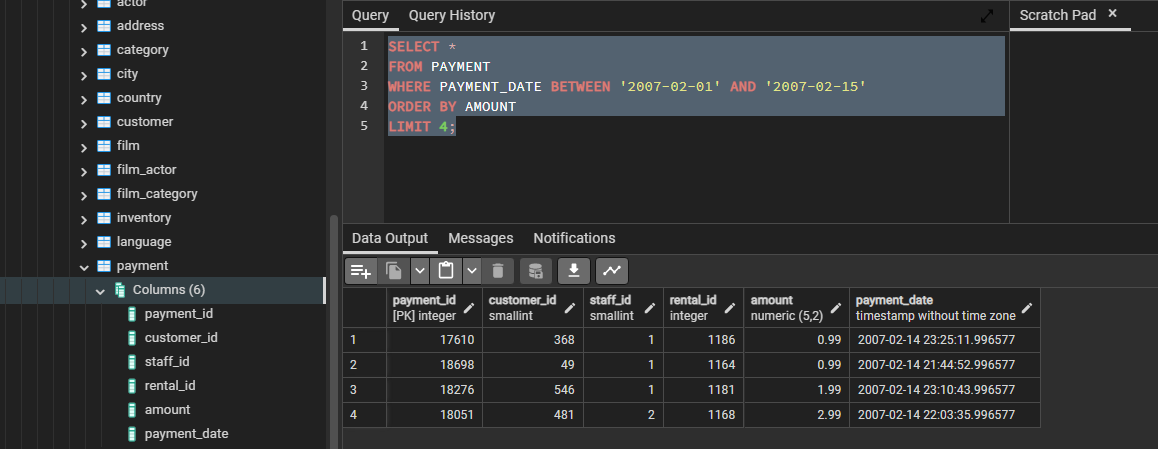
SELECT \*

FROM PAYMENT

WHERE PAYMENT\_DATE BETWEEN '2007-02-01' AND '2007-02-15'

ORDER BY AMOUNT

LIMIT 4;

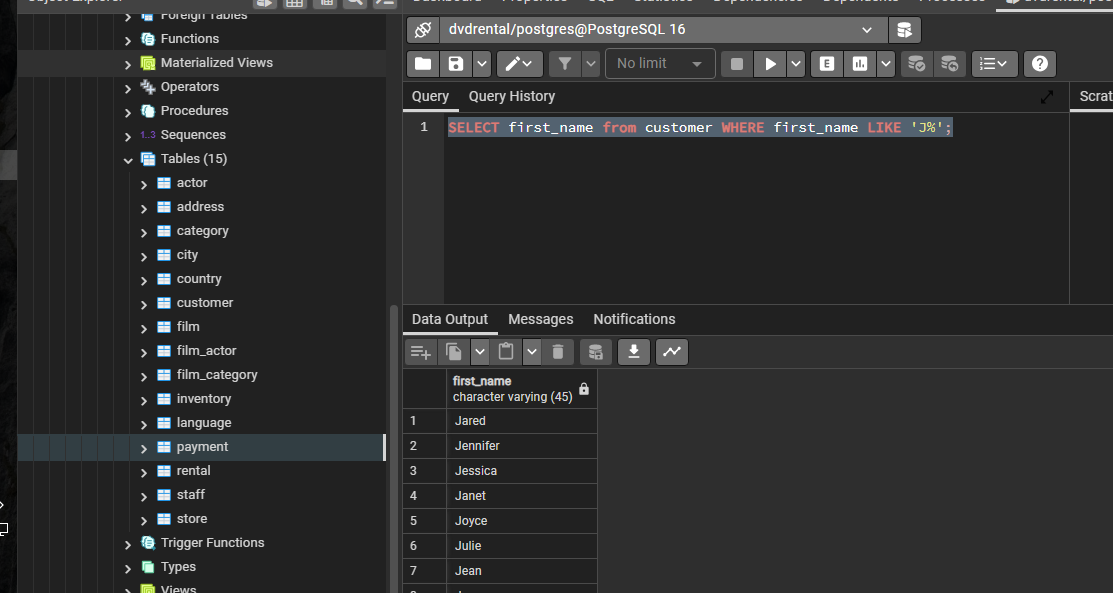


### LIKE and ILIKE Keyword

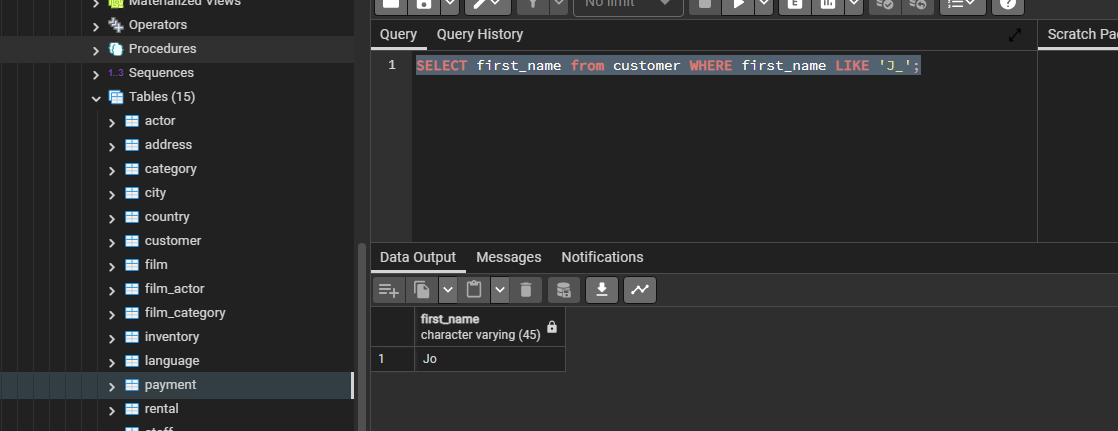
Like allows us to use pattern matching in SQL (% - matches any sequence of characters, ‘\_’ matches any single character)

LIKE is case sensitive & ILIKE is case insensitive

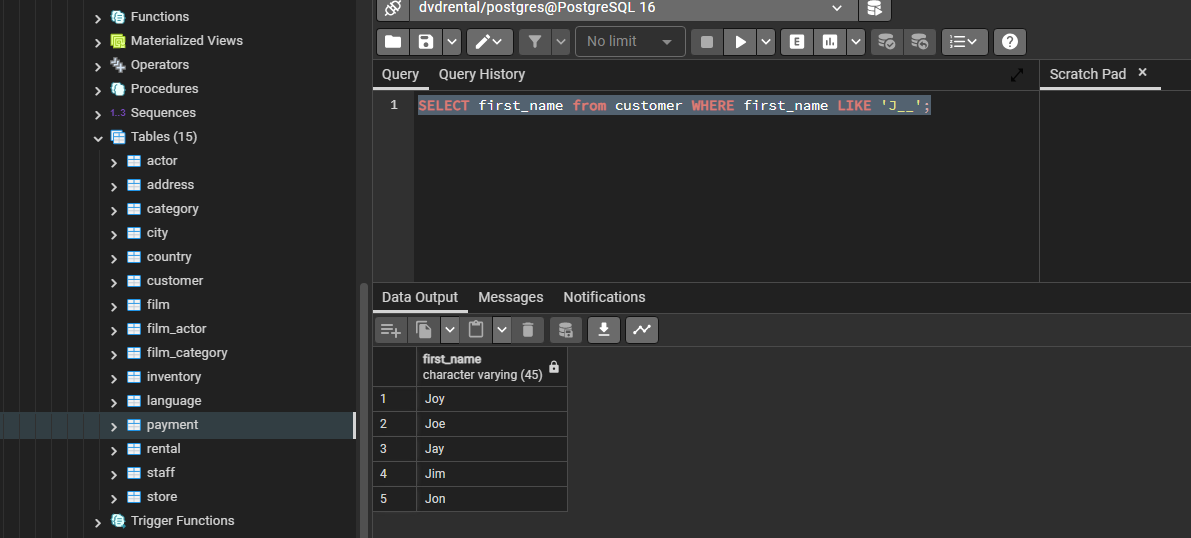
SELECT first\_name from customer WHERE first\_name LIKE 'J%';



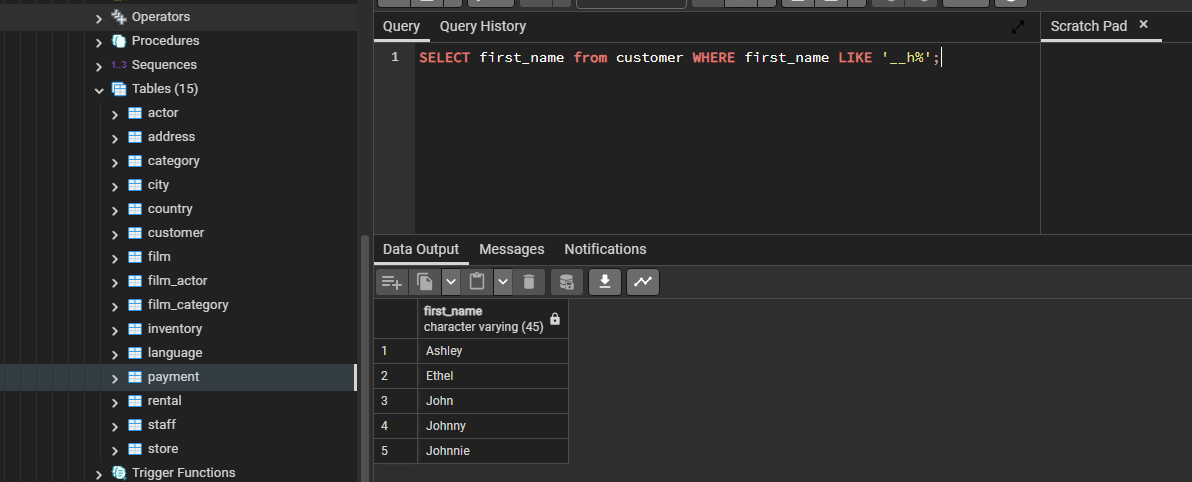
SELECT first\_name from customer WHERE first\_name LIKE 'J\_';



SELECT first\_name from customer WHERE first\_name LIKE 'J\_\_';



SELECT first\_name from customer WHERE first\_name LIKE '\_\_h%';

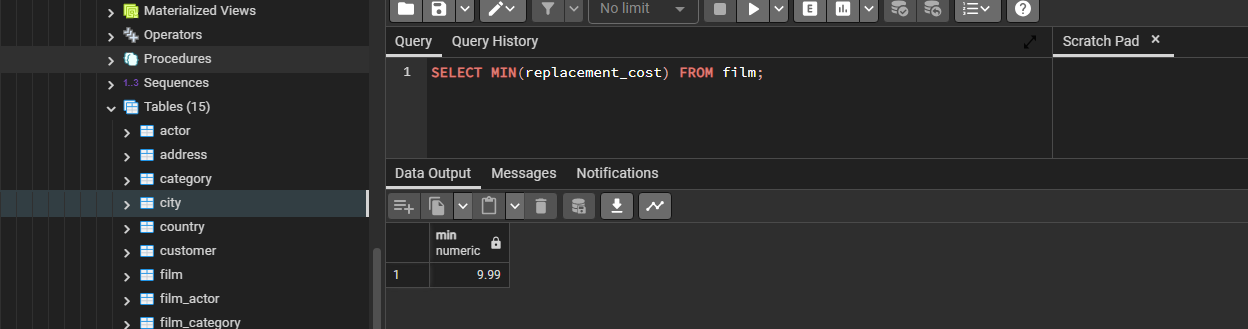


### Aggregate Functions

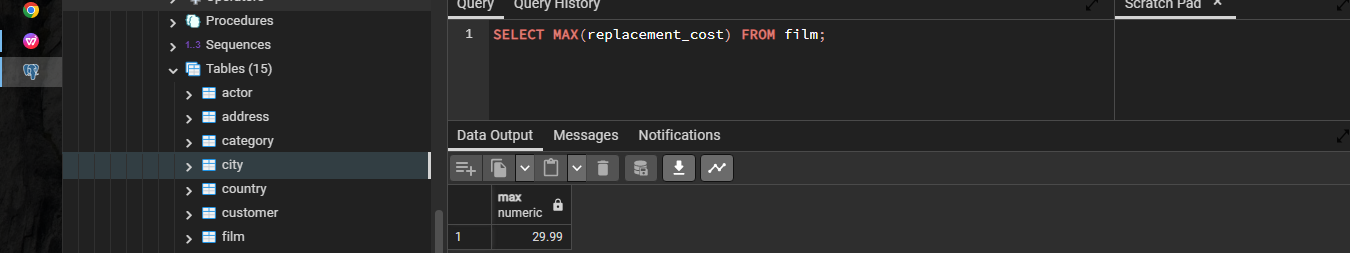
* AVG()
* COUNT()
* MAX()
* DATE()
* MIN()
* SUM()

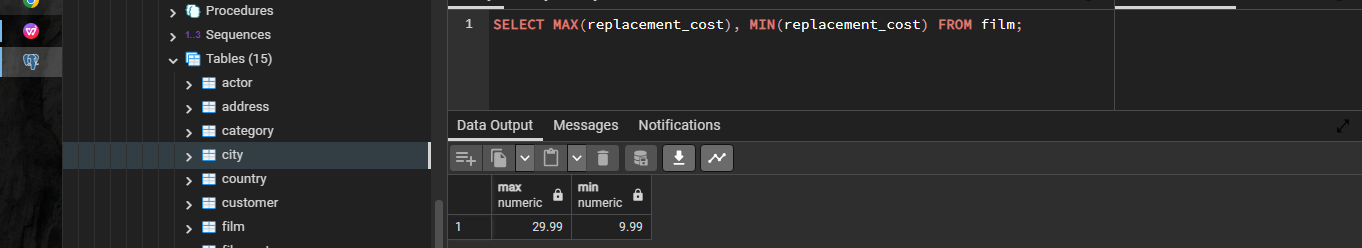
Works only in SELECT and HAVING

MIN()

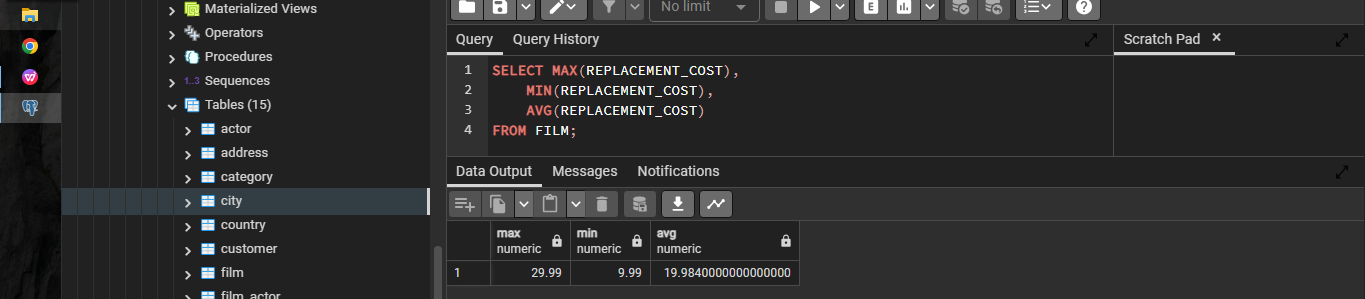


MAX()



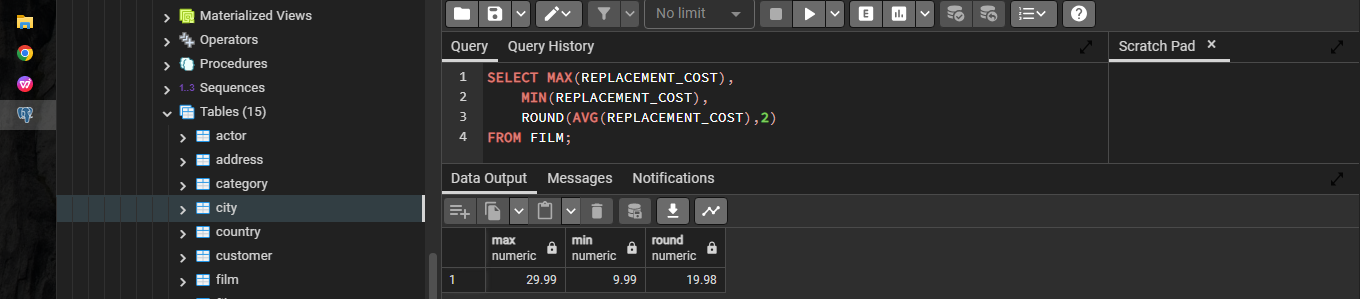


AVG()

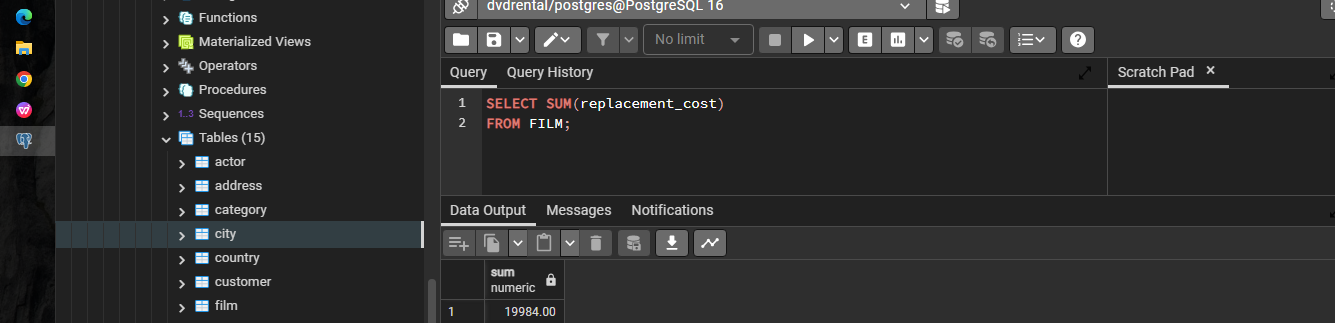


ROUND()

Syntax : ROUND(val, dec places)



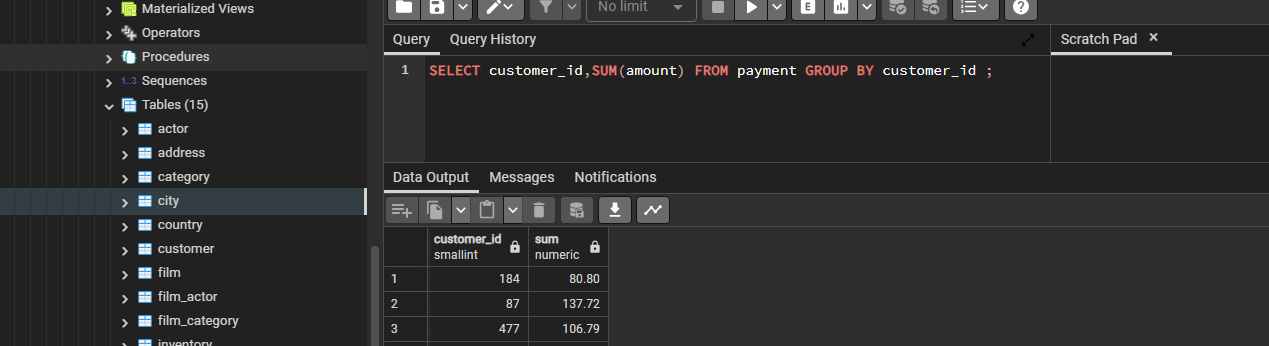
SUM()



### GROUP BY Keyword

Syntax:

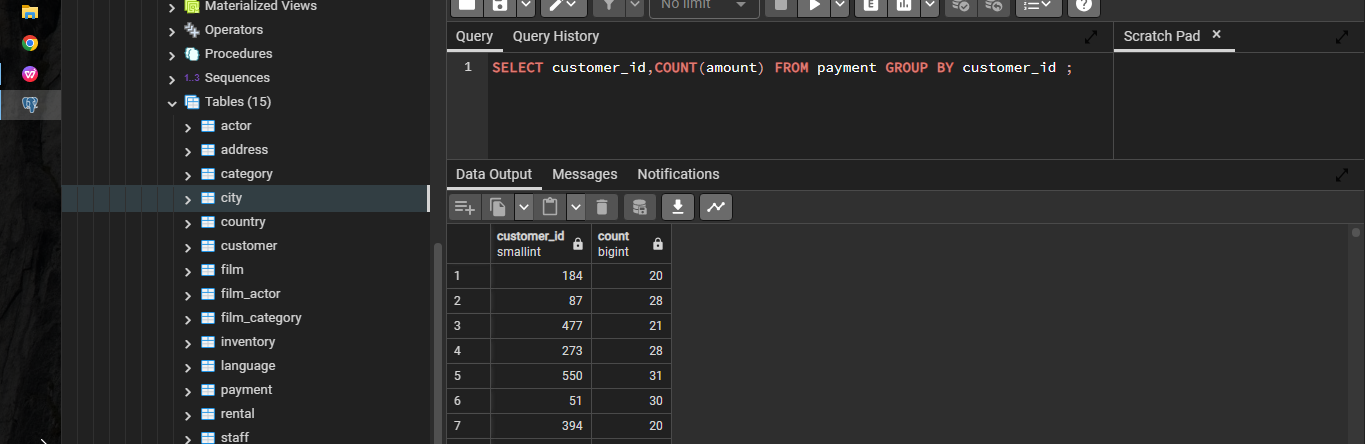
SELECT category\_col, AGG(data\_col) FROM table GROUP BY category\_col



SELECT customer\_id,SUM(amount) FROM payment GROUP BY customer\_id ;

Shows the total amount spend by each customer

Getting the count of transactions of each customer



SELECT CUSTOMER\_ID,

STAFF\_ID,

SUM(AMOUNT)

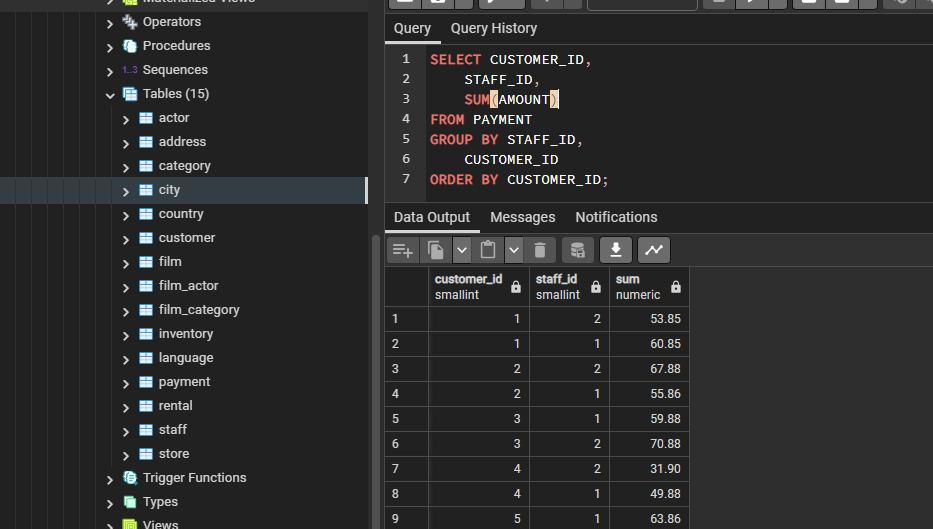
FROM PAYMENT

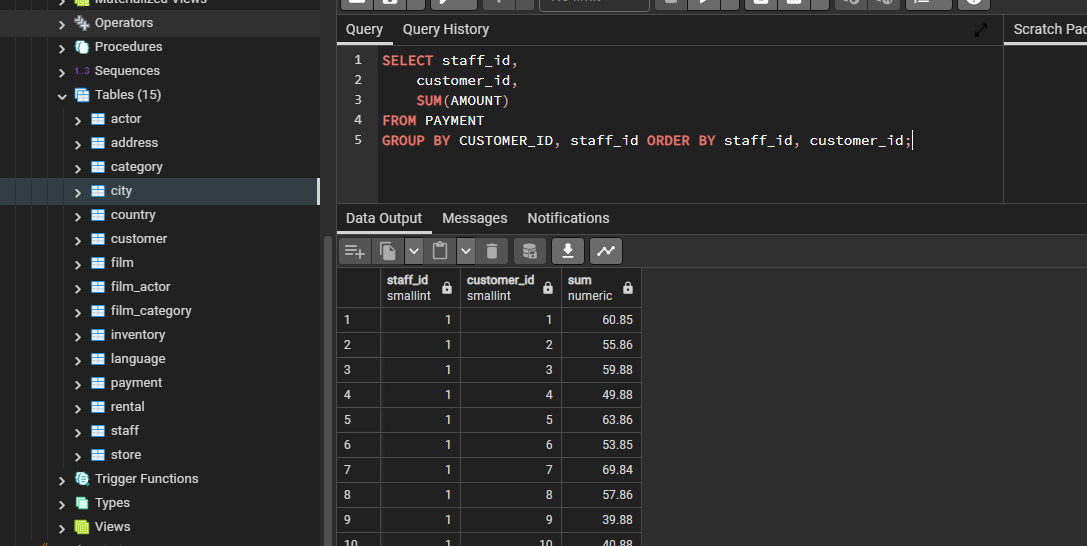
GROUP BY STAFF\_ID,

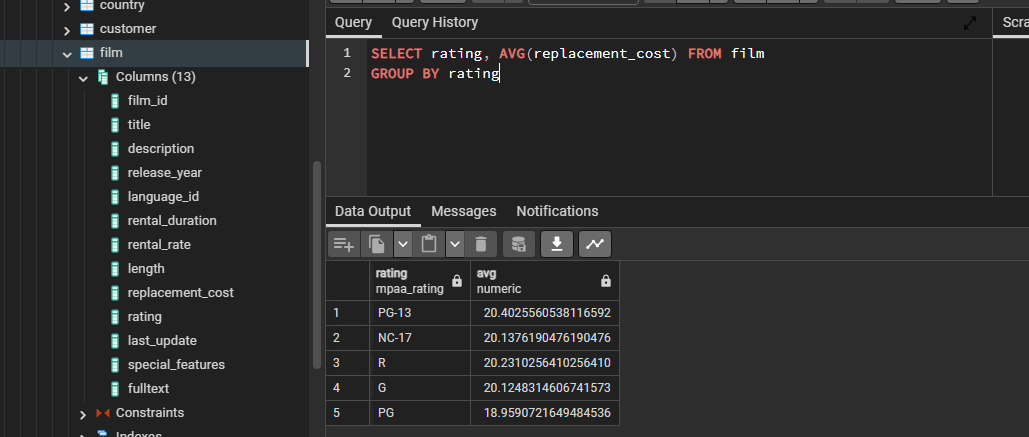
CUSTOMER\_ID

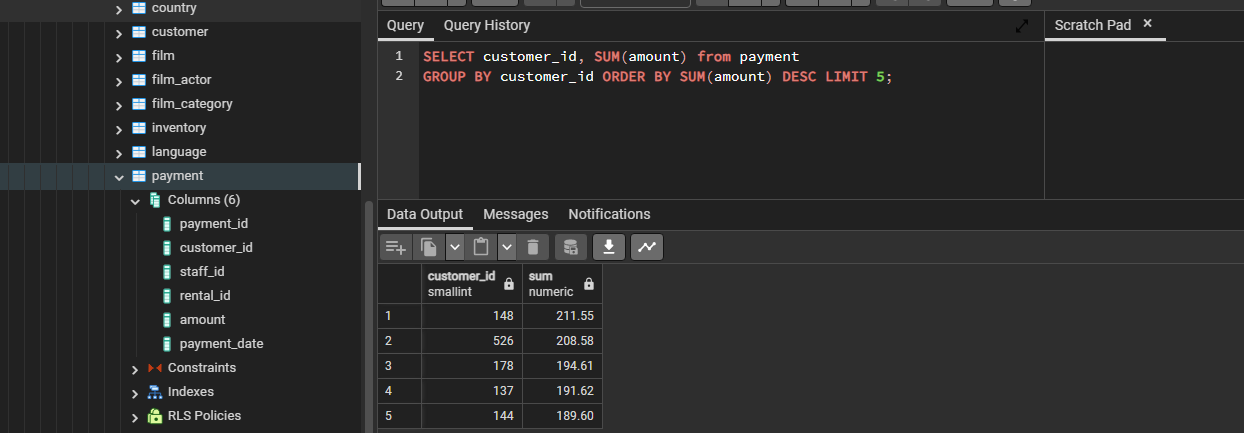
ORDER BY CUSTOMER\_ID;

This shows sales done by each customer with each staff

\







## HAVING Keyword

Allows to filter after an aggregation is over.

SELECT customer\_id,SUM(amount) FROM payment

GROUP BY customer\_id

HAVING SUM(amount) > 100

SELECT store\_id, count(customer\_id) from customer

GROUP BY store\_id

HAVING count(customer\_id) > 300

## **JOINS**

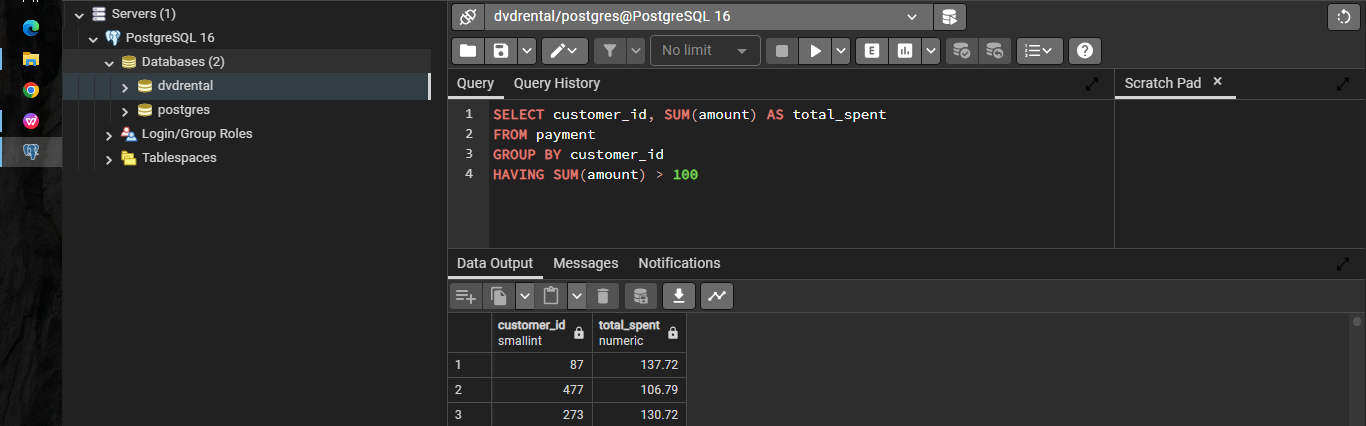
**AS Statement**

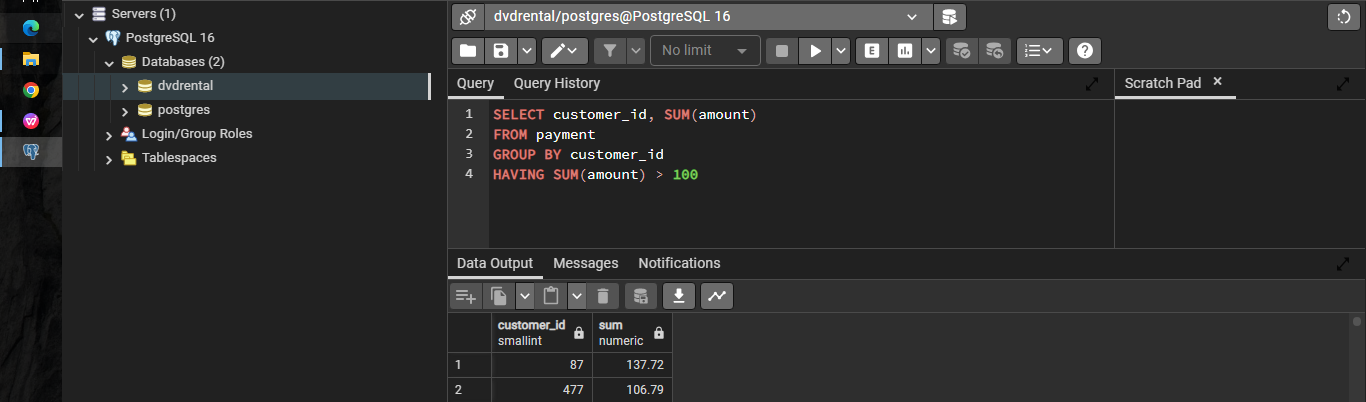
AS is used to give aliases

SELECT SUM(amount) AS net\_revenue FROM payment;

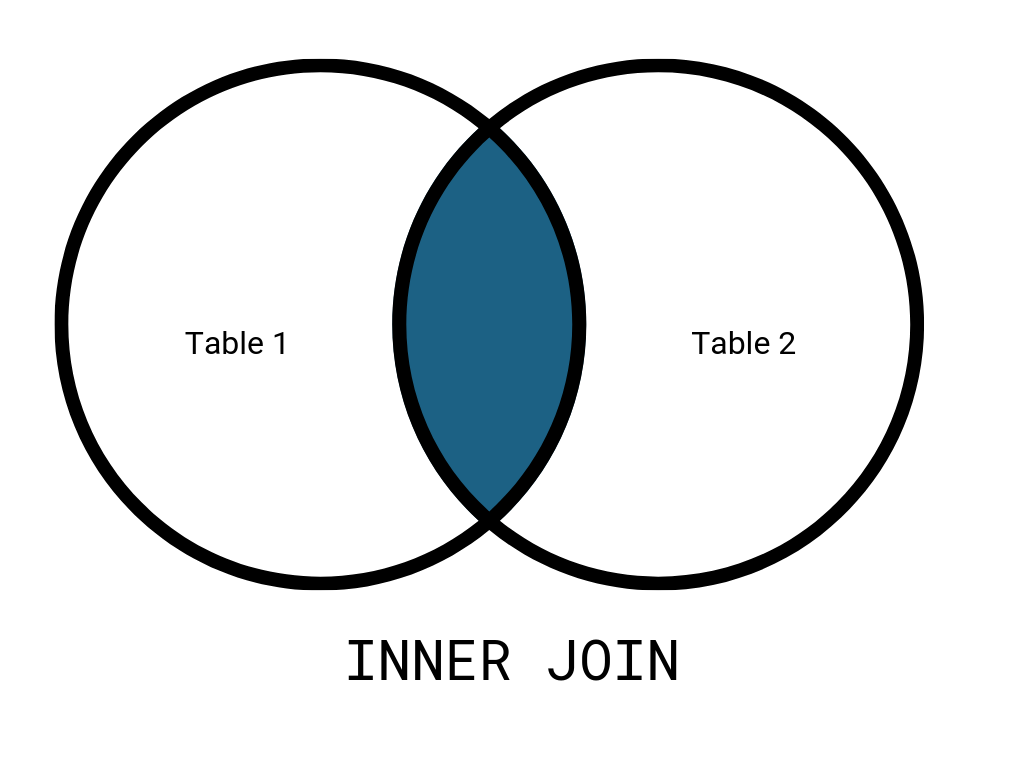
AS gets executed at the end of a query so it cannot be used within WHERE Operator

SELECT COUNT(amount) AS num\_transactions FROM payment;





## INNER JOIN



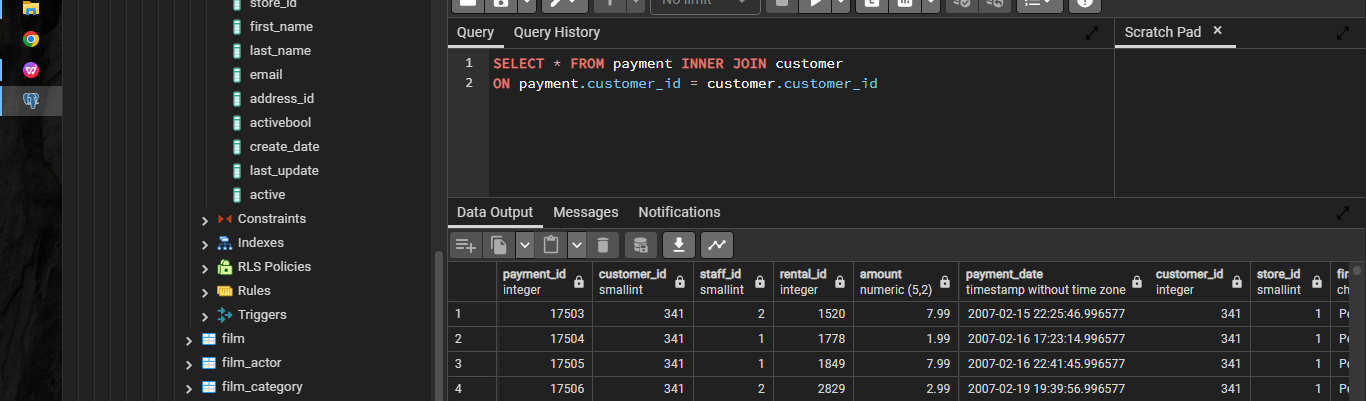
JOINS allow to combine multiple tables

SELECT \* FROM TableA INNER JOIN TableB

ON TableA.col\_match = TableB.col\_match

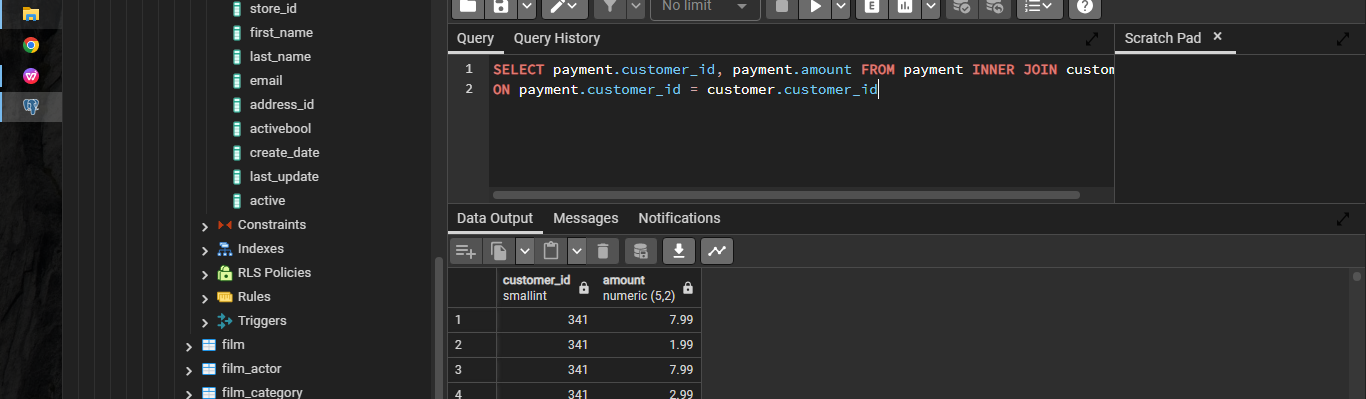
*SELECT \* FROM payment INNER JOIN customer*

*ON payment.customer\_id = customer.customer\_id*



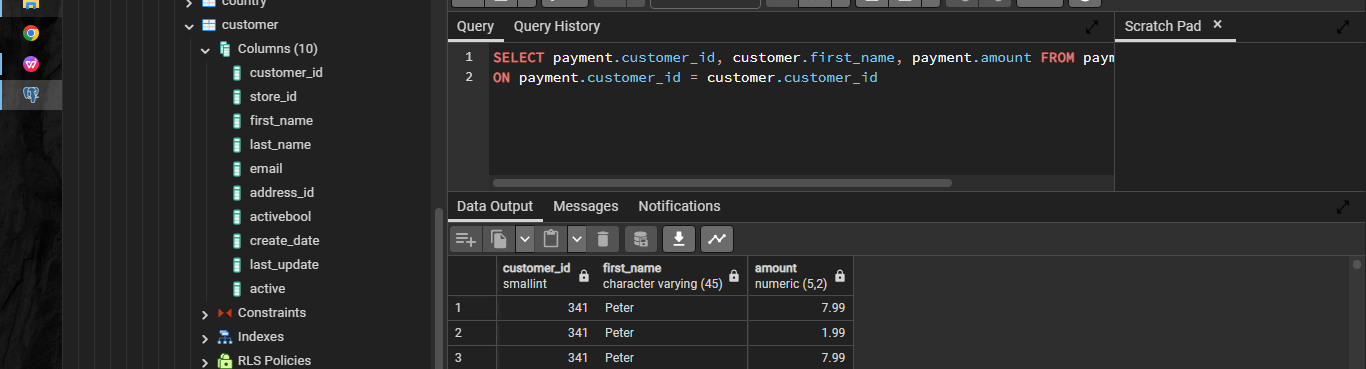
*SELECT payment.customer\_id, payment.amount FROM payment INNER JOIN customer*

*ON payment.customer\_id = customer.customer\_id*



*SELECT payment.customer\_id, customer.first\_name, payment.amount FROM payment INNER JOIN customer*

*ON payment.customer\_id = customer.customer\_id*

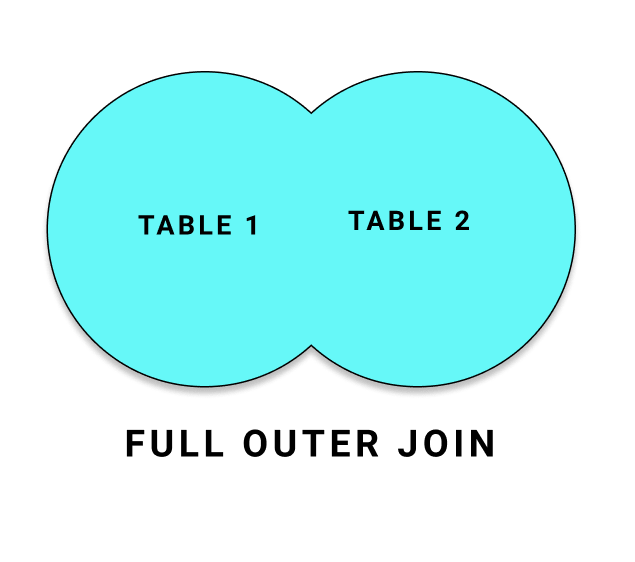


## OUTER JOINS

Opposite of Inner Joins

Three types… FULL OUTER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN

## FULL OUTER Join



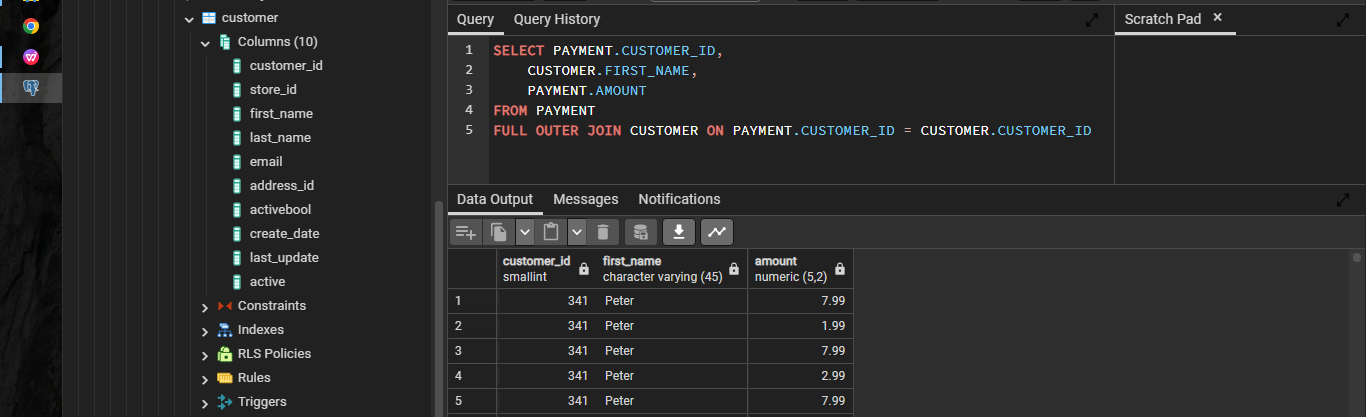
SELECT PAYMENT.CUSTOMER\_ID,

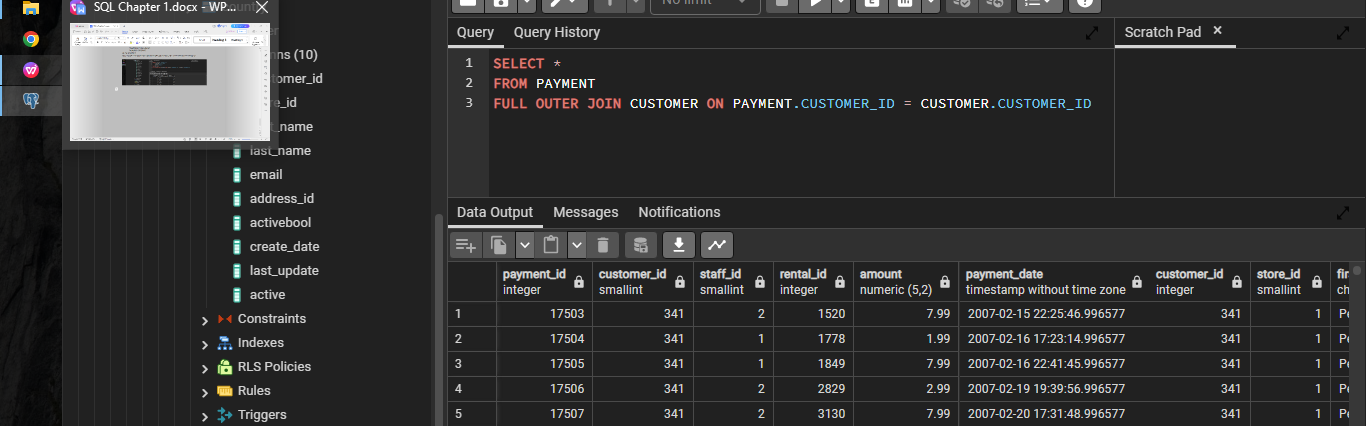
CUSTOMER.FIRST\_NAME,

PAYMENT.AMOUNT

FROM PAYMENT

FULL OUTER JOIN CUSTOMER ON PAYMENT.CUSTOMER\_ID = CUSTOMER.CUSTOMER\_ID





## LEFT OUTER JOIN