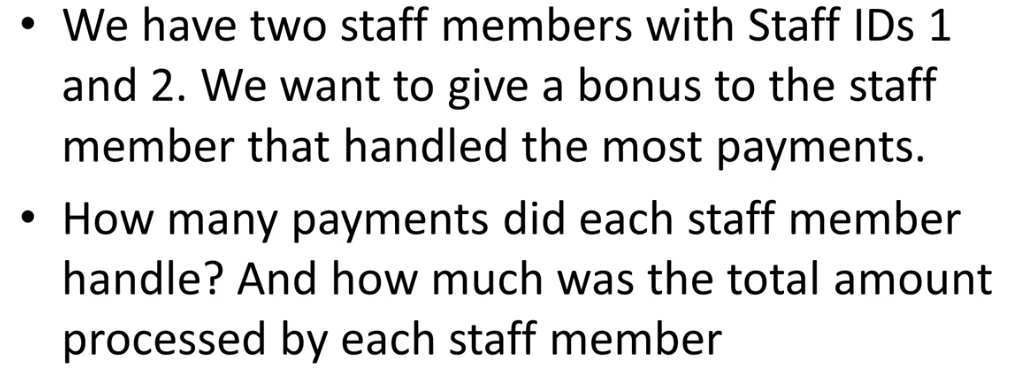
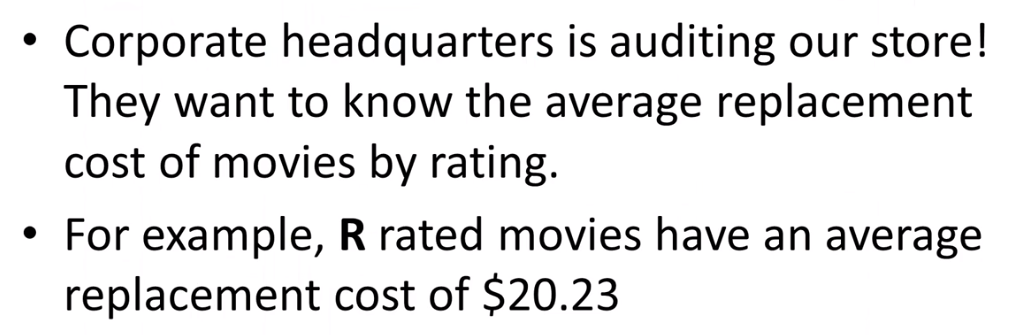


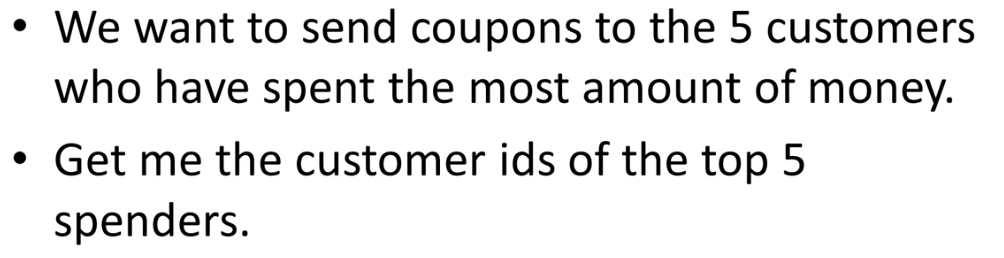
Now suppose you have to find MAX count from above result



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



SELECT rating, ROUND(AVG(replacement\_cost),3) FROM film GROUP BY rating;



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

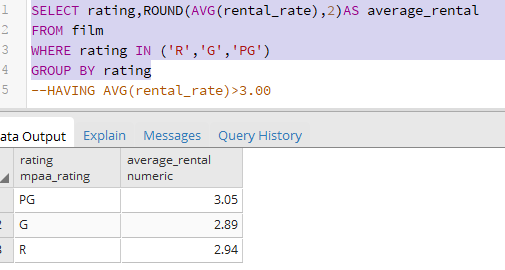
ORDER BY SUM(amount) DESC LIMIT 5;

WHERE and HAVING together

SELECT rating,ROUND(AVG(rental\_rate),2)AS average\_rental

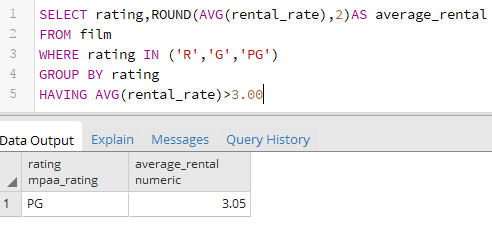
FROM film

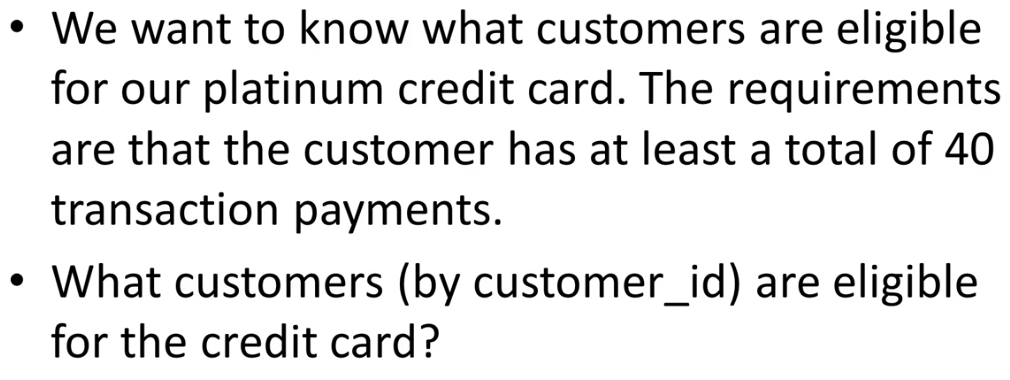
WHERE rating IN ('R','G','PG')

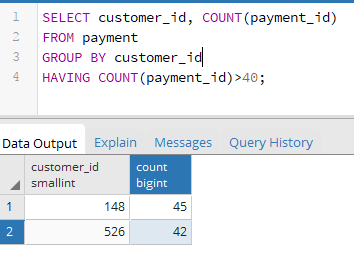


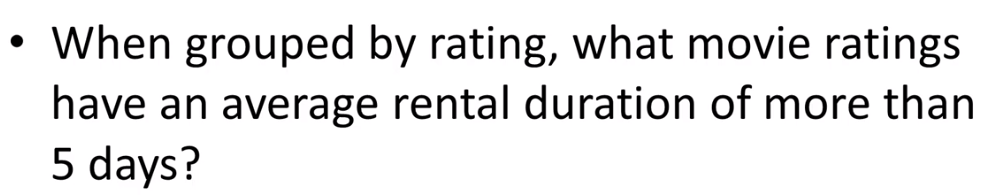
Based on where it has been filtered and we got records in group rows

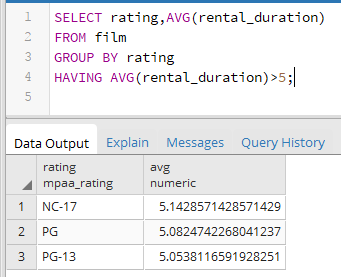
Now to filter group rows we will use HAVING clause

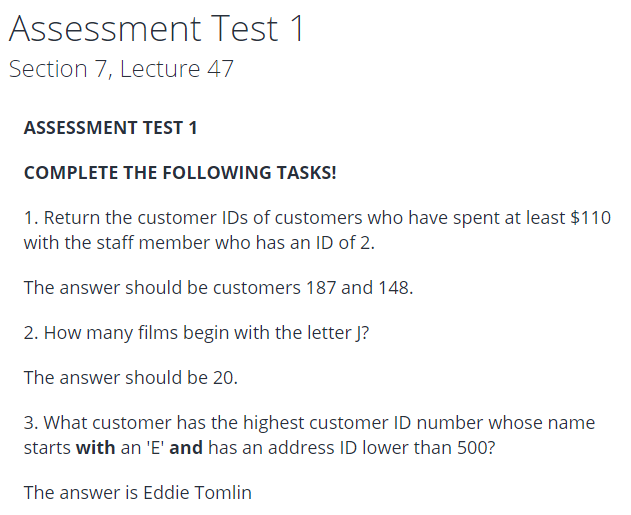


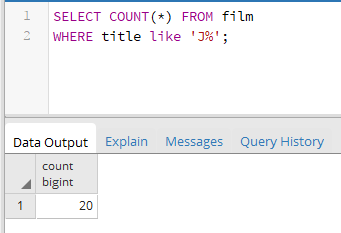


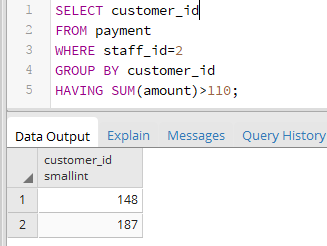


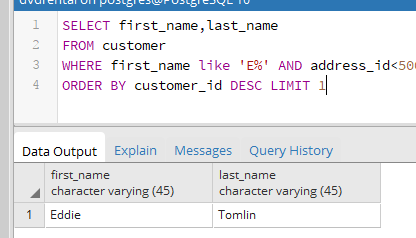








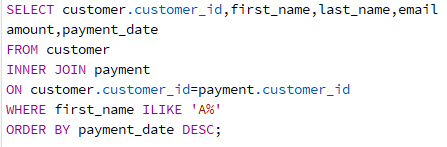




INNER JOIN

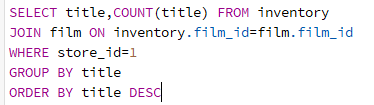


INNER JOIN and WHERE Together

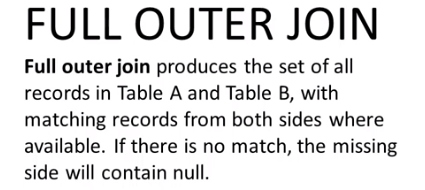


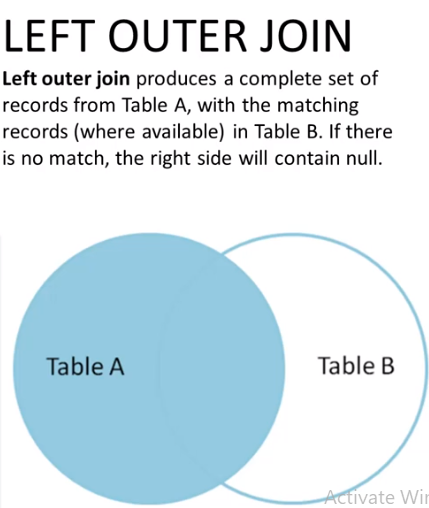
Note : in customer\_id column we have specified table name because it is present in both the tables. If not specified ambiguity exception will be thrown. If column names are different in both the tables just column name is enough. Suppose two same columns are present in both the tables and we want to fetch both the columns then better to use AS clause.

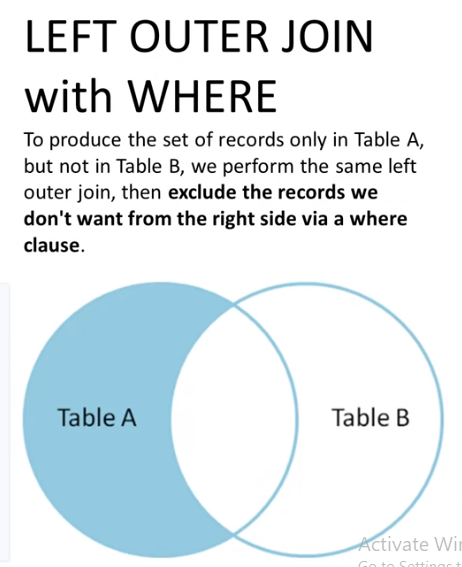
JOIN, WHERE AND GROUP BY Together

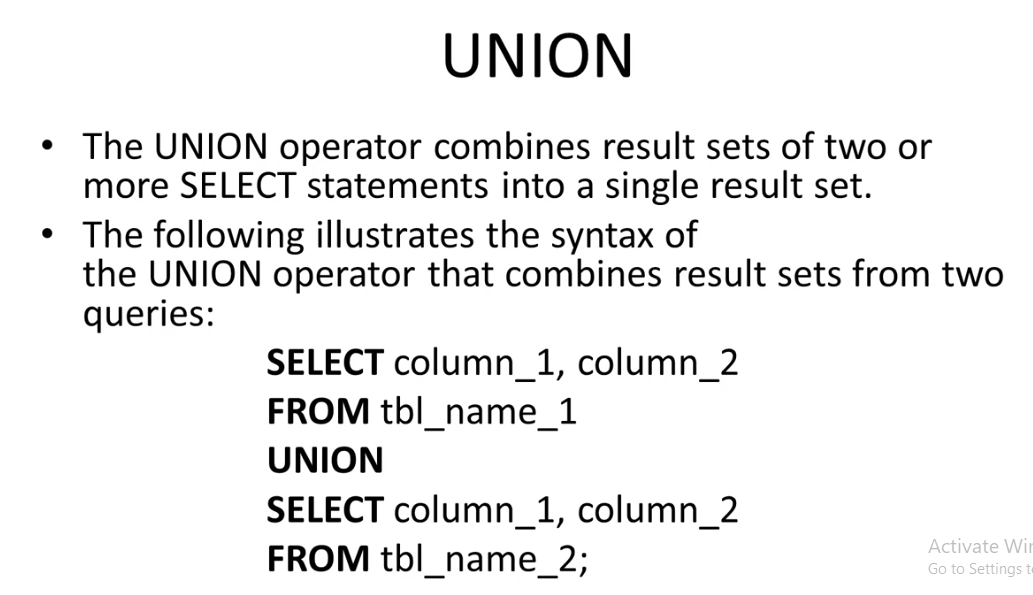


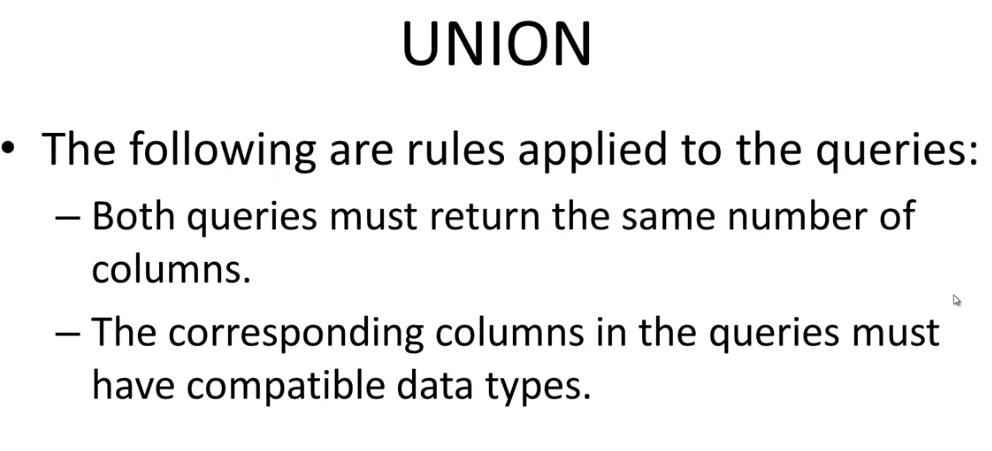
FULL OUTER JOIN

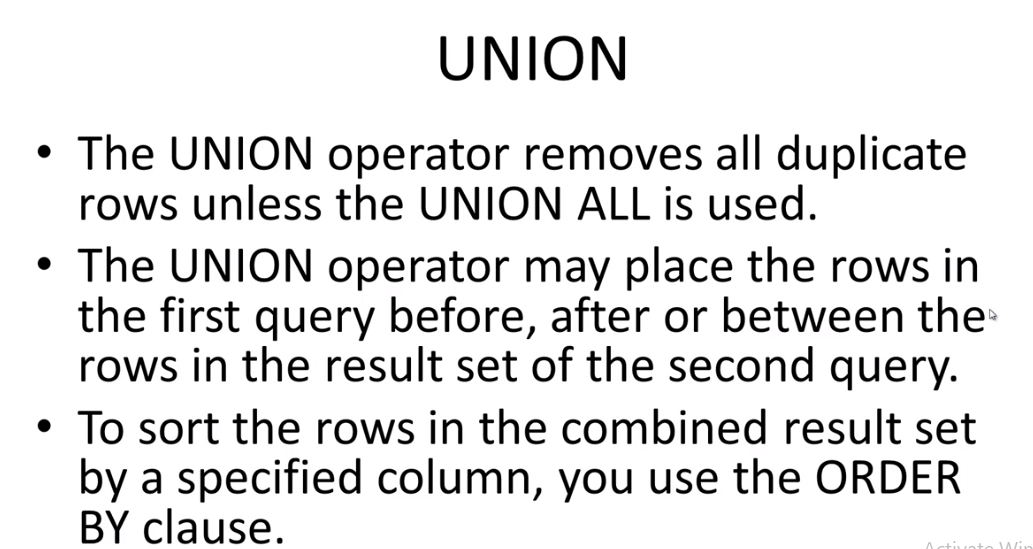


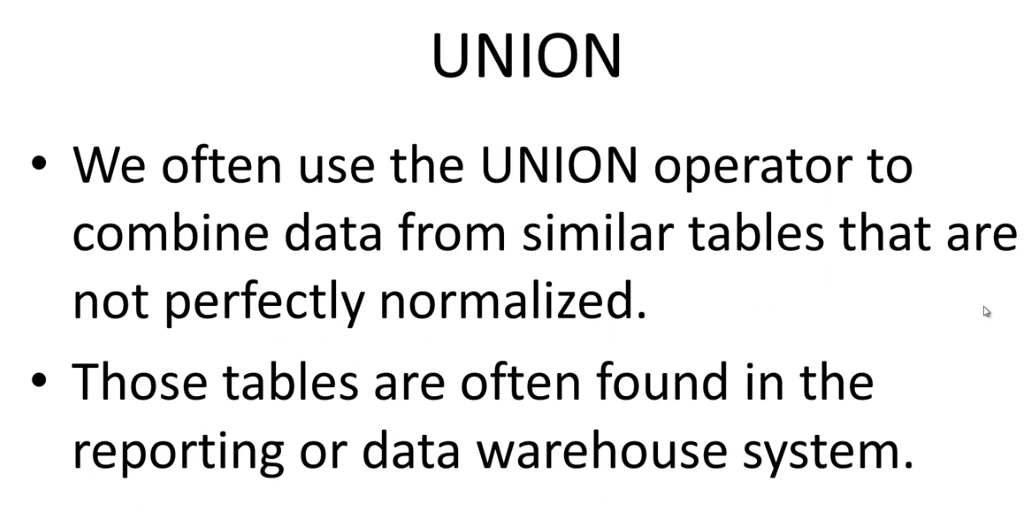






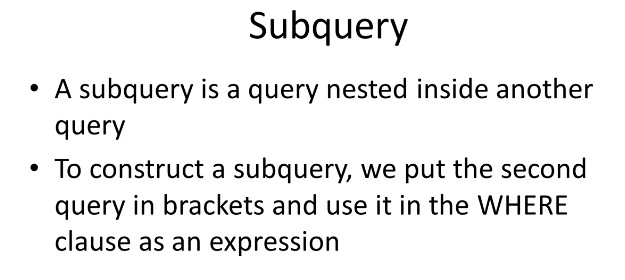


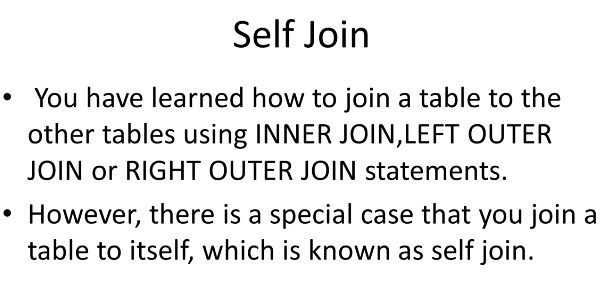


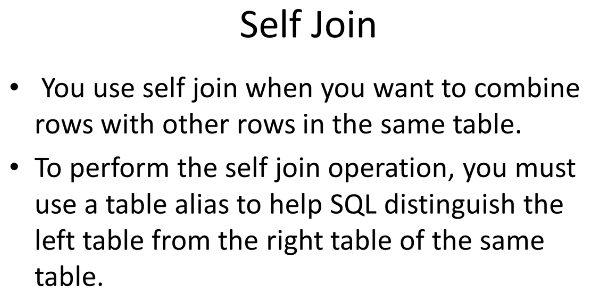


<http://www.sql-join.com/>

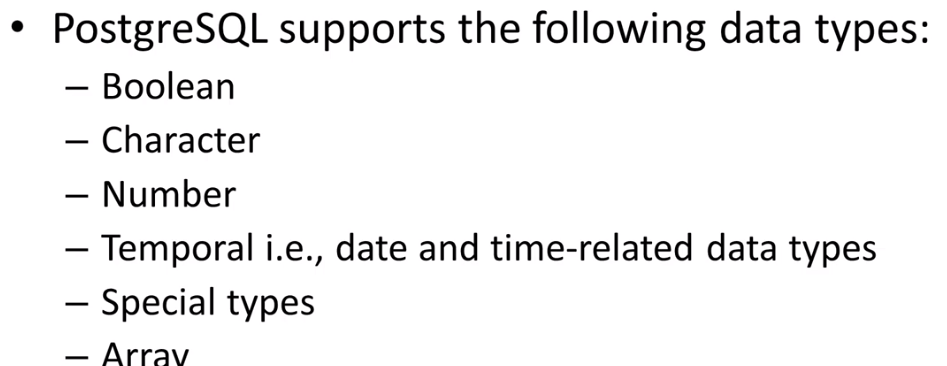
<https://blog.codinghorror.com/a-visual-explanation-of-sql-joins/>



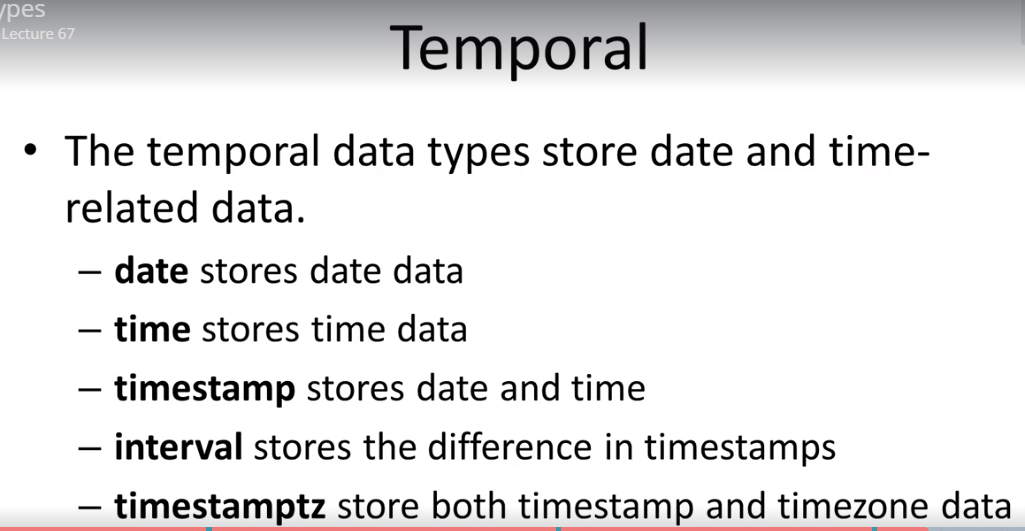


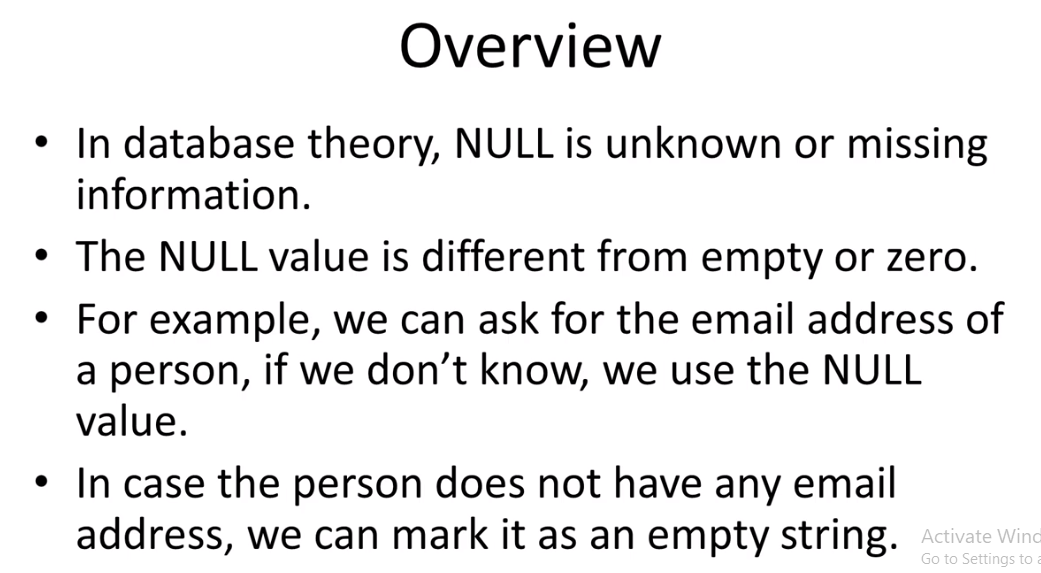


Postgres Data Types



In postgressql serial is same as AUTO\_INCREMENT in other databases.





VIEW

Main use of view is store a Complex query as a view so that we don’t have to re-write.

--CREATING A VIEW--

SELECT first\_name,last\_name,email,address,phone

FROM customer

JOIN address

ON customer.address\_id=address.address\_id;

--Suppose it is a frequent requirement we will create

--a view for the above query and save it as a view

CREATE VIEW customer\_info AS

SELECT first\_name,last\_name,email,address,phone

FROM customer

JOIN address

ON customer.address\_id=address.address\_id;

--Now use this view

SELECT \*FROM customer\_info;

--it is a virtual table it contains no data

--Here we saved this complex query as view

--Dropping

Drop VIEW IF EXISTS customer\_info;

Important Queries-

https://www.postgresql.org/docs/9.1/static/functions-datetime.html

https://www.postgresql.org/docs/9.5/static/functions-math.html

select count(distinct amount) from payment;

-------------------------------------------------------------

select \* from payment

where amount=4.99 limit 10;--returns top 10 rows

-------------------------------------------------------------------------

select first\_name,last\_name

from customer

order by first\_name

DESC;

----------------------------------------------------------------

select first\_name,last\_name

from customer

order by first\_name

ASC,last\_name desc;

It will select first\_name and last\_name from customer table and result will be order by first\_name ascending.

if there is more than one first\_name there last\_name name will be ordered by descending.

--------------------------------------------------------------------------------------------------------------------------------------------------

select first\_name

from customer

order by last\_name;

--we are selecting first\_name and ordering by

--last name. Here order by column is not

--displyaed. it is unique in Postgres

--which is not allowed in other sql database

--default last\_name order by Ascending.

in above query first\_name whoose last\_name starts with a will be selected first.

-------------------------------------------------------------------------------------------

select customer\_id

from payment

order by amount desc limit 10;

----------------------------------------------------------

select count(\*) from payment

where payment\_date between

'2007-02-07' and '2007-02-15';

-------------------------------------------------------

select customer\_id,rental\_id,return\_date

from rental

where customer\_id not in(7,13,10)// here in statement equivalent to customer\_id=7 or customer\_id=13 or customer\_id=10;

order by return\_date desc;

----------------------------------------------------------------

SELECT first\_name,last\_name FROM customer where first\_name like '%er%';

----------------------------------------------------------------------------------------------

SELECT first\_name,last\_name FROM customer where first\_name like '\_her%';--in result \_ will be replaced by any character

-----------------------------------------------------------------------------------------------------------------------------------------------------------

SELECT first\_name,last\_name FROM customer where first\_name not like 'Jen%'

----------------------------------------------------------------------------------------------------------------------------------------------

Challenges Solution

-----------------------------------------------------------------

select count(amount) from payment where amount>5.00;

select count(first\_name) from actor where first\_name like 'P%';

SELECT COUNT(DISTINCT district) FROM address;

SELECT DISTINCT district FROM address;

SELECT COUNT(\*) FROM film WHERE rating='R' AND replacement\_cost BETWEEN 5 AND 15;

---------------------------------------------------------------------------------------------------------------------------------

SELECT ROUND (AVG(amount),3) FROM payment;--round the average to 3 decimal

SELECT ROUND (MIN(amount),3) FROM payment;--round the minimum to 3 decimal

SELECT ROUND (MAX(amount),2) FROM payment;--round the average to 3 decimal

--------------------------------------------------------------------------------------------------------------------------

GROUP BY

---------------------------------------------------------

The GROUP by clause divides the rows returned from the select statement into groups.

for each group we can apply aggregate functions.

SELECT customer\_id FROM payment GROUP BY customer\_id;--gives unique id's

SELECT MAX(amount),customer\_id FROM payment GROUP BY customer\_id order by customer\_id desc;--

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

please note in other sql engines we have to include group by column in select

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

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

SELECT staff\_id,COUNT(\*) FROM payment GROUP BY staff\_id;--number of transactions made by staffs

SELECT rating,COUNT(\*) FROM film GROUP BY rating;

-------------------------------------------------------------------------------------------------

GROUP BY challange

-----------------------------------------------------------------------------------------------

HAVING

-------------------------------------------------------------------

We often use the HAVING clause in conjunction with the group by clause to filter group rows that do not satisfy a specified condition.

Difference between HAVING and WHERE

----------------------------------------------------------------------------------------------

--The HAVING clause sets the condition for group rows created by the GROUP BY clause

--after the GROUP BY clause applies

--While

----the WHERE clause sets the condition for the individual rows before GROUP BY clause applies

SELECT customer\_id, SUM(amount)

FROM payment

GROUP BY customer\_id

HAVING SUM(amount)>200;

------------

SELECT customer\_id, SUM(amount)

FROM payment

GROUP BY customer\_id

HAVING customer\_id<100;

-------

----------WHERE AND HAVING TOGETHER------

-----------------------------------------------------

INNER JOIN

---------------------------------------------------------

SELECT payment\_id,first\_name,last\_name

FROM payment

INNER JOIN staff ON payment.staff\_id=staff.staff\_id;

--------

SELECT film.film\_id,film.title,inventory\_id

FROM film

LEFT OUTER JOIN inventory ON inventory.film\_id=film.film\_id

WHERE inventory\_id IS NULL

ORDER BY film.film\_id

--These films are not in inventory

-------------------------------------------------------------

select customer\_id,extract ( day from payment\_date) from payment;--all dates

select SUM(amount),extract (month from payment\_date) AS month\_wise

FROM payment

GROUP BY month\_wise

ORDER BY SUM(amount) DESC

LIMIT 1;--Highest payment for a month

----------------------------------------------------------

Mathematical functions

select customer\_id+rental\_id as

new\_id from payment; (creating new id from existing value)

round function already covered

String Functoins

------------------------------------------------------

select first\_name, char\_length(first\_name)

from customer;--returns length of first\_name

--SUB QUERY--

--Want to find films whose rental rate is higher than average rental rate

-----

--FIRST WAY---

SELECT round(AVG(rental\_rate),2) from film;

---result is 2.98

SELECT title,rental\_rate from film

where rental\_rate>2.98

------------------------------------

second way using sub query

SELECT title,rental\_rate

FROM film

WHERE rental\_rate>(SELECT AVG(rental\_rate) FROM film);

---------------------------------------------------------

--SUBQUERY WITH JOINING A TABLE---

SELECT film\_id,title FROM film WHERE film\_id IN

(SELECT inventory.film\_id

FROM rental

INNER JOIN inventory ON inventory.inventory\_id=rental.inventory\_id

WHERE

return\_date BETWEEN '2005-05-29' AND '2005-05-30');

------------------

--Here

SELECT inventory.film\_id

FROM rental

INNER JOIN inventory ON inventory.inventory\_id=rental.inventory\_id

WHERE

return\_date BETWEEN '2005-05-29' AND '2005-05-30

--returns multiple values so IN is used in subQuery

-------------------------------------

SELF JOIN

ALL QUERIES HERE GIVES SAME RESULT

--USING SUBQUERY--

SELECT customer\_id,first\_name,last\_name

FROM customer

WHERE first\_name IN

(SELECT last\_name from customer);

-----------------------------------------------

--USING SELF JOIN--

--------------------------------

SELECT a.customer\_id,a.first\_name,a.last\_name,b.customer\_id,b.first\_name,b.last\_name

FROM customer AS a, customer AS b

WHERE a.first\_name=b.last\_name

-----------------------------------

--USING JOIN---

SELECT a.customer\_id,a.first\_name,a.last\_name,b.customer\_id,b.first\_name,b.last\_name

FROM customer AS a JOIN customer AS b

ON a.first\_name=b.last\_name

--------------------------------------------------

CREATE TABLE account(

user\_id serial PRIMARY KEY,

username VARCHAR (50) UNIQUE NOT NULL,

password VARCHAR (50) NOT NULL,

email VARCHAR (355) UNIQUE NOT NULL,

created\_on TIMESTAMP NOT NULL,

last\_login TIMESTAMP

);

-------------------------------------

CREATE TABLE role(

role\_id serial PRIMARY KEY,

role\_name VARCHAR (255) UNIQUE NOT NULL

);

-----------------------------

CREATE TABLE account\_role

(

user\_id integer NOT NULL,

role\_id integer NOT NULL,

grant\_date timestamp without time zone,

PRIMARY KEY (user\_id, role\_id),

CONSTRAINT account\_role\_role\_id\_fkey FOREIGN KEY (role\_id)

REFERENCES role (role\_id) MATCH SIMPLE

ON UPDATE NO ACTION ON DELETE NO ACTION,

CONSTRAINT account\_role\_user\_id\_fkey FOREIGN KEY (user\_id)

REFERENCES account (user\_id) MATCH SIMPLE

ON UPDATE NO ACTION ON DELETE NO ACTION

)

----------------------------------------------