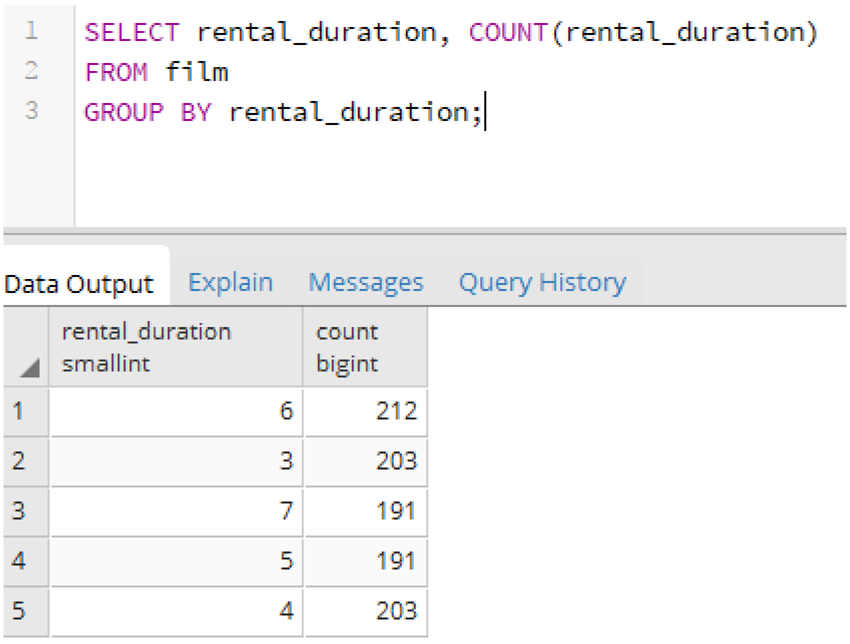


```sql
SELECT rental_duration, COUNT(rental_duration)
FROM film
GROUP BY rental_duration;
```

Data Output | Explain | Messages | Query History

	rental_duration smallint	count bigint
1	6	212
2	3	203
3	7	191
4	5	191
5	4	203

Now suppose you have to find MAX count from above result

- We have two staff members with Staff IDs 1 and 2. We want to give a bonus to the staff member that handled the most payments.
- How many payments did each staff member handle? And how much was the total amount processed by each staff member

SELECT staff_id, COUNT(amount),SUM(amount) FROM payment GROUP BY staff_id;

- Corporate headquarters is auditing our store! They want to know the average replacement cost of movies by rating.
- For example, **R** rated movies have an average replacement cost of $20.23

SELECT rating, ROUND(AVG(replacement_cost),3) FROM film GROUP BY rating;

- We want to send coupons to the 5 customers who have spent the most amount of money.
- Get me the customer ids of the top 5 spenders.

SELECT customer_id, SUM(amount) FROM payment GROUP BY customer_id

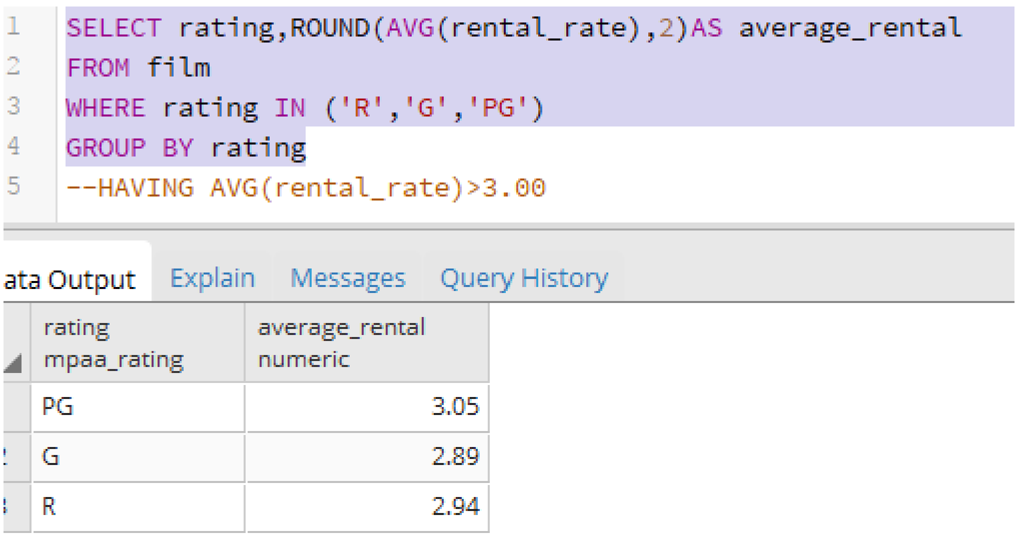
ORDER BY SUM(amount) DESC LIMIT 5;

WHERE and HAVING together

SELECT rating,ROUND(AVG(rental_rate),2)AS average_rental

FROM film

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

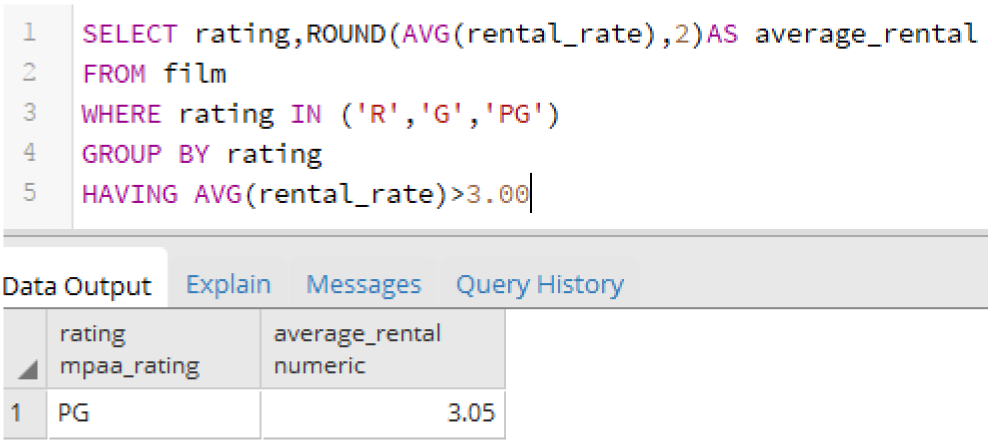


```
SELECT rating,ROUND(AVG(rental_rate),2)AS average_rental
FROM film
WHERE rating IN ('R','G','PG')
GROUP BY rating
--HAVING AVG(rental_rate)>3.00
```

	rating mpaa_rating	average_rental numeric
	PG	3.05
!	G	2.89
;	R	2.94

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

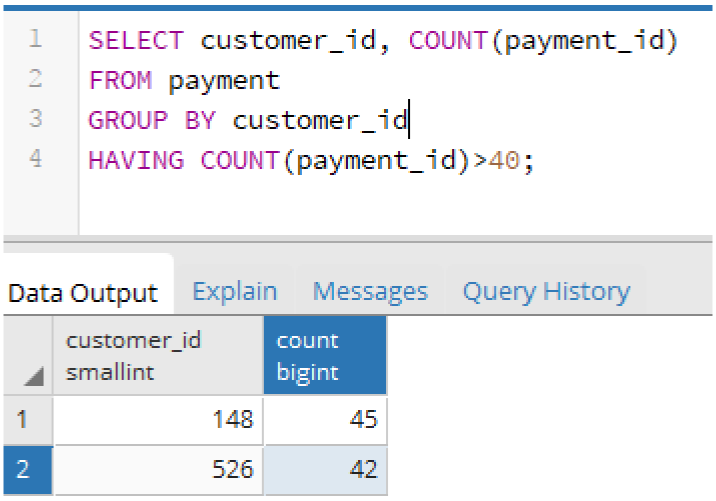
Now to filter group rows we will use HAVING clause



```
SELECT rating,ROUND(AVG(rental_rate),2)AS average_rental
FROM film
WHERE rating IN ('R','G','PG')
GROUP BY rating
HAVING AVG(rental_rate)>3.00
```

	rating mpaa_rating	average_rental numeric
1	PG	3.05

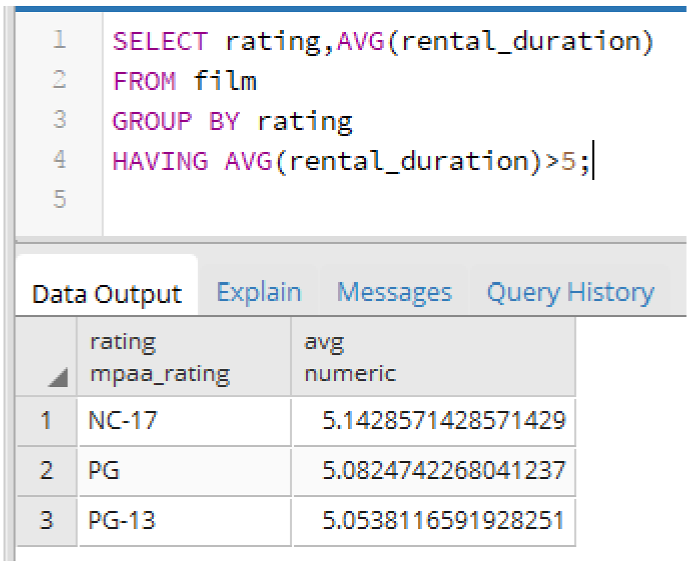
- We want to know what customers are eligible for our platinum credit card. The requirements are that the customer has at least a total of 40 transaction payments.
- What customers (by customer_id) are eligible for the credit card?



```
SELECT customer_id, COUNT(payment_id)
FROM payment
GROUP BY customer_id
HAVING COUNT(payment_id)>40;
```

	customer_id smallint	count bigint
1	148	45
2	526	42

- When grouped by rating, what movie ratings have an average rental duration of more than 5 days?



```
SELECT rating,AVG(rental_duration)
FROM film
GROUP BY rating
HAVING AVG(rental_duration)>5;
```

Data Output | Explain | Messages | Query History

	rating mpaa_rating	avg numeric
1	NC-17	5.1428571428571429
2	PG	5.0824742268041237
3	PG-13	5.0538116591928251

Assessment Test 1

Section 7, Lecture 47

ASSESSMENT TEST 1

COMPLETE THE FOLLOWING TASKS!

1. Return the customer IDs of customers who have spent at least $110 with the staff member who has an ID of 2.

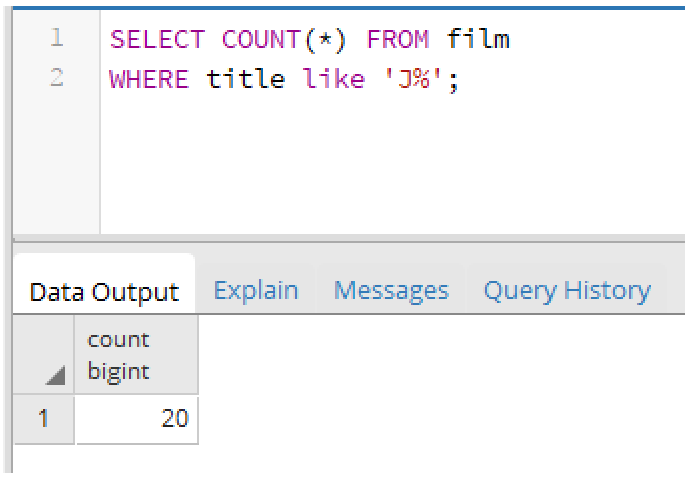
The answer should be customers 187 and 148.

2. How many films begin with the letter J?

The answer should be 20.

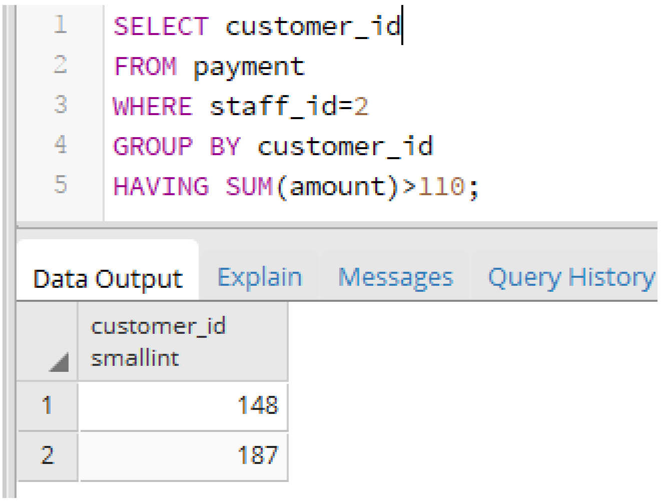
3. What customer has the highest customer ID number whose name starts **with** an 'E' **and** has an address ID lower than 500?

The answer is Eddie Tomlin



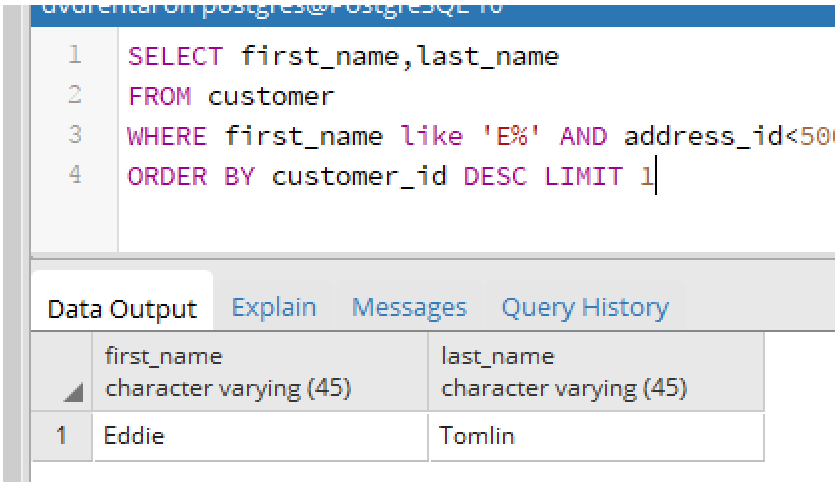
```sql
SELECT COUNT(*) FROM film
WHERE title like 'J%';
```

```
   count
   bigint
1      20
```



```sql
SELECT customer_id
FROM payment
WHERE staff_id=2
GROUP BY customer_id
HAVING SUM(amount)>110;
```

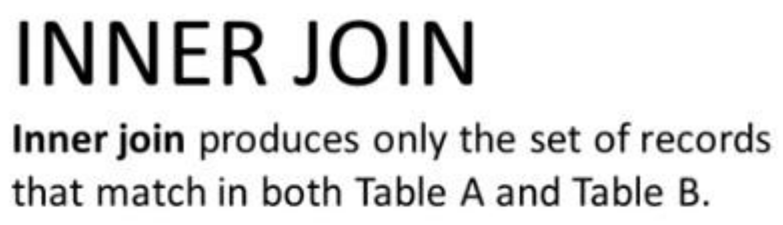
```
   customer_id
   smallint
1          148
2          187
```



```sql
SELECT first_name,last_name
FROM customer
WHERE first_name like 'E%' AND address_id<50
ORDER BY customer_id DESC LIMIT 1
```

```
   first_name               last_name
   character varying (45)   character varying (45)
1  Eddie                    Tomlin
```

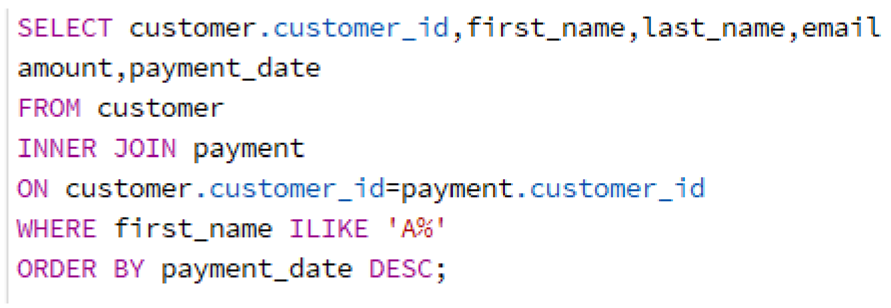
INNER JOIN



INNER JOIN

Inner join produces only the set of records that match in both Table A and Table B.

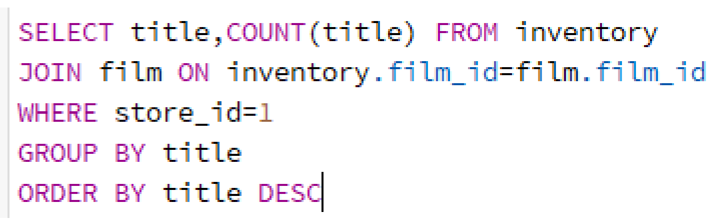
INNER JOIN and WHERE Together



```
SELECT customer.customer_id,first_name,last_name,email
amount,payment_date
FROM customer
INNER JOIN payment
ON customer.customer_id=payment.customer_id
WHERE first_name ILIKE 'A%'
ORDER BY payment_date DESC;
```

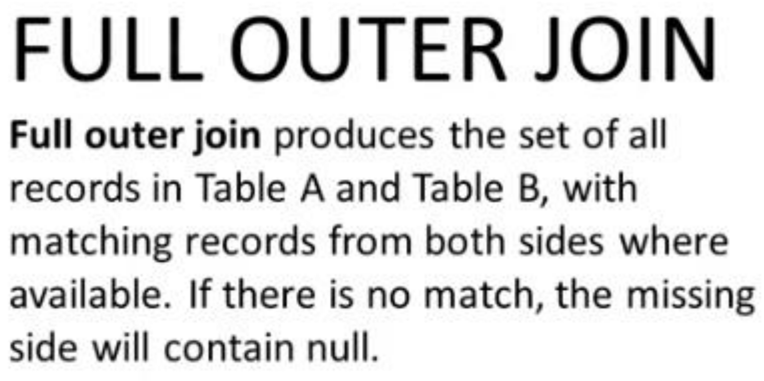
Note : in customer_id column we have specified table name because it is present in both the tables. If not specified ambiguity exception will be thrown. If column names are different in both the tables just column name is enough. Suppose two same columns are present in both the tables and we want to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

JOIN, WHERE AND GROUP BY Together



```
SELECT title,COUNT(title) FROM inventory
JOIN film ON inventory.film_id=film.film_id
WHERE store_id=1
GROUP BY title
ORDER BY title DESC
```

FULL OUTER JOIN

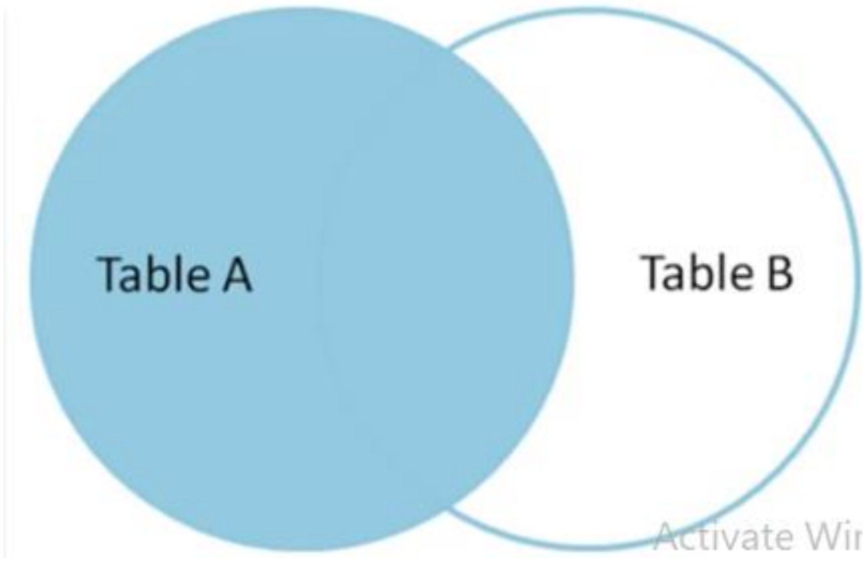


FULL OUTER JOIN

Full outer join produces the set of all records in Table A and Table B, with matching records from both sides where available. If there is no match, the missing side will contain null.

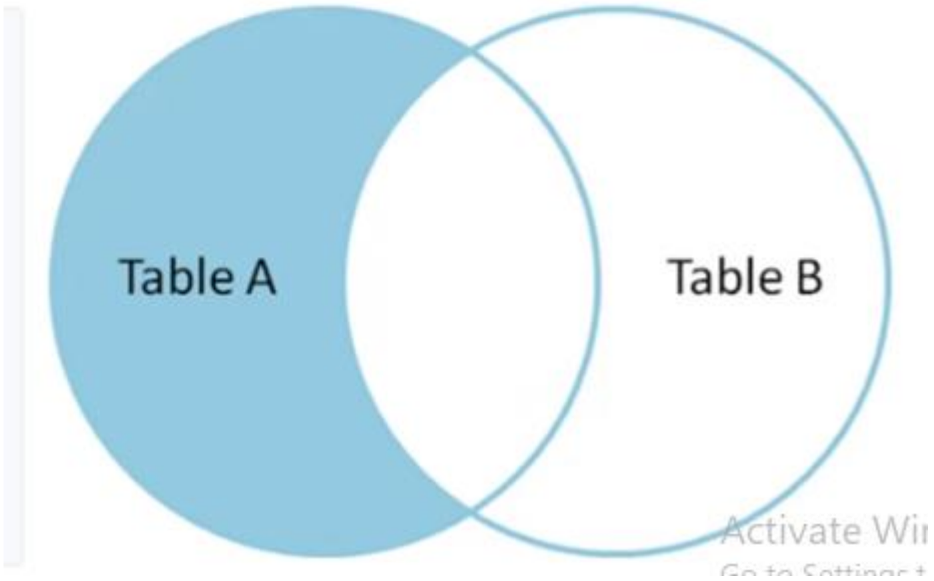
LEFT OUTER JOIN

Left outer join produces a complete set of records from Table A, with the matching records (where available) in Table B. If there is no match, the right side will contain null.



LEFT OUTER JOIN with WHERE

To produce the set of records only in Table A, but not in Table B, we perform the same left outer join, then **exclude the records we don't want from the right side via a where clause**.



UNION

- The UNION operator combines result sets of two or more SELECT statements into a single result set.
- The following illustrates the syntax of the UNION operator that combines result sets from two queries:

```
SELECT column_1, column_2
FROM tbl_name_1
UNION
SELECT column_1, column_2
FROM tbl_name_2;
```

UNION

- The following are rules applied to the queries:
  - Both queries must return the same number of columns.
  - The corresponding columns in the queries must have compatible data types.

UNION

- The UNION operator removes all duplicate rows unless the UNION ALL is used.
- The UNION operator may place the rows in the first query before, after or between the rows in the result set of the second query.
- To sort the rows in the combined result set by a specified column, you use the ORDER BY clause.

UNION

- We often use the UNION operator to combine data from similar tables that are not perfectly normalized.
- Those tables are often found in the reporting or data warehouse system.

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

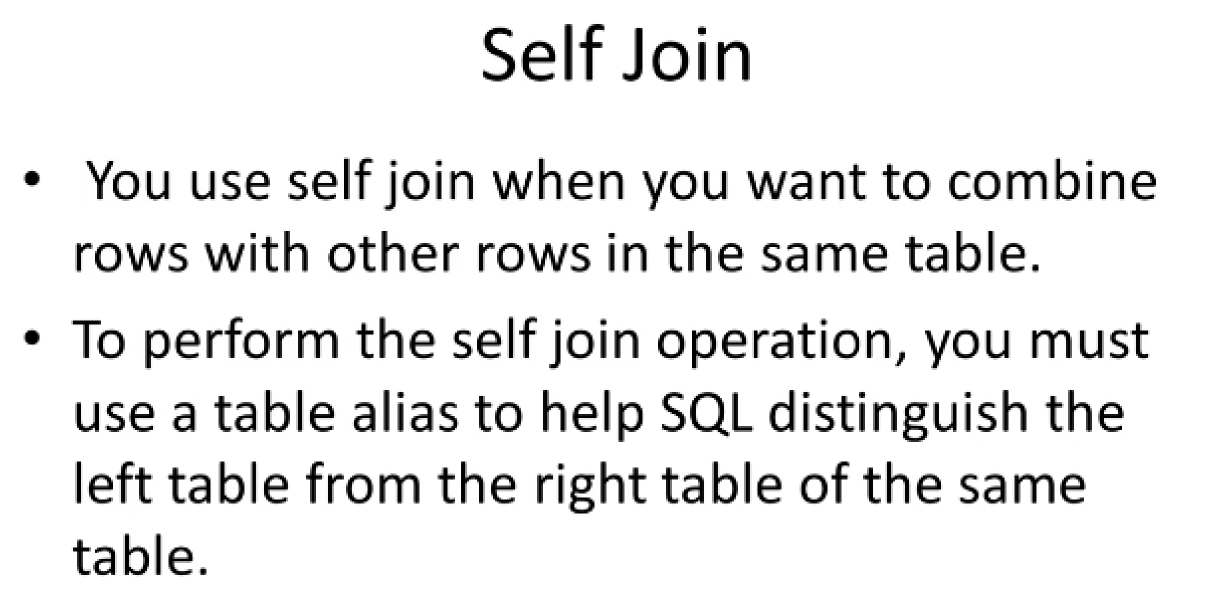
https://blog.codinghorror.com/a-visual-explanation-of-sql-joins/

Subquery

- A subquery is a query nested inside another query
- To construct a subquery, we put the second query in brackets and use it in the WHERE clause as an expression

Self Join

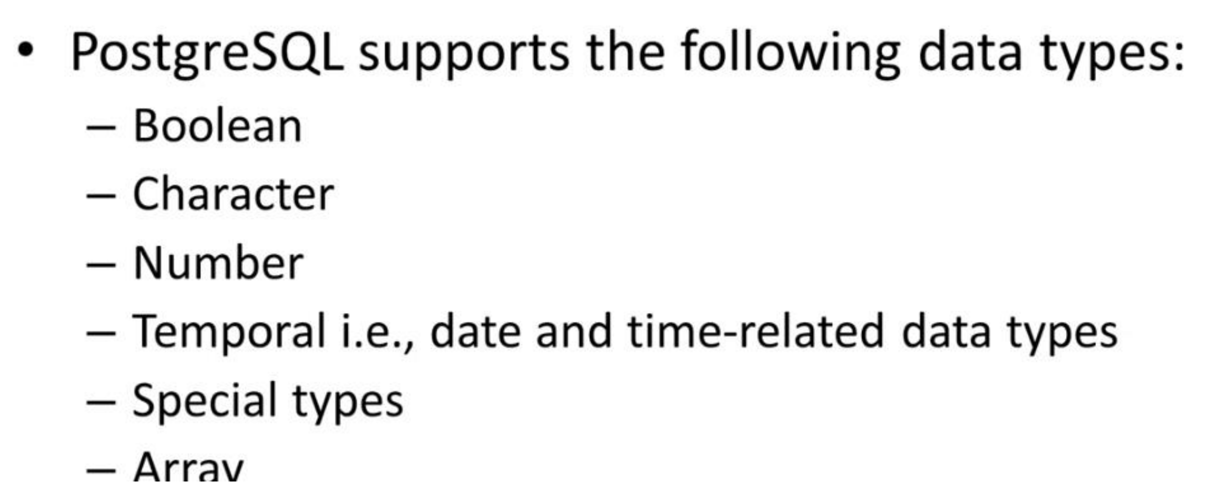
- You have learned how to join a table to the other tables using INNER JOIN,LEFT OUTER JOIN or RIGHT OUTER JOIN statements.
- However, there is a special case that you join a table to itself, which is known as self join.



Self Join

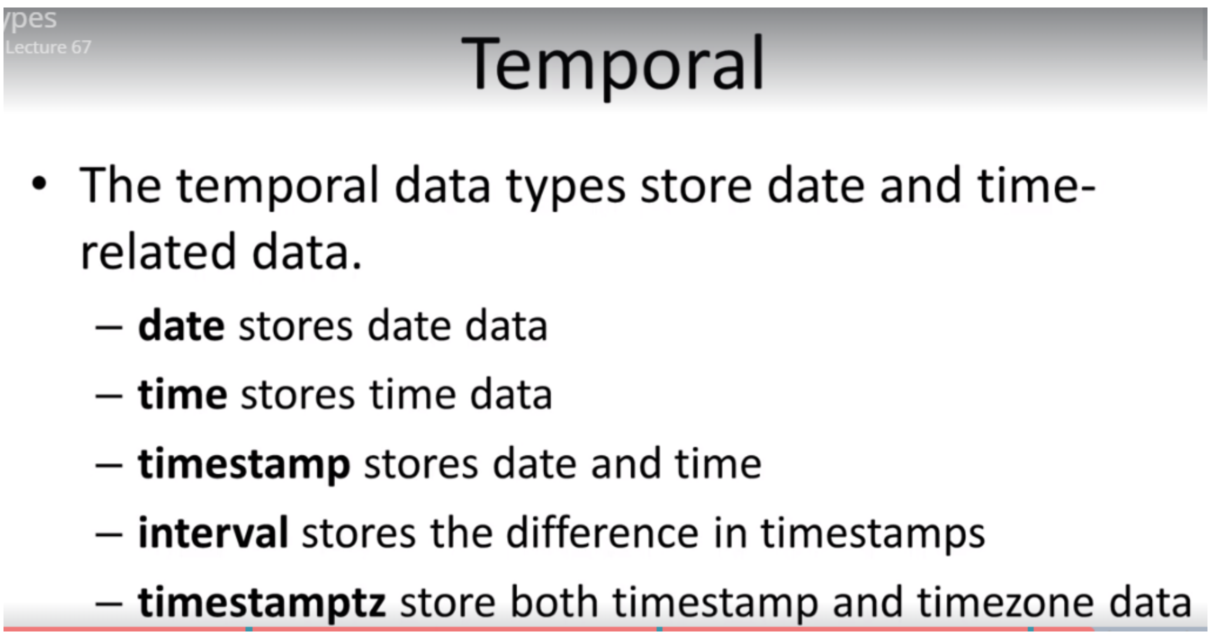
- You use self join when you want to combine rows with other rows in the same table.
- To perform the self join operation, you must use a table alias to help SQL distinguish the left table from the right table of the same table.

Postgres Data Types



- PostgreSQL supports the following data types:
  – Boolean
  – Character
  – Number
  – Temporal i.e., date and time-related data types
  – Special types
  – Array

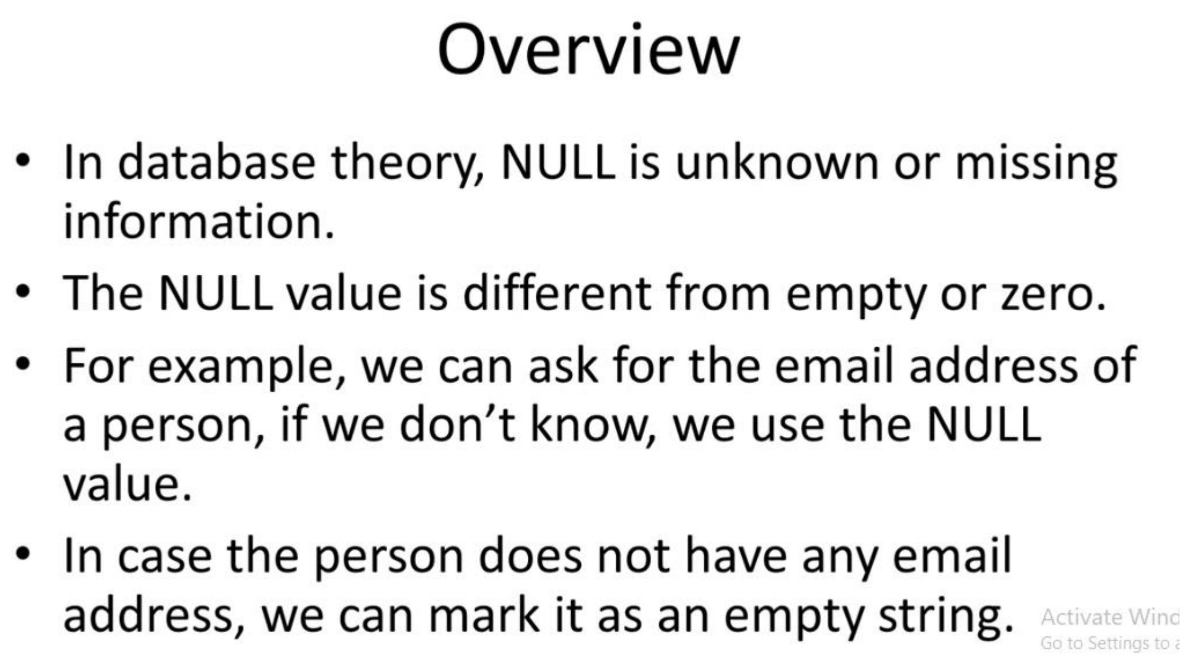
In postgressql serial is same as AUTO_INCREMENT in other databases.



ypes
Lecture 67

Temporal

- The temporal data types store date and time-related data.
  – **date** stores date data
  – **time** stores time data
  – **timestamp** stores date and time
  – **interval** stores the difference in timestamps
  – **timestamptz** store both timestamp and timezone data



Overview

- In database theory, NULL is unknown or missing information.
- The NULL value is different from empty or zero.
- For example, we can ask for the email address of a person, if we don’t know, we use the NULL value.
- In case the person does not have any email address, we can mark it as an empty string.

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
)
-----------------------------------------------
```