SELECT

DISTINCT

LIMIT 2

WHERE

AND

OR

=

>=

<=

!=

ORDER BY last\_name **ASC/DESC**; (ascending or descending)

ORDER BY first\_name ASC, last\_name DESC;

BETWEEN and NOT BETWEEN

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

SELECT customer\_id, amount, payment\_date, staff\_id from payment

WHERE staff\_id NOT IN (1,4)

ORDER BY customer\_id

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

SELECT first\_name, last\_name FROM customer

WHERE first\_name LIKE '%er%';

This technique is called pattern matching

SELECT first\_name, last\_name FROM customer

WHERE first\_name NOT LIKE '%en';

LIKE is case sensitive ILIKE is not!

AVG

SUM

MIN

MAX

ROUND

SELECT ROUND(AVG(amount),5) FROM payment;

\*\* USE AVG with ROUND always

SELECT customer\_id, SUM(amount) FROM payment

GROUP BY customer\_id

ORDER BY SUM(amount) DESC;

The difference between HAVING and WHERE:

HAVIG condition comes after GROUP BY and WHERE is applied before GROUP BY

SELECT customer\_id, SUM(amount) from payment

GROUP BY customer\_id

HAVING SUM(amount) > 200;

SELECT rating, ROUND(AVG(rental\_rate),2) FROM film

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

GROUP BY rating

HAVING ROUND(AVG(rental\_rate),2) < 3

;

Use where when you wanna filter something related to non-aggregate function and use HAVING when you wanna filter on the aggregate function

Assignment 1 Solutions:

SELECT customer\_id, SUM(amount) from payment

WHERE staff\_id=2

GROUP BY customer\_id

HAVING SUM(amount)>=110

;

SELECT COUNT(title) FROM film

WHERE title LIKE 'J%'

;

SELECT first\_name, last\_name FROM customer

WHERE first\_name LIKE 'E%' AND address\_id < 500 AND customer\_id=434

;

SELECT first\_name, last\_name FROM customer

WHERE first\_name LIKE 'E%' AND address\_id < 500

ORDER BY customer\_id DESC

LIMIT 1;

SELECT customer\_id, SUM(amount) AS Total\_Spent FROm payment

GROUP BY customer\_id;

***JOIN***

INNER JOIN:

SELECT

customer.customer\_id,

first\_name,

last\_name,

email

FROM customer

INNER JOIN payment ON customer.customer\_id = payment.customer\_id

Lets say we want to add first name, last name and email address of the customers ids FROM customer table TO the payment table, for the customer\_ids which are in both columns!

SELECT

customer.customer\_id,

first\_name,

last\_name,

email

FROM customer

INNER JOIN payment ON customer.customer\_id = paym

ent.customer\_id

WHERE first\_name LIKE 'A%'

ORDER BY first\_name;

NOTE: Always use ORDER BY AFTER WHERE, otherwise You will get error!

SELECT title, COUNT(title) AS copies\_store FROM inventory

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

WHERE store\_id=1

GROUP BY title;

ORDER BY title;

NOTE: WE cannot SELECT a column which has been set in WHERE condition, for example, in the example above we cannot query for store\_id, since its in WHERE clause.

INNER JOIN WITH AS:

SELECT title, name AS movie\_language

FROM film

INNER JOIN language AS lng ON lng.language\_id=film.language\_id;

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

FROM film

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

WHERE inventory.film\_id IS NULL;

MATHEMATICAL FUNCTIONS:

1. extract

Link: <https://www.postgresql.org/docs/9.6/functions-datetime.html>

SELECT SUM(amount) AS total\_amount, extract(month from payment\_date) AS month

FROM payment

GROUP BY month

ORDER BY SUM(amount) DESC

LIMIT 1;

2. Mathematical operators:

Link: <https://www.postgresql.org/docs/10/functions-math.html>

SELECT customer\_id + rental\_id AS new\_id

FROM payment

;

3. String

Link: <https://www.postgresql.org/docs/9.1/functions-string.html>

SELECT first\_name || ' ' ||last\_name as concat FROM customer;

SUBQUERY:

SELECT title, rental\_rate

FROM film

where rental\_rate > (SELECT AVG(rental\_rate) FROM film)

SELECT film\_id, title

FROM film

WHERE film\_id IN

(SELECT inventory.film\_id

FROM rental

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

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

SELF JOIN

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

FROM customer AS a, customer AS b

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

Assignemt 2:

SELECT facid, name, membercost, monthlymaintenance FROM cd.facilities

WHERE membercost < 1/50 \* monthlymaintenance

;

SELECT \* FROM cd.facilities

WHERE name LIKE '%Tennis%';

SELECT \* FROM cd.facilities

WHERE facid IN (1,5);

SELECT \* FROM cd.bookings

WHERE starttime > '2012-08-31';

SELECT DISTINCT(surname) FROM cd.members

ORDER BY surname DESC

LIMIT 10

;

SELECT \* FROM cd.members

ORDER BY joindate DESC

LIMIT 1;

SELECT COUNT(facid) FROM cd.facilities

WHERE guestcost > 10

;

SELECT facid, SUM(slots) FROM cd.bookings

WHERE starttime BETWEEN '2012-09-01' AND '2012-09-30'

GROUP BY facid

ORDER BY SUM(slots) DESC;

SELECT facid, SUM(slots) FROM cd.bookings

GROUP BY facid

ORDER BY facid

;

SELECT \* FROM cd.facilities

INNER JOIN cd.bookings ON cd.facilities.facid = cd.bookings.facid

WHERE name LIKE '%Tennis%' AND cd.bookings.starttime >= '2012-09-21' AND cd.bookings.starttime < '2012-09-22'

ORDER BY starttime DESC

;

SELECT \* FROM cd.bookings

INNER JOIN cd.members ON cd.bookings.memid = cd.members.memid

WHERE firstname = 'David' AND surname = 'Farrell'

;

CREATING TABLES AND DATABASES:

Primary key is one and only one

Good to have in every table

**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  
);**

Serial automatically is filled by SQL

VARCHAR(fixed n)

**CREATE TABLE leads1(**

**user\_id serial PRIMARY KEY,**

**first\_name VARCHAR (50) NOT NULL,**

**last\_name VARCHAR (50) NOT NULL,**

**email VARCHAR (355) UNIQUE NOT NULL,**

**sign\_up TIMESTAMP NOT NULL,**

**time\_spent integer NOT NULL**

**);**

\*\*\*\*\*\*\*\*\* NOW having the Table, trying to fill

1. INSERT

INSERT INTO link(url, name)

VALUES

('www.google.com', 'Google');

--Inserting multiple entries:

INSERT INTO link(url, name)

VALUES

('www.bing.com','bing'),

('www.amazon.com','amzon');

---Making another Table like the previously created one

CREATE TABLE link\_copy (LIKE link);

-- inserting from another table

INSERT INTO link\_copy

SELECT \* FROM link

WHERE name='Bing';

1. Update:

UPDATE link

SET rel = 'starts with b'

WHERE name LIKE 'n%';

-- Updating based upon another column in the same table

UPDATE link

SET rel = name;

WHERE name LIKE 'a%';

-- returning if you wanna get the row itself as well

UPDATE link

SET rel = 'Google'

WHERE id = 1

RETURNING id, url, name, rel;

1. DELETE

-- delete rows from delete (with no WHERE, it will delete the whole table)

DELETE FROM link

WHERE name LIKE 'a%'

DELETE FROM link

WHERE id = 2

RETURNING \*;

1. ALTER TABLE

-- Add a columns

ALTER TABLE link ADD COLUMN active boolean;

-- removing the column

ALTER TABLE link DROP COLUMN active;

-- rename columns

ALTER TABLE link RENAME COLUMN title TO new\_title\_name;

-- renaming the TABLE name

ALTER TABLE link RENAME TO url\_table;

5.

DROP TABLE IF EXISTS link;

1. CHECK conditions

CREATE TABLE new\_users(

id serial PRIMARY KEY,

first\_name VARCHAR(50),

birth\_date DATE CHECK(birth\_date > '1900-01-01'),

join\_date DATE CHECK(join\_date > birth\_date),

salary integer CHECK(salary > 0)

);

\*\*\*\*\*\*\*\*\*\*\*\*\* Assignment 3

INSERT INTO students (student\_id, first\_name, last\_name, phone, email, graduation\_year, homeroom\_number)

VALUES (1, 'Mark', 'Watney', '777-555-1234', Null, '2035-01-01', 5);

\*\*\*\*\*\*\*\*\*\* VIEW

If you wanna view a table you have created using join for example:

CREATE VIEW customer\_info AS

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

FROM customer

INNER JOIN address

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

\*\*\*\*\*\*\*\*\*\* python and PostgreSQL

Link: https://wiki.postgresql.org/wiki/Psycopg2\_Tutorial