

SQL

Igor Arambasic

SQL = Structured Query Language

- ISO standard
- but there are some differences between the systems that use it
- declaring language (WHAT to do, not HOW to do it)
- insensible to CAPITAL letters

Postgres

@ command line

- `psql -l` → **list** databases and exit
- `psql -d name` → connect to specific DB
- `\q` --> desconect to DB
- `psql -c 'sql query'` → **execute** and return the result
- `psql -f file.name` → **execute** commands **from file**, then exit
- `psql -d dsc -c '\l'`

@ psql

```
dsc=# this is one line
dsc=# notice the minus (-) in this line
dsc=# again -
dsc=# until I put ;
ERROR:  syntax error at or near "this"
LINE 1: this is one line
```

- you cannot return to previous line while writing a command!
- The command is interpreted when ";" is encountered!
- CTRL+C stops psql from interpreting the command
- there exist command history
- BUT all CTRL+C commands are NOT saved in history

@ psql

Non SQL commands start with: \

- \l = *list* = returns the list of the available DB
- \c = *connect* = returns the name of the actual DB
- \c name = connect to specific DB
- \d = *describe* = returns the list of the tables
- \d nombre = returns more information of the specific table
- \i file.sql = *execute* sql from file
- \q = quit (Ctrl+d)
- \h = list SQL commands (\h +command = command help)
- \? = list psql commands

We are in the shell!!!

All shell commands start with: **\!**

- \!pwd
- \! ls -l
- \! cd ..
- \! ls -l

- \cd ..

Create and Delete a DB

```
CREATE DATABASE Networking;  
\d
```

```
DROP DATABASE Networking;  
\d
```

Tables

Data is organized in tables.

Tables consist of columns, which can be of different types (text, number, date, etc).

```
CREATE TABLE friends (  nombre VARCHAR,  
                          edad INT,  
                          email VARCHAR);
```

```
\d friends
```

```
DROP TABLE friends ;
```

```
\d
```


Populating the tables

```
CREATE TABLE friends (      nombre VARCHAR,  
                             edad INT,  
                             email VARCHAR);
```

```
INSERT INTO friends VALUES ('Lionel Messi', 29, 'messi@fcbarcelona.es');
```

```
INSERT INTO friends VALUES ('Donald Tramp', 71, 'trump@twitter.com');
```

```
INSERT INTO friends VALUES ('Ivanka Tramp', 35);
```

```
INSERT INTO friends VALUES ('Melania Tramp', 'melania@supersmile');
```

```
INSERT INTO friends (nombre, email) VALUES ('Melania Tramp', 'melania@supersmile');
```

```
INSERT INTO friends VALUES ('Mario',25), ('Marina',24) , ('Renata',1, 'notiene');
```

Extracting the information

```
SELECT * FROM friends ;
```

```
SELECT nombre, edad FROM friends;
```

Filtering information

```
SELECT * FROM friends WHERE edad < 25;
```

```
SELECT nombre, edad FROM friends WHERE edad <> 24;
```

```
SELECT nombre, edad FROM friends WHERE edad != 24;
```

```
SELECT * FROM friends WHERE nombre LIKE '%T' ;
```

```
SELECT * FROM friends WHERE UPPER(nombre) LIKE '%T' ;
```

```
SELECT * FROM friends WHERE UPPER(nombre) NOT LIKE '%T%' ;
```

Special data type: NULL

```
SELECT * FROM friends WHERE edad IS NULL;
```

Deleting Information

```
DELETE FROM friends ;
```

```
DELETE FROM friends WHERE edad < 18;
```

Diferencias entre DROP y DELETE

- We have seen and used DROP and DELETE commands. What is the difference between them? When would you use one and when the other?
- DROP completely deletes the table and you cannot eliminate the data selectively. We use this when we are sure not to need the table any more.
- DELETE eliminates selectively the table rows, and leaves the rest of rows intact. We use it when for example we want to eliminate a user from our system.

Exercise 1

- Generate 'Facebook' table where every person is identified with the name, age, city of residence, and email. The table should at least have 5 rows.
- Write a query to obtain Facebook friends which are younger than <18 or older than 65 years.
- Write a query to obtain all facebook friends which do not reside in Madrid.

Primary Key

- Primary key uniquely identifies each row of the table.
- For example DNI could be seen as excellent primary key in a table consisting of personal information.
- Primary key cannot be empty
- There cannot be two rows with the same primary key
- Once created, the primary key of a row should not be changed

Primary Key

```
CREATE TABLE facebook (nombre VARCHAR,  
                           email VARCHAR,  
                           edad INT,  
                           PRIMARY KEY (email));
```

Primary key can also be added to the existing table:

```
ALTER TABLE friends ADD PRIMARY KEY (email);
```

Editing the table structure

Adding a new column:

```
ALTER TABLE friends ADD COLUMN telefono VARCHAR;
```

Deleting just one column:

```
ALTER TABLE friends DROP COLUMN telefono;
```

Updating the information inside the table

Change city of residence for all Facebook friends:

```
UPDATE facebook SET residencia = 'Getafe';
```

Block underage users:

```
UPDATE facebook SET state= 'blocked' WHERE edad < 18;
```

Exercise 2

- Add phone and username columns to your Facebook table.
- What is the content of these two columns after their creation?
- Update the phone number of all of your facebook friends
- All your Facebook friends use their email as user name, so that both columns coincide. How can you take advantage of this fact when updating the user name column

Wget

@Linea de comandos:

- wget
https://raw.githubusercontent.com/masterdatascience/postgres/master/my_fb_friends.csv
- wget
https://raw.githubusercontent.com/masterdatascience/postgres/master/my_ldin_contacts.csv

Git

@Linea de comandos:

- `git clone https://github.com/masterdatascience/postgres`

@Postgres:

```
ALTER TABLE facebook RENAME TO old_facebook;  
CREATE TABLE facebook AS SELECT  
    nombre,  
    edad,  
    residencia,  
    email FROM old_facebook;
```

```
\! cat ./my_fb_friends.csv  
ALTER TABLE facebook ADD PRIMARY KEY(email);  
\copy facebook from './my_fb_friends.csv' delimiter '^' csv header;
```

Alias

- Alias is used to change momentarily the name of something, during a query
- Alias is specified with **AS**

```
SELECT *, (edad*2) as doub_edad from facebook where (edad*2)>49;  
SELECT * FROM facebook AS t WHERE t.residencia = 'Valencia';
```


Eliminating the duplicates

Cities present in Facebook table:

```
SELECT DISTINCT residencia FROM facebook;
```

What is the difference with respect to this query?

```
SELECT residencia FROM facebook;
```

Sorting the results

When we launch a query the order of the obtained results is not guaranteed!

We can define the order by using order by.

from lowest to highest

```
SELECT * FROM facebook ORDER BY edad;
```

from highest to lowest:

```
SELECT * FROM facebook ORDER BY edad DESC;
```

Sorting by multiple options

```
SELECT * FROM facebook  
ORDER BY edad, nombre;
```

```
SELECT * FROM facebook  
ORDER BY edad DESC, nombre ASC;
```

Limiting the number of results

```
SELECT * FROM facebook  
ORDER BY edad DESC  
LIMIT 5;
```

Limiting the number of results

- With offset we can start counting at the specific line

```
SELECT * FROM facebook  
ORDER BY edad DESC  
OFFSET 5  
LIMIT 3;
```

Logical Operators

AND

NOT

BETWEEN

OR

IN

```
SELECT * FROM facebook  
WHERE edad BETWEEN 18 AND 30;
```

```
SELECT * FROM facebook  
WHERE edad NOT IN (15, 16,17);
```

Aggregate function

compute a single result value from a set of input values.

AVG

COUNT

MAX

MIN

SUM

```
SELECT COUNT(*) FROM facebook;
```

```
SELECT AVG(edad), MIN(edad), MAX(edad) FROM facebook;
```

Group by

- is used to group together those rows in a table that share the same values in all the columns listed.
- The effect is to combine each set of rows sharing common values into one group row that is representative of all rows in the group
- This is done to eliminate redundancy in the output and/or compute aggregates that apply to these groups.

```
SELECT residencia, COUNT(*), AVG(edad)  
FROM facebook  
GROUP BY residencia;
```


Group by

```
SELECT residencia, edad, COUNT(*) FROM facebook  
GROUP BY residencia, edad;
```

```
SELECT residencia, COUNT(*), AVG(edad) FROM facebook  
GROUP BY residencia HAVING AVG(edad)>20;
```

Multiple queries

- The result of a query is nothing more than a temporal table. Hence we can apply a query over the result of another query.

```
SELECT *, ( SELECT AVG(edad)
            FROM facebook
            WHERE residencia='Madrid' ) AS Mad_average
FROM facebook;
```

```
SELECT *, ( SELECT AVG(edad)
            FROM facebook
            WHERE residencia='Madrid' ) AS Mad_average
FROM facebook
WHERE edad < (SELECT AVG(edad)
              FROM facebook
              WHERE residencia='Madrid') ;
```

Multiple queries

- WHERE + IN, ANY, ALL

```
SELECT *  
FROM facebook  
WHERE residencia IN (SELECT residencia  
                      FROM facebook  
                      GROUP BY residencia  
                      HAVING AVG(edad)>35) ;
```

```
SELECT * FROM facebook  
WHERE usuario = ANY (SELECT nombre FROM friends WHERE nombre  
LIKE '%a%');
```

Linkedin

@Linea de comandos:

- `echo "drop table if exists linkedin">psql_create_contactos.sql`
- `mv my_ldin_contacts.csv linkedin`
- `csvsql -d '^ linkedin -i postgresql >> psql_create_contactos.sql`
- `echo "\\copy linkedin from './linkedin' delimiter '^' csv header; " >> psql_create_contactos.sql`
- `cat ./psql_create_contactos.sql`
- `psql -d networking -f psql_create_contactos.sql`

@Postgres:

- \d linkedin
- select * from linkedin;

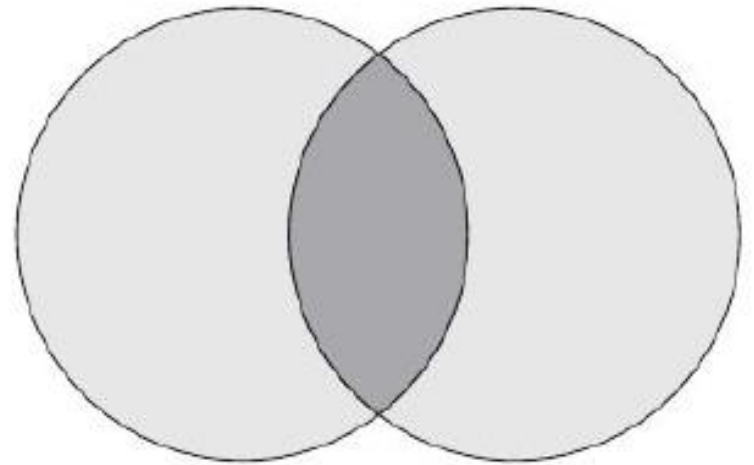
JOINS

- Joins the result of different queries
- Join combination can be:
 - INNER JOIN,
 - LEFT OUTER JOIN,
 - RIGHT OUTER JOIN,
 - FULL JOIN
- Multiple queries (especially join) can be very slow and can consume a lot of memory.
- Data model should facilitate multiple queries, but try NOT to duplicate too much the data 😊

INNER JOIN

- Combines the data of various tables and returns the overlap

```
SELECT fb.email  
FROM facebook AS fb  
INNER JOIN linkedin AS ln  
ON fb.email=ln.email;
```



INNER JOIN

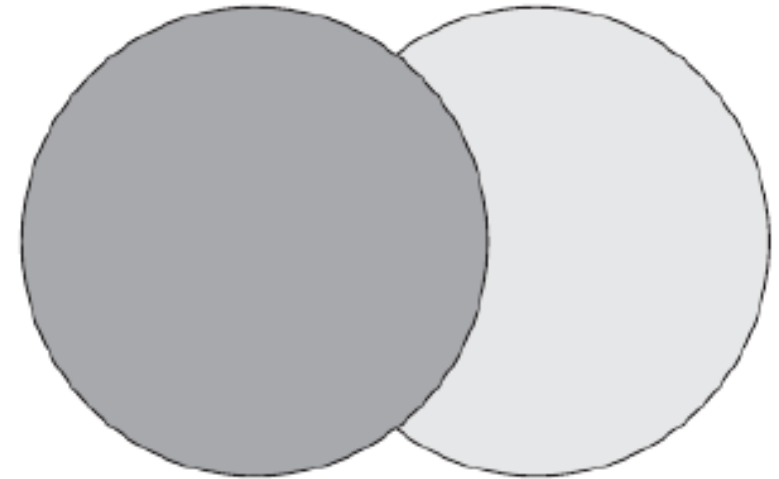
```
create table twitter (email VARCHAR, account VARCHAR);  
insert into twitter values ('isra@centerofworld.com', 'isra_oops');  
insert into twitter values ('kiko@centerofworld.com', 'Kiko');
```

```
SELECT fb.email AS email, fb.nombre AS fb, ln.contact AS ln, tw.account AS tw  
FROM facebook AS fb  
INNER JOIN linkedin AS ln ON fb.email=ln.email  
INNER JOIN twitter AS tw ON tw.email=fb.email;
```


LEFT OUTER JOIN

- All rows from the LEFT table with the additional data of the RIGHT table

```
SELECT fb.*, ln.contact, ln.company  
FROM facebook as fb  
LEFT OUTER JOIN  
linkedin AS ln  
ON fb.email=ln.email;
```

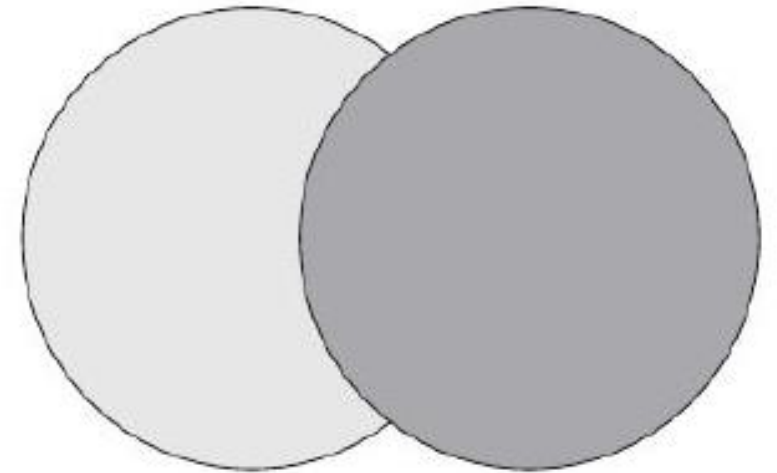


DO SANITY CHECK!!!!

RIGHT OUTER JOIN

- All rows from the RIGHT table with the additional data of the LEFT table

```
SELECT ln.*, fb.nombre, fb.edad, fb.residencia  
FROM facebook as fb  
RIGHT OUTER JOIN  
linkedin AS ln  
ON fb.email=ln.email;
```

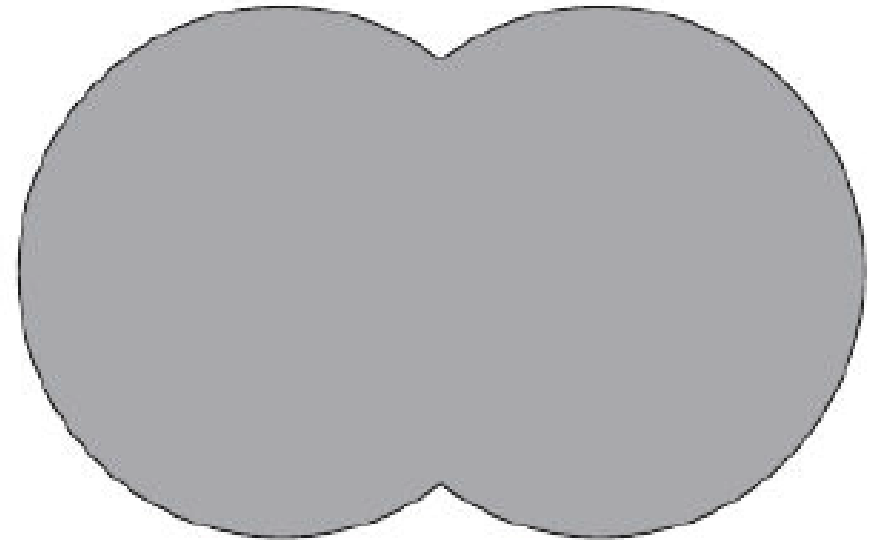


DO SANITY CHECK!!!!

FULL JOIN

- Combination of LEFT OUTER JOIN and RIGHT OUTER JOIN

```
SELECT ln.*, fb.*  
FROM facebook as fb  
FULL JOIN  
linkedin AS ln  
ON fb.email=ln.email;
```



Exercise 3

- Import `optd_aircraft.csv` and `optd_airlines.csv` in postgres
(/Data/opentraveldata/)
- Which airplane has the highest number of engines?
(`optd_aircraft`)
- What number of engines is most common on airplanes?
(`optd_aircraft`)

History

- `/home/dsc/.psql_history`