**Real python**

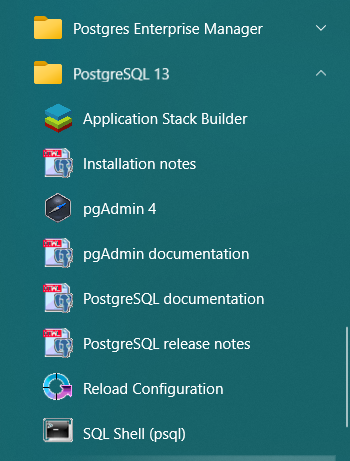
***Python and MySQL Database: A Practical Introduction***

# Notas – PostgreSQL

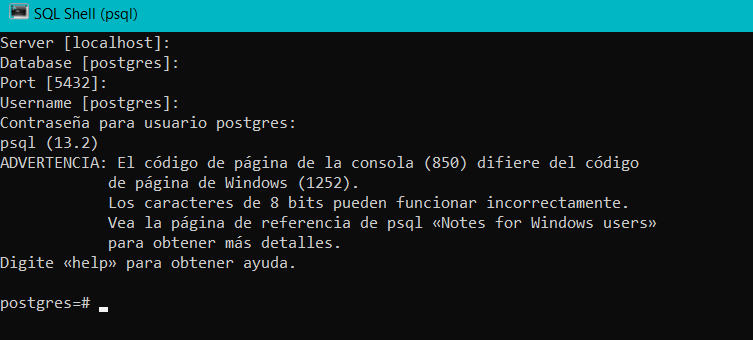
* One of the most popular
* SQL stands for Structured query Language, use for managing db
* Open source
* La contraseña de postgrestSQL es 32557

**Fecha: 08-05-2021**

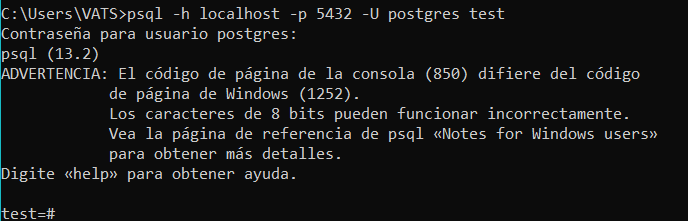
* Hoy intenté ingresar a PostgreSQL y no recordaba la contraseña, que Gil
* Vale, quiere decir que está instalado, ni me acordaba
* Debo ir a la carpeta que esta en inicio que contiene todos los programas instalados



* Para ingresar desde consola. Tenemos todos los valores predeterminados, por lo que sol basta con dar enter:



* Intento instalar psycopg con conda pero no trabaja
* Comando: conda install -c anaconda psycopg2
* **libpq** is the C application programmer's interface to PostgreSQL. libpq is a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries.
* Sigo el tutorial de esta pagina: <https://www.postgresqltutorial.com/postgresql-python/connect/>
* Ya puedo conectar con la base de datos desde el terminal



* Agregué la ruta de la carpeta bin de postgres al path de las variables del sistema.
* Ahora puedo ejecutar postgres desde el cmd. Debo tener cuidado porque toma el usuario de la maquina por defecto(VATS) y ese usuario no está creado. El que está creado es postgres con 32557
* Ahora sé como crear tablas sin restricciones. Pero eso puede ser un problema por los valores NULOS
* Necesito practicar el tipo de restricciones según el tipo de dato
* Ahora veré el video de como insertar valores a la tabla
* Tuve inconvenientes, como siempre, agregando las rutas de los archivos. De esta forma lo conseguí: \i 'D:\\xx - Github\\My-python-Workbook\\28 - Databases - PostgreSQL\\db\\person.sql'

O

\i 'D:/xx – Github/My-python-Workbook/28 - Databases – PostgreSQL/db/person.sql'

Fecha: 09-05-2020

* Sigo realizando ejercicios con la base de datos generada en: <https://www.mockaroo.com/>
* Ahora se como filtrar por columna y organizar de orden ascendente y descendente
* Algo que también aprendí fue limpiar la ventana del prompt: \! Cls

Fecha 10-05-2021

* Comando IN



The same



SELECT \* FROM person IN (‘China’,’Brazil’,’Colombia’) ORDER BY country;

* BETWEEN

Filtering using a range

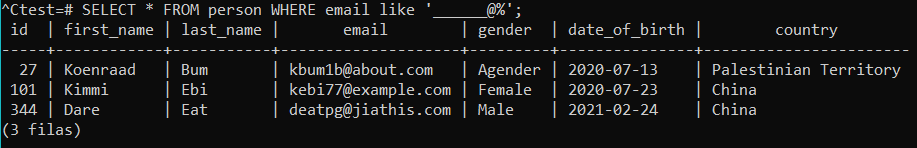


SELECT \* FROM person WHERE date\_of\_birth BETWEEN DATE '2020-09-21' AND '2021-12-31';

* Command LIKE

SELECT \* FROM person WHERE email like '%.com';

SELECT \* FROM person WHERE email like '%@google.%';



Using “\_”.

* Command ILIKE

Do the same that LIKE, but is not case sensitive

* GROUP BY

Help you to group values, in the case os filtering by DISTINCT, for example

e.g SELECT country, COUNT(\*) FROM person GROUP BY country;

* SELECT DISTINCT country FROM person;

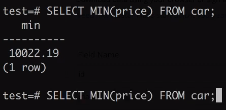
Shows the unique values

* GROUPING BY HAVING

SELECT country, COUNT(\*) FROM person GROUP BY country HAVING COUNT(\*) > 5 ORDER BY country;



* Max, Min, AVG



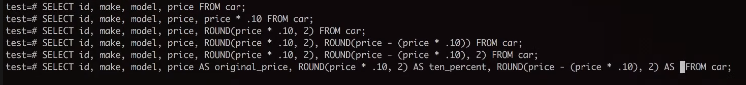
SELECT MAX(price) FROM car;

* ROUND(value)

SELECT ROUND(MAX(price)) FROM car;

* AS

Using “AS” as a alias:



* Coalesce
* NULLIF

## Commands

* \q
* \l
* CREATE DATABASE name;
* Psql -h localhost -p 5432 -U postgres dbName
* \c dbName – Connect to a listed database
* DROP DATABASE dbName – Eliminate the db
* CREATE TABLE tableName(

Id int,

First\_name VARCHAR(50),

Date\_of\_birth DATE);

* \d – Describe all the tables in the db, for tables
* \d tableName – Describe the table
* SELCT \* FROM person – get into a table, so far.
* INSERT INTO tableName(fields, fields, fields, fields,) VALUES(fieldValues, fieldValues, fieldValues, fieldValues);

e.g

INSERT INTO vendors(vendor\_name) VALUES('Probando') RETURNING vendor\_id;

* \i FILENAME – cargar indicaciones desde archivo
* \! Cls or \! Clear – Limpiar pantalla
* ORDER BY

SELECT \* FROM vendors ORDER BY vendor\_id asc;

* DISTINCT – para ver los valores únicos en nuestra tabla. E.g SELECT DISTINC country FROM person ORDER BY country;
* WHERE for filtering rows. E.g SELECT \* FROM person WHERE gender = ‘Female’;

Conwhere se pueden usar compuertas and, or..

WHERE for filtering rows. E.g SELECT \* FROM person WHERE gender = ‘Female’ AND country = ‘Colombia’;

* SELECT 1 = 1;  
  using select for making comparison. <> not equal, the rest is the same than other languages
* LIMIT, OFFSET AND FETCH

SELECT \* FROM person OFFSET 5 LIMIT 5;

SQL standard   
SELECT \* FROM person OFFSET 5 FETCH FIRST 5 ROW ONLY;

* IN - SELECT \* FROM person WHERE country IN ('China', 'Brazil', 'France');
* \i 'D:\\xx - Github\\PostgreSQL+Python\\source\_6\_CallingFunction\\function.sql'

Para leer instrucciones desde un archivo.

* RETURNING

UPDATE prueba SET prueba\_name = 'Victor'

WHERE prueba\_name = 'Alfonso'

RETURNING prueba\_name, id\_prueba AS new\_id;

* ON UPDATE CASCADE ON DELETE CASCADE

ON UPDATE CASCADE ON DELETE CASCADE means that if you UPDATE OR DELETE the parent, the change is cascaded to the child. This is the equivalent of ANDing the outcomes of first two statements.

* FOREIGN KEY (part\_id)

Telling to the db manager you’ll use an external ID

* REFERENCES parts (part\_id)

Telling to the db manager you’ll use an external ID and that gonna be your reference

CREATE TABLE vendor\_parts (

vendor\_id INTEGER NOT NULL,

part\_id INTEGER NOT NULL,

PRIMARY KEY (vendor\_id , part\_id),

FOREIGN KEY (vendor\_id)

REFERENCES vendors (vendor\_id)

ON UPDATE CASCADE ON DELETE CASCADE,

FOREIGN KEY (part\_id)

REFERENCES parts (part\_id)

ON UPDATE CASCADE ON DELETE CASCADE

* CREATE OR REPLACE FUNCTION

CREATE OR REPLACE FUNCTION increment(i integer) RETURNS integer AS $$

BEGIN

RETURN i + 1;

END;

$$ LANGUAGE plpgsql;

* RETURNS TABLE
* BEGIN
* INNER JOIN

# PostgreSQL + Python

## Connect to PostgreSQL Database Server

* Estoy siguiendo un tutorial en el que me instruyen como estructurar la comunicación.
* Creo 3 archivos, de momento. Un archivo almaceno las credenciales de la base de datos, otro donde proceso la configuración y un tercero que es donde se lleva a cabo la conexión
* El archivo de conexión hace un llamado al de configuración y el de config llama al de credenciales
* Tengo algunas dudas con el archivo de config
  + ConfigParser

This module provides the [ConfigParser](https://docs.python.org/3/library/configparser.html" \l "configparser.ConfigParser" \o "configparser.ConfigParser) class which implements a basic configuration language which provides a structure similar to what’s found in Microsoft Windows INI files. You can use this to write Python programs which can be customized by end users easily.

**Note:** This library does not interpret or write the value-type prefixes used in the Windows Registry extended version of INI syntax.

* Dudas en connect
  + psycopg2.connect(\*\*params)   
    por qué con dobe estrella?

The **\*args** and **\*\*kwargs** is a common idiom to allow arbitrary number of arguments to functions as described in the section more on defining functions in the Python documentation.

The \*args will give you all function parameters as a tuple:

def foo(\*args):

for a in args:

print(a)

foo(1)

# 1

foo(1,2,3)

# 1

# 2

# 3

The \*\*kwargs will give you all keyword arguments except for those corresponding to a formal parameter as a dictionary.

def bar(\*\*kwargs):

for a in kwargs:

print(a, kwargs[a])

bar(name='one', age=27)

# name one

# age 27

Both idioms can be mixed with normal arguments to allow a set of fixed and some variable arguments:

def foo(kind, \*args, \*\*kwargs):

pass

It is also possible to use this the other way around:

def foo(a, b, c):

print(a, b, c)

obj = {'b':10, 'c':'lee'}

foo(100,\*\*obj)

# 100 10 lee

Another usage of the \*l idiom is to unpack argument lists when calling a function.

def foo(bar, lee):

print(bar, lee)

l = [1,2]

foo(\*l)

# 1 2

In Python 3 it is possible to use \*l on the left side of an assignment (Extended Iterable Unpacking), though it gives a list instead of a tuple in this context:

first, \*rest = [1,2,3,4]

first, \*l, last = [1,2,3,4]

Also Python 3 adds new semantic (refer PEP 3102):

def func(arg1, arg2, arg3, \*, kwarg1, kwarg2):

pass

Such function accepts only 3 positional arguments, and everything after \* can only be passed as keyword arguments.

Note:

A Python dict, semantically used for keyword argument passing, are arbitrarily ordered. However, in Python 3.6, keyword arguments are guaranteed to remember insertion order.

"The order of elements in \*\*kwargs now corresponds to the order in which keyword arguments were passed to the function." - What’s New In Python 3.6

In fact, all dicts in CPython 3.6 will remember insertion order as an implementation detail, this becomes standard in Python 3.7.

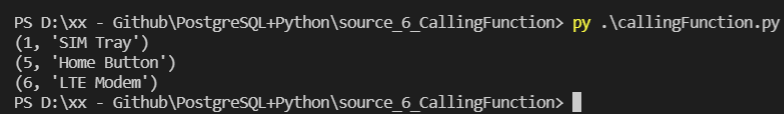
## Create Tables

1. First, construct CREATE TABLE statements.
2. Next, connect to the PostgreSQL database by calling the connect() function. The connect() function returns a connection object.
3. Then, create a cursor object by calling the cursor() method of the connection object.
4. After that, execute the CREATE TABLE by calling the execute() method of the cursor object.
5. Finally, close the communication with the PostgreSQL database server by calling the close() methods of the cursor and connection objects.

* Instalo conda install -c anaconda psycopg2
* Si lo tenía instalado, solo que al ejecutar desde el botón de play dio error

## Call PostgreSQL Functions

* Para poder reproducir la función, la guarde en un archivo con extensión .sql



* La idea es crear la función en el manager de base de dato, luego se corre el programa en python

## Handling BLOB Data

tandard SQL defines BLOB as the binary large object for storing binary data in the database. With the BLOB data type, you can store the content of a picture, a document, etc. into the table.

1. First, read data from a file.
2. Next, [connect to the PostgreSQL database](https://www.postgresqltutorial.com/postgresql-python/connect/) by creating a new connection object from the connect() function.
3. Then, create a cursor object from the connection object.
4. After that, execute the [INSERT](https://www.postgresqltutorial.com/postgresql-insert/) statement with the input values. For BLOB data, you use the Binary object of the psycopg module
5. Finally, commit the changes permanently to the PostgreSQL database by calling the commit() method of the connection object.