Dauphine UNIVERSITÉ PARIS | PSL

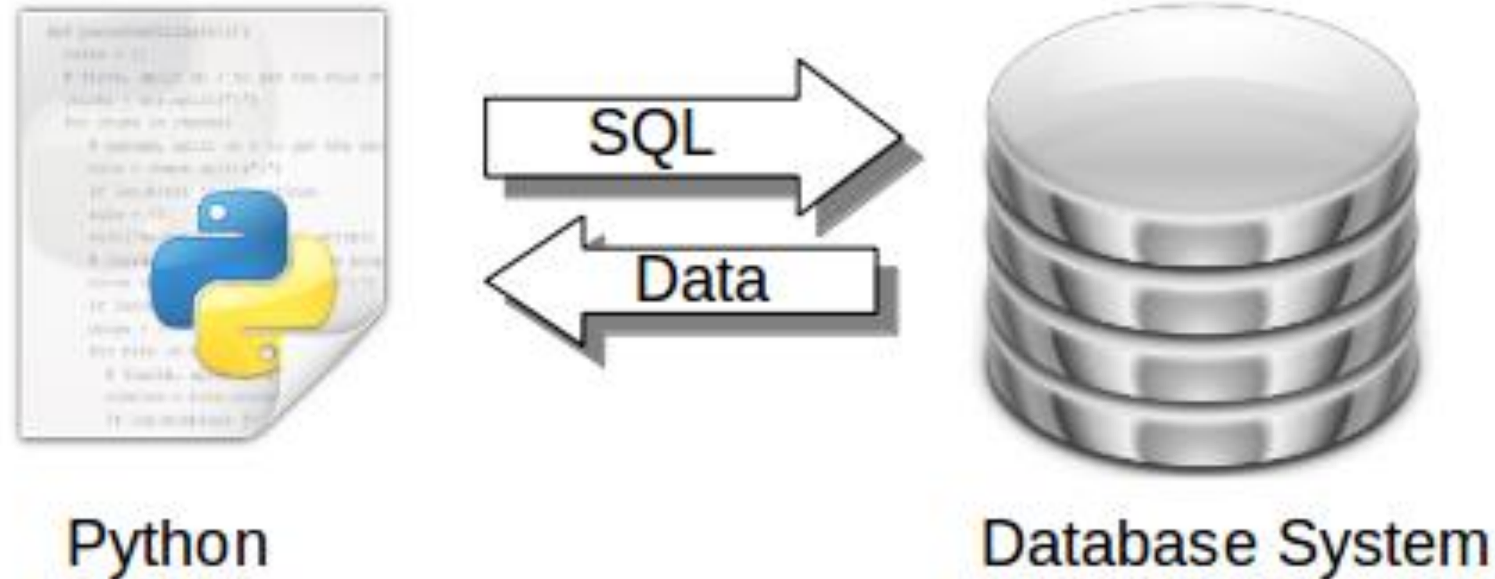
Executive Master Statistique et Big Data

ACCÉDER À UNE BASES DE DONNÉES SQL EN PYTHON

Maude Manouvrier

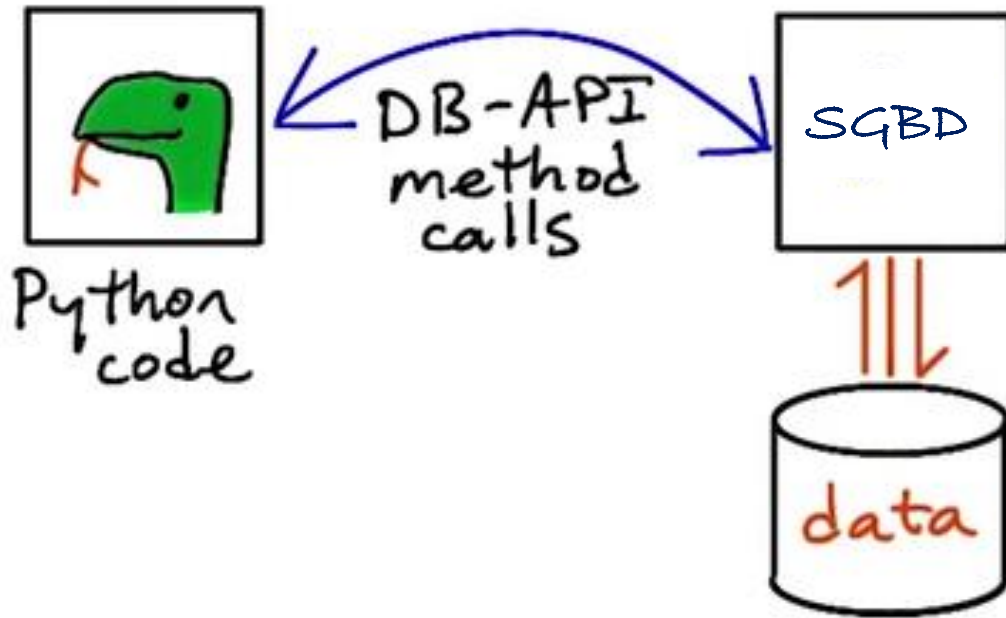
Moodle Exec Master Statistique et Big Data_P5_20-21_Accueil
Module 2 - Bases de données sous SQL - M. MANOUVRIER

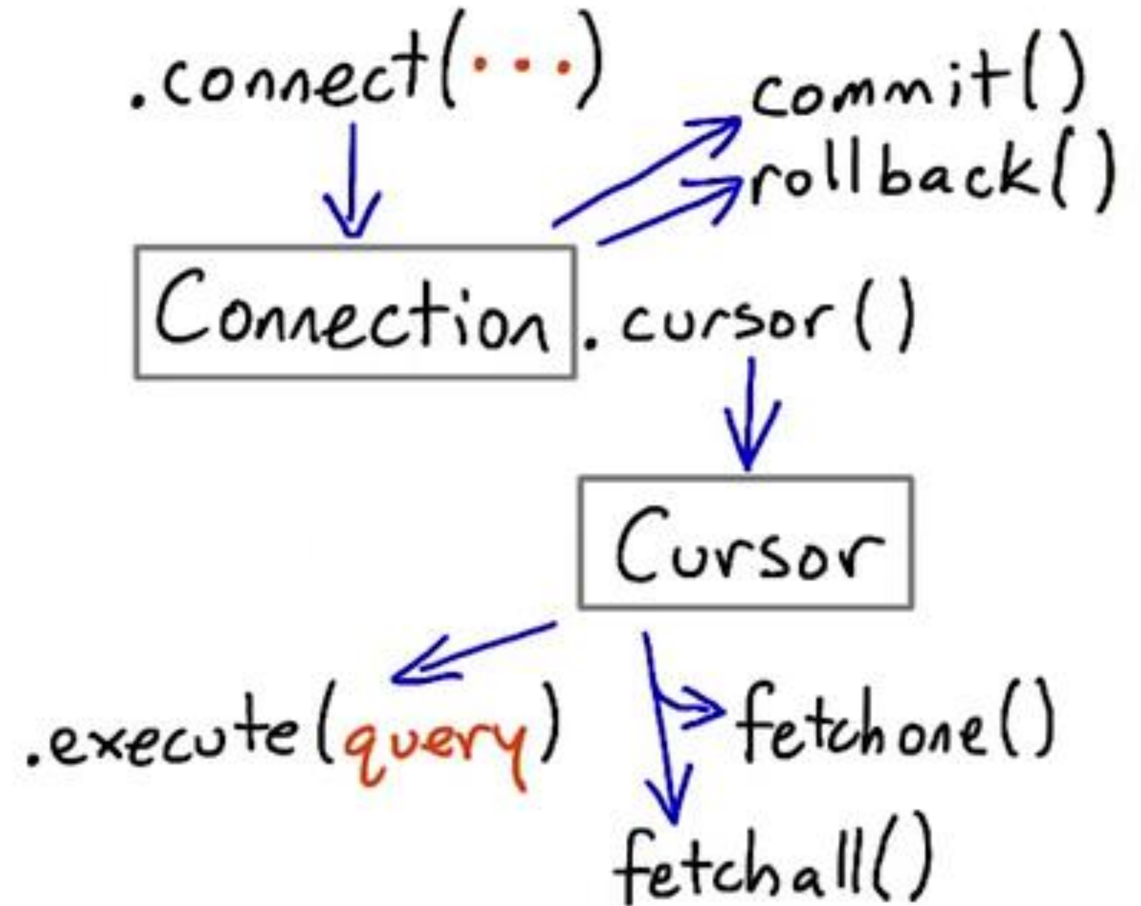
Accéder à une base de données en Python



Nécessité d'utiliser une interface standard pour les modules d'accès à une base de données ou *Database Application Programming Interface* (DB-API) de Python

Méthodes de DB-API Python





Méthodes de DB-API Python

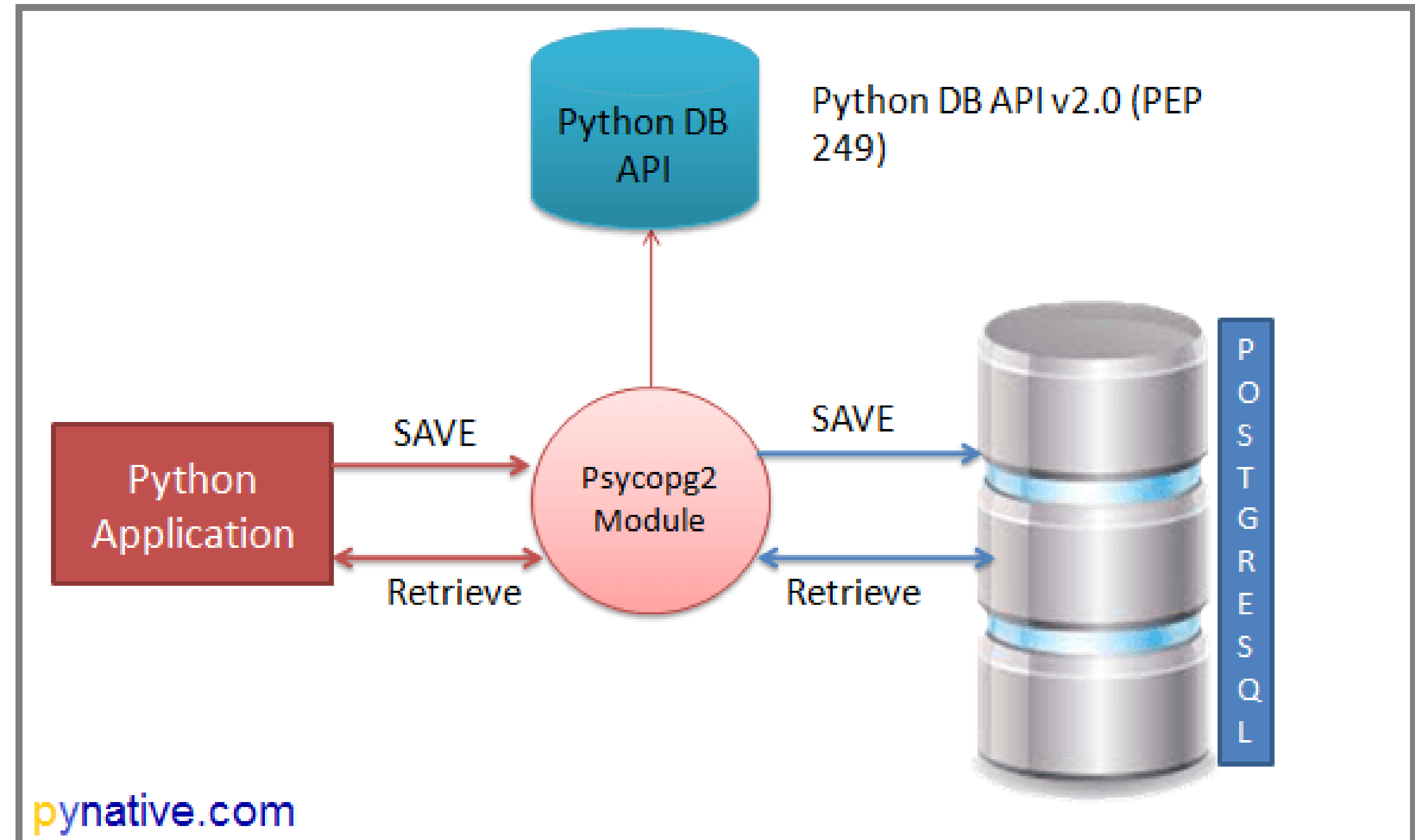
- **`module.connect(...)`** : pour se connecter au SGBD et à la base de données – la méthode renvoie un descripteur de connexion ou génère une exception
- **`connection.cursor()`** : pour créer un curseur pour exécuter une requête SQL et le cas échéant afficher le résultat
- **`connection.commit()`** : pour valider l'exécution d'une transaction (INSERT, UPDATE, DELETE)
- **`connection.rollback()`** : pour annuler une transaction (en cas d'exception par exemple)
- **`connection.close()`** : pour fermer la connexion
- **`cursor.execute(statement)`** : pour exécuter une requête
- **`cursor.fetchall() cursor.fetchone()`** : pour récupérer les nuplets résultats d'une requête SELECT

DB-API Python / SGBD

- `sqlite3` pour SQLite,
- `psycopg2` pour PostgreSQL
- `mysql-python` pour MySQL …

Pour installer le module :

```
>>> pip install psycopg2
```



Exemple de programme Python utilisant PostgreSQL

```
#cf https://librecours.net/module/dwh/etl01/pyt2c02.xhtml

import psycopg2

HOST = "localhost"
USER = "postgres"
PASSWORD = "****"
DATABASE = "postgres"

# Open connection
conn = psycopg2.connect("host=%s dbname=%s user=%s
password=%s" % (HOST, DATABASE, USER, PASSWORD))

# Open a cursor to send SQL commands
cur = conn.cursor()

# Testing
sql = "SELECT * FROM public.personne"
cur.execute(sql)
print(cur.fetchall())

#Close connection
conn.close()
```

`fetchall()` renvoie une liste de tuples :

```
>>> (executing file "<tmp 3>")
[(1, 'Gamotte', 'Albert'), (2, 'Pabien', 'Yvon'), (3,
'Computing', 'Claude'), (4, 'Slatable', 'Deborah'), (5
, 'Suffit', 'Sam'), (6, 'Debece', 'Gilles')]
```

Exemple de programme Python utilisant PostgreSQL

Une fonction pour récupérer les informations de connexion dans un fichier :

```
import psycopg2
from configparser import ConfigParser

def config(filename='database.ini', section='postgresql'):
    # create a parser
    parser = ConfigParser()
    # read config file
    parser.read(filename)

    # get section, default to postgresql
    db = {}
    if parser.has_section(section):
        params = parser.items(section)
        for param in params:
            db[param[0]] = param[1]
    else:
        raise Exception('Section {0} not found in the {1}
file'.format(section, filename))

    return db
```

Contenu du fichier `database.ini` :

```
[postgresql]
host=localhost
database=postgres
user=postgres
password=****
```

Exemple de programme Python utilisant PostgreSQL

Programme utilisant la fonction `config()` :

```
#Connect to the PostgreSQL database server
conn = None
try:

    # read connection parameters
    params = config()

    # connect to the PostgreSQL server
    print('Connecting to the PostgreSQL database...')
    conn = psycopg2.connect(**params)

    # Testing
    cur = conn.cursor()
    sql = "SELECT * FROM public.personne"
    cur.execute(sql)
    L = cur.fetchall()
    print(L)

except (Exception, psycopg2.DatabaseError) as error:
        print(error)
finally:
        if conn is not None:
            conn.close()
            print('Database connection closed.')
```

Exemple de programme Python utilisant PostgreSQL

Exemple d'utilisation de `fetchone`() qui renvoie un tuple :

```
# Open a cursor to send SQL commands
cur = conn.cursor()

# Testing
sql = "SELECT * FROM public.personne"
cur.execute(sql)

# Fetch data line by line
raw = cur.fetchone()

while raw:
    print (raw[0])
    print (raw[1])
    print (raw[2])
    raw = cur.fetchone()

#Close connection
conn.close()
```

```
>>> (executing file "<tmp 3>")
1
Gamotte
Albert
2
Pabien
Yvon
3
Computing
Claude
4
Slatable
Deborah
5
Suffit
Sam
6
Debece
Gilles
```

Attention aux requêtes de mise à jour

Sans utiliser `commit()` :

```
# Open a cursor to send SQL commands
cur = conn.cursor()

sql = "INSERT INTO public.personne VALUES
(7,'Debece','Aude')"

cur.execute(sql)

# Testing
sql = "SELECT * FROM public.personne"
cur.execute(sql)
print(cur.fetchall())

#Close connection
conn.close()
```

Mise à jour OK en mémoire :

```
>>> (executing file "<tmp 3>")
[(1, 'Gamotte', 'Albert'), (2, 'Pabien', 'Yvon'), (3,
'Computing', 'Claude'), (4, 'Slatable', 'Deborah'), (5
, 'Suffit', 'Sam'), (6, 'Debece', 'Gilles'), (7, 'Debe
ce', 'Aude')]
```

Mais aucune modification sur le disque :

	pid [PK] integer	nom character varying (20)	prenom character varying (20)
1	1	Gamotte	Albert
2	2	Pabien	Yvon
3	3	Computing	Claude
4	4	Slatable	Deborah
5	5	Suffit	Sam
6	6	Debece	Gilles

Attention aux requêtes de mise à jour

En utilisant `commit()` :

```
# Open a cursor to send SQL commands
cur = conn.cursor()

sql = "INSERT INTO public.personne VALUES
(7,'Debece','Aude')"

cur.execute(sql)

# Testing
sql = "SELECT * FROM public.personne"
cur.execute(sql)
print(cur.fetchall())

# COMMIT!
conn.commit()
```

```
#Close connection
conn.close()
```

Mise à jour OK en mémoire :

```
>>> (executing file "<tmp 3>")
[(1, 'Gamotte', 'Albert'), (2, 'Pabien', 'Yvon'), (3,
'Computing', 'Claude'), (4, 'Slatable', 'Deborah'), (5
, 'Suffit', 'Sam'), (6, 'Debece', 'Gilles'), (7, 'Debe
ce', 'Aude')]
```

Répercussion de la mise à jour sur le disque :

	pid [PK] integer	nom character varying (20)	prenom character varying (20)
1	1	Gamotte	Albert
2	2	Pabien	Yvon
3	3	Computing	Claude
4	4	Slatable	Deborah
5	5	Suffit	Sam
6	6	Debece	Gilles
7	7	Debece	Aude

En utilisant SQLAlchemy

`SQLAlchemy` = un outils de correspondance objet-relationnel ou ORM (*Object Relational Mapper*)

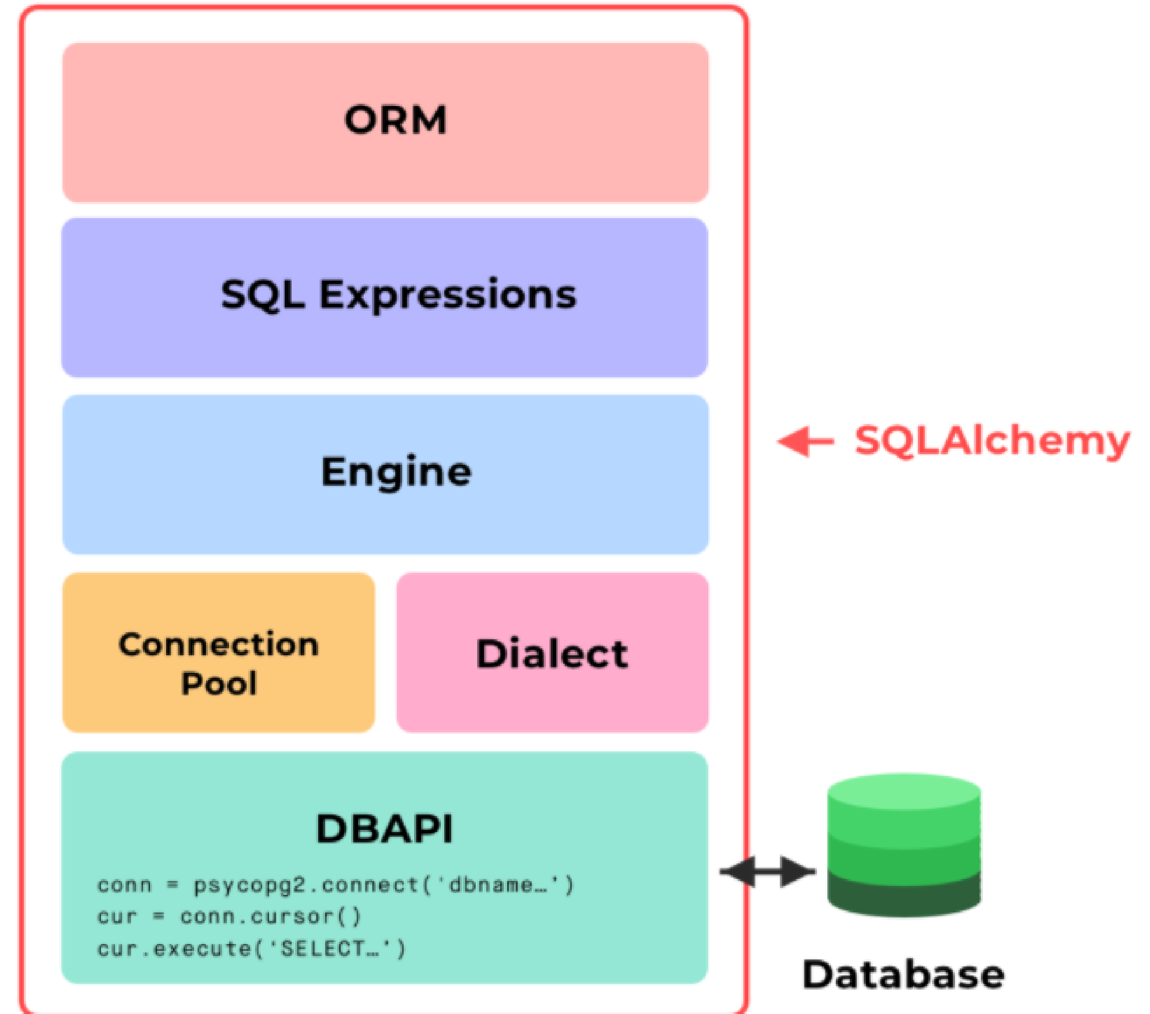
Cf. https://www.sqlalchemy.org/

Un ORM :

- permet à un programme de dialoguer avec un SGBD sans émettre d'ordres SQL
- masque au programme les particularités de chaque SGBD

Pour installer le module :

```
>>> pip install SQLAlchemy
```



En utilisant SQLAlchemy

Possibilité d'utiliser `SQLAlchemy` pour faire du SQL standard :

```
from sqlalchemy import create_engine

engine =
create_engine('postgresql://login:passwd@localhost:5432/data
baseName')
conn = engine.connect()

result = conn.execute("SELECT * FROM public.personne")

rows = result.fetchall()
print("rows:",rows)


table_names = engine.table_names()
print("\ntables : ",table_names)

result.close()
conn.close()
```

```
>>> (executing file "AvecSQLAlchemy.py")
rows: [(1, 'Gamotte', 'Albert'), (2, 'Pabien', 'Yvon
'), (3, 'Computing', 'Claude'), (4, 'Slatable', 'Deb
orah'), (5, 'Suffit', 'Sam'), (6, 'Debece', 'Gilles'
), (7, 'Debece', 'Aude')]

tables :  ['personne', 'ticket', 'tombola', 'membre'
, 'atelier', 'inscription', 'departement', 'ville',
'bureauvote', 'electeur', 'entreprise', 'apprentis',
'apprentissage', 'enfant', 'banque', 'binome', 'pers
on_event', 'test', 'person_email_addr', 'person', 's
alle', 'etudiant', 'events', 'event', 'cours', 'ense
ignant', 'reservation', 'universite', 'batiment', 't
ypeeval', 'relevenotes', 'seance', 'compte', 'vireme
nt', 'formation', 'r1', 'commentaire', 'r2', 'r3', '
station', 'publi', 'consultation', 'amitie', 'utilis
ateur', 'event_person', 'autorisationconsultation',
'bus', 'arretbus']
```

Activer Windows

En utilisant pandas

Possibilité d'utiliser `psycopg2` et `pandas` pour faire du SQL standard :

```
import psycopg2
from configparser import ConfigParser
import pandas as pd
import pandas.io.sql as sqlio
```

```
#Connect to the PostgreSQL database server
conn = None
try:

    # read connection parameters
    params = config()

    # connect to the PostgreSQL server
    print('Connecting to the PostgreSQL database...')
    conn = psycopg2.connect(**params)

    # Testing
    sql = "SELECT * FROM public.personne"
    dat = sqlio.read_sql_query(sql, conn)
    print(dat)

except (Exception, psycopg2.DatabaseError) as error:
        print(error)
finally:
        if conn is not None:
            conn.close()
            print('Database connection closed.')
```

```
>>> (executing file "AccesPostgreSQLPythonPanda.py")
Connecting to the PostgreSQL database...
   pid       nom   prenom
0    1   Gamotte   Albert
1    2    Pabien     Yvon
2    3  Computing   Claude
3    4  Slatable  Deborah
4    5    Suffit      Sam
5    6    Debece   Gilles
6    7    Debece     Aude
Database connection closed.
```

Activer Windows

En utilisant pandas

Possibilité d'utiliser `SQLAlchemy` et `pandas` pour faire du SQL standard :

```
from sqlalchemy import create_engine
import pandas as pd

engine =
create_engine('postgresql://login:passwd@localhost:5432/data
baseName')
conn = engine.connect()

df = pd.read_sql_query("SELECT * FROM public.personne",
engine)

print(df)

conn.close()
```

```
>>> (executing file "AvecSQLAlchemyETPanda.py")
   pid       nom   prenom
0    1   Gamotte   Albert
1    2    Pabien     Yvon
2    3 Computing   Claude
3    4  Slatable  Deborah
4    5    Suffit      Sam
5    6    Debece   Gilles
6    7    Debece     Aude
```

Activer Windows

En utilisant pandas

```
conn = engine.connect()

rs = conn.execute("SELECT * FROM public.personne")

df = pd.DataFrame(rs.fetchall())
df.columns = rs.keys()

print("df.columns :", df.columns)
print("\nmax de pid :", df['pid'].max())
print("\ndf.values :",df.values)

conn.close()
```

```
>>> (executing file "AvecSQLAlchemyETPanda.py")
df.columns : Index(['pid', 'nom', 'prenom'], dtype='
object')

max de pid : 7

df.values : [[1 'Gamotte' 'Albert']
 [2 'Pabien' 'Yvon']
 [3 'Computing' 'Claude']
 [4 'Slatable' 'Deborah']
 [5 'Suffit' 'Sam']
 [6 'Debece' 'Gilles']
 [7 'Debece' 'Aude']]
```

Activer Windows

Liens

- Pour le DB-API MySQL :
  - https://ichi.pro/fr/connexion-a-une-base-de-donnees-python-savoir-comment-se-connecter-a-une-base-de-donnees-198984543081596
  - https://www.w3schools.com/python/python_mysql_getstarted.asp
  - https://www.freecodecamp.org/news/connect-python-with-sql/
- Pour le DB-API PostgresSQL :
  - https://www.postgresqltutorial.com/postgresql-python/
  - https://towardsdatascience.com/python-and-postgresql-how-to-access-a-postgresql-database-like-a-data-scientist-b5a9c5a0ea43
- Pour le DB-API SQLite :
  - https://pythonspot.com/python-database-programming-sqlite-tutorial/

Liens

- Pour SQLAlchemy:
  - https://www.sqlalchemy.org/
  - https://hackersandslackers.com/python-database-management-sqlalchemy/
  - https://shravan-kuchkula.github.io/sqlalchemy-layers/
  - https://tahe.developpez.com/tutoriels-cours/python-flask-2020/?page=utilisation-de-l-orm-sqlalchemy
- Pour Pandas :
  - https://pandas.pydata.org/docs/getting_started/comparison/comparison_with_sql.html
  - https://pythontic.com/pandas/serialization/postgresql
  - https://datatofish.com/sql-to-pandas-dataframe/
  - https://www.sqlshack.com/exploring-databases-in-python-using-pandas/