

# **Application de bases de données, principes et exemples avec Python**

# Table des matières

|                                                                                          |           |
|------------------------------------------------------------------------------------------|-----------|
| <b>Objectifs</b>                                                                         | <b>3</b>  |
| <b>Introduction</b>                                                                      | <b>4</b>  |
| <b>I - Cours</b>                                                                         | <b>5</b>  |
| 1. Applications et bases de données.....                                                 | 5         |
| 1.1. Architecture générale d'une application de base de données .....                    | 5         |
| 1.2. Exemple d'application de base de données .....                                      | 6         |
| 1.3. Méthode générale d'accès à une BD en écriture par un langage de programmation ..... | 8         |
| 1.4. Exemple d'accès à une BD en écriture par un langage de programmation .....          | 9         |
| 1.5. Méthode générale d'accès à une BD en lecture par un langage de programmation.....   | 10        |
| 1.6. Exemples d'accès à une BD en lecture par un langage de programmation.....           | 11        |
| 2. Application avec Python et PostgreSQL .....                                           | 12        |
| 2.1. Psycopg : module Python pour PostgreSQL .....                                       | 12        |
| 2.2. Connexion Python-PostgreSQL .....                                                   | 13        |
| 2.3. Écriture dans PostgreSQL avec Python .....                                          | 13        |
| 2.4. Exercice .....                                                                      | 14        |
| 2.5. Lecture de PostgreSQL avec Python.....                                              | 14        |
| 2.6. Exercice .....                                                                      | 15        |
| <b>II - Exercice</b>                                                                     | <b>16</b> |
| 1. Exercice .....                                                                        | 16        |
| <b>Glossaire</b>                                                                         | <b>18</b> |
| <b>Abréviations</b>                                                                      | <b>19</b> |

# Objectifs

---



Savoir utiliser une base de données depuis un langage de programmation

# Introduction

---



Si les SGBD offrent les technologies de modélisation et de gestion des données, ils nécessitent la plupart du temps d'être interfacés avec des applications qui fournissent un accès orienté métier aux utilisateurs, notamment à travers des IHM évoluées. Même des systèmes comme Oracle ou PostgreSQL qui proposent des langages procéduraux intégrés au dessus de la couche SQL (comme PL/pgSQL), ne sont pas auto-suffisants. Les langages évolués comme Python, Java ou C++ sont couramment utilisés pour implémenter la couche applicative d'exploitation des BD.



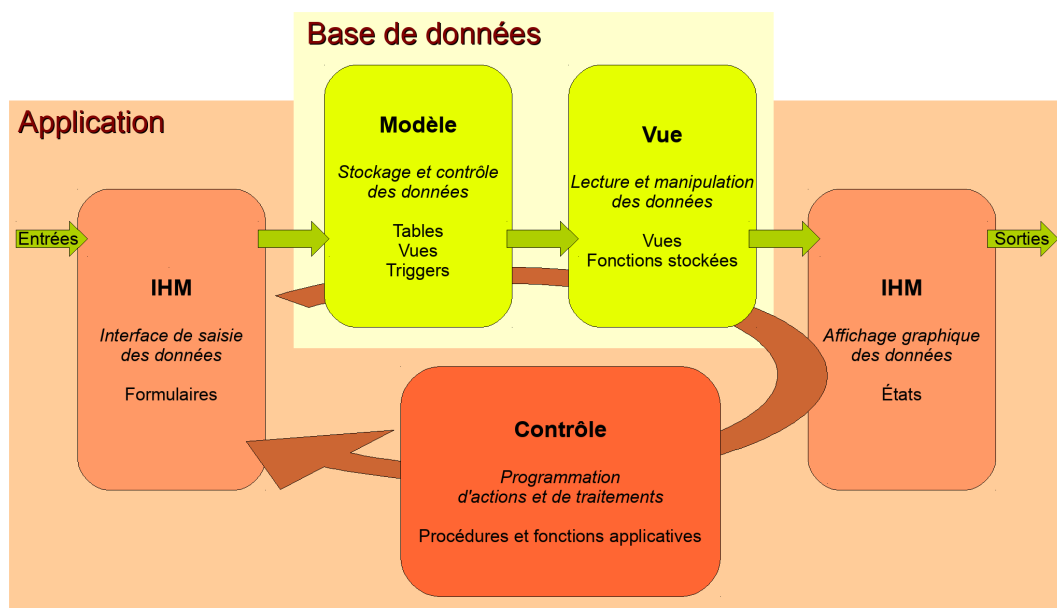
## 1. Applications et bases de données

### 1.1. Architecture générale d'une application de base de données



Une application de base de données comporte :

1. Une base de données : Elle pose le modèle de données, stocke les données, définit des vues sur les données (préparation des modalités de lecture)
2. Une application : Elle définit les interfaces homme-machine pour écrire et lire les données et contient le programme informatique de traitement des données avant insertion dans la base ou présentation à l'utilisateur.



Architecture générale d'une application de base de données

### Procédure générale de développement



1. Développement de la **BD\*** : Conception en **UML\*** ou **EA\***, puis traduction en **R\*** (ou **NoSQL\***), puis implémentation **SQL\***
  - Créer les **tables**
  - Ajouter des **vues** et **triggers** (lorsque le SGBD le permet, comme Oracle ou PostgreSQL) pour implémenter les contraintes complexes non exprimables en SQL

- Ajouter les **vues** utiles pour simplifier l'accès aux données en lecture
  - Implémenter des **fonctions stockées** (lorsque le SGBD le permet) permettant de renvoyer les valeurs calculées prévues par le modèle conceptuel
2. Développement de l'application : traitements, formulaires et états
- Les traitements sont réalisés dans le langage applicatif choisi (PHP, Python, Java, C++...)
  - Les *IHM\** sont parfois réalisées avec le même langage, mais de plus en plus souvent, on privilégie des IHM Web (HTML, CSS, JavaScript)

### Exemple de technologies

? Exemple

- BD :
  - SQL pour les tables et les vues
  - PL/SQL (Oracle), PL/pgSQL (PostgreSQL) pour les triggers et les fonctions stockées
- Formulaires
  - PHP, HTML (balise `<form>`), HTTP/POST (Web)
  - Formulaires Access
  - Input Python
- États
  - PHP, HTML (`<table>`), HTTP/GET (Web)
  - États Access
  - Print Python
- Contrôle :
  - Python, PHP, JavaScript, Java, C++, Macros & VBA (Access)

## 1.2. Exemple d'application de base de données

### Modèle

? Exemple

```
1 CREATE TABLE philosopher (
2 surname TEXT PRIMARY KEY,
3 givenname TEXT,
4 century INTEGER);
```

### Vue

? Exemple

```
1 CREATE VIEW v_philosopher (name, century) AS
2 SELECT
3 COALESCE(givenname, '') ||
4 CASE WHEN givenname IS NOT NULL THEN ' ' ELSE '' END
5 || surname,
6 century || 'e'
7 FROM philosopher;
```

## Contrôle (menu.py)

? Exemple

```

1 #!/usr/bin/python3
2
3 import psycpg2
4 import forms
5 import reports
6
7 HOST = "localhost"
8 USER = "me"
9 PASSWORD = "secret"
10 DATABASE = "mydb"
11
12 conn = psycpg2.connect("host=%s dbname=%s user=%s password=%s" % (HOST, DATABASE,
    USER, PASSWORD))
13
14 choice = '1'
15 while choice == '1' or choice == '2':
16     print ("Pour ajouter un philosophe à la base, entrez 1")
17     print ("Pour voir la liste des philosophes, entrez 2")
18     print ("Pour sortir, entrez autre chose")
19     choice = input()
20     if choice == '1':
21         forms.addPhilo(conn)
22     if choice == '2':
23         reports.printPhilo(conn)
24     print(choice)
25
26 conn.close()

```

Terminal - stc@hal9017: ~/pyt

Fichier Édition Affichage Terminal Onglets Aide

```

stc@hal9017:~/pyt$ ./menu.py
Pour ajouter un philosophe à la base, entrez 1
Pour voir la liste des philosophes, entrez 2
Pour sortir, entrez autre chose

```

## Formulaire

? Exemple

```

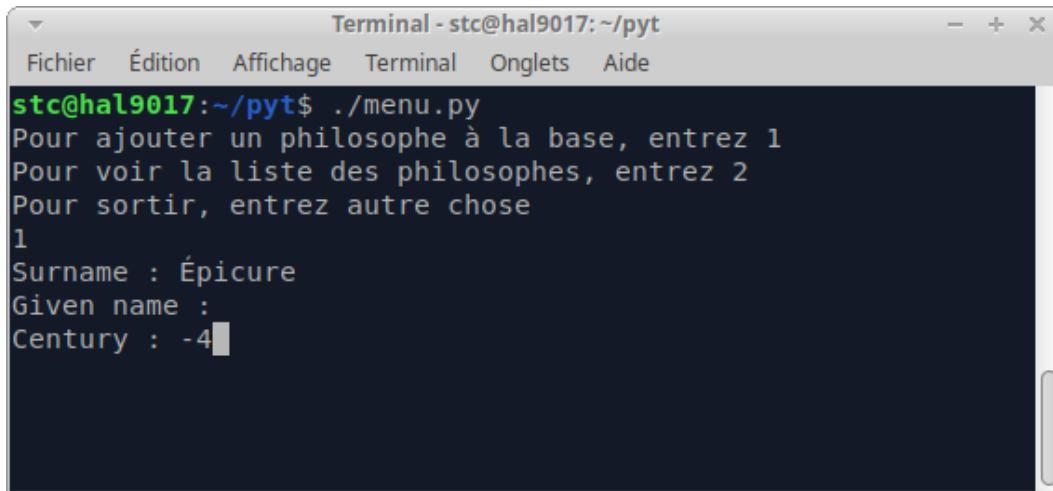
1 #!/usr/bin/python3
2
3 def quote(s):
4     if s:
5         return '\'' + s + '\''
6     else:
7         return 'NULL'
8
9 def addPhilo(conn):
10     surname = quote(input("Surname : "))
11     givenname = quote(input("Given name : "))
12     century = int(input("Century : "))
13     # Connect, execute SQL, close

```

```

14 cur = conn.cursor()
15 sql = "INSERT INTO philosopher VALUES (%s, %s, %i)" % (surname, givenname,
century)
16 cur.execute(sql)
17 conn.commit()

```



```

Terminal - stc@hal9017: ~/pyt
Fichier  Édition  Affichage  Terminal  Onglets  Aide
stc@hal9017:~/pyt$ ./menu.py
Pour ajouter un philosophe à la base, entrez 1
Pour voir la liste des philosophes, entrez 2
Pour sortir, entrez autre chose
1
Surname : Épicure
Given name :
Century : -4

```

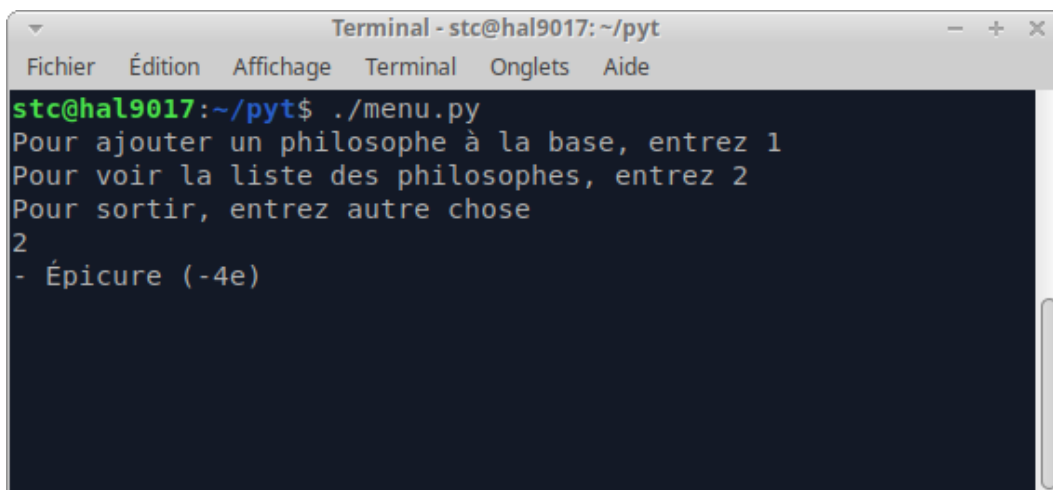
## État (report.py)

? Exemple

```

1 #!/usr/bin/python3
2
3 def printPhilo(conn):
4     # Connect and retrieve data
5     cur = conn.cursor()
6     sql = "SELECT name, century FROM v_philosopher"
7     cur.execute(sql)
8     # Fetch data line by line
9     raw = cur.fetchone()
10    while raw:
11        print ("- %s (%s)" % (raw[0], raw[1]))
12        raw = cur.fetchone()

```



```

Terminal - stc@hal9017: ~/pyt
Fichier  Édition  Affichage  Terminal  Onglets  Aide
stc@hal9017:~/pyt$ ./menu.py
Pour ajouter un philosophe à la base, entrez 1
Pour voir la liste des philosophes, entrez 2
Pour sortir, entrez autre chose
2
- Épicure (-4e)

```

## 1.3. Méthode générale d'accès à une BD en écriture par un langage de programmation

✂ Méthode

1. Connexion à la base de données et récupération d'un identifiant de connexion
2. Écriture de la requête d'insertion ou de mise à jour de données



3. Exécution de la requête sur la connexion ouverte (et éventuelle récupération d'un résultat d'exécution, par exemple : le nombre de lignes modifiés, ou un booléen selon que la requête s'est exécutée ou non avec succès)
4. Test du résultat et dialogue avec l'utilisateur an cas d'erreur ou de résultat incorrect
5. Clôture de la connexion

### Pseudo-code



Méthode

```

1 // Connexion à la base de données
2 host = "foo.fr"
3 port = "6666"
4 data = "myDatabase"
5 user = "me"
6 pass = "secret"
7 conn = CONNECT(host, port, data, user, pass)

1 // Connexion à la base de données
2 host = "foo.fr"
3 port = "6666"
4 data = "myDatabase"
5 user = "me"
6 pass = "secret"
7 conn = CONNECT(host, port, data, user, pass)

1 // Écriture de la requête
2 sql= "INSERT INTO t (a) VALUES (1) ;"

1 // Exécution de la requête
2 result = QUERY(conn, sql)

1 // Test du résultat
2 IF (NOT result) THEN MESSAGE("Échec de l'exécution")

1 // Clôture de la connexion
2 CLOSE (conn)

```

## 1.4. Exemple d'accès à une BD en écriture par un langage de programmation

### Python (sans gestion d'erreur)



Exemple

```

1 #!/usr/bin/python3
2
3 import psycopg2
4
5 HOST = "localhost"
6 USER = "me"
7 PASSWORD = "secret"
8 DATABASE = "mydb"
9
10 # Open connection
11 conn = psycopg2.connect("host=%s dbname=%s user=%s password=%s" % (HOST, DATABASE,
    USER, PASSWORD))
12
13 # Open a cursor to send SQL commands
14 cur = conn.cursor()
15
16 # Execute a SQL INSERT command
17 sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, -4)"
18 cur.execute(sql)
19 conn.commit()
20
21 # Close connection

```

22 `conn.close()`

Remarque

Ce code n'intègre aucune gestion d'erreur. En Python on pourra introduire une gestion d'exceptions pour traiter les cas d'erreur (tentative d'insertion d'un doublon dans une clé par exemple).

## 1.5. Méthode générale d'accès à une BD en lecture par un langage de programmation

### Pointeur sur un résultat de requête



Définition

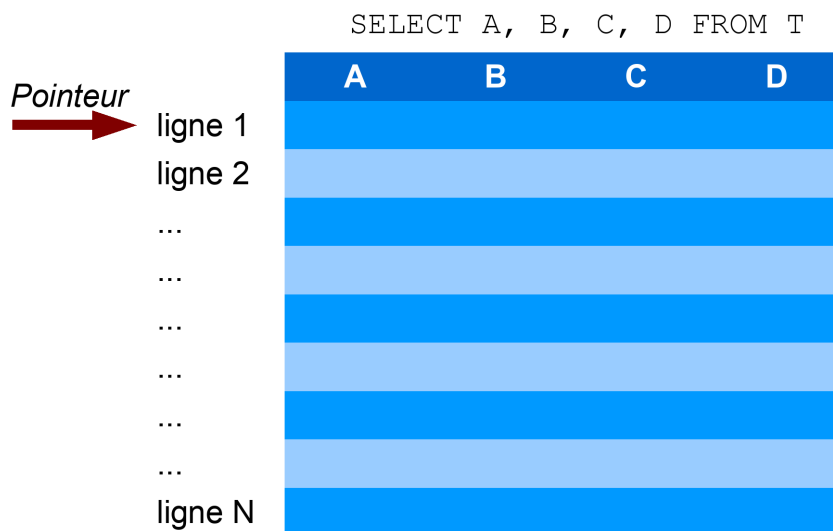
Lorsqu'un programme exécute une requête dans une BD, il récupère un pointeur sur un résultat de requête permettant de parcourir le tableau résultant de l'exécution de la requête ligne par ligne.

En effet :

- Une requête SQL retourne toujours un tableau (même si parfois il n'a qu'une ligne et une colonne), c'est le fondement du modèle relationnel
- En général il n'est pas souhaitable que ce tableau soit transmis directement sous la forme d'une structure en mémoire (`array`), car le résultat d'une requête peut être très volumineux (et ne pas tenir en mémoire primaire)
- La solution générale est donc que la BD fournisse le résultat ligne par ligne

Le pointeur permet de :

- Retourner la ligne pointée sous la forme d'un vecteur (tableau à une seule ligne)
- Passer à la ligne suivante
- Savoir si l'on a atteint la fin du tableau (ligne N)



*Pointeur sur un résultat de requête*



Méthode

1. Connexion à la base de données et récupération d'un identifiant de connexion
2. Écriture de la requête de sélection
3. Exécution de la requête sur la connexion ouverte et récupération d'un pointeur sur un résultat de requête

4. Parcours du pointeur dans une boucle permettant de rapporter et traiter (afficher par exemple) le tableau ligne par ligne
5. Clôture de la connexion

### Pseudo-code



Méthode

```

1 // Connexion à la base de données
2 host = "foo.fr"
3 port = "6666"
4 data = "myDatabase"
5 user = "me"
6 pass = "secret"
7 conn = CONNECT(host, port, data, user, pass)

1 // Écriture de la requête
2 sql = "SELECT a, b FROM t;"

1 // Exécution de la requête
2 pointeur = QUERY (conn, sql)

1 // Traitement du résultat
2 WHILE (pointeur)
3   FETCH pointeur IN result
4   PRINT result[1]
5   PRINT result[2]
6   NEXT pointeur
7 END WHILE

1 // Clôture de la connexion
2 CLOSE (conn)

```

## 1.6. Exemples d'accès à une BD en lecture par un langage de programmation

### Python (sans gestion d'erreur)



Exemple

```

1 #!/usr/bin/python3
2
3 import psycopg2
4
5 HOST = "localhost"
6 USER = "me"
7 PASSWORD = "secret"
8 DATABASE = "mydb"
9
10 # Open connection
11 conn = psycopg2.connect("host=%s dbname=%s user=%s password=%s" % (HOST, DATABASE,
    USER, PASSWORD))
12
13 # Open a cursor to send SQL commands
14 cur = conn.cursor()
15
16 # Execute a SQL SELECT command
17 sql = "SELECT name, century FROM v_philosopher"
18 cur.execute(sql)
19
20 # Fetch data line by line
21 raw = cur.fetchone()
22 while raw:
23     print (raw[0], raw[1])
24     raw = cur.fetchone()

```



Ce code n'intègre aucune gestion d'erreur. En Python on pourra introduire une gestion d'exceptions pour traiter les cas d'erreur (requête SELECT mal formée par exemple).

## 2. Application avec Python et PostgreSQL

### 2.1. Psycopg : module Python pour PostgreSQL

#### Pilote de SGBD



Un langage de programmation a besoin d'un pilote (*driver*) spécifique pour interagir avec un SGBD. L'usage de ce pilote varie selon les langages et les pilotes, mais c'est une opération simple consistant à ajouter un librairie à son programme.



Pour connecter Python et PostgreSQL, il existe plusieurs pilotes (modules Python), on utilisera dans le cadre de ce cours le pilote *Psycopg2*.

« Psycopg is a PostgreSQL adapter for the Python programming language. It is a wrapper for the libpq, the official PostgreSQL client library. »



```
1#!/usr/bin/python3
2import psycopg2
```

#### Documentation



<https://www.psycopg.org/docs>

#### Installation



```
1pip3 install --user psycopg2-binary
```

#### Pour mettre à jour pip3



```
1pip3 install --user --upgrade pip
```

#### Pour installer pip3 sous Debian



```
1apt-get install python3-pip
```

## 2.2. Connexion Python-PostgreSQL

### § Syntaxe

```
1#!/usr/bin/python3
2import psycopg2
3
4# Connect to an existing database
5conn = psycopg2.connect("host=%s dbname=%s user=%s password=%s" % ("localhost",
    "database", "user", "password"))
```

### Préparation de la base PostgreSQL

### ? Exemple

Préparer une base de données PostgreSQL *mydb* appartenant à l'utilisateur *me* avec le mot de passe *secret*. Ajouter une table *t* avec les attributs *a* et *b*.

```
1#!/usr/bin/bash
2psql -c "CREATE USER me WITH PASSWORD 'secret';"
3psql -c "CREATE DATABASE mydb OWNER=me";
4psql -h localhost -U me -d mydb -c "CREATE TABLE t (a VARCHAR,b INTEGER);"
```

### Connexion à la base de données en Python (connect.py)

### ? Exemple

```
1#!/usr/bin/python3
2
3import psycopg2
4
5HOST = "localhost"
6USER = "me"
7PASSWORD = "secret"
8DATABASE = "mydb"
9
10# Connect to an existing database
11conn = psycopg2.connect("host=%s dbname=%s user=%s password=%s" % (HOST, DATABASE,
    USER, PASSWORD))
12print (conn)
13
14# Close connection
15conn.close()
16print (conn)

1$ ./connect.py
2<connection object at 0x7f752e56fe88; dsn: 'user=me password=xxx dbname=mydb
    host=localhost', closed: 0>
3<connection object at 0x7f752e56fe88; dsn: 'user=me password=xxx dbname=mydb
    host=localhost', closed: 1>
```

### Documentation du module psycopg2

### + Complément

<https://www.psycopg.org/docs/usage.html>

## 2.3. Écriture dans PostgreSQL avec Python

### Insertion (insert.py)

### ? Exemple

```
1#!/usr/bin/python3
2
3import psycopg2
4
5HOST = "localhost"
```

```

6 USER = "me"
7 PASSWORD = "secret"
8 DATABASE = "mydb"
9
10 # Open connection
11 conn = psycopg2.connect("host=%s dbname=%s user=%s password=%s" % (HOST, DATABASE,
    USER, PASSWORD))
12
13 # Open a cursor to send SQL commands
14 cur = conn.cursor()
15
16 # Execute a SQL INSERT command
17 sql = "INSERT INTO t VALUES ('Hello',1)"
18 cur.execute(sql)
19
20 # Commit (transactionnal mode is by default)
21 conn.commit()
22
23 # Close connection
24 conn.close()

1 $ ./insert.py
2 [('Hello', 1)]

```

### Documentation du module psycopg2



<https://www.psycopg.org/docs/usage.html>

## 2.4. Exercice

Soit une base de données composée d'une unique table `t`, contenant un unique attribut `a` de type chaîne de caractère.

```
1 CREATE TABLE t (a TEXT PRIMARY KEY);
```

Compléter le programme Python permettant d'ajouter la valeur *something* dans la base de données implémentée sous PostgreSQL.

```
#!/usr/bin/python3
```

```
import psycopg2
```

```
conn = psycopg2.connect("host=%s dbname=%s user=%s password=%s" % ("localhost", "mydb", "me",
"secret"))
```

```
cur = conn.cursor()
```

```
sql = "INSERT INTO _____ VALUES ( _____ )"
_____
```

```
_____.execute( _____ )
```

```
conn. _____
```

```
conn. _____
```

## 2.5. Lecture de PostgreSQL avec Python

### Sélection (select.py)



```

1 #!/usr/bin/python3
2
3 # http://initd.org/psycopg/docs/usage.html
4
5 import psycopg2
6

```

```

7 HOST = "localhost"
8 USER = "me"
9 PASSWORD = "secret"
10 DATABASE = "mydb"
11
12 # Open connection
13 conn = psycopg2.connect("host=%s dbname=%s user=%s password=%s" % (HOST, DATABASE,
    USER, PASSWORD))
14
15 # Open a cursor to send SQL commands
16 cur = conn.cursor()
17
18 # Execute a SQL SELECT command
19 sql = "SELECT * FROM t"
20 cur.execute(sql)
21
22 # Fetch data line by line
23 raw = cur.fetchone()
24 while raw:
25     print (raw[0])
26     print (raw[1])
27     raw = cur.fetchone()
28
29 # Close connection
30 conn.close()
31
1 $ ./select.py
2 Hello
3 1

```

## Documentation du module psycopg2



<https://www.psycopg.org/docs/usage.html>

## 2.6. Exercice

Soit une base de données composée d'une unique table `t`, contenant un unique attribut `a`.

```
1 CREATE TABLE t (a TEXT PRIMARY KEY);
```

Compléter le programme Python permettant d'interroger la base de données implémentée sous PostgreSQL.

```
#!/usr/bin/python3
```

```
import psycopg2
```

```
conn = psycopg2.connect("host=%s dbname=%s user=%s password=%s" % ("localhost", "mydb", "me",
    "secret"))
```

```
cur = conn.cursor()
```

```
sql = "SELECT _____ FROM _____"
```

```
_____.execute(_____)
```

```
data = _____
```

```
while _____:
```

```
    print(_____)
```

```
    data = _____
```

```
conn._____
```

# Exercice



## 1. Exercice

[45 min]

Soit la base de données *country* définie par le code SQL ci-après.

```
1 CREATE TABLE country (  
2   name VARCHAR,  
3   gdp INTEGER,  
4   PRIMARY KEY (name)  
5 );  
6  
7 CREATE TABLE region (  
8   country VARCHAR,  
9   name VARCHAR,  
10  population INTEGER,  
11  area INTEGER,  
12  PRIMARY KEY (country, name),  
13  FOREIGN KEY (country) REFERENCES country(name)  
14 );  
15  
16 CREATE TABLE log (  
17   id INTEGER,  
18   country VARCHAR NOT NULL,  
19   region VARCHAR NOT NULL,  
20   date DATE NOT NULL,  
21   temperature DECIMAL(3,1) NOT NULL,  
22   humidity DECIMAL(3,1) NOT NULL,  
23   PRIMARY KEY (id),  
24   UNIQUE (country, region, date),  
25   FOREIGN KEY (country, region) REFERENCES region (country, name)  
26 );  
  
1 INSERT INTO country VALUES ('France', 38128);  
2 INSERT INTO country VALUES ('Norway', 70392);  
3 INSERT INTO country VALUES ('Spain', 26609);  
4  
5 INSERT INTO region VALUES ('France', 'Hauts-de-France', 5973098, 31813);  
6 INSERT INTO region VALUES ('France', 'Île-de-France', 12005077, 12012);  
7 INSERT INTO region VALUES ('France', 'Normandie', 3322757, 29906);  
8  
9 INSERT INTO log VALUES (1, 'France', 'Hauts-de-France',  
10  TO_DATE('23052017', 'DDMMYYYY'), 21.1, 51.4);  
11 INSERT INTO log VALUES (2, 'France', 'Hauts-de-France',  
12  TO_DATE('24052017', 'DDMMYYYY'), 23.1, 62.4);  
13 INSERT INTO log VALUES (3, 'France', 'Hauts-de-France',  
14  TO_DATE('25052017', 'DDMMYYYY'), 17.5, 71.1);
```

### Question 1

Écrire un programme Python qui affiche la liste des régions de la base de données, en les triant par ordre inverse de densité de population (la densité est la population divisée par la surface).



**Question 2**

Proposez un programme Python qui affiche *Yes* s'il existe un pays qui commence par *Fr*, et *No* sinon.

**Question 3**

Proposez un programme Python qui affiche tous les pays qui commencent par *Fr*.

# Glossaire

---



## NoSQL

NoSQL désigne une famille de systèmes de gestion de base de données (SGBD) qui s'écarte du paradigme classique des bases relationnelles. (Wikipédia)

# Abréviations

---



**BD** : Base de Données

**E-A** : Entité-Association

**IHM** : Interaction Homme Machine ou Interface Homme Machine

**R** : Relationnel

**SQL** : Structured Query Language

**UML** : Unified Modeling Language