Application de bases de données avec Python

Table des matières

Objectifs

Savoir utiliser une base de données avec Python

Cours

1. Gestion des erreurs SQL

1.1. Notion d'exception

Exception *Définition*

Une exception ou erreur est générée automatiquement par le programme Python lorsque que quelque chose se passe mal, par exemple :

- lorsqu'on essaye de se connecter à une base de données qui n'existe pas
- lorsqu'on essaie d'insérer une clé qui existe déjà.
- lorsqu'on insère une valeur du mauvais type
- ...

Par défaut une exception interrompt immédiatement le programme.

Insertion de valeur du mauvais type *Exemple*

```
#!/usr/bin/python3

import psycopg2

conn = psycopg2.connect("host='localhost' dbname='mydb' user='me'
password='secret'")
cur = conn.cursor()

sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, 'Quatrième avant JC')"
cur.execute(sql)
conn.commit()

print("Fin du programme")

conn.close()
```

```
Traceback (most recent call last):
  File "./sortie1.py", line 14, in <module>
    cur.execute(sql)
psycopg2.DataError: invalid input syntax for integer: "Premier"
LINE 1: INSERT INTO philosopher VALUES ('Épicure', NULL, 'Premier')
```

Attention

La ligne `print("Fin du programme")` n'est jamais atteinte, l'exception a entraîner l'interruption du programme.

1.2. Gestion d'exception

Méthode

Il est possible d'interception une exception et ainsi :

- empêcher le programme de s'arrêter ;
- et faire des traitements personnalisés en cas d'erreur.

Try/Except *Syntaxe*

```
try:
  # code pouvant générer une erreur
except type_de_l_erreur as e:
  # traitements spécifiques
  print(e) # affichage de l'erreur
```

Insertion de valeur du mauvais type (psycopg2.DataError) *Exemple*

```
#!/usr/bin/python3

import psycopg2

conn = psycopg2.connect("host='localhost' dbname='mydb' user='me' password='secret'")
cur = conn.cursor()

try:
  sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, 'Premier')"
  cur.execute(sql)
  conn.commit()
except psycopg2.DataError as e:
  print("Message personnalisé : Contrainte non respectée")
  print("Message système :", e)

print("Fin du programme")

conn.close()
```

```
Message personnalisé : Contrainte non respectée
Message système : invalid input syntax for integer: "Premier"
LINE 1: INSERT INTO philosopher VALUES ('Épicure', NULL, 'Premier')
                                                         ^
Fin du programme
```

1.3. Exceptions et transactions

Rappel

Transactions en SQL *(cf. p.12)*

Duplication de clé

Exemple

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

import psycopg2

conn = psycopg2.connect("host='localhost' dbname='mydb' user='me' password='secret'")
cur = conn.cursor()

sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, -4)"
cur.execute(sql)
conn.commit()

sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, -4)"
cur.execute(sql)
conn.commit()

print("Fin du programme")

conn.close()
```

```
Traceback (most recent call last):
  File "./sortie1.py", line 18, in <module>
    cur.execute(sql)
psycopg2.IntegrityError: duplicate key value violates unique constraint "philosopher_pkey"
DETAIL:  Key (surname)=(Épicure) already exists.
```

Duplication de clé (gestion de l'exception)

Exemple

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

import psycopg2

conn = psycopg2.connect("host='localhost' dbname='mydb' user='me' password='secret'")
cur = conn.cursor()

try:
  sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, -4)"
  cur.execute(sql)
  conn.commit()
except psycopg2.IntegrityError as e:
  print("Message système :", e)

try:
  sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, -4)"
  cur.execute(sql)
  conn.commit()
except psycopg2.IntegrityError as e:
  print("Message système :", e)

print("Fin du programme")

conn.close()
```

```
Message système : duplicate key value violates unique constraint "philosopher_pkey"
DETAIL:  Key (surname)=(Épicure) already exists.

Traceback (most recent call last):
  File "./sortie1.py", line 22, in <module>
```

```
    cur.execute(sql)
psycopg2.InternalError: current transaction is aborted, commands ignored until end
of transaction block
```

Attention

L'erreur de duplication est bien gérée, mais une seconde exception est levée car lors de la seconde insertion, la première transaction est encore en cours, car l'instruction `conn.commit()` ne s'est pas exécutée.

Méthode

On va ajouter un ordre de *rollback* pour terminer la transaction en cas d'erreur.

Try/Except

Syntaxe

```
try:
  # code pouvant générer une erreur
except type_de_l_erreur as e:
  # traitements spécifiques
  print(e) # affichage de l'erreur
  conn.rollback()
```

Exemple

```
#!/usr/bin/python3

import psycopg2

conn = psycopg2.connect("host='localhost' dbname='mydb' user='me'
password='secret'")
cur = conn.cursor()

try:
  sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, -4)"
  cur.execute(sql)
  conn.commit()
except psycopg2.IntegrityError as e:
  conn.rollback()
  print("Message système :", e)

try:
  sql = "INSERT INTO philosopher VALUES ('Épicure', NULL, -4)"
  cur.execute(sql)
  conn.commit()
except psycopg2.IntegrityError as e:
  conn.rollback()
  print("Message système :", e)

print("Fin du programme")

conn.close()
```

2. Bonne pratiques

2.1. Insérer des entiers

Méthode

La fonction *input* retourne une chaîne de caractère.

Afin d'obtenir des entiers on utilise la fonction de conversion de type `int()`.

Saisie d'un entier

Syntaxe

```
#!/usr/bin/python3

i = int(input("Entrez un entier : "))
```

Exemple

```
#!/usr/bin/python3

import psycopg2

conn = psycopg2.connect("host='localhost' dbname='mydb' user='me'
password='secret'")
cur = conn.cursor()

century = int(input("Entrez un entier : "))

try:
  sql = "INSERT INTO philosopher VALUES ('Platon', NULL, %i)" % century
  cur.execute(sql)
  conn.commit()
except psycopg2.IntegrityError as e:
  conn.rollback()
  print("Message système :", e)

conn.close()
```

Attention

Si la valeur entrée n'est pas convertible en entier, alors le programme lève une exception (et s'interrompt si l'exception n'est pas gérée).

3. Compléments

3.1. INSERT, DELETE, UPDATE avec valeur de retour

La clause SQL RETURNING à la fin d'une instruction INSERT, UPDATE ou DELETE, permet de retourner un résultat de requête (qui peut être lors traité comme le résultat d'un SELECT).

Syntaxe

```
sql = "INSERT INTO t (a, b, c) VALUES (1, 2, 3) RETURNING a"
cur.execute(sql)
res = cur.fetchone()[0]
```

Exemple

```
#!/usr/bin/python3

import psycopg2

conn = psycopg2.connect("host='localhost' dbname='mydb' user='me'
password='secret'")
cur = conn.cursor()

try:
  sql = "INSERT INTO philosopher VALUES ('Aristote', NULL, -4) RETURNING surname"
  cur.execute(sql)
  res = cur.fetchone()[0]
  print(res)
  conn.commit()
except psycopg2.IntegrityError as e:
  conn.rollback()
  print("Message système :", e)

conn.close()
```

Méthode

Cette fonction est très utile pour récupérer la valeur d'une clé artificielle générée automatiquement (par une séquence typiquement).

3.2. Créer un menu

Exemple

```
#!/usr/bin/python3

import psycopg2
import forms
import reports

HOST = "localhost"
USER = "me"
PASSWORD = "secret"
DATABASE = "mydb"

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

choice = '1'
while choice == '1' or choice == '2':
  print ("Pour ajouter un philosophe à la base, entrez 1")
  print ("Pour voir la liste des philosophes, entrez 2")
  print ("Pour sortir, entrez autre chose")
  choice = input()
  if choice == '1':
    forms.addPhilo(conn)
  if choice == '2':
    reports.printPhilo(conn)
  print(choice)

conn.close()
```

3.3. Fetchall

La méthode *Fetchall* est une alternative à *Fetchone*, qui permet de récupérer d'un coup l'ensemble du résultat d'une requête. Lorsque le volume attendu est suffisamment faible pour tenir en mémoire, c'est une alternative intéressante, le résultat est plus facile à manipuler, et les performances sont améliorées (suppression des multiples accès pour accéder aux données une par une).

Exemple

```
#!/usr/bin/python3

import psycopg2

conn = psycopg2.connect("host='localhost' dbname='mydb' user='me' password='secret'")
cur = conn.cursor()

sql = "SELECT name, century FROM v_philosopher"
cur.execute(sql)
res = cur.fetchall()

print(res)
```

```
[('Aristote', '-4e'), ('Epicure', '-4e'), ('Platon', '-5e')]
```

Exemple

```
#!/usr/bin/python3

import psycopg2

conn = psycopg2.connect("host='localhost' dbname='mydb' user='me' password='secret'")
cur = conn.cursor()

sql = "SELECT name, century FROM v_philosopher"
cur.execute(sql)
res = cur.fetchall()

for raw in res:
  print (raw[0], raw[1])
```

```
Aristote -4e
Epicure -4e
Platon -5e
```

Exercice

1. Exercice

Question

Améliorer le programme Python ci-après :

- Intégrer une gestion d'exception pour la phase de connexion
- Utiliser *fetchall* à la place de *fetchone*
- Afficher – `Île-de-France (d=999)` à la place de `Île-de-France 999`

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

import psycopg2

conn = psycopg2.connect("dbname='mydb' user='me' host='localhost' password='secret'")

sql = "SELECT name, population/area AS density FROM region ORDER BY density DESC";

resultat = conn.cursor()
resultat.execute(sql)

ligne = resultat.fetchone()
while ligne:
        name = ligne[0]
        density = ligne[1]
        print(name, density)
        ligne = resultat.fetchone()
```

Contenus annexes

1. Transactions en SQL

Introduction

Le langage *SQL*∗ fournit trois instructions pour gérer les transactions.

Début d'une transaction

§ *Syntaxe*

```
1 BEGIN TRANSACTION (ou BEGIN) ;
```

Cette syntaxe est optionnelle (voire inconnue de certains *SGBD*∗), une transaction étant débutée de façon **implicite** dès qu'instruction est initiée sur la *BD*∗.

Fin correcte d'une transaction

§ *Syntaxe*

```
1 COMMIT TRANSACTION (ou COMMIT) ;
```

Cette instruction SQL signale la fin d'une transaction couronnée de succès. Elle indique donc au gestionnaire de transaction que l'unité logique de travail s'est terminée dans un état cohérent est que les données peuvent effectivement être modifiées de façon durable.

Fin incorrecte d'une transaction

§ *Syntaxe*

```
1 ROLLBACK TRANSACTION (ou ROLLBACK) ;
```

Cette instruction SQL signale la fin d'une transaction pour laquelle quelque chose s'est mal passé. Elle indique donc au gestionnaire de transaction que l'unité logique de travail s'est terminée dans un état potentiellement incohérent et donc que les données ne doivent pas être modifiées en annulant les modifications réalisées au cours de la transaction.

Programme

Remarque

Un programme est généralement une séquence de plusieurs transactions.

Abréviations

BD : Base de Données
SGBD : Système de Gestion de Bases de Données
SQL : Structured Query Language

Index