# Работа с бази данни в Python

д-р Филип Андонов

6 юли 2022 г.

# Съдържание

# SQL API

Съществуват множество SQL бази данни и част от тях имат клиентски модули в Python. Тъй като обаче API се различават, смяната на СУБД води до необходимост от промяна на кода. Решението на проблема е в стандартен API за бази данни, което трябва да се реализира от модули, осъществяващи достъп до бази данни Текущата версия на API е 2.0

# SQL API

Всеки модул, съвместим с DB API, версия 2.0 трябва да има три глобални променливи, които да описват особеностите на модула.

| Променлива | Употреба |
|---|---|
| apilevel | версията на Python DB API |
| threadsafety | колко безопасен за работа с нишки е модулът |
| paramstyle | какъв стил на параметрите се използва в SQL заявките |

# SQL API

API level (apilevel) е символна константа, даваща версията на използваното API Версията на DB API може да бъде или '1.0' или '2.0'. Ако променливата не е налична, модулът не е съвместим с версия 2.0 и трябва да предположите че е версия 1.0

# SQL API

threadsafety е цяло число между 0 и 3 включително.
Стойност 0 означава, че нишките не могат да споделят модула изобщо, а 3 означава, че модулът е напълно thread-safe.
Стойност 1 означава, че нишките могат да споделят модула, но не и връзките
Стойност 2 означава, че нишките могат да споделят модула и връзките, но не и курсорите.
Ако не използвате нишки, не е необходимо да се интересуват от тази променлива

Стилът на параметрите (paramstyle) означава как параметрите се подават на SQL заявките.
Стойността 'format' означава стандартно форматиране на символни низове в стил C, така че се вмъква %s където искате да поставите параметър.
Стойност 'pyformat' индикира разширен формат, подобен на работа с речници, например %(foo)s.

# SQL API

В допълнение към тези Python стилове, има три начина да се индикират полетата: 'qmark' означава, че се използват въпросителни знаци
'numeric' означава полета от вида :1 или :2 (където числата са номера на параметъра)
'named' означава полета от типа :foobar, където foobar е името на параметъра

# Свързване

За да имате достъп до СУБД-то, първо трябва да се свържете с него. За целта се използва функцията connect.
Тя приема няколко параметъра, в зависимост от използваната СУБД
Следващата таблица описва често употребяваните параметри и тяхното значение Всички аргументи са символни низове

# Свързване

| Име на параметър | Описание | Задължителен |
|---|---|---|
| dsn | име на източника на данни | Да |
| user | потребител | Не |
| password | парола | Не |
| host | хост | Не |
| database | име на база данни | Не |

# Свързване

Функцията connect връща обект - връзка. Той представлява текущата сесия в базата данни. Поддържаните методи са:

| Име на метод | Описание |
|---|---|
| close() | Затваря връзката. Както обектът connection, така и курсорите не могат да се ползват |
| commit() | предаване на данните от чакащите транзакции |
| rollback() | отмяна на чакащите транзакции |
| cursor() | Връщане на обект cursor към връзката |

# SQLite

- SQLite е вградена СУБД, не изисква сървър
- Всяка SQLite база данни се съхранява в един файл
- SQLite поддържа и in-memory бази данни

Инсталация под Ubuntu:

sudo apt-get install python-sqlite

Примерната база данни се намира на адрес:
http://chinookdatabase.codeplex.com/

## Внасяне на модула

```
import sqlite3
```

## Отваряне на връзка

```
conn=sqlite3.connect('Chinook_Sqlite.sqlite')
```

Ако файлът не съществува, се създава празна база данни

## Създаване на курсор

```
curs=conn.cursor()
```

Изпълнение на заявка и връщане на един ред

Listing 1: Програмен код

```
query = 'SELECT * FROM Customer'
curs.execute(query)
row = curs.fetchone()
if row!=None:
    firstname, lastname = row[1], row[2]
```

Изпълнение на заявка и връщане на всички редове (курсорът е итерируем)

Listing 2: Програмен код

```
1 for row in curs:
2     print "firstname: %s, lastname: %s, company: %
      s, address: %s, city: %s, state: %s,
      country: %s" % (row[1], row[2], row[3], row
      [4], row[5], row[6], row[7])
```

## Свойство Cursor.connection

Това е атрибут само за четене, който връща референция към обект Connection, върху който е създаден курсора Използва се, когато едновременно съществуват множество курсори

# Обектът Cursor

## Метод Cursor.execute ( operation [, parameters ])

Подготвя и изпълнява операция по базата данни (заявка или команда)

Параметрите се подават като последователност или съответствие

Курсорът пази референция към операцията. Ако същия обект за операция бъде подаден отново, курсорът оптимизира изпълнението. Това е полезно в случаи, в които една и съща заявка се извършва множество пъти и единствената разлика са различните стойности на параметрите

Ако трябва да се добавят множество данни наведнъж има наличен метод .executemany()

## Метод Cursor.fetchone()

Връща следващия ред от резултата от заявката.
В случай че няма повече данни връща None
Поражда изключение, ако предходното изпълнение на .execute метода не е породил резултат или изобщо не е бил извикван

## Метод Cursor.fetchall()

Връща всички оставащи записи от резултата от заявката
Резултатът е списък от кортежи
Поражда изключение, ако предходното изпълнение на .execute метода не е породил резултат или изобщо не е бил извикван

## Метод Cursor.next()

Връща следващия ред от текущо изпълняваната заявка със същатат семантика като .fetchone()

## Свойство Cursor.rowcount

Това е атрибут само за четене, който съдържа броя редове от изпълнението на последния execute() метод.
Стойността е -1 в случай, че не е изпълняван метод execute() или броят редове не може да бъде установен от интерфейса

## Свойство Cursor.lastrowid

Това е свойство само за четене, което съдържа стойността на ключа на последния променен ред (например от единична INSERT заявка). Ако операцията не установява ключ, атрибутът е със стойност None.

Стойността на атрибута е неопределена, ако последно изпълнената команда променя повече от един ред, например INSERT с .executemany().

## Метод Cursor.close()

Затваря курсора веднага.

Курсорът е неизползваем от този момент нататък

# Заявки

Заявки с параметри cur.execute(SQL, параметри) SQL – Python символен низ параметри – редица(кортеж, списък) или съответствие(речник)

# Заявки

Използване на ? за заместване на параметрите и последователност за стойностите

Listing 3: Програмен код

```
1 curs.execute("insert into Customer(firstname, lastname,email) values (?, ?,?)",("Gerhard", "Haering","blabla@abv.bg"))
```

Listing 4: Програмен код

```
1 import sqlite3
2 conn=sqlite3.connect('Chinook_Sqlite.sqlite')
3 curs=conn.cursor()
4 query = 'SELECT * FROM Customer'
5 print query
6 print sqlite3.apilevel
7 curs.execute(query)
```

Listing 5: Програмен код

```
row = curs.fetchone()
if row!=None:
firstname, lastname = row[1], row[2]
print firstname,lastname
for row in curs:
print "firstname: %s, lastname: %s, company: %s, address: %s, city: %s, state: %s, country: %s" % (row[1], row[2], row[3], row[4], row[5], row[6], row[7])
```

Listing 6: Програмен код

```
1 curs.execute("insert into Customer(firstname, lastname,email) values (?, ?,?)",("Gerhard", "Haering","blabla@abv.bg"))
2 conn.commit()
3 conn.close()
```

Всяка стойност в SQLite база данни принадлежи на един от следните класове:

- NULL. Стойността е NULL.
- INTEGER. Стойността е цяло число със знак, съхранено в 1, 2, 3, 4, 6 или 8 байта, в зависимост от размера на стойността.
- REAL. Стойността е число с плаваща запетая, съхранено в 8-байтов IEEE floating point.
- TEXT. Стойността е текстов низ, съхранен с кодиране на самата база данни(UTF-8, UTF-16BE or UTF-16LE).
- BLOB. Стойността е необработваемо парче от числови данни, съхранено както е въведено.

В SQLite таблиците имат уникален идентификатор за всеки ред, който е 64 битова целочислена стойност със знак
Идентификаторът може да се достъпва със специалните имена ROWID, _ROWID_, or OID, освен ако самата таблица не съдържа колони с тези имена.
Ако таблицата съдържа колона от тип INTEGER PRIMARY KEY, тогава колоната става псевдоним за ROWID.

Когато се вмъква нов ред в таблицата, ROWID може да бъде специфициран като част от INSERT заявката или да получи автоматично стойност.
Ако не се зададе ROWID на INSERT заявка, или ROWID има стойност NULL, тогава съответната стойност се създава автоматично. Алгоритъмът дава на новосъздадения ред ROWID с единица по-голямо от най-голямата стойност преди INSERT-а

Операторът LIKE извършва сравнение с шаблон. Операндът отдясно на оператора LIKE съдържа шаблона, а този отляво – символният низ, в който търсим шаблона.
Символът за процент ("%") замества нула или повече символи в символния низ.
Символът за подчертаване ("_") замества единичен символ в символния низ. Всеки друг символ съответства на себе си или своя еквивалент в горен/долен регистър (не е чувствително към регистъра).

# Задача

Отворете и parse-нете файла retn5_dat.txt Таблицата на USDA за задържане на хранителните вещества Данните за състава на храната са необходими както за храната в суров вид, така и в сготвен. Обикновено обаче данните за хранителните вещества в сготвени храни липсват. Именно затова тези данни могат да се използват, за да се определи хранителната стойност на сготвена храна. Състава на хранителните вещества в сготвената храна може да се изчисли от несготвената храна като се приложи retention фактора.

# Задача

Във файла всички полета са разделени с (^) и текстовите полета са оградени с тилда (~).

| име на полето | описание |
|---|---|
| Retn_Code | 4-цифров код, уникално идентифициращ факторите за задържане |
| Desc | описание на хранителната категория и начин на приготвяне |
| Nutr_No | 3-цифров уникален код за хранителното вещество |
| NutrDesc | името на храната |
| Retn_Factor | Количеството на хранителния компонент, който остава след приготвянето на храната |
| N | брой опити |
| StdDev | стандартно отклонение |

# Задача

Създайте нова SQLite база данни и създайте таблица, която да бъде със структура като тази на първите пет атрибута в таблицата в текстовия файл. Parse-вайки файла попълнете таблицата с данните от файла. Напишете заявка, която извежда всички храни, които съдържат телешко ('VEAL')

Благодаря за вниманието!

Въпроси?

# References