



DaneSoul 15 февраля 2017 в 03:23

Python: Работа с базой данных, часть 1/2: Используем DB-API

Python*, Программирование*, SQL*, SQLite*

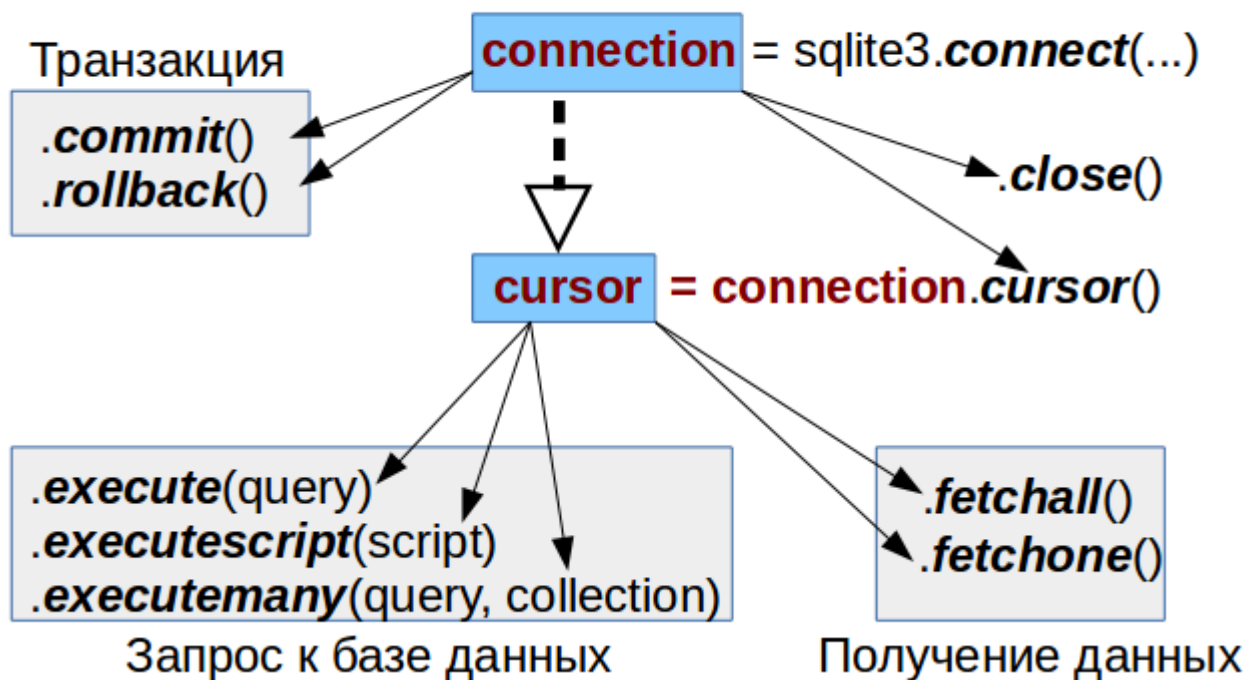
Tutorial

часть 1/2: Используем DB-API

часть 2/2: Используем ORM

Python DB-API — это не конкретная библиотека, а набор правил, которым подчиняются отдельные модули, реализующие работу с конкретными базами данных. Отдельные нюансы реализации для разных баз могут отличаться, но общие принципы позволяют использовать один и тот же подход при работе с разными базами данных.

Python DB-API методы



В статье рассмотрены основные методы DB-API, позволяющие полноценно работать с базой данных. Полный список можете найти по ссылкам в конце статьи.

Требуемый уровень подготовки: базовое понимание синтаксиса SQL и Python.

Готовим инвентарь для дальнейшей комфортной работы

- Python имеет встроенную поддержку SQLite базы данных, для этого вам не надо ничего дополнительно устанавливать, достаточно в скрипте указать импорт стандартной библиотеки

```
import sqlite3
```

- Скачаем тестовую базу данных, с которой будем работать. В данной статье будет использоваться открытая (MIT лицензия) тестовая база данных “Chinook”. Скачать ее можно с репозитория:

github.com/lerocha/chinook-database
Нам нужен для работы только бинарный файл “Chinook_Sqlite.sqlite”:
github.com/lerocha/chinook-database/blob/master/ChinookDatabase/DataSources/Chinook_Sqlite.sqlite

- Для удобства работы с базой (просмотр, редактирование) нам нужна программа браузер баз данных, поддерживающая SQLite. В статье работа с браузером не рассматривается, но он поможет Вам наглядно видеть что происходит с базой в процессе наших экспериментов.

Примечание: внося изменения в базу не забудьте их применить, так как база с непримененными изменениями остается залоченной.

Вы можете использовать (последние два варианта кросс-платформенные и бесплатные):

- Привычную вам утилиту для работы с базой в составе вашей IDE;
- [SQLite Database Browser](#)
- [SQLiteStudio](#)

Python DB-API модули в зависимости от базы данных

База данных	DB-API модуль
SQLite	sqlite3
PostgreSQL	psycopg2
MySQL	mysql.connector
ODBC	pyodbc

Соединение с базой, получение курсора

Для начала рассмотрим самый базовый шаблон DB-API, который будем использовать во всех дальнейших примерах:

```
# Импортируем библиотеку, соответствующую типу нашей базы данных
import sqlite3

# Создаем соединение с нашей базой данных
# В нашем примере у нас это просто файл базы
conn = sqlite3.connect('Chinook_Sqlite.sqlite')

# Создаем курсор - это специальный объект который делает запросы и получает их резул
cursor = conn.cursor()

# ТУТ БУДЕТ НАШ КОД РАБОТЫ С БАЗОЙ ДАННЫХ
# КОД ДАЛЬНЕЙШИХ ПРИМЕРОВ ВСТАВЛЯТЬ В ЭТО МЕСТО

# Не забываем закрыть соединение с базой данных
conn.close()
```

При работе с другими базами данных, используются дополнительные параметры соединения, например для PostgreSQL:

```
conn = psycopg2.connect( host=hostname, user=username, password=password, dbname=dat
```

Чтение из базы

```
# Делаем SELECT запрос к базе данных, используя обычный SQL-синтаксис
cursor.execute("SELECT Name FROM Artist ORDER BY Name LIMIT 3")

# Получаем результат сделанного запроса
results = cursor.fetchall()
results2 = cursor.fetchall()

print(results)  # [('A Cor Do Som',), ('Aaron Copland & London Symphony Orchestra',
print(results2) # []
```

Обратите внимание: После получения результата из курсора, второй раз без повторения самого запроса его получить нельзя — вернется пустой результат!

Запись в базу

```
# Делаем INSERT запрос к базе данных, используя обычный SQL-синтаксис
cursor.execute("insert into Artist values (Null, 'A Aagrh!') ")
```

```
# Если мы не просто читаем, но и вносим изменения в базу данных - необходимо сохранить
conn.commit()

# Проверяем результат
cursor.execute("SELECT Name FROM Artist ORDER BY Name LIMIT 3")
results = cursor.fetchall()
print(results) # [('A Aagrh!',), ('A Cor Do Som',), ('Aaron Copland & London Sympho
```

Примечание: Если к базе установлено несколько соединений и одно из них осуществляет модификацию базы, то база SQLite заочивается до завершения (метод соединения **.commit()**) или отмены (метод соединения **.rollback()**) транзакции.

Разбиваем запрос на несколько строк в тройных кавычках

Длинные запросы можно разбивать на несколько строк в произвольном порядке, если они заключены в тройные кавычки — одинарные ("...") или двойные ("\"...\"")

```
cursor.execute("""
    SELECT name
    FROM Artist
    ORDER BY Name LIMIT 3
""")
```

Конечно в таком простом примере разбивка не имеет смысла, но на сложных длинных запросах она может кардинально повышать читаемость кода.

Объединяем запросы к базе данных в один вызов метода

Метод курсора **.execute()** позволяет делать только один запрос за раз, при попытке сделать несколько через точку с запятой будет ошибка.

▼ Для тех кто не верит на слово:

```
cursor.execute("""
    insert into Artist values (Null, 'A Aagrh!');
    insert into Artist values (Null, 'A Aagrh-2!');
""")
# sqlite3.Warning: You can only execute one statement at a time.
```

Для решения такой задачи можно либо несколько раз вызывать метод курсора **.execute()**

```
cursor.execute("""insert into Artist values (Null, 'A Aagrh!');""")
cursor.execute("""insert into Artist values (Null, 'A Aagrh-2!');""")
```

Либо использовать метод курсора **.executescript()**

```
cursor.executescript("""
insert into Artist values (Null, 'A Aagrh!');
insert into Artist values (Null, 'A Aagrh-2!');
""")
```

Данный метод также удобен, когда у нас запросы сохранены в отдельной переменной или даже в файле и нам его надо применить такой запрос к базе.

Делаем подстановку значения в запрос

Важно! Никогда, ни при каких условиях, не используйте конкатенацию строк (+) или интерполяцию параметра в строке (%) для передачи переменных в SQL запрос. Такое формирование запроса, при возможности попадания в него пользовательских данных – это ворота для SQL-инъекций!

Правильный способ – использование второго аргумента метода **.execute()**

Возможны два варианта:

```
# С подстановкой по порядку на места знаков вопросов:
cursor.execute("SELECT Name FROM Artist ORDER BY Name LIMIT ?", ('2'))

# И с использованием именованных замен:
cursor.execute("SELECT Name from Artist ORDER BY Name LIMIT :limit", {"limit": 3})
```

Примечание 1: В PostgreSQL (UPD: и в MySQL) вместо знака '?' для подстановки используется: %s

Примечание 2: Таким способом не получится заменять имена таблиц, одно из возможных решений в таком случае рассматривается тут: stackoverflow.com/questions/3247183/variable-table-name-in-sqlite/3247553#3247553

UPD: Примечание 3: Благодарю @Igelko за упоминание параметра **paramstyle** — он определяет какой именно стиль используется для подстановки переменных в данном модуле. Вот ссылка с полезным приемом для работы с разными стилями подстановок.

Делаем множественную вставку строк проходя по коллекции с помощью метода курсора **.executemany()**

```
# Обратите внимание, даже передавая одно значение - его нужно передавать кортежем!
# Именно по этому тут используется запятая в скобках!
new_artists = [
    ('A Aagrh!',),
    ('A Aagrh!-2',),
    ('A Aagrh!-3',),
]
cursor.executemany("insert into Artist values (Null, ?);", new_artists)
```

Получаем результаты по одному, используя метод курсора *.fetchone()*

Он всегда возвращает кортеж или None. если запрос пустой.

```
cursor.execute("SELECT Name FROM Artist ORDER BY Name LIMIT 3")
print(cursor.fetchone())    # ('A Cor Do Som',)
print(cursor.fetchone())    # ('Aaron Copland & London Symphony Orchestra',)
print(cursor.fetchone())    # ('Aaron Goldberg',)
print(cursor.fetchone())    # None
```

Важно! Стандартный курсор забирает все данные с сервера сразу, не зависимо от того, используем мы *.fetchall()* или *.fetchone()*

Курсор как итератор

```
# Использование курсора как итератора
for row in cursor.execute('SELECT Name from Artist ORDER BY Name LIMIT 3'):
    print(row)
# ('A Cor Do Som',)
# ('Aaron Copland & London Symphony Orchestra',)
# ('Aaron Goldberg',)
```

UPD: Повышаем устойчивость кода

Благодарю @paratagas за ценное дополнение:

Для большей устойчивости программы (особенно при операциях записи) можно оборачивать инструкции обращения к БД в блоки «try-except-else» и использовать встроенный в sqlite3 «родной» объект ошибок, например, так:

```
try:
    cursor.execute(sql_statement)
    result = cursor.fetchall()
except sqlite3.DatabaseError as err:
```

```
print("Error: ", err)
else:
    conn.commit()
```

UPD: Использование with в psycopg2

Благодарю [@KurtRotzke](#) за ценное дополнение:

Последние версии psycopg2 позволяют делать так:

```
with psycopg2.connect("dbname='habr'") as conn:
    with conn.cursor() as cur:
```

Некоторые объекты в Python имеют `__enter__` и `__exit__` методы, что позволяет «чисто» взаимодействовать с ними, как в примере выше.

UPD: Использование row_factory

Благодарю [@remzalp](#) за ценное дополнение:

Использование `row_factory` позволяет брать метаданные из запроса и обращаться в итоге к результату, например по имени столбца.

По сути — callback для обработки данных при возврате строки. Да еще и полезнейший `cursor.description`, где есть всё необходимое.

Пример из документации:

```
import sqlite3

def dict_factory(cursor, row):
    d = {}
    for idx, col in enumerate(cursor.description):
        d[col[0]] = row[idx]
    return d

con = sqlite3.connect(":memory:")
con.row_factory = dict_factory
cur = con.cursor()
cur.execute("select 1 as a")
print(cur.fetchone()["a"])
```

Дополнительные материалы (на английском)

- Краткий бесплатный он-лайн курс — Udacity — Intro to Relational Databases —
Рассматриваются синтаксис и принципы работы SQL, Python DB-API – и теория и практика в одном флаконе. Очень рекомендую для начинающих!
- Advanced SQLite Usage in Python
- SQLite Python Tutorial на tutorialspoint.com
- A thorough guide to SQLite database operations in Python
- UPD: The Novice's Guide to the Python 3 DB-API
- Справочные руководства по SQLite он-лайн:
 - www.tutorialspoint.com/sql/index.htm
 - www.tutorialspoint.com/sqlite
 - www.sqlitetutorial.net

часть 1/2: Используем DB-API

часть 2/2: Используем ORM

Приглашаю к обсуждению:

- Если я где-то допустил неточность или не учёл что-то важное — пишите в комментариях, важные комментарии будут позже добавлены в статью с указанием вашего авторства.
- Если какие-то моменты не понятны и требуется уточнение — пишите ваши вопросы в комментариях — или я или другие читатели дадут ответ, а дельные вопросы с ответами будут позже добавлены в статью.

Теги: python, sql, sqlite, db, db-api

Хабы: Python, Программирование, SQL, SQLite



+19



455K



483



38



Редакторский дайджест

Присылаем лучшие статьи раз в месяц



Электронпочта

**64**

Карма

2

Рейтинг

Александр @DaneSoul

Веб-программирование, Python

[Задонатить](#)**Комментарии 38****smple**

15.02.2017 в 04:37



эх собиру минусы за упоминания того что нельзя упоминать, но все же

Чем ваш connection отличается от <http://php.net/manual/ru/class.pdo.php> (да знаю сейчас отличается, и текст запроса в курсоре, но тот вариант что по ссылкам кажется более продуман тем то что описано у вас)

ну и соответственно cursor от <http://php.net/manual/ru/class.pdostatement.php>

мне кажется немного обсуждений и придете к тому же интерфейсу что я указал по ссылкам



0

[Ответить](#)**DaneSoul**

15.02.2017 в 05:39



А почему общие подходы решения стандартных задач обязательно должны отличаться в разных языках? Наоборот, чем меньше отличий, тем проще с этим работать.

Так если посмотреть, то и принцип работы с файлами, и принципы парсинга JSON или XML, те же регулярные выражения очень похоже реализованы в разных языках.

Проблема то здесь в чем?



+3

[Ответить](#)**ls1**

15.02.2017 в 11:16



1) cursor.execute(«SELECT * FROM 100MBtable»)

2) results = cursor.fetchall()

Интересует, что происходит «под капотом»

Правильно ли я понимаю, что первая команда создаст на сервере БД структуру размером 100МБ, а вторая — перекачает её по сети клиенту? Или это как-то иначе работает?



+1

[Ответить](#)**DaneSoul**

15.02.2017 в 12:39



Если используется стандартный курсор, то да, причем:

* Объем данных будет значительно больше чем размер самой базы

* Даже если используется fetchone() все равно будет загружена вся информация целиком сразу

Для решения таких проблем существуют специальные классы курсоров (например SSCursor), которые позволяют хранить результат запроса на сервере.

Вот подобные вопросы на StackOverflow:

<http://stackoverflow.com/questions/4559402/the-memory-problem-about-mysql-select>

<http://stackoverflow.com/questions/337479/how-to-get-a-row-by-row-mysql-resultset-in-python>

◆ +1 Ответить



kilgur

15.02.2017 в 14:13

```
cur.execute("SELECT * FROM Test WHERE testID > :tid", {'tid': 10})
```

получаю:

You have an error in your SQL syntax;...

пакеты MySQLdb и mysql.connector, оба ругаются на "?":

```
cur.execute("SELECT * FROM Test WHERE testID > ?", (10,))
```

Not all parameters were used in the SQL statement

Если не указать запятую при одном параметре (10,) параметр будет передан как есть, а не в кортеже. MySQLdb выругается вот так:

not all arguments converted during string formatting

а mysql.connector ругается на ошибку sql-синтаксиса (near ?)

python v2.7

mariadb 10.1

Судя по документации, именованные параметры в mysql должны задаваться через "@". Но в таком случае, хотя и нет исключений при вызове execute, результат пустой.

У себя в скриптах использую .format(), но их запускаю только я, поэтому sql-инъекции маловероятны.

◆ 0 Ответить



DaneSoul


15.02.2017 в 14:17

У меня код тестировался под SQLite, StackOverflow советует для MySQL использовать для подстановки %s

<http://stackoverflow.com/questions/775296/python-mysql-parameterized-queries>

◆ 0 Ответить



 15.02.2017 в 14:34

Действительно, %s работает как позиционный аргумент. Спасибо за информацию!

 +2 Ответить

 Igelko
15.02.2017 в 19:41

это должно быть можно настраивать через paramstyle

 +1 Ответить

 kilgur
16.02.2017 в 01:35


Гениально! Спасибо тебе, добрый человек.

В MySQLdb paramstyle == 'format', по умолчанию. Выставил в 'pyformat', теперь работает так:

```
cur.execute("SELECT * FROM Test WHERE testID > %(tid)s", {'tid': 10})
```

Единственное, в данном модуле не получится использовать %d, например. П.ч. все аргументы прогоняются через db.literal(), который возвращает строку.

 0 Ответить

 DaneSoul
17.02.2017 в 07:05

А как Вы смогли изменить этот параметр?

На форумах в нескольких местах находил информацию, что он информационный и «вшит» в сам модуль базы данных.

Я пробовал менять его для SQLite — значение параметра меняется, но работать с новым типом подстановок он не начинает:

```
import sqlite3

print(sqlite3.paramstyle)      # qmark
sqlite3.paramstyle = 'format'
print(sqlite3.paramstyle)      # format

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

cursor.execute("SELECT Name FROM Artist ORDER BY Name LIMIT %s", ('2'))
# sqlite3.OperationalError: near "%": syntax error
```

Вот кстати полезная ссылка с изящным обходным решением проблемы:

<http://stackoverflow.com/questions/12184118/python-sqlite3-placeholder>

 0 Ответить


 **Igelko**
17.02.2017 в 16:04

в том-то и запутанность ситуации, что PEP утверждает, что в этой переменной уровня модуля должен содержаться используемый модулем paramstyle, но ни слова не говорит, можно или нельзя его изменить, и если можно, то как это делать, а также не накладывает ограничений на те варианты paramstyle, который реализовать обязательно.

и да, я не прав, про то, что это можно нормально настраивать. Там на самом деле обычный pyformat.

Автор вкатил тупой if в этом месте и подстановку в запрос без всякой защиты от инъекций <https://github.com/farcepest/MySQLdb1/blob/master/MySQLdb/cursors.py#L183>

 +1 Ответить

 **kilgur**
18.02.2017 в 04:12

Все параметры «прогоняются» через db.literal(), который, в свою очередь, вызывает escape() у _mysql.connection.


```
print("LIKE %s" % conn.escape("' ; select 1"))
```

ВЫВОДИТ

```
LIKE '\'; select 1'
```

Т.е. защита от инъекций все-таки есть.

 0 Ответить

 **kilgur**
18.02.2017 в 04:05

Вот уж совпало, так совпало...

Я ловил ошибки, пытаюсь применить именованные параметры по докам mysql (вида @param). Бегло прочитал про paramstyle и установил его в pyformat. Попробовал %(param)s — сработало, обрадовался, хотя paramstyle в действительности не причем. Если я правильно понимаю, то этот атрибут модуля носит информативный характер, т.е. автор модуля этим атрибутом указывает способ форматирования параметров. Копание в исходниках показывает, что он нигде не используется и никуда не передается.

 +1 Ответить

 **paratagas**
15.02.2017 в 16:01

Для большей устойчивости программы (особенно при операциях записи) можно оборачивать инструкции обращения к БД в блоки «try-except-else» и использовать встроенный в sqlite3 «родной» объект ошибок, например, так:

```
try:
    cursor.execute(sql_statement)
    result = cursor.fetchall()
```

```
except sqlite3.DatabaseError as err:
    print("Error: ", err)
else:
    conn.commit()
```

◆ +1 Ответить



remzalp

15.02.2017 в 16:36

И снова ограничиваемся скучным и унылым Tuple.

За что любил PHP — при любой правке порядка полей в SQL запросе, если поле хотя бы есть, конструкция вида \$row['field'] вполне успешно отдаёт его значение. В случае с простой реализацией запросов в питоне уже не всё так радужно, но можно сделать чуть посложнее.

Такую приятную вещь, как **row_factory** Вы забыли. Она позволяет брать метаданные из запроса и обращаться в итоге к результату, например по имени столбца.

По сути — callback для обработки данных при возврате строки. Да еще и полезнейший cursor.description, где есть всё необходимое.

▶ Фрагмент из справки

◆ +1 Ответить



Igelko

15.02.2017 в 19:39

а это уже разное от драйвера к драйверу, чем меня спецификация db-api2 изрядно расстраивает. Очень не хватает четкой спеки на параметры, специфицированного autocommit в стандарте, serverside-курсоров,

dict или объекты во всех драйверах тоже реализованы, но по-разному и это печально.

Нет connection pooling и асинхронщины, что встаёт костью в горле, когда скрещиваешь это с каким-нибудь asyncio (я в своё время пытался скрещивать с twisted, примерно такого же порядка проблемы). Пришло время описывать db api3 =)

◆ 0 Ответить

○ НЛО прилетело и опубликовало эту надпись здесь



whintu

16.02.2017 в 02:55

Не знаю зачем вам такое, но это работает:

```
cursor.execute('''INSERT INTO t1(f) VALUES('v1'); INSERT INTO t2(f) VALUES('v2'); ''')
```

python 3.6, psycopg 2.6

Лучше всё-таки не лепить три запроса в один скрипт, а: выполнить коннект, три раза отдельно выполнить execute с правильной интерполяцией значений, а потом выполнить commit.

А если будут траблы со вставкой множества значений в одну таблицу (там действительно ужасный overхед, вставлял как-то котиловки), то здесь поможет рецепт.

 +1 Ответить



◦ НЛО прилетело и опубликовало эту надпись здесь

◦  worldmind
15.02.2017 в 22:33

> conn.commit()

а почему все операции через курсор делаются, а коммит у коннекта вызывается?

 0 Ответить



◦  BubaVV
16.02.2017 в 04:04

Потому что может быть несколько курсоров на один коннект, и судя по всему нет возможности коммитить по отдельности

 0 Ответить




◦  worldmind
16.02.2017 в 14:11

Если это так и нет осмысленного объяснения почему это так, то это баг в API

 0 Ответить



◦  worldmind
15.02.2017 в 22:35 

Я в питоне новичок, но как понимаю курсор тоже надо закрывать и правильно это делать как-то так

```
with closing(connection.cursor()) as cursor:
```

 0 Ответить



◦  worldmind
15.02.2017 в 22:36

ну и с коннектом видимо также нужно поступать

 0 Ответить




◦ НЛО прилетело и опубликовало эту надпись здесь

◦  worldmind
16.02.2017 в 14:13

Благодарю за дополнение, но вопрос с коммитами про другое был — почему коммит вызывается у коннекта, а не у курсора

 0 Ответить



 **Igelko**
16.02.2017 в 18:47

Потому что существуют СУБД, умеющие несколько курсоров в одной транзакции. Например может вернуться несколько курсоров как результат выполнения хранимой процедуры у какого-нибудь Oracle.

Но при этом авторы спецификации не стали заморачиваться и объединили сущность транзакции и соединения, чем обломали кайф любителям firebird, у них общепринятая практика — открыть одну длинную читающую транзакцию в read committed и записи делать короткими пишущими транзакциями внутри одного и того же соединения.

На самом деле мне тоже не нравится подобная ситуация — можно закрыть транзакцию, сломав вдребезги напололам ещё не закрытые и недочитанные курсоры.

 0 Ответить

 **worldmind**
17.02.2017 в 02:14

Да даже если бы это ничего не ломало это был бы косяк в API, в питоне обычно всё логично устроено, может мы чего-то не знаем?

 0 Ответить

 **Igelko**
17.02.2017 в 15:10

PEP-249 ничего по этому поводу не говорит, но я понимаю, почему авторы вытащили курсоры в отдельную сущность — их точно даже может существовать больше одного на соединение в некоторых БД.


MySQLdb и psycopg2 емнип делают close всем открытым курсорам этого соединения при commit.

 0 Ответить

 **worldmind**
17.02.2017 в 15:18

Мы что-то о разном, я не говорю что курсор не нужен, я говорю о том, что раз мы работаем через курсор, то и коммит надо делать через курсор

 0 Ответить

 **Igelko**
17.02.2017 в 16:10

я вот и говорю, что в одной транзакции может быть теоретически открыто больше одного объекта курсора в зависимости от СУБД и это нормально. Транзакция при этом будет одна, общая.

Допустим, мы хотим почитать из нескольких табличек параллельно так, чтобы данные из обеих были консистентны. Допустим мы делаем джойн или какую-то сверку данных из двух больших таблиц на клиенте.


Нужно открыть транзакцию режиме в SNAPSHOT ISOLATION и читать с помощью serverside cursor кусочками, потом оба курсора закрыть и выбросить.

0 Ответить

 **worldmind**
18.02.2017 в 03:38

Если транзакция общая на несколько курсоров, то было бы логичнее её явно начинать, а потом явно коммитить, а по дефолту транзакция на каждый курсор отдельная должна быть

 0 Ответить

 **Igelko**
20.02.2017 в 17:47

Я очень не уверен, что все СУБД поддерживают несколько транзакций в одном соединении, так что возможность открывать по транзакции на курсор фича попросту нереализуемая в большинстве реализаций и мешающая распространению стандарта. Стандарт — это всегда какое-то общее подмножество того, что есть на рынке.

На самом деле мы сейчас гадаем на кофейной гуще, пытаюсь понять, что двигало авторами спеки. Вероятнее всего уже до PER уже существовала какая-то реализация, где всё ограничивалось фичами конкретной СУБД, и которую почти без изменений втащили в стандарт.

Судя по автору PER, уши растут из драйвера mxODBC.

 +1 Ответить

 **andreios**
17.02.2017 в 04:14

А подскажите пожалуйста, причину вот этого:

>> # Обратите внимание, даже передавая одно значение — его нужно передавать кортежем! Не совсем понимаю, почему обязательно делать список из кортежей, вместо обычных строк?

 +1 Ответить

 **DaneSoul**
17.02.2017 в 04:46

Это особенность реализации метода `.executemany()`, который работает с коллекцией коллекций, а не коллекцией единичных значений, поэтому даже для единичного значения его надо передавать как коллекцию, хотя не обязательно кортеж.

<http://stackoverflow.com/questions/19154324/psycpg2-executemany-with-simple-list>

<http://stackoverflow.com/questions/5331894/i-cant-get-pythons-executemany-for-sqlite3-to-work-properly>

 0 Ответить

 **DaneSoul**
17.02.2017 в 07:43

Дополнения из комментариев от @Igelko, @paratagas, @KurtRotzke, @remzalp были добавлены в статью с указанием авторства. Большое спасибо за такие полезные дополнения!

 0 Ответить



17.02.2017 в 16:10

Ёжик оказался не прав.

https://habrahabr.ru/post/321510/#comment_10074610



0

Ответить



VijayDeveloper

13.02.2019 в 14:07



It looks good to me after a complete reading but you can add few more resources like:

- <https://www.python.org/dev/peps/pep-0249/>
- <https://docs.python.org/2/library/sqlite3.html>
- <https://docs.python.org/3/library/sqlite3.html>

Listing a lot of resources is the good way for all your audience may be they are not able to understand through those resource which you have listed.



+1

Ответить



Только полноправные пользователи могут оставлять комментарии. [Войдите](#), пожалуйста.

Публикации

ЛУЧШИЕ ЗА СУТКИ

ПОХОЖИЕ



alexopryshko сегодня в 17:58

Школу закончил в 14, Бауманку в 18: почему, зачем и какие последствия



+41



8.1K



20



57 +57



MaFrance351 сегодня в 15:01

Считываем и эмулируем карты с магнитной полосой



+40



3.3K



28



19 +19



elena_pastukhova сегодня в 19:01

Как мы съедаем 15 тонн воды в день



+29



3.6K



7



12 +12



Sagidullin сегодня в 03:53

Big bada boom отменяется? Подводные интернет-магистралы выдержат наступление «События Кэррингтона»

♦ +29

👁 2.5K

📖 5

💬 12 +12



sergey__pushkin сегодня в 14:01

Язык диаграмм

♦ +27

👁 2.3K

📖 26

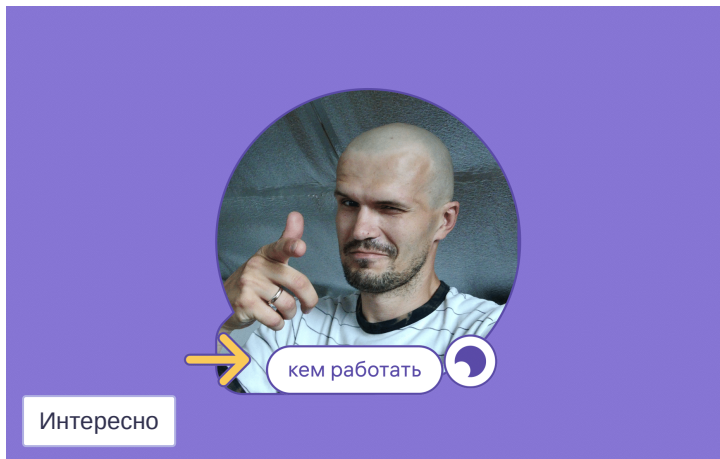
💬 7 +7

Итоги сезона Data Mining: тем много, но в топе NLP и гайды

Мегапост

МИНУТОЧКУ ВНИМАНИЯ

Разместить



Кем работать в IT: бэкенд-разработчик



Нужен ваш лучший пост за год: с экспертизой и котами

КУРСЫ



Python-разработчик с нуля

8 декабря 2022 · 90 750 ₽ · Нетология



Углубленный курс по Python

10 декабря 2022 · 45 000 ₽ · GB



SQL и получение данных

19 декабря 2022 · 24 850 ₽ · Нетология



Основы программирования на Python. Уровень 2

22 декабря 2022 · 19 500 ₽ · Level UP

ЧИТАЮТ СЕЙЧАС

Как школьники МЭШ взломали

34K 69 +69

Школу закончил в 14, Бауманку в 18: почему, зачем и какие последствия

8K 57 +57

Ловушка алгоритмизации, или как 44-ФЗ породил коррупцию

2.3K 36 +36

Lumia 640 — всё ещё достоин?

1.9K 7 +7

Как мы съедаем 15 тонн воды в день

3.6K 12 +12

До конца года можно выбить ачивку Технотекста 2022

Событие

ИСТОРИИ


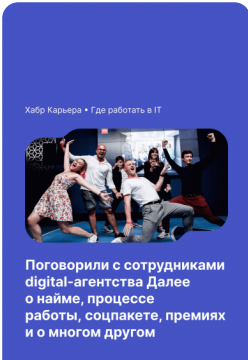
iFixit опубликовала видеоролик, объясняющий причины возгорания и взрыва аккумулятора



Причины возгорания аккумулятора

Хабр Карьера • Где работать в IT

Поговорили с сотрудниками digital-агентства Далее о найме, процессе работы, соцпакете, премиях и о многом другом



Где работать в IT: Далее

Замыкаем магический круг



Сезон Data Mining от Хабра и SM Lab закончен! Подводим итоги чёрной мессы, отмечаем выдающиеся заклятия посты, награждаем сильнейшего data-рекрутера



Итоги сезона Data Mining


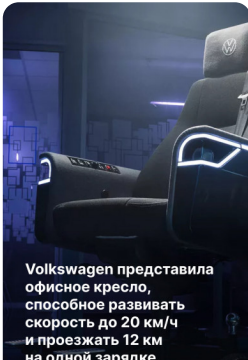
Работа не в ущерб жизни

Статьи о продуктивности, лайфхаки и советы по управлению временем



Не работай через силу


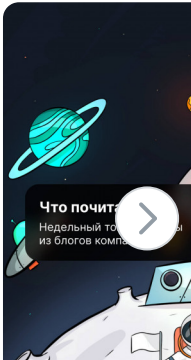
Volkswagen представила офисное кресло, способное развивать скорость до 20 км/ч и проезжать 12 км на одной зарядке



Офисное кресло Volkswagen

Что почитать

Недельный топ из блогов компаний



Недельный топ гаджетов от компаний

Python разработчик
164 вакансии

Data Scientist
122 вакансии

Django разработчик
49 вакансий

Все вакансии

Ваш аккаунт	Разделы	Информация	Услуги
Войти	Публикации	Устройство сайта	Корпоративный блог
Регистрация	Новости	Для авторов	Медийная реклама
	Хабы	Для компаний	Нативные проекты
	Компании	Документы	Образовательные программы
	Авторы	Соглашение	Стартапам
	Песочница	Конфиденциальность	Мегапроекты



Настройка языка

Техническая поддержка

Вернуться на старую версию