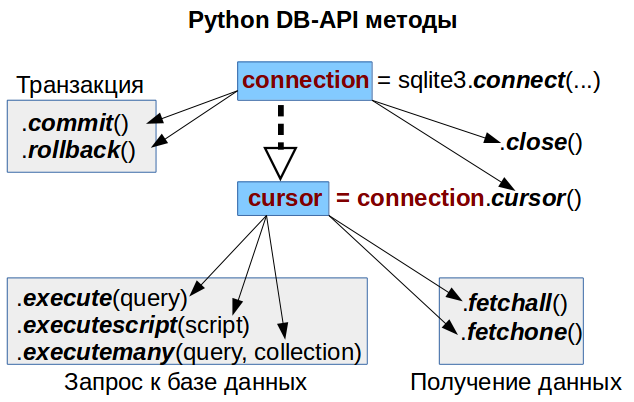
**Python DB-API** – это не конкретная библиотека, а набор правил, которым подчиняются отдельные модули, реализующие работу с конкретными базами данных. Отдельные нюансы реализации для разных баз могут отличаться, но общие принципы позволяют использовать один и тот же подход при работе с разными базами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



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

* 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.  
    
  Вы можете использовать (последние два варианта кросс-платформенные и бесплатные):
  + Привычную вам утилиту для работы с базой в составе вашей IDE;
  + [SQLite Database Browser](http://sqlitebrowser.org/)
  + [SQLiteStudio](https://sqlitestudio.pl/index.rvt)

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 результаты*

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 PostrgeSQL:

conn = psycopg2.connect( host=hostname, user=username, password=password, dbname=database)

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',), ('Aaron Goldberg',)]*

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 транзакцию*

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 Symphony Orchestra',)]*

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 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 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:

*# C подставновкой по порядку на места знаков вопросов:*

cursor.execute("SELECT Name FROM Artist ORDER BY Name LIMIT ?", ('2'))

*# И с использованием именнованных замен:*

cursor.execute("SELECT Name from Artist ORDER BY Name LIMIT :limit", {"limit": 3})

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

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)

**Кортеж (tuple)** в **Python** — это последовательность неизменяемых объектов. Кортежи очень схожи со списками, с той разницей — что элементы кортежа не могут быть изменены и используют обычные скобки, вместо квадратных, как в списках.

Кортеж создаётся простым присваиванием элементов, разделённых запятыми. Так же — их можно указывать в скобках. Например:

|  |  |  |
| --- | --- | --- |
| 1 | >>> tup1 =('physics','chemistry',1997,2000); | |
| 2 | >>> tup2 =(1,2,3,4,5); |

|  |  |
| --- | --- |
| 3 | >>> tup3 ="a","b","c","d"; |

Пустой кортеж создаётся указанием пустых скобок:

|  |  |
| --- | --- |
| 1 | >>> tup1 = (); |

Что бы создать кортеж, содержащий только один элемент — тоже необходимо добавлять запятую:

|  |  |
| --- | --- |
| 1 | >>> tup1 =(50,); |
| 2 | >>> print tup1[0] | |

|  |  |
| --- | --- |
| 3 | 50 |

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',)*

Повышаем устойчивость кода

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()

Ипользование row\_factory

Использование [row\_factory](https://docs.python.org/3/library/sqlite3.html" \l "sqlite3.Connection.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"])