Moscow Coding School

Python как первый язык

Преподаватель: Захарчук Сергей Сергеевич

Сегодня

- БД
- ORM
- MVC
- Логирование

Но сначала…

Чем плохи файлы для хранения данных?

- нет многопользовательского доступа;
- медленно работают;
- очень медленно работают при большом количестве данных;
- нет валидации данных;
- не гарантируется целостность данных;
- нет прав доступа;
- нет транзакций;
- …

Реляционные базы данных

- Реляционная база данных это набор таблиц с данными.

- Таблица - это прямоугольная матрица, состоящая из строк и столбцов. Таблица задает отношение (relation).

- Строка - запись, состоящая из полей - столбцов. В каждом поле может содержаться некоторое значение, либо специальное значение NULL (пусто). В таблице может быть произвольное количество строк. Для реляционной модели порядок расположения строк не определен и не важен. Каждый столбец в таблице имеет собственное имя и тип.

Популярные СУБД

- sqlite
- MySQL
- PostgreSQL
- MongoDB

SQL

SELECT, WHERE, GROUP BY, HAVING, ORDER BY, JOIN, INSERT, UPDATE, CREATE, DELETE

SQL

SELECT, WHERE, GROUP BY, HAVING, ORDER BY, JOIN, INSERT, UPDATE, CREATE, DELETE

http://sql-ex.ru/

Что такое DB-API 2

Вынесенная в заголовок аббревиатура объединяет два понятия:

DB (Database, база данных) и

API (Application Program Interface, интерфейс прикладной программы).

Таким образом, DB - API определяет интерфейс прикладной программы с базой данных. Этот интерфейс, описываемый ниже, должен реализовывать все модули расширения, которые служат для связи Python - программ с базами данных.

Единый API (в настоящий момент его вторая версия) позволяет абстрагироваться от марки используемой базы данных, при необходимости довольно легко менять одну СУБД на другую, изучив всего один набор функций и методов.

Интерфейс модуля

Доступ к базе данных осуществляется с помощью объекта - соединения(connection object).

DB -API - совместимый модуль должен предоставлять функцию - конструктор connect() для класса объектов - соединений.

Конструктор должен иметь следующие именованные параметры:

- Dsn - Название источника данных в виде строки
- User -Имя пользователя
- Password - Пароль
- Host - Адрес хоста, на котором работает СУБД
- Database - Имя базы данных.

Интерфейс модуля

Модуль определяет константы, содержащие его основные характеристики:

- Apilevel
- Threadsafety
- Paramstyle

- Модуль должен определять ряд исключений для обозначения типичных исключительных ситуаций:
- Warning (предупреждение),
- Error (ошибка),
- InterfaceError (ошибка интерфейса),
- DatabaseError (ошибка, относящаяся к базе данных).

Объект - соединение

Объект - соединение, получаемый в результате успешного вызова функции connect(), должен иметь следующие методы:

- close() Закрывает соединение с базой данных.
- commit() Завершает транзакцию.
- rollback() Откатывает начатую транзакцию (восстанавливает исходное состояние). Закрытие соединения при незавершенной транзакции автоматически производит откат транзакции.
- cursor() Возвращает объект - курсор, использующий данное соединение. Если база данных не поддерживает курсоры, модуль сопряжения должен их имитировать.

Объект - курсор

Под транзакцией понимается группа из одной или нескольких операций, которые изменяют базу данных. Транзакция соответствует логически неделимой операции над базой данных, а частичное выполнение транзакции приводит к нарушению целостности БД.

Объект - курсор

Курсор (от англ. cursor - CURrrent Set Of Records, текущий набор записей) служит для работы с результатом запроса. Результатом запроса обычно является одна или несколько прямоугольных таблиц со столбцами- полями и строками- записями. Приложение может читать и обрабатывать полученные таблицы и записи в таблице по одной, поэтому в курсоре хранится информация о текущей таблице и записи. Конкретный курсор в любой момент времени связан с выполнением одной SQL - инструкции.

Объект - курсор

Атрибуты объекта - курсора тоже определены DB - API:

arraysize - Атрибут, равный количеству записей, возвращаемых методом fetchmany(). По умолчанию равен 1.

callproc(procname[, params]) - Вызывает хранимую процедуру procname с параметрами из изменчивой после довательности params. Хранимая процедура может изменить значения некоторых параметров последовательности. Метод может возвратить результат, доступ к которому осуществляется через fetch - методы.

Объект - курсор

close() - Закрывает объект - курсор. description - Этот доступный только для чтения атрибут является последовательностью из семиэлементных последовательностей.

Каждая из этих последовательностей содержит информацию, описывающую один столбец результата:

(name, type_code, display_size, internal_size, precision, scale, null_ok)

Первые два элемента (имя и тип) обязательны, а вместо остальных (размер для вывода, внутренний размер, точность, масштаб, возможность задания пустого значения) может быть значение None. Этот атрибут может быть равным None для операций, не возвращающи х значения.

Объект - курсор

execute(operation[, parameters])Исполняет запрос к базе данных или команду СУБД. Параметры (parameters) могут быть представлены в принятой в базе данных нотации в соответствии с атрибутом paramstyle, описанным выше.

executemany(operation, seq_of_parameters) Выполняет серию запросов или команд, подставляя параметры в заданный шаблон. Параметр seq_of_parameters задает последовательность наборов параметров.

Объект - курсор

fetchall() - Возвращает все (или все оставшиеся) записи результата запроса.

fetchmany([size]) - Возвращает следующие несколько записей из результатов запроса в виде последовательности последовательностей. Пустая последовательность означает отсутствие данных. Необязательный параметр size указывает количество возвращаемых записей (реально возвращаемых записей может быть меньше). По умолчанию size равен атрибуту arraysize объекта - курсора.

Объект - курсор

fetchall() - Возвращает все (или все оставшиеся) записи результата запроса.

fetchmany([size]) - Возвращает следующие несколько записей из результатов запроса в виде последовательности последовательностей. Пустая последовательность означает отсутствие данных. Необязательный параметр size указывает количество возвращаемых записей (реально возвращаемых записей может быть меньше). По умолчанию size равен атрибуту arraysize объекта - курсора.

Объект - курсор

fetchone() -Возвращает следующую запись (в виде последовательности) из результата запроса или None при отсутствии данных.

nextset() - Переводит курсор к началу следующего набора данных, полученного в результате запроса (при этом часть записей в предыдущем наборе может остаться непрочитанной). Если наборов больше нет, возвращает None. Не все базы данных поддерживают возврат нескольких наборов результатов за одну операцию.

Объект - курсор

rowcount() - Количество записей, полученных или затронутых в результате выполнения последнего запроса. В случае отсутствия execute - запросов или невозможности указать количество записей равен - 1.

Объекты - типы

API 2.0 предусматривает названия для объектов

- STRING - Строка и символ
- BINARY -Бинарный объект
- NUMBER -Число
- DATETIME - Дата и время
- ROWID - Идентификатор записи
- None – NULL значение (отсутствующее значение)

Объекты - типы

Схематично работа с базой данных может выглядеть примерно так:

- Подключение к базе данных (вызов connect() с получением объекта - соединения).
- Создание одного или нескольких курсоров (вызов метода объекта -соединения cursor() с получением объекта - курсора).
- Исполнение команды или запроса (вызов метода execute() или его вариантов).
- Получение результатов запроса (вызов метода fetchone() или его вариантов).
- Завершение транзакции или ее откат (вызов метода объекта - соединения commit() или rollback()).
- Когда все необходимые транзакции произведены, подключение закрывается вызовом метода close() объекта - соединения.

Знакомство с СУБД

- Допустим, программное обеспечение установлено правильно, и можно работать с модулем sqlite. Стоит посмотреть, чему будут равны константы:

```
>>> import sqlite3
>>> sqlite3.apilevel
'2.0'
>>> sqlite3.paramstyle
' qmark'
>>> sqlite3.threadsafety
1
```

Знакомство с СУБД

- Допустим, программное обеспечение установлено правильно, и можно работать с модулем sqlite. Стоит посмотреть, чему будут равны константы:

```
>>> import sqlite3
>>> sqlite3.apilevel
'2.0'
>>> sqlite3.paramstyle
' qmark'
>>> sqlite3.threadsafety
1
```

Отсюда следует, что sqlite поддерживает DB - API 2.0, подстановка параметров выполняется в стиле строки форматирования языка Python, а соединения нельзя совместно использовать из различных потоков управления (без блокировок).

Проверка sqlite

$ sqlite3 test.db sqlite

.tables sqlite

.exit $ python

>>> import sqlite3

>>> sqlite3.version

Создание базы данных

```
import sqlite3
con = sqlite3.connect('users.db')
cur = con.cursor()
cur.execute('CREATE TABLE users (id INTEGER PRIMARY KEY, firstName VARCHAR(100), secondName VARCHAR(30))')
con.commit()
cur.execute('INSERT INTO users (id, firstName, secondName) VALUES(NULL, "Guido", "van Rossum")')
con.commit()
print (cur.lastrowid)
cur.execute('SELECT * FROM users')
print (cur.fetchall())
con.close()
```

Создание базы данных

Порядок работы:

- Создание соединения с базой данных. Если БД не существует то она будет создана, иначе файл будет открыт.
- Создание объекта курсора для взаимодействия с БД.
- Вставка кортежа со значениями, в зависимости от пользовательского вода

Создание базы данных

Порядок работы:

- Создание соединения с базой данных. Если БД не существует то она будет создана, иначе файл будет открыт.
- Создание объекта курсора для взаимодействия с БД.
- Вставка кортежа со значениями, в зависимости от пользовательского вода

cur.execute('INSERT INTO users VALUES (null, ?, ?)', (firstName, secondName)) con.commit()

Выборка данных

Выборка нескольких строк с данными:

```
import sqlite3
con = sqlite3.connect('users.db')
cur = con.cursor()
cur.execute('SELECT * FROM users')
print (cur.fetchall())
cur.execute('SELECT * FROM users')
for row in cur:
        print ('-'*10)
        print ('ID:', row[0])
        print ('First name:', row[1])
        print ('Second name:', row[2])
        print ('-'*10)
con.close()
```

Выборка данных

Выборка одной строки с данными:

```
import sqlite3
con = sqlite3.connect('users.db')
with con:
        cur = con.cursor()
        cur.execute('SELECT * FROM names')
        data = cur.fetchone()
        print (data[0])
con.close()
```

Есть возможность выбрать заданное количество строк, передав желаемое значение в курсор:

```
cur.execute('SELECT * FROM names')
print (cur.fetchmany(2))
```

Работа с БД

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
""")
```

Работа с БД

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

```
cursor.execute("""
insert into users.usersname values (Null, '1');
 insert into users.usersname values (Null, '2');
""")
# sqlite3.Warning: You can only execute one statement at a time.
```

Заменить cursor.execute() на cursor.executescript()

Работа с БД

Подстановка значений из словаря:

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

или

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

Пример множественной подстановки

```
new_name = [
('1',),
('2',),
('3',),
]
cursor.executemany("insert into Artist values (Null, ?);", new_name)
```

Пример множественной выборки как итератором

```
>>>for row in cursor.execute('SELECT Name from Users ORDER BY
Name LIMIT 3'):
        print(row)
```

Работа с БД

```
engine.execute( "SELECT * FROM users WHERE login='iname';" )
```

Работа с БД

```
users = engine.execute( "SELECT * FROM users WHERE login='iname';"
).fetchall()
```

Работа с БД

```
def get_users(login):
    return engine.execute( "SELECT * FROM users WHERE
    login='%s';" % login ).fetchall()
```

Работа с БД

```
def get_users(login):
    return engine.execute( "SELECT * FROM users WHERE
    login='%s';" % login ).fetchall()
```

SQL injection

```
def get_users(login):
        return engine.execute( "SELECT * FROM users WHERE
        login='%s';" % login ).fetchall()
```

```
get_users("' OR '1' = '1") # вернёт ВСЕХ пользователей
# SELECT * FROM users WHERE login='' OR '1' = '1'
```

SQL injection

```
def get_users(login):
    return engine.execute( "SELECT * FROM users WHERE
    login='%s';" % login ).fetchall()
```

SQL injection

```
def get_users(login):
        return engine.execute( "SELECT * FROM users WHERE login=?;",
        login ).fetchall()
```

SQL injection

```
def get_users(login):
        return engine.execute( "SELECT * FROM users WHERE login=?;",
        login ).fetchall()
```

ORM

```
def get_users(login, only_active=False, with_statistics=False):
        query = "SELECT * FROM users WHERE login=?"
        if only_active:
                query += " AND is_active”
         if with_statistics:
                query += "JOIN user_stat ON user_stat.user_id = user.id"
        return engine.execute(query, login).fetchall()
```

ORM

```
def get_users(login, only_active=False):
        users = User.filter(login=login)
         if only_active:
                users = users.active()
        return users
```

ORM

```
for user in get_users('BD_NAME'):
        print user.id
```

Популярные ORM

- Django ORM
- SQLAlchemy

ORM

```
class User(AbstractCore):

id = Column(BigInteger, primary_key=True)

email = Column(String(255), nullable=False)

first_name = Column('fname', String(255, convert_unicode=True),
nullable=True)

last_name = Column('lname', String(255, convert_unicode=True),
nullable=True)

middle_name = Column('mname', String(255, convert_unicode=True),
nullable=True)
```

Модель

```
class User(AbstractCore):
        id = Column(BigInteger, primary_key=True)
        ...
        def is_user_online(self):
                online_user_ids = [s.user_id for s in get_sessions()]
                return self.id in online_user_ids
```

Model-View-Controller

- модель
- шаблон

Model-View-Controller

- модель
- представление
- контроллер

Model-View-Controller

- модель (предоставляет данные и методы работы с ними)
- представление (отвечает за отображение информации)
- контроллер (обеспечивает связь между пользователем, моделью и представлением)

Model-View-Controller

- fat models;
- thin views;
- stupid templates.

Логирование

Чем logging круче, print:

- легко форматировать сообщение (например, добавить timestamp);
- у каждого сообщения есть свой уровень критчности (debug, error, critical, ...);
- вместо удаления ненужных print можно отключить логирование сообщений указанного уровня критичности;
- можно легко перенаправить вывод логов в файл, syslog, внешний сервис (или куда-то ещё).

Пример

```
import logging
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
for friend in vk_api.get_friends():
        logger.debug('processing %s...' % friend)
        try:
                print vk_api.get_audios(friend.id)
        except VkApiError:
                logger.error('API error occured')
        logger.debug('\tfinished')
```