Python. Подготовка к собеседованию

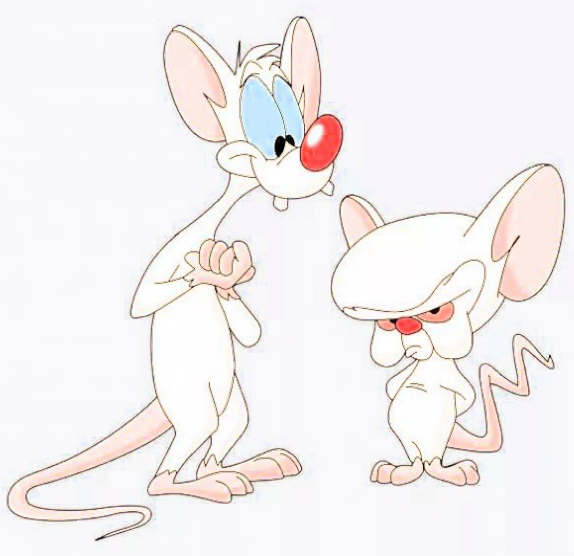
Урок 6

Работа с базами данных в Python

Особенности использования баз данных в Python и Django.

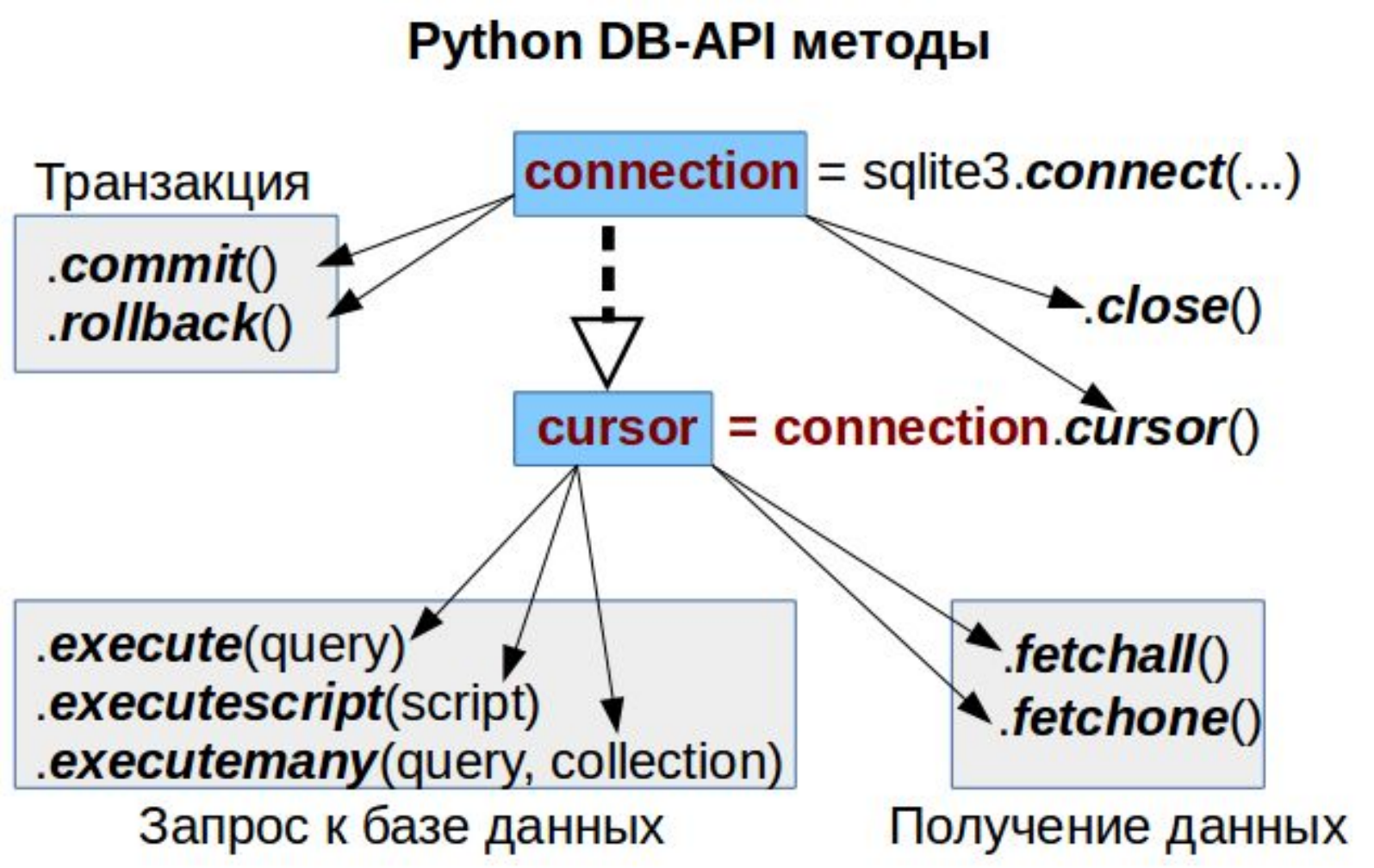
Цели урока

Повторить механизм взаимодействия с базами данных в Python и Django:

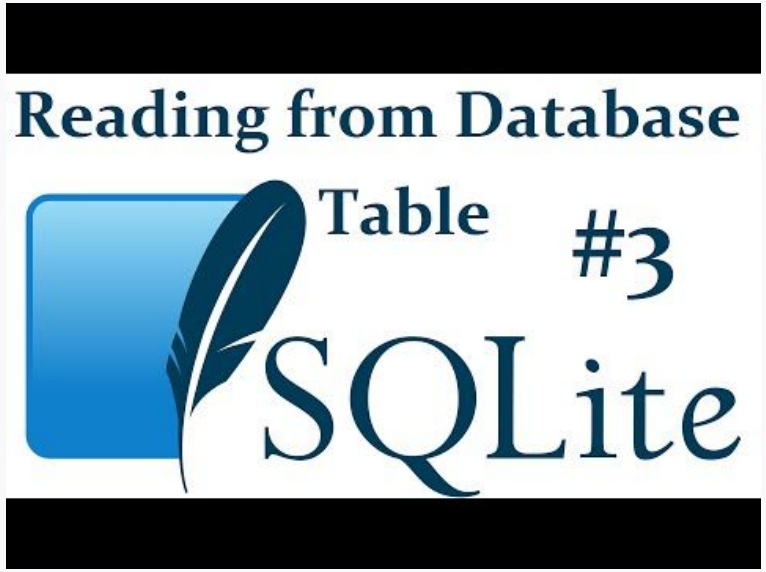


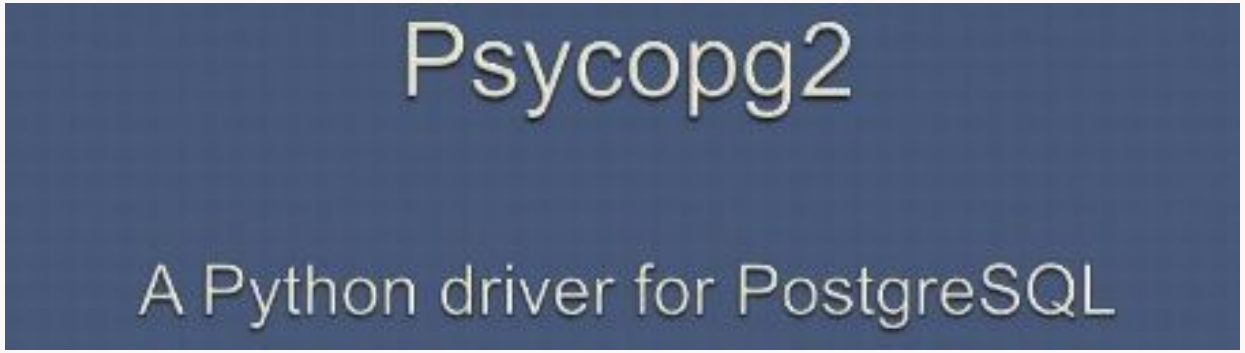
- Python DB-API для реляционных баз данных;
- Модули, реализующие Python DB-API;
- Преимущества SQLite;
- Браузеры баз данных;
- Обработка ошибок;
- Использование SQLAlchemy;
- Взаимодействие с базами данных в Django;
- SQLite или PostgreSQL;
- Mongo DB: ее преимущества, недостатки и области применения, реализация масштабирования;
- Установка и использование MongoDB.

Python DB-API для реляционных баз данных



Модули, реализующие Python DB-API





pyodbc

Преимущества SQLite

- Мультиплатформенность.
- Минимальная настройка.
- Экономичность.
- Отсутствие необходимости в сервере.
- Автономное приложение.
- Свободная лицензия.
- Безопасность хранения.
- Возможность использования с различными языками программирования.

Браузеры баз данных

SQLite Studio

pgAdmin Version 4
Management Tools for PostgreSQL

Обработка ошибок

Использование ORM-библиотеки SQLAlchemy

Установка:

pip install SQLAlchemy

или:

загружаем архив с официального сайта и выполняем: python setup.py install

Подключение:

SQLite: engine = create engine('sqlite:///:memory:', echo=True)

PostgreSQL:
```
engine = create_engine(
    'postgresql+psycopg2://user:password@/dbname'
)
```

MySQL:
```
engine = create_engine(
    'mysql+pymysql://user:pass@127.0.0.1:3306/db', pool_recycle=3600
)
```

Традиционный подход к созданию таблиц, классов и отображений (часть 1)

Описываем таблицу:

```
объект_таблицы = Table(
    название_таблицы,
    объект_коллекции_метаданных
    Column(имя_столбца, тип_данных)
    ...
)
```

Создаем таблицу:

```
metadata.create_all(engine)
```

Традиционный подход к созданию таблиц, классов и отображений (часть 2)

Создаем класс для подготовленной таблицы (через него данные будут передаваться в таблицу):

```
class имя_класса:

    def __init__(self, имя_столбца, ...):
        self.имя_столбца = имя_столбца

        ...

    def __repr__(self):
        return "<имя_таблицы('%s')>" % (self.имя_столбца)
```

Традиционный подход к созданию таблиц, классов и отображений (часть 3)

Создаем отображение между таблицей и классом (связываем таблицу и класс):

```
from sqlalchemy.orm import mapper

mapper(имя_класса,                                    имя_таблицы)
```

Декларативный подход к созданию таблиц, классов и отображений

Все в одном общем классе (для этого используем функцию-конструктор declarative_base):

```
объект_класса_родителя  = declarative_base()

class имя_пользовательского_класса(объект_класса_родителя):
    __tablename__ = имя_таблицы
    имя_столбца = Column(тип_данных)

    def __init__(self, имя_столбца, ...):
        self.имя_столбца = имя_столбца
    ...

  def __repr__(self):
    return "<имя_пользовательского_класса('%s')>" % (self.имя_столбца)
```

Добавление данных в таблицу в SQLAlchemy

Создаем экземпляр класса (запись базы данных):

```
экземпляр_класса = имя_класса(значение_столбца, ...)
```

Создаем класс-генератор объектов сессий:

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker(bind=engine)
```

Создаем объект класса-генератора:

```
session = Session()
```

Добавляем созданный объект к имеющейся сессии:

```
session.add(экземпляр_класса )
```

Завершаем транзакцию:

```
session.commit()
```

Взаимодействие с базами данных в Django

Миграции, их назначение и создание

Анализ изменений в моделях и создание миграций:

python manage.py makemigrations

Применение новых миграций к базе данных:

python manage.py migrate

SQLite или PostgreSQL

- Приложения, не требующие расширения (игры, мобильные приложения).
- Небольшие настольные приложения (расчетные системы, программы учета).
- Приложения для тестирования.

- Приложения со значительной нагрузкой.
- Приложения с числом пользователей более 30.
- Приложения, требующие выполнения операций записи в больших объемах.

MongoDB, ее преимущества и недостатки

- Использование простой и мощной JSON-подобной схемы хранения данных.
- Наличие гибкого языка для формирования запросов.
- Наличие эффективной системы хранения двоичных данных больших объемов.
- Наличие системы логирования операций.
- Поддержка механизмов отказоустойчивости и масштабируемости данных.

- Запросы строятся не на SQL, а на собственном языке.
- Несовершенные инструменты перевода SQL-запросов в MongoDB.
- Продолжительное время установки, требовательность к ресурсам компьютера.
- Сложность использования при решении многокомпонентных задач.

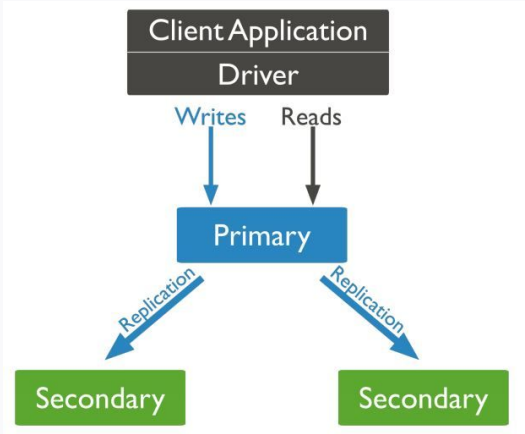
Области применения MongoDB

- Высокопроизводительные распределенные веб-приложения, работающие с большими объемами данных и имеющие требования к гибкости структуры.
- Приложения с пропорциональным распределением нагрузки.
- Приложения с минимальным объемом транзакций.
- Расширяемые проекты.

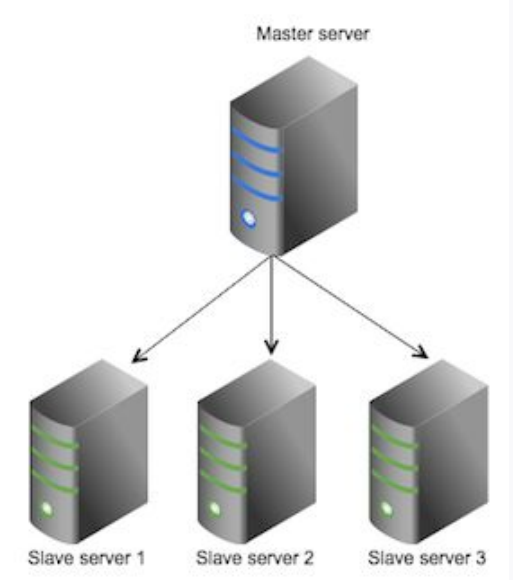
Масштабирование и его реализация в MongoDB

Репликация в MongoDB:

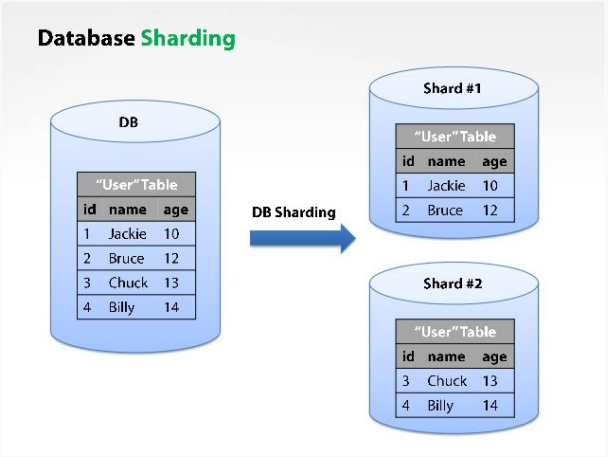
Реплисеты:



Ведущий-ведомый:



Шардинг в MongoDB:



Установка MongoDB и ее использование в Python (часть 1)

pip install pymongo

Устанавливаем соединение с сервером базы данных

```
import pymongo
объект_соединения = pymongo.MongoClient(имя_хоста, номер_порта)
```

Получаем список всех баз данных

```
имя_списка = pymongo.MongoClient().database_names()
```

Подключаемся к одной из баз данных

```
объект_БД = объект_соединения[имя_БД]
```

Выводим все коллекции для базы данных

```
переменная_список_коллекций = объект_БД.collection_names()
```

Получаем доступ к определенной коллекции базы данных

```
переменная_коллекции = объект_БД[имя_коллекции]
```

Установка MongoDB и ее использование в Python (часть 2)

Вставка документа в коллекцию

```
объект_БД.имя_коллекции.insert({"ключ":"значение"})
```

Получение документа

```
переменная_документа = переменная_коллекции.find_one({"ключ": "значение"})
```

Получение всех документов из коллекции

```
for документ_коллекции in переменная_коллекции.find():
    print(документ_коллекции)
```

Обновление документа

```
объект_БД.имя_коллекции.update(
    {"ключ":"старое_значение"},{"ключ":"новое_значение"}
)
```

Удаление документа

```
объект_БД.имя_коллекции.remove({"ключ":"значение"})
```

Практическое задание

Практическое задание (часть 1)

Практическое задание шестого и седьмого урока — разработать десктопное приложение с графическим интерфейсом пользователя, предназначенное для ведения простого складского учета. Это приложение с оконным интерфейсом будет реализовано в привязке к СУБД SQLite и позволит заносить в базу данных сведения о номенклатуре товаров, поставщиках и сотрудниках предприятия.

На данном этапе работы с проектом выполним первую часть практического задания: подготовим фрагменты программного кода, отвечающие за создание таблиц базы данных. В седьмом уроке отображение этих таблиц реализуем в специальных виджетах. Ниже приведено описание необходимых блоков программного кода.

Создать файл базы данных

С помощью одного из менеджеров баз данных (например, SQLiteStudio) создать пустой файл SQLite-базы данных.

Создать подключение к базе данных

Выполнить импорт модуля с Python DB-API для реализации взаимодействия с СУБД SQLite. Создать подключение к базе данных, путь к которой записан в переменную **db_path**. Создать объект-курсор для выполнения операций с данными.

Практическое задание (часть 2)

Создать вспомогательные таблицы:

- ***Категории товаров.*** Написать запрос создания таблицы **categories** (с проверкой ее существования). Таблица должна содержать два поля: **category_name** (категория), **category_description** (описание). Все поля должны быть не пустыми. Поле **category** должно быть первичным ключом.
- ***Единицы измерения товаров.*** Написать запрос создания таблицы **units** с проверкой ее существования. Таблица должна содержать одно поле — **unit** (единица измерения). Оно должно быть не пустым и выступать первичным ключом.
- ***Должности***. Написать запрос создания таблицы **positions** (с проверкой ее существования). Таблица должна содержать одно поле — **position** (должность). Оно должно быть не пустым и выступать первичным ключом.

Практическое задание (часть 3)

Создать основные таблицы:

- ***Товары.*** Написать запрос создания таблицы **goods** с проверкой ее существования. Таблица должна содержать четыре поля: **good_id** (номер товара — первичный ключ), **good_name** (название товара), **good_unit** (единица измерения товара — внешний ключ на таблицу **units**), **good_cat** (категория товара — внешний ключ на таблицу **categories**).
- ***Сотрудники.*** Написать запрос создания таблицы **employees** с проверкой ее существования. Таблица должна содержать три поля: **employee_id** (номер сотрудника — первичный ключ), **employee_fio** (ФИО сотрудника), **employee_position** (должность сотрудника — внешний ключ на таблицу **positions**).
- ***Поставщики.*** Написать запрос создания таблицы **vendors** с проверкой ее существования. Таблица должна содержать шесть полей: **vendor_id** (номер поставщика — первичный ключ), **vendor_name** (название поставщика), **vendor_ownerchipform** (форма собственности поставщика), **vendor_address** (адрес поставщика), **vendor_phone** (телефон поставщика), **vendor_email** (email поставщика).

Дополнительные материалы

1. https://habr.com/post/237889/.
2. http://markov.site/2016/11/28/sqlalchemy-%D1%81%D1%82%D0%B0%D1%80%D1%82/.
3. https://www.8host.com/blog/sozdanie-prilozheniya-django-i-podklyuchenie-k-baze-dannyx/.
4. https://toster.ru/q/198141.
5. https://metanit.com/python/django/5.2.php.
6. http://python-3.ru/page/mongodb-i-python.
7. https://jsehelper.blogspot.com/2016/05/mongodb.html.