IT CAREER HUB

Урок 43.
MongoDB и Python. Модули

Работа с MongoDB из Python

MongoDB — это документоориентированная база данных, с которой можно удобно работать и из Python.

Подключение к базе

Для работы с MongoDB в Python используется библиотека `pymongo`, которая позволяет выполнять запросы к базе в привычной форме: через словари, как в консоли MongoDB.

Установка библиотеки

```python
pip install pymongo
```

Подключение к серверу

Для работы с базой данных MongoDB необходимо создать **объект** `MongoClient`, передав в него строку подключения. Она содержит все параметры для подключения: логин, пароль, адрес сервера и настройки авторизации.

```python
from pymongo import MongoClient

client = MongoClient(
    "mongodb://ich_editor:verystrongpassword"
    "@mongo.itcareerhub.de/?readPreference=primary"
    "&ssl=false&authMechanism=DEFAULT&authSource=ich_edit"
)
```

Основные параметры:

- `ich_editor` — имя пользователя
- `verystrongpassword` — пароль
- `mongo.itcareerhub.de` — адрес сервера
- `authSource=ich_edit` — имя базы, в которой хранятся учётные данные

Проверка подключения

```python
client.admin.command("ping")
print("Connection successful!")
```

Выбор базы данных

```python
db = client["ich_edit"]
```

Особенности:

- Если база не существует, она будет создана **при первом добавлении данных**.
- Подключение устанавливается **лениво** — реально соединение происходит при первом запросе.

Выбор коллекции

```python
products = db["products"]
```

- Коллекции тоже создаются автоматически **при первом добавлении данных**

Закрытие соединения

```python
client.close()
```

MongoClient сам закроет соединение при завершении программы, но при необходимости его можно закрыть вручную.

IT CAREER HUB

Добавление данных

В MongoDB все данные хранятся в **документах**, которые представляют собой обычные Python-словарики (`dict`). MongoDB позволяет добавлять документы несколькими способами:

1. Добавление одного документа — insert_one

```python
product = {
    "name": "Notebook",
    "price": 5.99,
    "stock": 120
}

result = products.insert_one(product)
print("Inserted ID:", result.inserted_id)
```

- Метод `insert_one()` вставляет **один документ**.
- Возвращает объект `InsertOneResult`, из которого можно получить `_id` добавленного документа.

2. Добавление нескольких документов — insert_many

```python
items = [
    {"name": "Pen", "price": 1.50, "stock": 300},
    {"name": "Pencil", "price": 0.99, "stock": 500},
    {"name": "Eraser", "price": 0.75, "stock": 200},
]

result = products.insert_many(items)
print("Inserted IDs:", result.inserted_ids)
```

- Метод `insert_many()` принимает список словарей и добавляет их сразу.
- Возвращает `InsertManyResult` с перечнем `_id` всех вставленных документов.

Особенности:

- Если поле `_id` не указано вручную, MongoDB создаёт его **автоматически**.
- Вы можете **вставить любой словарь**, если структура документов в коллекции **не фиксирована**.
- В отличие от SQL, **необязательно заранее описывать структуру коллекции**.

IT CAREER HUB

Чтение данных

Для получения документов из коллекции используются методы `find_one()` и `find()`.

Примеры

```python
# Получить один документ
doc = products.find_one()
print(doc)

# Получить все документы
docs = products.find()
print(docs)
for item in docs:
    print(item)
```

Особенности:

- Метод `find_one()` возвращает **первый попавшийся** документ (или None, если ничего не найдено).
- Можно передавать **условие поиска** — словарь с нужными параметрами:

```python
docs = products.find({"price": {"$lt": 5}})

for item in docs:
    print(item)
```

Работа с курсором

Метод `find()` возвращает **объект-курсор**, а не список. Это позволяет **не загружать сразу все документы в память**, а перебирать их по одному — как итератор.

Пример

```python
cursor = products.find({"price": {"$gt": 1}})

for doc in cursor:
    print(doc)
```

Особенности:

- Курсор можно использовать в `for`, как обычный итератор.
- После окончания перебора курсор становится **пустым**.
- Можно настроить **лимит, сортировку и пропуск** через методы `.limit()`, `.sort()`, `.skip()`:

```python
for doc in products.find().sort("price", -1).skip(1).limit(2):
    print(doc)
```

Проекция полей

По умолчанию `find()` и `find_one()` возвращают **все поля** документа. Но можно указать, **какие поля нужны**, — это называется **проекцией**.

Примеры

```python
# Вернёт только name, price и id (по умолчанию)
for doc in products.find({}, {"name": 1, "price": 1}):
    print(doc)

# Исключить _id:
for doc in products.find({}, {"_id": 0}):
```

```
    print(doc)

# Оставить только name (все остальные исключаются, включая _id)
for doc in products.find({}, {"_id": 0, "name": 1}):
    print(doc)
```

IT CAREER HUB

Обновление данных

Чтобы изменить документ в MongoDB, используются методы `update_one()` и `update_many()`. Они принимают **фильтр** (поиск нужного документа) и **модификатор** (`$set`, `$inc`, и др.).

Пример: изменить цену одного товара

```python
result = products.update_one(
    {"name": "Notebook"},       # фильтр
    {"$set": {"price": 24.99}}  # изменение
)

print("Matched:", result.matched_count)
print("Modified:", result.modified_count)
```

- Метод `update_one()` изменяет **только первый подходящий** документ.
- Модификатор `$set` обновляет указанное поле.
- `matched_count` — сколько документов подошли под условие.
- `modified_count` — сколько документов реально изменились.

Пример: увеличить цену всех товаров

```python
result = products.update_many(
    {},                     # пустой фильтр – все документы
    {"$inc": {"price": 1}}  # увеличить поле price на 1
)

print("Matched:", result.matched_count)
print("Modified:", result.modified_count)
```

- Метод `update_many()` изменяет **все подходящие** документы.

- `$inc` используется для **увеличения числового значения**.
- Возвращаемый объект также содержит `matched_count` и `modified_count`.

IT CAREER HUB

Удаление данных

MongoDB позволяет удалять как один документ, так и все, подходящие под условие.

Пример: удалить один документ

```python
result = products.delete_one({"name": "Notebook"})
print("Deleted:", result.deleted_count)
```

- `delete_one()` удаляет **первый** документ, подходящий под фильтр
- `deleted_count` покажет, сколько документов было удалено (0 или 1)

Пример: удалить все товары по фильтру

```python
result = products.delete_many({"price": {"$lt": 2}})
print("Deleted:", result.deleted_count)
```

- `delete_many()` удаляет **все документы**, удовлетворяющие условию

IT CAREER HUB

Обработка ошибок

При работе с базой MongoDB через pymongo могут возникать исключения, например, при проблемах с подключением или ошибках в запросах. Чтобы программа не завершалась с ошибкой, нужно использовать блоки `try / except`.

Пример

```python
from pymongo import MongoClient, errors

try:
    client = MongoClient(

"mongodb://ich_editor:wrong_pass@mongo.itcareerhub.de/?authSource=ich_edit"
    )
    db = client["store"]
    products = db["products"]
    products.insert_one({"name": "Lamp", "price": 15.99})
except errors.ConnectionFailure:
    print("Ошибка подключения к MongoDB")
except errors.OperationFailure:
    print("Ошибка авторизации или запроса")
```

Основные типы исключений

- `pymongo.errors.ConnectionFailure` — ошибка подключения к серверу MongoDB
- `pymongo.errors.OperationFailure` — ошибка выполнения запроса (например, неправильные права)
- `pymongo.errors.DuplicateKeyError` — попытка вставить документ с уже существующим `_id`
- `pymongo.errors.PyMongoError` — базовый класс для всех исключений pymongo

IT CAREER HUB

Задания для закрепления 1

1. Какими способами можно добавить данные в коллекцию MongoDB?

a. `insert_one()`
b. `insert_many()`
c. `add_documents()`
d. `create_one()`

Посмотреть ответ

2. Какой тип данных возвращается методом find()?

a. Список
b. Курсор (итератор)
c. Генератор
d. Строка JSON
e. Словарь

Посмотреть ответ

IT CAREER HUB

Модули

Модуль — это любой `.py` файл, содержащий переменные, функции, классы и другие конструкции.

Зачем нужны модули

- Упрощают структуру программы и позволяют разделять код по смыслу
- Повышают читаемость и удобство сопровождения кода
- Облегчают повторное использование и помогают избегать дублирования
- Позволяют использовать готовые решения из стандартной библиотеки Python
- Упрощают тестирование

Виды модулей

- **Встроенные модули** — входят в стандартную библиотеку Python (например, `math`, `random`, `datetime`).
- **Сторонние модули** — устанавливаются через менеджер пакетов `pip`.
- **Пользовательские модули** — это любые `.py`-файлы, написанные самостоятельно.

Работа с собственным модулем

Чтобы импортировать свой модуль, он должен находиться в той же папке или в доступном пути.

Пример

Создадим файл `math_utils.py` со следующим содержимым:

```python
print("Inside math_utils.py")

def average(numbers):
    return sum(numbers) / len(numbers)
```

```
def maximum(numbers):
    return max(numbers)
```

Теперь в другом файле (в той же папке), например `main.py`, можно использовать этот модуль:

```python
import math_utils
from math_utils import maximum

values = [10, 20, 30]

print(math_utils.average(values))
print(maximum(values))
```

Прямой запуск модуля

Иногда модуль может использоваться и как **библиотека**, и как самостоятельный **скрипт**, запускаемый напрямую. Чтобы различать эти случаи, в Python используют конструкцию:

```python
if __name__ == "__main__":
    # Код, который будет выполняться только при прямом запуске
```

Когда модуль запускается напрямую, переменная `__name__` получает значение `"__main__"`. Если модуль **импортируется** в другой файл — `__name__` будет равно имени модуля.

Пример

Файл `math_utils.py`:

```python
# Это сообщение будет выведено при прямом запуске или при импорте
print("Inside math_utils.py")

def average(numbers):
    return sum(numbers) / len(numbers)

def maximum(numbers):
    return max(numbers)

# Это сообщение будет выведено только при прямом запуске
if __name__ == "__main__":
    print("Inside main file!")
```

- Только если запустить `math_utils.py` напрямую — будет выведено `"Inside main file!"`.

Компиляция модулей

Когда Python выполняет импорт модуля, он автоматически компилирует его в байт-код — промежуточный формат, который быстрее загружается при следующем запуске.

Скомпилированный файл сохраняется с расширением `.pyc` и помещается в папку `__pycache__/`.

Пример

Допустим, у нас есть модуль `math_utils.py`. При первом импорте Python создаст файл:

```
__pycache__/math_utils.cpython-312.pyc
```

Этот файл содержит байт-код, и Python будет использовать его сам при импорте.

Когда пересоздаётся .pyc файл

- При первом импорте модуля
- Если содержимое `.py`-файла изменилось
- Если обновилась версия Python

IT CAREER HUB

Пакеты и папки

Пакет — это папка, которая содержит модули и файл **`__init__.py`. Он позволяет группировать модули по смыслу и создавать вложенную структуру.**

Файл `__init__.py` может быть пустым, но чаще в нём явно указывают, какие модули или функции доступны при импорте пакета.

Пример структуры

```
Python
modules/
├── main.py
├── math_utils.py
├── analyzer.py
└── tools/
    ├── __init__.py
    ├── text_utils.py
    └── helpers/
        ├── __init__.py
        └── string_tools.py
```

Содержимое `tools/text_utils.py`:

```
Python
def count_words(text):
    return len(text.split())
```

Содержимое `tools/helpers/string_tools.py`:

```
Python
def reverse(text):
    return text[::-1]
```

Теперь в `tools/__init__.py` можно импортировать нужные функции, чтобы использовать пакет как единое целое:

```python
from .text_utils import count_words
from .helpers.string_tools import reverse
```

- Благодаря этому можно обращаться к функциям напрямую через `tools.count_words()` или `tools.reverse()`, **не указывая имена вложенных файлов**.

Содержимое `analyzer.py` (вне пакета):

```python
import tools  # Импорт пакета как модуля

text = "What a beautiful day to learn Python!"

print(tools.count_words(text))
print(tools.reverse(text))
```

IT CAREER HUB

Задания для закрепления 2

1. Сопоставь термин и определение

1. Модуль
2. Пакет
3. `__pycache__`
4. `__init__.py`

a. Папка с модулями и `__init__.py`, объединёнными по смыслу
b. Любой `.py` файл с переменными, функциями, классами и т.д.
c. Файл, обозначающий папку как пакет и управляющий импортом
d. Папка, где Python сохраняет байткод модулей

Посмотреть ответ

2. Что делает __pycache__/math_utils.cpython-312.pyc?

a. Хранит текст модуля в сжатом виде
b. Является скомпилированной версией `.py` файла
c. Позволяет запускать программу без Python
d. Содержит переменные окружения

Посмотреть ответ

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Добавление данных в коллекцию MongoDB	Ответ: a, b
2. Тип данных, метод `find()`	Ответ: b
Задания на закрепление 2	Вернуться к заданиям
1. Термин и его определение	Ответ: 1-b, 2-a, 3-d, 4-c
2. Действие `__pycache__/math_utils.cpython-312.pyc`	Ответ: b

IT CAREER HUB

Практическая работа

1. Поиск заказов с маленькой суммой

Прочитайте все документы из коллекции `orders`, у которых сумма (`amount`) **меньше 10**. Выведите каждый найденный заказ построчно.

Пример вывода:

```python
{'_id': ObjectId('...'), 'id': 3, 'customer': 'Olga', 'product': 'Kiwi',
'amount': 9.6, 'city': 'Berlin'}
{'_id': ObjectId('...'), 'id': 5, 'customer': 'Olga', 'product': 'Banana',
'amount': 8, 'city': 'Madrid'}
```

Решение:

```python
from pymongo import MongoClient

client = MongoClient(
    "mongodb://ich_editor:verystrongpassword"
    "@mongo.itcareerhub.de/?readPreference=primary"
    "&ssl=false&authMechanism=DEFAULT&authSource=ich_edit"
)

db = client["ich_edit"]
orders = db["orders"]

results = list(orders.find({"amount": {"$lt": 10}}))

for order in results:
    print(order)
```

IT CAREER HUB

2. Сохранение результатов в другую коллекцию

Сохраните все найденные заказы в новую коллекцию `orders_lesson_43`. После записи выведите, сколько документов было добавлено.

Пример вывода:

```
Python
6 documents inserted into 'orders_lesson_43'.
```

Решение:

```python
orders_lesson_43 = db["orders_lesson_43"]
orders_lesson_43.delete_many({})  # очищаем перед вставкой

if results:
    insert_result = orders_lesson_43.insert_many(results)
    print(f"{len(insert_result.inserted_ids)} documents inserted into
'orders_lesson_43'.")
else:
    print("No documents to insert.")
```