**Модуль 8: NoSQL бази даних**

**Заняття 1: Основи роботи з MongoDb**

**Вступ**

Реляційні бази даних мають ряд неприємних особливостей:

* реляційні бази погано масштабуються, з ними дуже складно створювати розподілені сховища;
* проектування великих баз з безліччю компонентів потребує значних зусиль. Це приведення сутностей до нормальних форм і складності у відображенні зв'язків типу багато-до-багатьох. Такі схеми важко читати і розуміти їхнє бізнес-застосування;
* еволюція схеми даних майже завжди відстає від нових потреб бізнесу. Часто вона встигає застаріти ще до випуску нової фічі. Міграція на оновлену схему займає потворно довгі години, протягом яких «сервер лежить».

INFO

Фіча (від англійської feature – особливість, незвичайна властивість) – це певна деталь або частина продукту, яка має специфічні характеристики. Також фічами називають механізми, що дозволяють додавати нові функціональності, веб-частини та елементи дизайну в продукт.

Причини такої поведінки RDb лежать у виконанні принципів ACID:

* Atomicity - Атомарність гарантує, що жодна транзакція не буде зафіксована в системі частково. Будуть або виконані її підоперації, або не виконано жодної. Оскільки на практиці неможливо одночасно і атомарно виконати всю послідовність операцій всередині транзакції, вводиться поняття «відкочування» (rollback): якщо транзакцію не вдається повністю завершити, результати всіх її досі зроблених дій будуть скасовані і система повернеться до «зовнішнього вихідного» стану — зі сторони здаватиметься, що транзакції і не було. Звичайно, лічильники, індекси та інші внутрішні структури можуть змінитися, але якщо СУБД запрограмована без помилок, це не вплине на зовнішню її поведінку.
* Consistency - Узгодженість. Транзакція, що досягає свого нормального завершення (EOT - end of transaction, завершення транзакції) і, тим самим, фіксує свої результати, зберігає узгодженість бази даних. Інакше кажучи, кожна успішна транзакція за визначенням фіксує лише допустимі результати. Ця умова є необхідною для підтримки четвертої властивості.
* Isolation - Ізольованість. Під час виконання транзакції паралельні транзакції не повинні впливати на її результат. Ізольованість - вимога вартісна, тому в реальних БД існують режими, що не повністю ізолюють транзакцію (рівні ізольованості Repeatable Read і нижче).
* Durability - Довговічність. Незалежно від проблем на нижніх рівнях (наприклад, знеструмлення системи або збої в обладнанні) зміни, зроблені успішно завершеною транзакцією, повинні залишитися збереженими після повернення системи до роботи. Іншими словами, якщо користувач отримав підтвердження від системи, що транзакція виконана, він може бути впевнений, що зроблені ним зміни не будуть скасовані через будь-який збій.

Що робити, якщо потрібно реалізувати кластер та рознести базу на фізично розділені машини?

[**CAP**](https://en.wikipedia.org/wiki/CAP_theorem)**теорема** (https://en.wikipedia.org/wiki/CAP\_theorem)

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/intro_01#cap-%D1%82%D0%B5%D0%BE%D1%80%D0%B5%D0%BC%D0%B0)Будь-яке сховище даних має три базові властивості:

* Узгодженість даних (Consistency). Тобто дані повинні бути повними та несуперечливими (включно і у всіх вузлах кластера).
* Доступність (Availability). Грубо кажучи, це швидкість відповіді сервера на наш запит (для запису або читання).
* Стійкість до поділу (Partition tolerance). Це означає, що у разі поділу системи на кілька частин, кожна з них, якщо вона доступна, повинна мати можливість працювати автономно, віддаючи коректний відгук та надаючи свої дані. Обрив зв'язків у кластері не повинен впливати на підсумкову роботу.

Теорема CAP повідомляє нам, що із цих трьох компонентів ми можемо отримати лише два.

Реляційні бази даних реалізують CA комбінацію та є нестійкими до поділу.

NoSQL технології були народжені з метою вирішення проблеми стійкості до поділу, тобто ефективно працювати на кластерах. Реляційна модель не в змозі впоратися з цим завданням, оскільки була створена для інших цілей та інших умов. Вам не вдасться «просто відпиляти пару-трійку таблиць або спокійно їх поділити у сусідній кластер».

Сховища NoSQL за своєю природою можуть бути легко розділені на кластер через специфічну структуру зберігання даних.

Справжня сутність теореми CAP проявляється саме в умовах розподіленої системи. Очевидно, що створювати нестійкий до поділу кластер — позбавлено практичної користі. Тобто кластер апріорі повинен створюватися стійким до поділу. Розуміння цього факту дозволяє нам побачити теорему CAP у новому світлі: з узгодженості та доступності можна вибрати тільки щось одне — або використовувати розумний компроміс між цими двома пунктами (а не трьома, як можна подумати з оригінального визначення).

Друге завдання, яке намагаються вирішити ідеологи NoSQL технологій, - підвищення доступності, тобто отримувати швидку відповідь сервера.

**Типи NoSQL баз даних**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/intro_01#%D1%82%D0%B8%D0%BF%D0%B8-nosql-%D0%B1%D0%B0%D0%B7-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)

**Key-value/Key-cache**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/intro_01#key-valuekey-cache)База є, власне, словником з унікальними ключами, за якими відбувається пошук значень. Пошук відбувається за константний час *O(n)*. Значенням може бути будь-що: кеш [**Memcached**](https://memcached.org/) (https://memcached.org/), структуровані дані [**Redis**](https://redis.io/) (https://redis.io/)тощо.

**Document Store**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/intro_01#document-store)Кожним рядком (записом) у базі є структурований документ зі стандартизованим або відмінним набором полів. Представник [**MongoDB**](https://www.mongodb.com/) (https://www.mongodb.com/)

**Graph**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/intro_01#graph)Різновид баз даних із реалізацією мережевої моделі у вигляді графа та його узагальнень. Основними елементами моделі є вузли та зв'язки. Графові бази даних застосовуються для моделювання соціальних графів (соціальних мереж), в біоінформатиці, а також для семантичної павутини. Представник [**Neo4j**](https://neo4j.com/) (https://neo4j.com/)

**Object Db**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/intro_01#object-db)База даних, у якій дані моделюються як об'єкти, їх атрибути, методи і класи. Об'єктно-орієнтовані бази даних зазвичай рекомендовані для тих випадків, коли потрібна високопродуктивна обробка даних, що мають складну структуру. Представник:[**Realm**](https://realm.io/) (https://realm.io/)

**Wide Column Store**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/intro_01#wide-column-store)Те саме, що і RDb база даних у сенсі застосування таблиць, що містять рядки (записи) та стовпці, але тип даних в одному стовпці необов'язково буде єдиним для всіх записів. В якомусь сенсі можна уявити таку таблицю як двовимірний словник. Представники:[**Amazon DynamoDB**](Amazon%20DynamoDB)(<https://aws.amazon.com/dynamodb/?nc1=h_ls>)**,** [**Apache Accumulo**](https://accumulo.apache.org/) (https://accumulo.apache.org/)**,**[**Apache Cassandra**](https://cassandra.apache.org/_/index.html) (https://cassandra.apache.org/\_/index.html)

**Основи MongoDB**

Сучасному вебзастосунку необхідне сховище даних. Традиційно для цього використовувалися SQL бази даних, але прогрес не стоїть на місці та з'явилася альтернатива у вигляді NoSQL баз даних. Ці бази даних взяли до уваги розподілену природу мережі Інтернет і натомість зосередилися на паралелізмі для масштабування продуктивності. Ми розглянемо провідну документоорієнтовану базу даних MongoDB.

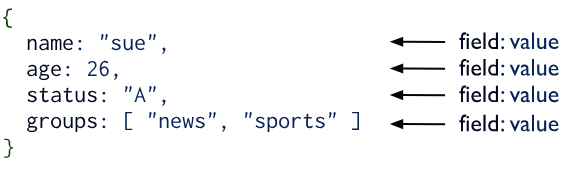
MongoDB є нереляційною базою даних типу NoSQL. База базується на моделі документів — об'єкти даних зберігаються у вигляді окремих документів у колекції.

**Колекція та документ**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/intro_mdb_02#%D0%BA%D0%BE%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D1%8F-%D1%82%D0%B0-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82)Дані MongoDB групуються в колекції. Колекція — це збірник документів, які мають однакове призначення. Колекція подібна до таблиці в SQL базі даних, але відрізняється тим, що для колекції немає суворої схеми, і документи колекції можуть мати різну структуру.

Документ — це представлення елемента інформації у базі даних. Вони можуть складатися з підпорядкованих документів, і ця модель даних більше підходить для веб-застосунків. Максимальний розмір документа обмежений до 16 Мбайт.

У MongoDB у якості мови запитів використовується JavaScript та JSON-структури. Вибір мови запиту пояснюється тим, що MongoDB використовує JSON-формат для представлення документів та виведення результатів. Фізично JSON-структури зберігаються у бінарному BSON-форматі.



Документи (тобто об'єкти) відповідають власним типам даних у багатьох мовах програмування. Вбудовані документи та масиви скорочують потребу у вартісних об'єднаннях.

У наведеній нижче таблиці представлені відмінності у термінології між SQL і MongoDB.



Первинний ключ. В SQL потрібно вказати будь-який унікальний стовпчик або комбінацію стовпчиків як первинний ключ. У MongoDB первинний ключ автоматично встановлюється у полі \_id. Фактично змінна \_id — це об'єкт типу ObjectId.

\_id: ObjectId('5f15996fbbde793a107af359')

Він містить 12 байт, кожен із яких формується певним чином.

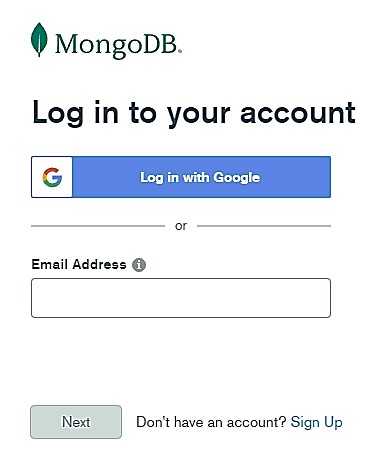
* 4 - байтове значення (5f15996f), що позначає секунди, починаючи з останнього запису
* 3 - байтове значення (bbde79), що позначає ідентифікатор машини
* 2 - байтове значення (3a10), що позначає ідентифікатор процесу
* 3 - байтовий лічильник (7af359), починаючи з випадкового значення

**MongoDB Atlas**

Замість встановлення бази даних локально, ми будемо використовувати хмарне сховище [MongoDB Atlas](https://www.mongodb.com/cloud/atlas) (https://www.mongodb.com/atlas/database).

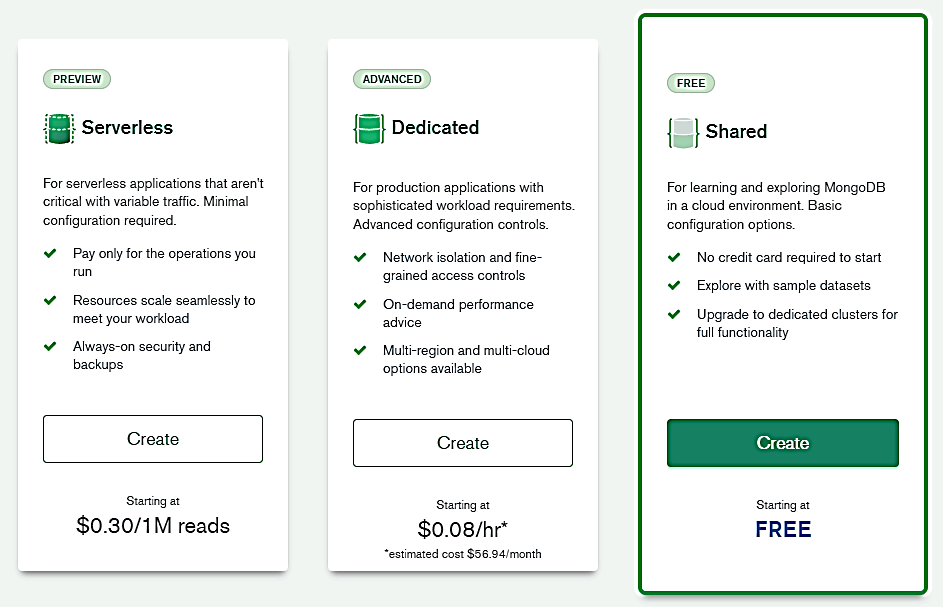
MongoDB Atlas — це варіант служби MongoDB, розміщеної у хмарі, яка не вимагає додаткових витрат на встановлення та пропонує безкоштовний рівень для початку роботи.

Пройдемо реєстрацію за цим [посиланням](https://account.mongodb.com/account/login) (<https://account.mongodb.com/account/> login). Найкраще це зробити за допомогою Google акаунта.



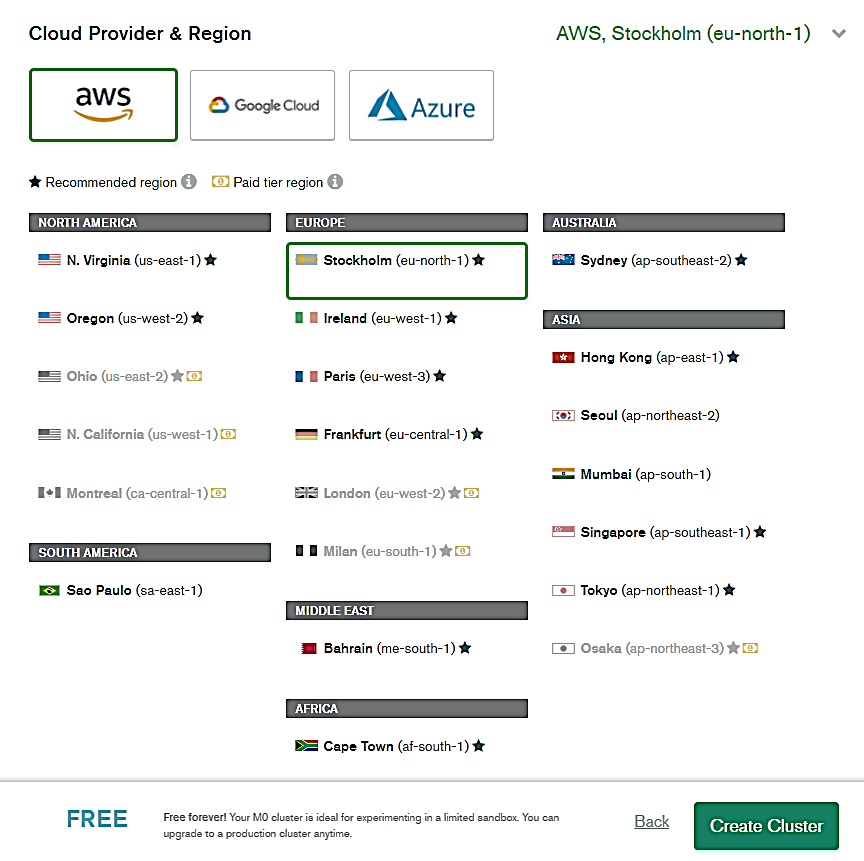
**Вибір тарифу**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/connect_03#%D0%B2%D0%B8%D0%B1%D1%96%D1%80-%D1%82%D0%B0%D1%80%D0%B8%D1%84%D1%83)Насамперед нас попросять вибрати тариф, що цікавить. Ми вибираємо безкоштовний тариф, він цілком підійде для наших завдань. Розмір виділеної бази даних (БД) складатиме 512Mb



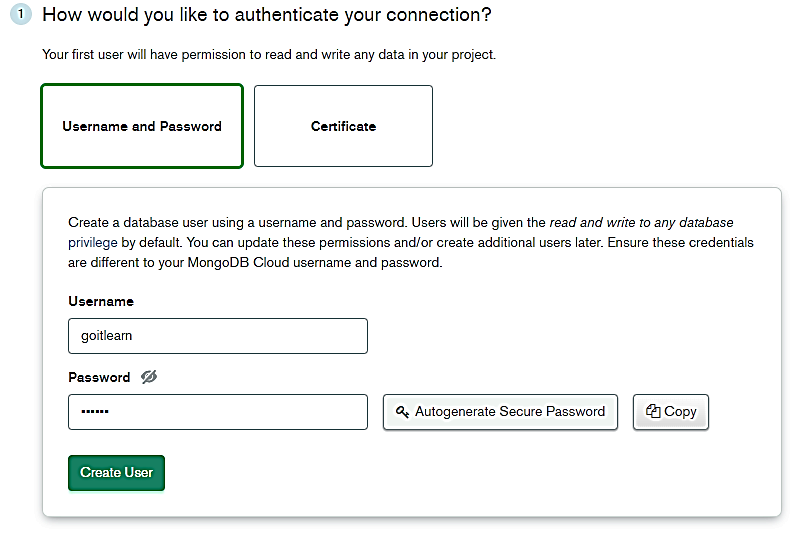
**Вибір провайдера**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/connect_03#%D0%B2%D0%B8%D0%B1%D1%96%D1%80-%D0%BF%D1%80%D0%BE%D0%B2%D0%B0%D0%B9%D0%B4%D0%B5%D1%80%D0%B0)На другому етапі з'явиться пропозиція обрати провайдера для нашої БД. Під час вибору є три найбільші хмарні постачальники: AWS, Google (Google cloud), Microsoft (Azure). Вибір компанії залишаємо на ваш розсуд, головне обрати регіон ближче до географічного розташування. На скріншоті нижче вибрано компанію AWS та регіон Стокгольм (Швеція). Після вибору натисніть зелену кнопку Create Cluster.



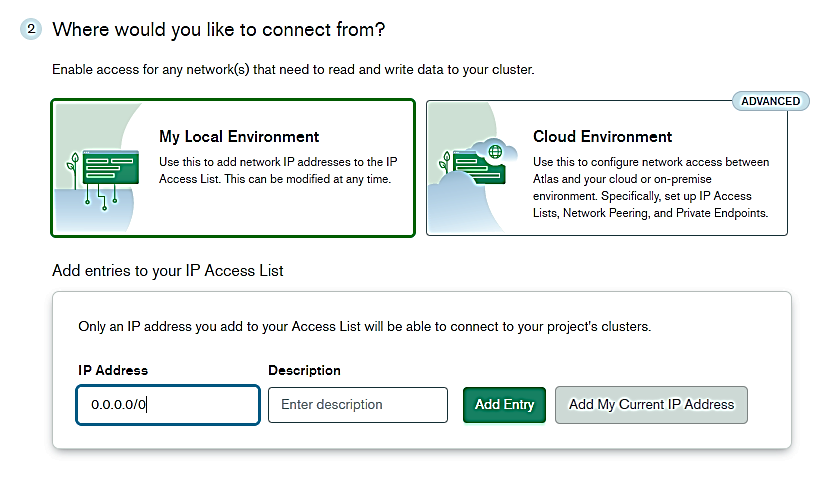
**Створення користувача**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/connect_03#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D1%83%D0%B2%D0%B0%D1%87%D0%B0)Наступний крок — створення користувача для доступу до нашої хмарної бази даних з програми — це так звані "креденшіалс" (credentials) (*Прим. ви ще не раз почуєте цей термін надалі під час роботи*). Їх ми використовуватимемо далі, коли перейдемо до розробки нашої програми. Ми вибрали ім'я користувача goitlearn. Пароль придумайте самі або скористайтеся функцією автогенерації. Обов'язково запам'ятайте та збережіть ім'я користувача та пароль, щоб використовувати їх далі.

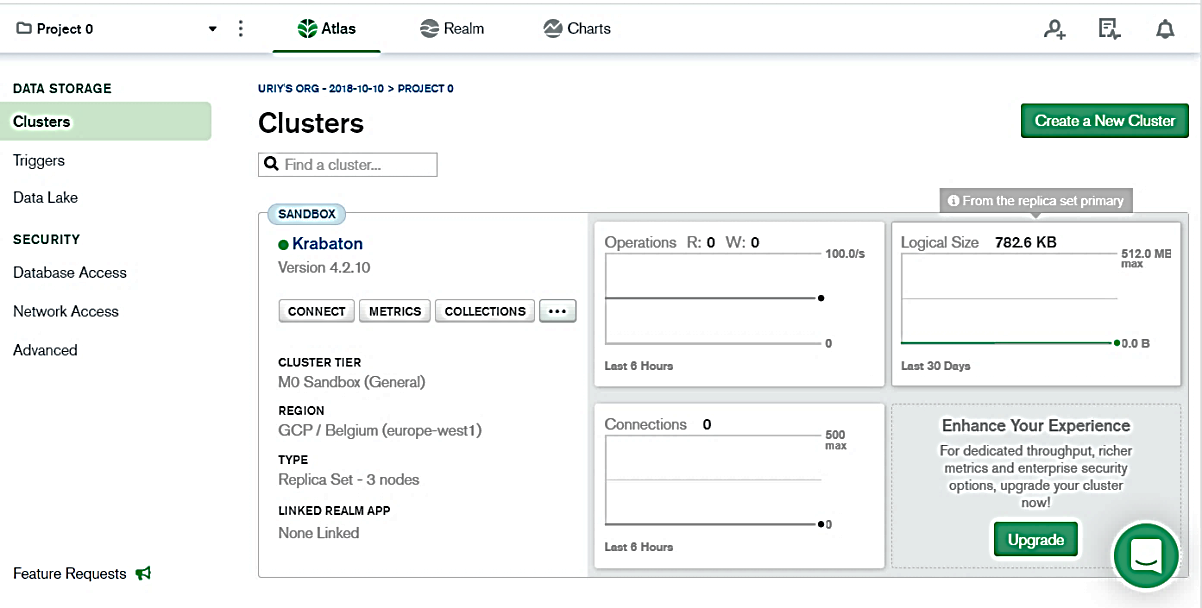


**Доступ з мережі**

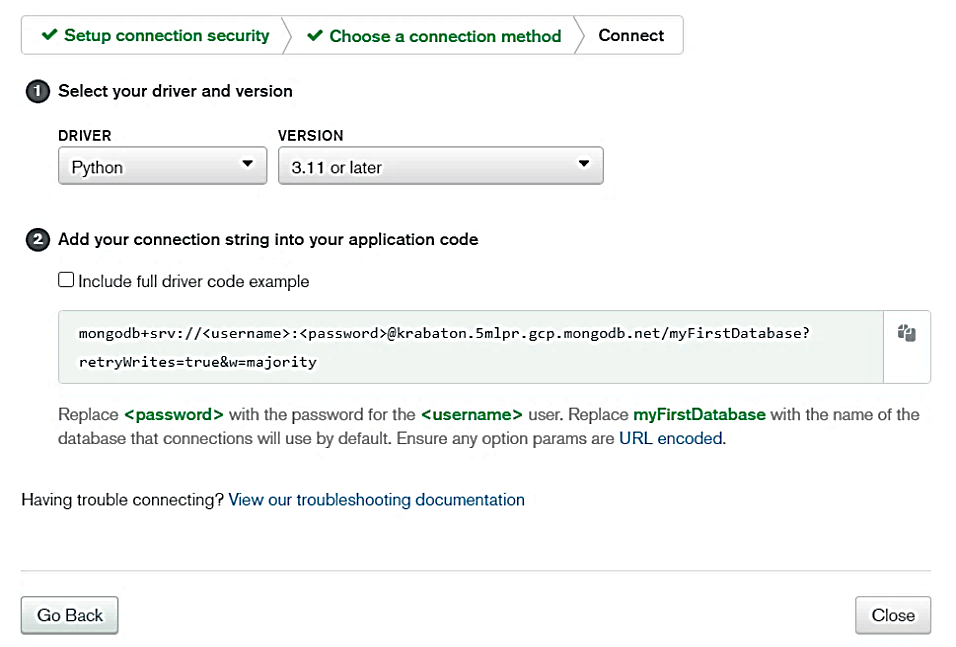
[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/connect_03#%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF-%D0%B7-%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D1%96)Хорошою практикою є обмеження доступу до БД, застосунку тощо за IP адресою. Таким чином ми підвищуємо безпеку нашого застосунку. Зловмиснику важче зламати наш будь-який сервіс не з нашої IP адреси. Тому на останньому кроці нас попросять додати IP, з якого ми можемо отримати доступ до нашої БД. Ми можемо внести свій IP, дізнавшись його, наприклад, за допомогою сервісу [myip](https://www.myip.com/) або, вказавши адресу 0.0.0.0/0, яка говорить, що можна використовувати будь-яку IP адресу.



Попереднє налаштування завершено і тепер можна повернутися на вкладку Cluster.



Тут вибираємо кнопку Connect. У вікні, що відкрилося, вибираємо Connect your application.



Потрібно вибрати драйвер для Python. Нам нададуть рядок для доступу до бази даних.

mongodb+srv://<username>:<password>@krabaton.5mlpr.gcp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority

У цьому рядку є три плейсхолдери:

* <username>
* — ім'я користувача, що ми зареєстрували (
* *Прим*. у нас це goitlearn)
* <password> — його пароль (*Прим*. ви його повинні були придумати, коли створювали користувача)
* myFirstDatabase — ім'я бази даних. (*Прим*. можна вибрати будь-яке, що відповідає вашому завданню)

За допомогою цього рядка ми будемо з'єднуватися з нашою хмарною базою даних MongoDB.

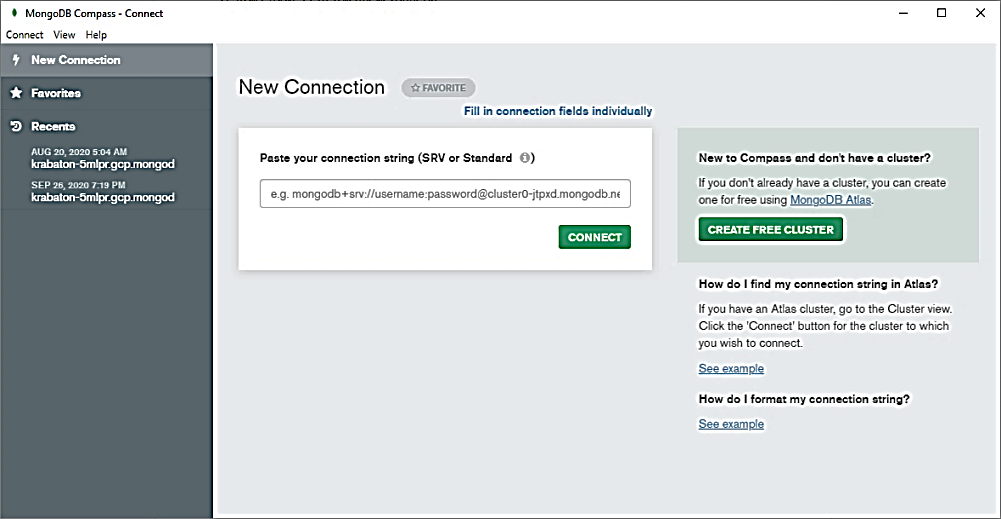
**MongoDB GUI**

Існує багато графічних інструментів управління MongoDB. Ці інструменти підвищують продуктивність наших завдань розробки та адміністрування MongoDB. Вони зручні і найчастіше надають внутрішню консоль для роботи з БД або її аналог. Розглянемо два популярні графічні інструменти MongoDB і розберемося, як у них підключитися до хмарної бази даних. Цих двох інструментів вистачить для всіх потреб під час проходження курсу.

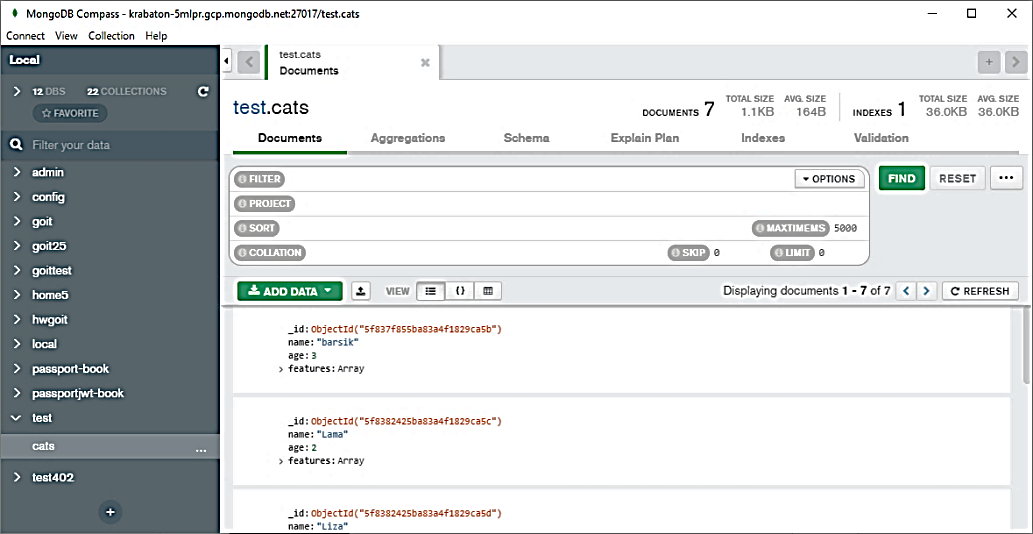
[**MongoDB Compass**](https://www.mongodb.com/products/compass) (https://account.mongodb.com/account/login)

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/gui_04#mongodb-compass)Графічний інтерфейс для MongoDB від самої компанії. Має непогану візуалізацію даних і має повну функціональністю CRUD. Доступний для платформ Linux, Mac або Windows.

Підключення до бази даних здійснюється за допомогою рядка SRV.



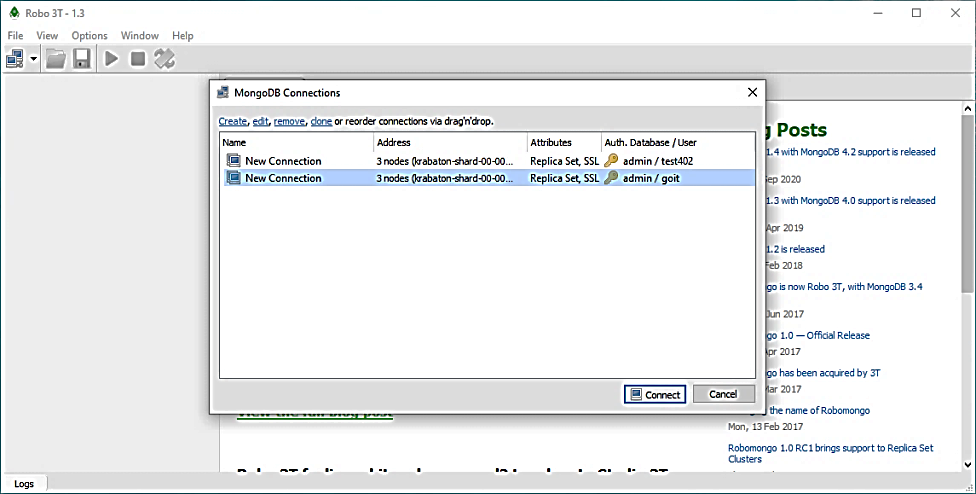
Необхідно вставити рядок із вже підставленими плейсхолдерами в блок підключення Compass та натиснути кнопку connect.



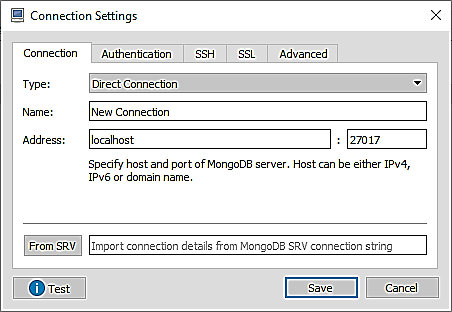
Ліворуч буде меню з нашими базами та колекціями. При виборі колекції ми потрапляємо до панелі управління колекцією. Замість консолі, Compass надає панель для пошуку та фільтрації за нашими документами. І повну функціональність по CRUD операціям над документами, без необхідності використання мови запитів. Compass досить простий та дуже добре підходить для новачка.

**R**[**obo 3T**](https://robomongo.org/) (https://robomongo.org/)

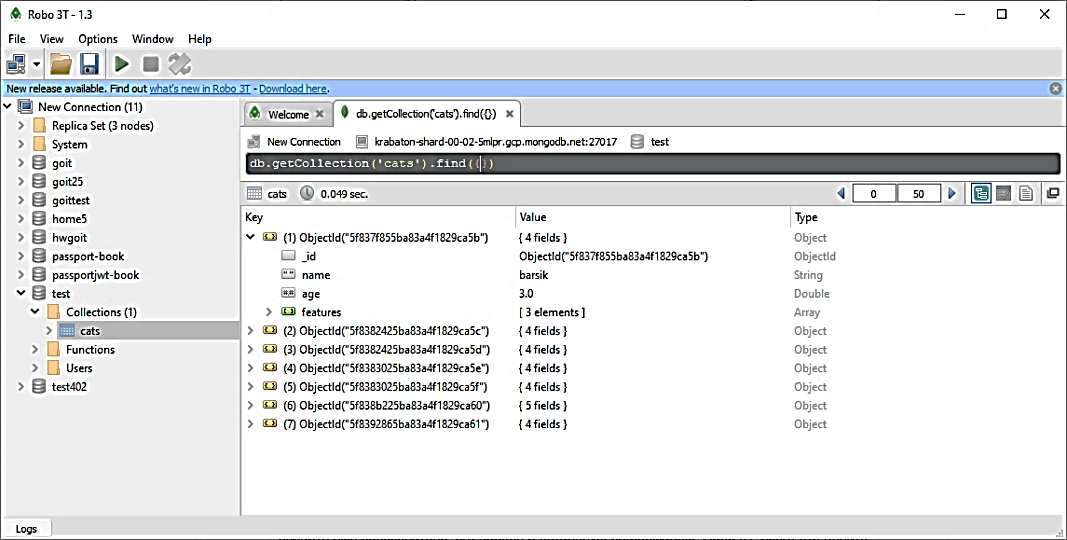
[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/gui_04#robo-3t)Популярний безкоштовний графічний інтерфейс для ентузіастів MongoDB. Цей легкий інструмент з відкритим кодом має кросплатформенну підтримку, а також вбудовує оболонку mongo у свій інтерфейс для забезпечення взаємодії як на основі оболонки, так і на основі графічного інтерфейсу. Він розроблений 3T Software, командою, що стоїть за Studio 3T, IDE для MongoDB, яка є повністю платною, на відміну від цієї версії.



На старті з'явиться вікно, що пропонує підключитися до бази даних. Тут є невеликий нюанс. Натисніть Create.



Ми побачимо вікно, що пропонує нам ввести credentials для приєднання. Не поспішайте вводити ім'я користувача, його пароль та іншу інформацію. Robo 3T вміє це робити сам із рядка підключення, який нам надає MongoDB Atlas. Вставте рядок у саме нижнє поле введення і натисніть From SRV, і Robo 3T сам підставить всюди потрібну інформацію. Дайте своєму підключенню ім'я, натисніть save, а потім виконайте connect до хмарної бази даних.



Інтерфейс схожий на попередню програму, але замість панелі вгорі, ми бачимо консоль із запитами до бази даних. Наприклад, виведення колекції було виконано командою

db.

getCollection

('cats').find({})

Команди виконуються кнопкою F5 або комбінацією клавіш Ctrl+Enter. Як ілюстрації ми будемо використовувати цей GUI. І вже в наступному розділі повною мірою ним скористаємося.

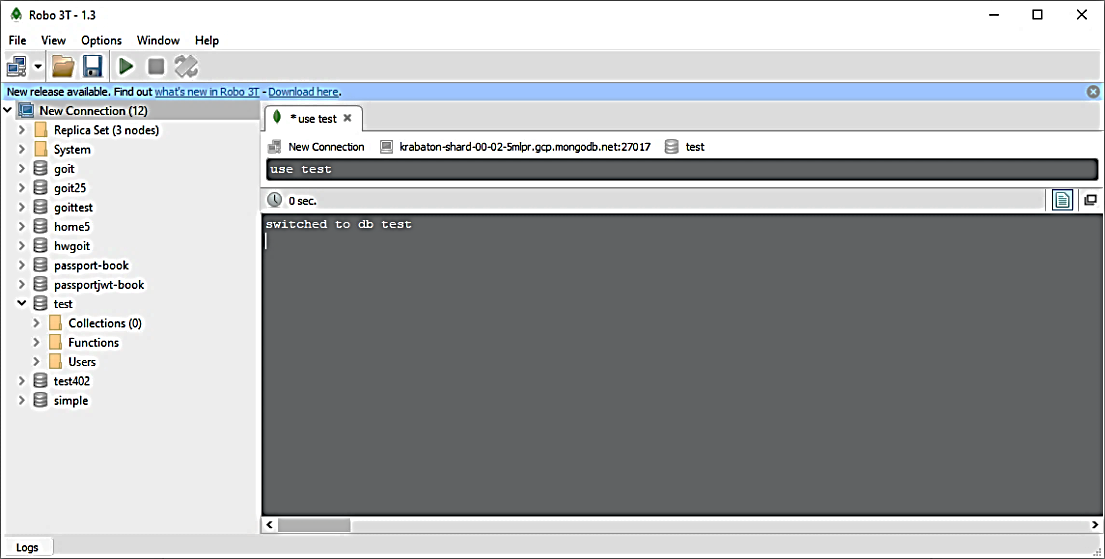
**Додавання даних в колекцію**

Після підключення до хмарної бази MongoDB, клацніть правою кнопкою миші на будь-якій базі в лівій панелі Robo 3T. В контекстному меню виберіть Open shell — відкриється консоль, в якій ми будемо працювати.

Перша команда буде

use test

Виконується вона натисканням клавіші F5 або комбінацією клавіш Ctrl+Enter. Тепер як поточна буде встановлена БД test.



Якщо ми хочемо дізнатися, яка бд використовується зараз, ми можемо скористатися командою db:

db

Використовуючи команду db.stats(), можна отримати статистику за поточною базою даних, а статистику за колекцією cats:

db.cats.stats()

Але, перш ніж отримати статистику з колекції, її потрібно створити.

Можна використовувати для цього три методи:

* insertOne: додає один документ
* insertMany: додає кілька документів
* insert: може додавати як один, так і декілька документів

Клікнемо правою кнопкою миші на базі test, у контекстному меню викличемо консоль і вставимо інформацію про першого кота:

db.cats.insertOne({

name: 'barsik',

age: 3,

features: ['ходить в капці', 'дає себе гладити', 'рудий'],

})

Нам повернеться результат:

{

"acknowledged": true,

"insertedId": ObjectId(

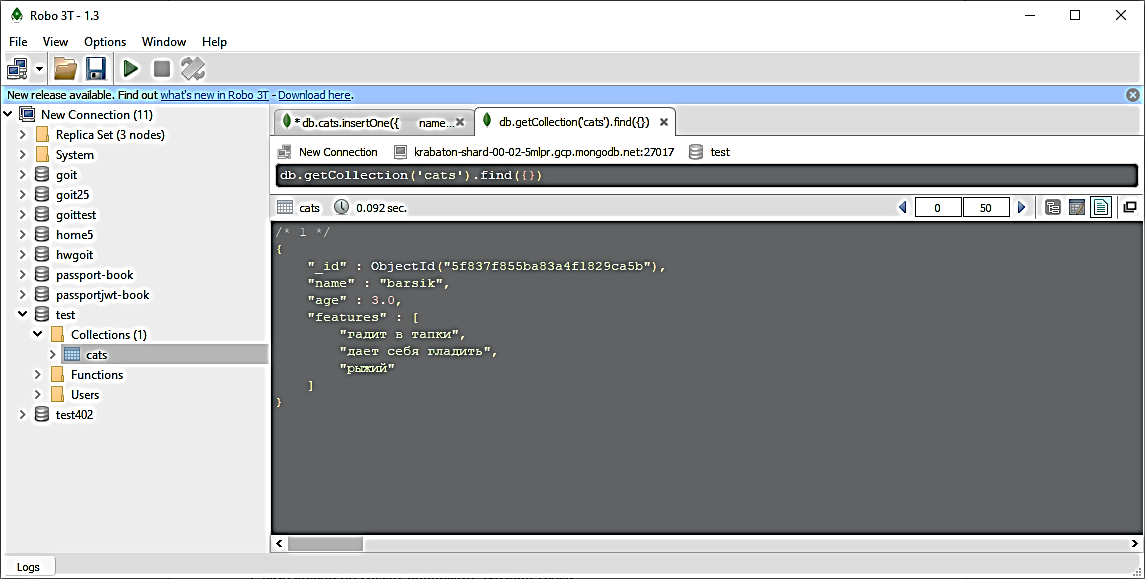
"5f837f855ba83a4f1829ca5b"

)

}

Він означає, що результат був доданий до колекції cats. Подвійним кліком на ім'я cats колекції в лівому меню ми відкриємо нову вкладку з консоллю, де автоматично виконається команда

db.getCollection('cats').find({})



Існують деякі обмеження під час використання імен ключів:

1. Символ $ не може бути першим символом в імені ключа
2. Ім'я ключа не може містити символ крапки .
3. Ім'я \_id не рекомендується використовувати

Використовуємо вставлення даних відразу за двома кішками

db.cats.insertMany([

{

name: 'Lama',

age: 2,

features: ['ходить в лоток', 'не дає себе гладити', 'сірий'],

},

{

name: 'Liza',

age: 4,

features: ['ходить в лоток', 'дає себе гладити', 'білий'],

},

])

Результат відповіді буде схожим на цей — нам повернули два унікальні ідентифікатори новостворених документів

{

"acknowledged": true,

"insertedIds": [

ObjectId(

"5f8382425ba83a4f1829ca5c"

),

ObjectId(

"5f8382425ba83a4f1829ca5d"

)

]

}

І останній варіант — це універсальне вставлення командою insert

db.cats.insert([

{

name: 'Boris',

age: 12,

features: ['ходить в лоток', 'не дає себе гладити', 'сірий'],

},

{

name: 'Murzik',

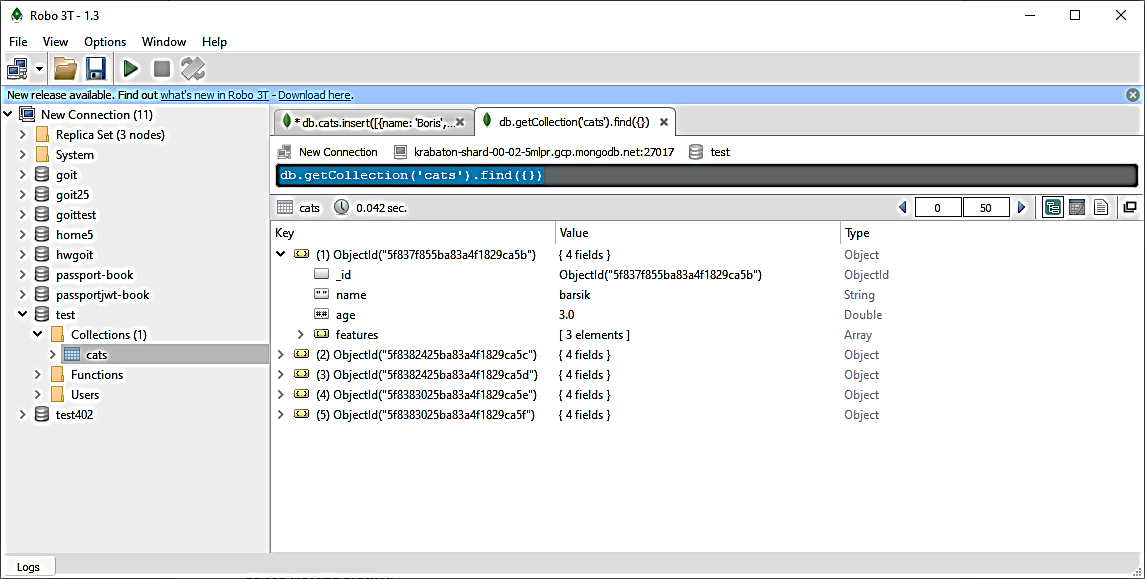
age: 1,

features: ['ходить в лоток', 'дає себе гладити', 'чорний'],

},

])

Після цього вставлення в нашій колекції повинно бути п'ять документів



**Пошук в колекції**

Для виведення документів використовується метод find:

db.cats.find()

Результат у консолі повинен бути на цей момент наступним

{

"\_id": ObjectId("5f837f855ba83a4f1829ca5b"),

"name": "barsik",

"age": 3.0,

"features": [

"ходить у капці",

"дає себе гладити",

"рудий"

]

}

{

"\_id": ObjectId("5f8382425ba83a4f1829ca5c"),

"name": "Lama",

"age": 2.0,

"features": [

"ходить в лоток",

"не дає себе гладити",

"сірий"

]

}

{

"\_id": ObjectId("5f8382425ba83a4f1829ca5d"),

"name": "Liza",

"age": 4.0,

"features": [

"ходить в лоток",

"дає себе гладити",

"білий"

]

}

{

"\_id": ObjectId("5f8383025ba83a4f1829ca5e"),

"name": "Boris",

"age": 12.0,

"features": [

"ходить в лоток",

"не дає себе гладити",

"сірий"

]

}

{

"\_id": ObjectId("5f8383025ba83a4f1829ca5f"),

"name": "Murzik",

"age": 1.0,

"features": [

"ходить в лоток",

"дає себе гладити",

"чорний"

]

}

У MongoDB у запитах можна використовувати умовні конструкції за допомогою операторів порівняння:

* $eq (дорівнює)
* $gt (більше ніж)
* $lt (менше ніж)
* $gte (більше або дорівнює)
* $lte (менше або дорівнює)

db.cats.find({age: {$lte: 3}, features: 'дає себе гладити'})

Результат:

{

"\_id": ObjectId("5f837f855ba83a4f1829ca5b"),

"name": "barsik",

"age": 3.0,

"features": [

"ходить у капці",

"дає себе гладити",

"рудий"

]

}

{

"\_id": ObjectId("5f8383025ba83a4f1829ca5f"),

"name": "Murzik",

"age": 1.0,

"features": [

"ходить в лоток",

"дає себе гладити",

"чорний"

]

}

**Проекція**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/find_06#%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%86%D1%96%D1%8F)Іноді нам потрібна не вся інформація з документа, а, наприклад, певні поля, або, навпаки, деякі поля потрібно виключити

db.cats.find({age: {$lte: 3}, features: 'дає себе гладити'}, {name: 0})

Результат — другим об'єктом у функції find ми виключили виведення поля name

{

"\_id": ObjectId("5f837f855ba83a4f1829ca5b"),

"age": 3.0,

"features": [

"ходить у капці",

"дає себе гладити",

"рудий"

]

}

{

"\_id": ObjectId("5f8383025ba83a4f1829ca5f"),

"age": 1.0,

"features": [

"ходить в лоток",

"дає себе гладити",

"чорний"

]

}

А можемо навпаки - залишити виведення лише двох необхідних полів

db.cats.find(

{age: {$lte: 3}, features: 'дає себе гладити'},

{name: 1, age: 1},

)

Результат

{

"\_id": ObjectId("5f837f855ba83a4f1829ca5b"),

"name": "barsik",

"age": 3.0

}

{

"\_id": ObjectId("5f8383025ba83a4f1829ca5f"),

"name": "Murzik",

"age": 1.0

}

Варто зауважити, що поле \_id виводиться завжди, якщо тільки примусово не заборонити його виведення \_id: 0

**Запит до вкладених об'єктів**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/find_06#%D0%B7%D0%B0%D0%BF%D0%B8%D1%82-%D0%B4%D0%BE-%D0%B2%D0%BA%D0%BB%D0%B0%D0%B4%D0%B5%D0%BD%D0%B8%D1%85-%D0%BE%D0%B1%D1%94%D0%BA%D1%82%D1%96%D0%B2)Давайте вставимо новий документ із полем об'єктом owners

db.cats.insert({

name: 'Dariy',

age: 10,

features: ['ходить в лоток', 'не дає себе гладити', 'сірий'],

owners: {name: 'Nata', age: 23, address: 'Poltava'},

})

Щоб здійснити пошук за вкладеним об'єктом, потрібно використовувати крапку owners.name

db.cats.find({'owners.name': 'Nata'})

Результат:

{

"\_id": ObjectId("5f838b225ba83a4f1829ca60"),

"name": "Dariy",

"age": 10.0,

"features": [

"ходить в лоток",

"не дає себе гладити",

"сірий"

],

"owners": {

"name": "Nata",

"age": 23.0,

"address": "Poltava"

}

}

**Додаткове налаштування запитів:**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/find_06#%D0%B4%D0%BE%D0%B4%D0%B0%D1%82%D0%BA%D0%BE%D0%B2%D0%B5-%D0%BD%D0%B0%D0%BB%D0%B0%D1%88%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B7%D0%B0%D0%BF%D0%B8%D1%82%D1%96%D0%B2)**Обмеження вибірки**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/find_06#%D0%BE%D0%B1%D0%BC%D0%B5%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B2%D0%B8%D0%B1%D1%96%D1%80%D0%BA%D0%B8)Щоб обмежити вибірку, використовується функція limit. Наприклад, показати перші три документи в колекції

db.cats.find().limit(3)

Щоб пропустити кілька документів у вибірці, використовується функція skip. Наприклад, пропустити три документи у вибірці

db.cats.find().skip(3)

**Сортування**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/find_06#%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)Сортування у вибірці виконується функцією sort, яка приймає об'єкт із полями для сортування і вони приймають значення: 1 за зростанням, -1 за спаданням

db.cats.find().sort({name: 1})

**Довжина колекції**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/find_06#%D0%B4%D0%BE%D0%B2%D0%B6%D0%B8%D0%BD%D0%B0-%D0%BA%D0%BE%D0%BB%D0%B5%D0%BA%D1%86%D1%96%D1%97)За допомогою функції count() можна отримати кількість елементів у колекції:

db.cats.count()

Результат: 6

**Модифікатори**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/find_06#%D0%BC%D0%BE%D0%B4%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%82%D0%BE%D1%80%D0%B8)Оператор $exists дозволяє витягти лише ті документи, в яких певний ключ присутній або відсутній.

db.cats.find({owners: {$exists: true}})

Оператор $type витягує лише ті документи, у яких певний ключ має значення певного типу, наприклад, рядок або число

db.cats.find({age: {$type: 'number'}})

Оператор $regex задає регулярний вираз, якому повинно відповідати значення поля.

db.cats.find({name: {$regex: 'L'}})

Результат:

{

"\_id": ObjectId("5f8382425ba83a4f1829ca5c"),

"name": "Lama",

"age": 2.0,

"features": [

"ходить в лоток",

"не дає себе гладити",

"сірий"

]

}

{

"\_id": ObjectId("5f8382425ba83a4f1829ca5d"),

"name": "Liza",

"age": 4.0,

"features": [

"ходить в лоток",

"дає себе гладити",

"білий"

]

}

**Логічні модифікатори**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/find_06#%D0%BB%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D1%96-%D0%BC%D0%BE%D0%B4%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%82%D0%BE%D1%80%D0%B8)Оператор логічного множення $or дозволяє об'єднати вибірки

db.cats.find({$or: [{name: {$regex: 'L'}}, {age: {$lte: 3}}]})

Результат:

{

"\_id": ObjectId("5f837f855ba83a4f1829ca5b"),

"name": "barsik",

"age": 3.0,

"features": [

"ходить у капці",

"дає себе гладити",

"рудий"

]

}

{

"\_id": ObjectId("5f8382425ba83a4f1829ca5c"),

"name": "Lama",

"age": 2.0,

"features": [

"ходить в лоток",

"не дає себе гладити",

"сірий"

]

}

{

"\_id": ObjectId("5f8382425ba83a4f1829ca5d"),

"name": "Liza",

"age": 4.0,

"features": [

"ходить в лоток",

"дає себе гладити",

"білий"

]

}

{

"\_id": ObjectId("5f8383025ba83a4f1829ca5f"),

"name": "Murzik",

"age": 1.0,

"features": [

"ходить в лоток",

"дає себе гладити",

"чорний"

]

}

Оператор логічного множення $and знаходить перетини вибірок

db.cats.find({$and: [{name: {$regex: 'L'}}, {age: {$lte: 3}}]})

Результат:

{

"\_id": ObjectId("5f8382425ba83a4f1829ca5c"),

"name": "Lama",

"age": 2.0,

"features": [

"ходить в лоток",

"не дає себе гладити",

"сірий"

]

}

**Курсори**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/find_06#%D0%BA%D1%83%D1%80%D1%81%D0%BE%D1%80%D0%B8)Результат вибірки, що отримується за допомогою функції find, називається курсором. Курсори інкапсулюють у собі набори об'єктів, що отримуються з БД. Використовуючи синтаксис мови JavaScript та методи курсорів, ми можемо вивести отримані документи на екран та якось їх обробити.

const cursor = db.cats.find()

while (cursor.hasNext()) {

obj = cursor.next()

print(obj['name'])

}

Результат:

barsik

Lama

Liza

Boris

Murzik

Dariy

**Зміна документів**

**Збереження документів**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/change_07#%D0%B7%D0%B1%D0%B5%D1%80%D0%B5%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2)Збереження документів можна виконати методом save. У новий документ як поле можна передати параметр \_id. Якщо метод знаходить документ з таким значенням \_id, то документ оновлюється. Якщо з таким \_id немає документів, то документ вставляється.

db.cats.save({name: 'Bars', age: 3})

**Оновлення документів**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/change_07#%D0%BE%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2)Детальніше налаштування під час оновлення пропонує функція update. Вона приймає три параметри:

1. query: приймає запит на вибірку документа, який потрібно оновити
2. objNew: надає документ з новою інформацією, який замістить старий під час оновлення
3. options: визначає додаткові параметри під час оновлення документів. Може приймати два аргументи: upsert та multi.

Якщо параметр upsert має значення true, то MongoDB буде оновлювати документ, якщо він знайдений, і створювати новий, якщо такого документа немає. Якщо він має значення false, то MongoDB не буде створювати новий документ, якщо запит на вибірку не знайде жодного документа.

Параметр multi вказує, чи повинен оновлюватися перший елемент у вибірці (використовується за замовчуванням, якщо цей параметр не вказано) або повинні оновлюватися всі документи у вибірці.

db.cats.update({name: 'Bars'}, {name: 'Tom', age: 5}, {upsert: true})

Використання оператора $set приводить до того, що якщо документ не містить оновлюване поле, то воно створюється. Інакше буде зроблено заміну документа.

db.cats.update(

{name: 'Tom'},

{$set: {features: ['ходить в лоток', 'не дає себе гладити', 'сірий']}},

)

Вказавши значення multi:true, ми можемо оновити всі документи вибірки {multi:true}

Для видалення окремого ключа використовується оператор $unset:

db.cats.update({name: 'Tom'}, {$unset: {age: 1}})

Метод updateOne схожий на метод update за тим винятком, що він оновлює лише один документ. Якщо необхідно оновити всі документи, що відповідають деякому критерію, то застосовується метод updateMany

**Видалення документа**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/change_07#%D0%B2%D0%B8%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0)Для видалення документів у MongoDB передбачений метод remove:

Видалення всіх документів із зазначеним запитом

db.cats.remove({name: 'Tom'})

Якщо потрібно видалити тільки один документ

db.cats.remove({name: 'Tom'}, true)

**Робота з масивами**

Оператор $in визначає масив можливих виразів і шукає ті ключі, значення яких є в масиві:

db.cats.find({age: {$in: [2, 10]}})

Результат:

{

"\_id": ObjectId("5f8382425ba83a4f1829ca5c"),

"name": "Lama",

"age": 2.0,

"features": [

"ходить в лоток",

"не дає себе гладити",

"сірий"

]

}

{

"\_id": ObjectId("5f838b225ba83a4f1829ca60"),

"name": "Dariy",

"age": 10.0,

"features": [

"ходить в лоток",

"не дає себе гладити",

"сірий"

],

"owners": {

"name": "Nata",

"age": 23.0,

"address": "Poltava"

}

}

Протилежним чином діє оператор $nin — він визначає масив можливих виразів і шукає ті ключі, значення яких відсутнє в цьому масиві

db.cats.find({age: {$nin: [2, 10]}})

Оператор $all схожий на $in: він також визначає масив можливих виразів, але вимагає, щоб документи мали увесь визначений набір виразів.

db.cats.find({features: {$all: ['ходить в лоток', 'дає себе гладити']}})

Результат:

{

"\_id": ObjectId("5f8382425ba83a4f1829ca5d"),

"name": "Liza",

"age": 4.0,

"features": [

"ходить в лоток",

"дає себе гладити",

"білий"

]

}

{

"\_id": ObjectId("5f8383025ba83a4f1829ca5f"),

"name": "Murzik",

"age": 1.0,

"features": [

"ходить в лоток",

"дає себе гладити",

"чорний"

]

}

Оператор $size використовується для знаходження документів, в яких масиви мають кількість елементів, що дорівнює значенню $size.

db.cats.find({features: {$size: 3}})

Оператор $push додає значення в масив

db.cats.updateOne({name: 'Tom'}, {$push: {features: 'смердючий'}})

Якщо потрібно додати відразу кілька значень

db.cats.updateOne(

{name: 'Tom'},

{$push: {features: {$each: ['хропить', 'злий']}}},

)

Оператор $addToSet подібно до оператора $push додає об'єкти в масив. Відмінність полягає в тому, що $addToSet додає дані, якщо їх ще немає в масиві:

db.cats.update({name: 'Lama'}, {$addToSet: {features: 'божевільний'}})

Оператор $pop дозволяє видаляти елемент з масиву:

db.cats.update({name: 'Tom'}, {$pop: {features: 1}})

1 — кінець масиву, -1 — початок масиву

Оператор $pull видаляє за значенням

db.cats.update({name: 'Tom'}, {$pull: {features: 'сірий'}})

А якщо ми хочемо видалити не одне значення, а відразу декілька, тоді ми можемо застосувати оператор $pullAll:

db.cats.update(

{name: 'Tom'},

{$pullAll: {features: ['не дає себе гладити', 'смердючий', 'хропить']}},

)

**Робота та підключення до Python**

Встановимо драйвер Python під назвою "PyMongo". Існує багато драйверів, написаних спільнотою, але PyMongo є офіційним драйвером Python для MongoDB. Детальну документацію драйвера можна знайти [тут](https://docs.mongodb.com/drivers/python/) (https://www.mongodb.com/docs/drivers/python-drivers/).

Найпростіший спосіб встановити драйвер — через систему управління пакетами pip. Виконайте в командному рядку наступне:

python -m pip install pymongo[srv,tls]

Або за допомогою Poetry

poetry add pymongo[srv,tls]

**Створення документів**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/pymongo_09#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2)Для створення документів MongoDB використовуються методи: insert\_one - для вставлення одного документа і insert\_many - для вставлення відразу кілька документів в колекцію.

from pymongo import MongoClient

from pymongo.server\_api import ServerApi

client = MongoClient(

"mongodb+srv://<username>:<password>@krabaton.5mlpr.gcp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority",

server\_api=ServerApi('1')

)

db = client.book

result\_one = db.cats.insert\_one(

{

"name": "barsik",

"age": 3,

"features": ["ходить в капці", "дає себе гладити", "рудий"],

}

)

print(result\_one.inserted\_id)

result\_many = db.cats.insert\_many(

[

{

"name": "Lama",

"age": 2,

"features": ["ходить в лоток", "не дає себе гладити", "сірий"],

},

{

"name": "Liza",

"age": 4,

"features": ["ходить в лоток", "дає себе гладити", "білий"],

},

]

)

print(result\_many.inserted\_ids)

Замість рядка mongodb+srv://<username>:<password> @krabaton.5mlpr.gcp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority, ви повинні підставити свій рядок підключення. За допомогою client.book ми створюємо базу даних book. Результатом виконання db.cats.insert\_one та db.cats.insert\_many буде \_id вставлених документів в колекцію cats.

60d24b783733b1ae668d4a77

[ObjectId('60d24b783733b1ae668d4a78'), ObjectId('60d24b783733b1ae668d4a79')]

**Отримання документів**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/pymongo_09#%D0%BE%D1%82%D1%80%D0%B8%D0%BC%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2)

Щоб отримати один документ, використовується метод find\_one

from bson.objectid import ObjectId

from pymongo import MongoClient

from pymongo.server\_api import ServerApi

client = MongoClient(

"mongodb+srv://<username>:<password>@krabaton.5mlpr.gcp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority",

server\_api=ServerApi('1')

)

db = client.book

result = db.cats.find\_one({"\_id": ObjectId("60d24b783733b1ae668d4a77")})

print(result)

Результат — словник виду:

{

"\_id": ObjectId("60d24b783733b1ae668d4a77"),

"name": "barsik",

"age": 3,

"features": ["ходить в капці", "дає себе гладити", "рудий"],

}

Отримати декілька документів — метод find:

from pymongo import MongoClient

from pymongo.server\_api import ServerApi

client = MongoClient(

"mongodb+srv://<username>:<password>@krabaton.5mlpr.gcp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority",

server\_api=ServerApi('1')

)

db = client.book

result = db.cats.find({})

for el in result:

print(el)

Результат:

{

"\_id": ObjectId("60d24b783733b1ae668d4a77"),

"name": "barsik",

"age": 3,

"features": ["ходить в капці", "дає себе гладити", "рудий"],

}

{

"\_id": ObjectId("60d24b783733b1ae668d4a78"),

"name": "Lama",

"age": 2,

"features": ["ходить в лоток", "не дає себе гладити", "сірий"],

}

{

"\_id": ObjectId("60d24b783733b1ae668d4a79"),

"name": "Liza",

"age": 4,

"features": ["ходить в лоток", "дає себе гладити", "білий"],

}

**Оновлення документів**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/pymongo_09#%D0%BE%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2)

Для оновлення документа можна використовувати метод update\_one

from bson.objectid import ObjectId

from pymongo import MongoClient

from pymongo.server\_api import ServerApi

client = MongoClient(

"mongodb+srv://<username>:<password>@krabaton.5mlpr.gcp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority",

server\_api=ServerApi('1')

)

db = client.book

db.cats.update\_one({"name": "barsik"}, {"$set": {"age": 4}})

result = db.cats.find\_one({"name": "barsik"})

print(result)

Результат:

{

"\_id": ObjectId("60d24b783733b1ae668d4a77"),

"name": "barsik",

"age": 4,

"features": ["ходить в капці", "дає себе гладити", "рудий"],

}

**Видалення документів**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/pymongo_09#%D0%B2%D0%B8%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%96%D0%B2)Для видалення документа з колекції використовується метод delete\_one

from bson.objectid import ObjectId

from pymongo import MongoClient

from pymongo.server\_api import ServerApi

client = MongoClient(

"mongodb+srv://<username>:<password>@krabaton.5mlpr.gcp.mongodb.net/myFirstDatabase?retryWrites=true&w=majority",

server\_api=ServerApi('1')

)

db = client.book

db.cats.delete\_one({"name": "barsik"})

result = db.cats.find\_one({"name": "barsik"})

print(result)

Результатом буде відсутність документа в колекції:

None

Ми виділили та використали кілька методів PyMongo для взаємодії з нашим хмарним сервером MongoDB зі скрипту Python.

Усі доступні методи можна знайти в офіційній документації [PyMongo](https://pymongo.readthedocs.io/en/stable/) (https://pymongo.readthedocs.io/en/stable/)

**Практичний приклад роботи з MongoDB**

У цьому практичному прикладі ми створимо базу даних (БД) **MongoDb** з використанням ODM пакета [**MongoEngine**](https://mongoengine-odm.readthedocs.io/tutorial.html) (https://mongoengine-odm.readthedocs.io/tutorial.html)**.**

Наша мета — отримати невелику БД для проекту "TODO", в якому будуть завдання — колекція notes, у завдань будуть підпорядковані пункти з описом і контролем виконання — внутрішній документ records. Також буде можливість додати до завдань список тегів — внутрішній документ tags.

**Підключення до БД**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/mongoengine_10#%D0%BF%D1%96%D0%B4%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%BE-%D0%B1%D0%B4)

Встановлюємо пакети**PyMongo** та **MongoEngine** для роботи з БД MongoDB

pip install pymongo mongoengine

Як ми вже сказали, як провідник даних ми будемо використовувати бібліотеку [**MongoEngine**](https://mongoengine-odm.readthedocs.io/tutorial.html) (https://mongoengine-odm.readthedocs.io/tutorial.html), яка надає **ODM** підхід до опису та управління об'єктами у БД **MongoDB**. По суті, MongoEngine є еквівалентом SQLAlchemy, але вже для MongoDB.

Нам знадобиться рядок підключення, ім'я користувача та пароль. Додайте рядок підключення до коду застосунку.

mongodb+srv://goitlearn:<password>@cluster0.smeju.mongodb.net/myFirstDatabase?retryWrites=true&w=majority

У цьому рядку потрібно замінити плейсхолдер <password> паролем для користувача goitlearn. Також можна замінити ім'я myFirstDatabase на своє ім'я бази даних, яку будемо використовувати в нашій програмі.

У цьому завданні ми використовуємо найпростіший спосіб приховування конфіденційних даних — це використання вбудованого модуля **configparser** та створення файлу *config.ini*

INFO

Основи роботи з модулем configparser можна знайти в офіційній документації

Структура файлу конфігурації, файл config.ini:

[DB]

USER=<підставте ім'я користувача>

PASS=<підставте сюди пароль користувача>

DB\_NAME=<підставте ім'я БД>

DOMAIN=<підставте ім'я домену>

Напишемо скрипт підключення до БД, файл connect.py:

**connect.py**

from mongoengine import connect

import configparser

config = configparser.ConfigParser()

config.read('config.ini')

mongo\_user = config.get('DB', 'user')

mongodb\_pass = config.get('DB', 'pass')

db\_name = config.get('DB', 'db\_name')

domain = config.get('DB', 'domain')

# connect to cluster on AtlasDB with connection string

connect(host=f"""mongodb+srv://{mongo\_user}:{mongodb\_pass}@{domain}/{db\_name}?retryWrites=true&w=majority""", ssl=True)

Використання цього модуля підключатиме нашу програму до хмарної БД MongoDB.

**Створення моделей**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/mongoengine_10#%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B5%D0%B9)Тепер можна створити наші моделі даних. Абстракція MongoEngine заснована на класах, і всі створені моделі є класами.

Оголошення деяких властивостей у класі еквівалентно створенню структури даних для збереження даних. Ці класи прийнято зберігати у сценарії як модуль моделі застосунку, ми зробимо це просто у файлі**models.py.**

**models.py**

from datetime import datetime

from mongoengine import EmbeddedDocument, Document

from mongoengine.fields import BooleanField, DateTimeField, EmbeddedDocumentField, ListField, StringField

class Tag(EmbeddedDocument):

name = StringField()

class Record(EmbeddedDocument):

description = StringField()

done = BooleanField(default=False)

class Notes(Document):

name = StringField()

created = DateTimeField(default=datetime.now())

records = ListField(EmbeddedDocumentField(Record))

tags = ListField(EmbeddedDocumentField(Tag))

meta = {'allow\_inheritance': True}

Тут у нас описані три схеми для документів Tag, Record та Notes. Причому схема документів Notes у нашому випадку включає документи Tag і Record. Ми не будемо повністю описувати кожен рядок коду. Для детальної інформації завжди можна звернутись до [документації](https://docs.mongoengine.org/guide/defining-documents.html) (https://docs.mongoengine.org/guide/defining-documents.html) модуля MongoEngine.

**Додаємо дані**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/mongoengine_10#%D0%B4%D0%BE%D0%B4%D0%B0%D1%94%D0%BC%D0%BE-%D0%B4%D0%B0%D0%BD%D1%96)Далі напишемо скрипт seeds.py, щоб наповнити даними нашу БД. Підключаємо всі моделі та імпортуємо модуль connect.py, для підключення до нашої хмарної БД MongoDB. Інші коментарі є в коді файлу seeds.py

**seeds.py**

from models import Notes, Record, Tag

import connect

# спочатку - створити об'єкт Tag

tag = Tag(name='Purchases')

# потім - створення об'єктів Record

record1 = Record(description='Buying sausage')

record2 = Record(description='Buying milk')

record3 = Record(description='Buying oil')

# Останнє - створюємо об'єкт Note і зберігаємо його

Notes(name='Shopping', records=[record1, record2, record3], tags=[tag, ]).save()

Notes(name='Going to the movies', records=[Record(description='Went to see the Avengers'), ], tags=[Tag(name='Fun'), ]).save()

Тут достатньо просто — ми створюємо тег 'Purchases', і потім окремо створюємо три записи Record. А далі ми створюємо екземпляр Notes і викликаємо на ньому метод save, щоб дані збереглися у БД.

tag = Tag(name='Purchases')

record1 = Record(description='Buying sausage')

record2 = Record(description='Buying milk')

record3 = Record(description='Buying oil')

Notes(name='Shopping', records=[record1, record2, record3], tags=[tag, ]).save()

Для наступного документа ми створюємо записи та теги безпосередньо під час виклику екземпляра класу Notes і одразу зберігаємо його методом save у базі даних.

Після виконання скрипту seeds.py, ми повинні отримати наступні схеми документів у колекції notes:

{

"\_id": {

"$oid": "6316635072902f5680fdc347"

},

"\_cls": "Notes",

"name": "Shopping",

"created": {

"$date": {

"$numberLong": "1662411598499"

}

},

"records": [

{

"description": "Buying sausage",

"done": false

},

{

"description": "Buying milk",

"done": false

},

{

"description": "Buying oil",

"done": false

}

],

"tags": [

{

"name": "Purchases"

}

]

}

{

"\_id": {

"$oid": "6316635072902f5680fdc348"

},

"\_cls": "Notes",

"name": "Going to the movies",

"created": {

"$date": {

"$numberLong": "1662411598499"

}

},

"records": [

{

"description": "Went to see the Avengers",

"done": false

}

],

"tags": [

{

"name": "Fun"

}

]

}

NOTE

У вашому випадку значення \_id очевидно відрізнятимуться

**Отримання даних**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/mongoengine_10#%D0%BE%D1%82%D1%80%D0%B8%D0%BC%D0%B0%D0%BD%D0%BD%D1%8F-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)Залишилось написати скрипт отримання даних. Це детально розписано в [цьому](https://docs.mongoengine.org/guide/querying.html) (https://docs.mongoengine.org/guide/querying.html) розділі документації.

Наприклад, щоб отримати всі записи з колекції notes:

notes = Notes.objects()

for note in notes:

print("-------------------")

records = [f'description: {record.description}, done: {record.done}' for record in note.records]

tags = [tag.name for tag in note.tags]

print(f"id: {note.id} name: {note.name} date: {note.created} records: {records} tags: {tags}")

Виведення:

id: 6316635072902f5680fdc347 name: Shopping date: 2022-09-05 20:59:58.499000 records: ['description: Buying sausage, done: False', 'description: Buying milk, done: False', 'description: Buying oil, done: False'] tags: ['Purchases']

id: 6316635072902f5680fdc348 name: Going to the movies date: 2022-09-05 20:59:58.499000 records: ['description: Went to see the Avengers, done: False'] tags: ['Fun']

INFO

Є спосіб відразу отримати результат у вигляді словника, використовуючи комбінацію методів to\_mongo та to\_dict:

notes = Notes.objects()

print("-------------------")

for note in notes:

print(note.to\_mongo().to\_dict())

Виведення:

-------------------

{'\_id': ObjectId('6316635072902f5680fdc347'), '\_cls': 'Notes', 'name': 'Shopping', 'created': datetime.datetime(2022, 9, 5, 20, 59, 58, 499000), 'records': [{'description': 'Buying sausage', 'done': False}, {'description': 'Buying milk', 'done': False}, {'description': 'Buying oil', 'done': False}], 'tags': [{'name': 'Purchases'}]}

{'\_id': ObjectId('6316635072902f5680fdc348'), '\_cls': 'Notes', 'name': 'Going to the movies', 'created': datetime.datetime(2022, 9, 5, 20, 59, 58, 499000), 'records': [{'description': 'Went to see the Avengers', 'done': False}], 'tags': [{'name': 'Fun'}]}

Якщо ми хочемо отримати записи тільки з тегом 'Fun', нам потрібно вказати критерій відбору tags\_\_name. Такий запис критерію говорить, що пошук потрібно провести в tags.name, як у викладеному документі, але крапку замінюють подвійним підкресленням.

notes = Notes.objects(tags\_\_name='Fun')

for note in notes:

records = [f'description: {record.description}, done: {record.done}' for record in note.records]

tags = [tag.name for tag in note.tags]

print(f"id: {note.id} name: {note.name} date: {note.created} records: {records} tags: {tags}")

Виведення:

id: 6316635072902f5680fdc348 name: Going to the movies date: 2022-09-05 20:59:58.499000 records: ['description: Went to see the Avengers, done: False'] tags: ['Fun']

Повний код файлу main.py для виведення даних:

**main.py**

from models import Notes

import connect

print('--- All notes ---')

notes = Notes.objects()

for note in notes:

records = [f'description: {record.description}, done: {record.done}' for record in note.records]

tags = [tag.name for tag in note.tags]

print(f"id: {note.id} name: {note.name} date: {note.created} records: {records} tags: {tags}")

print('--- Notes with tag Fun ---')

notes = Notes.objects(tags\_\_name='Fun')

for note in notes:

records = [f'description: {record.description}, done: {record.done}' for record in note.records]

tags = [tag.name for tag in note.tags]

print(f"id: {note.id} name: {note.name} date: {note.created} records: {records} tags: {tags}")

INFO

Якби ви хотіли отримати записи відразу з кількома тегами одночасно, то нотація, відповідно до документації: на поля вбудованих документів можна посилатися за допомогою синтаксису пошуку полів, використовуючи подвійне підкреслення, замість крапки в синтаксисі доступу до атрибутів об'єкта, була б такою:

Notes.objects(tags\_\_name\_\_in=['Fun', 'Purchases'])

**Оновлення та видалення даних**

[​](https://textbook.edu.goit.global/python-web-textbook/uk/docs/module-08/module-08-01/mongoengine_10#%D0%BE%D0%BD%D0%BE%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D1%82%D0%B0-%D0%B2%D0%B8%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)

Що стосується видалення та оновлення даних, то тут все досить просто і використовується стандартне найменування методів.

Оновлення:

\_id = '6318789480a92d0f3a49c7cb'

note = Notes.objects(id=\_id)

note.update(name='New name')

Видалення:

\_id = '6318789480a92d0f3a49c7cb'

note = Notes.objects(id=\_id).delete()

Ми розглянули підключення та роботу з хмарною БД MongoDB за допомогою ODM MongoEngine.