Лабораторна робота №7

Основи роботи з MongoDB

Мета роботи

Отримати базові навички роботи з MongoDB.

Загальні відомості

MongoDB — це нереляційне (NoSQL) сховище документів у форматі BSON (binary JSON — бінарний JSON), що має гнучку модель даних та забезпечує повну підтримку індексування, реплікації, сегментування та інших можливостей. Основною перевагою MongoDB є те, що вона зберігає дані у вигляді об'єктів, а не реляційних таблиць, а тому краще пристосована до моделювання більшості предметних областей, ніж класичні реляційні бази даних. В своїй основі MongoDB є розподіленою, тому висока доступність (high availability), горизонтальне масштабування (horizontal scaling) та географічна розподіленість (geographic distribution) присутні у ній «з коробки» та є відносно простими у використанні.

MongoDB доступна як хмарне рішення (MongoDB Atlas, а також версії у популярних хмарних провайдерів), корпоративне self-managed рішення (MongoDB Enterprise) та безкоштовна self-managed версія з відкритим кодом (MongoDB Community). В даній роботі ми використовуватимемо останню.

Основною сутністю запису у MongoDB є *документ*, що є структурою даних, утвореною сукупністю полів типу «ключ-значення». MongoDB зберігає документи у форматі BSON, що фактично є бінарним аналогом формату JSON. Окрім примітивних типів, значеннями полів також можуть бути інші об'єкти, масиви або масиви об'єктів, а також деякі спеціальні типи:

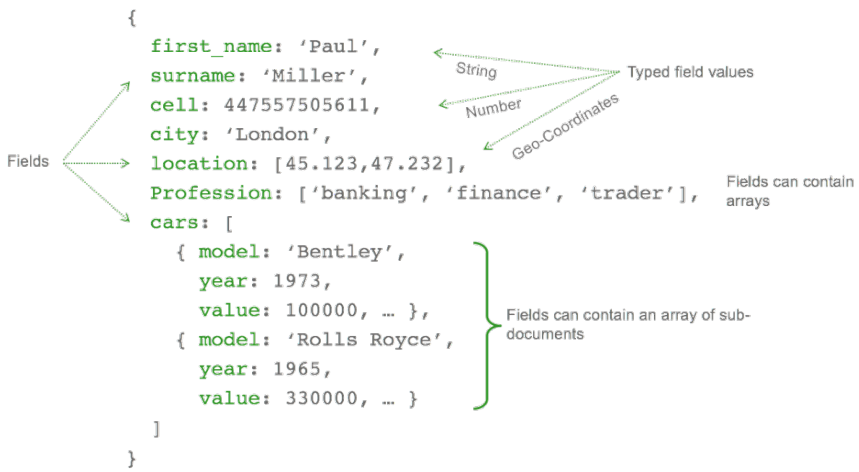

Рисунок 1 — Ілюстрація документу у MongoDB

Початок роботи

1. Завантажте та встановіть (або розпакуйте, якщо ви завантажили архів) community-версію MongoDB. Знайти її можна за цим посиланням: https://www.mongodb.com/try/download/community.
2. Завантажте та встановіть (або розпакуйте, якщо ви завантажили архів) MongoDB Shell — консольний клієнт для MongoDB — за цим посиланням: https://www.mongodb.com/try/download/shell.
3. Завантажте та встановіть повну версію MongoDB Compass — клієнту для MongoDB з графічним інтерфейсом користувача — за цим посиланням: https://www.mongodb.com/try/download/atlascli.
4. Запустіть сервер MongoDB (файл `mongod.exe`) з кроку 1. За замовчуванням MongoDB зберігає файли сховища у директорію `C:\data\db`. Створіть її, якщо вона відсутня. Змінити місце збереження даних можна запустивши сервер з ключем `--dbpath` та вказавши цільовий шлях.
5. Запустіть клієнти MongoDB Shell (файл `mongosh.exe` з кроку 2) та MongoDB Compass (`MongoDBCompass.exe` з кроку 3), та приєднайтеся з них до серверу MongoDB. За замовчуванням сервер MongoDB запускається на порту `27017` (налаштувати порт можна ключем `--port`), тому рядок підключення (connection string) для запущеного локально екземпляру серверу виглядатиме наступним чином: `mongodb://localhost:27017`. Обидва клієнти мають успішно з'єднатися з сервером MongoDB:

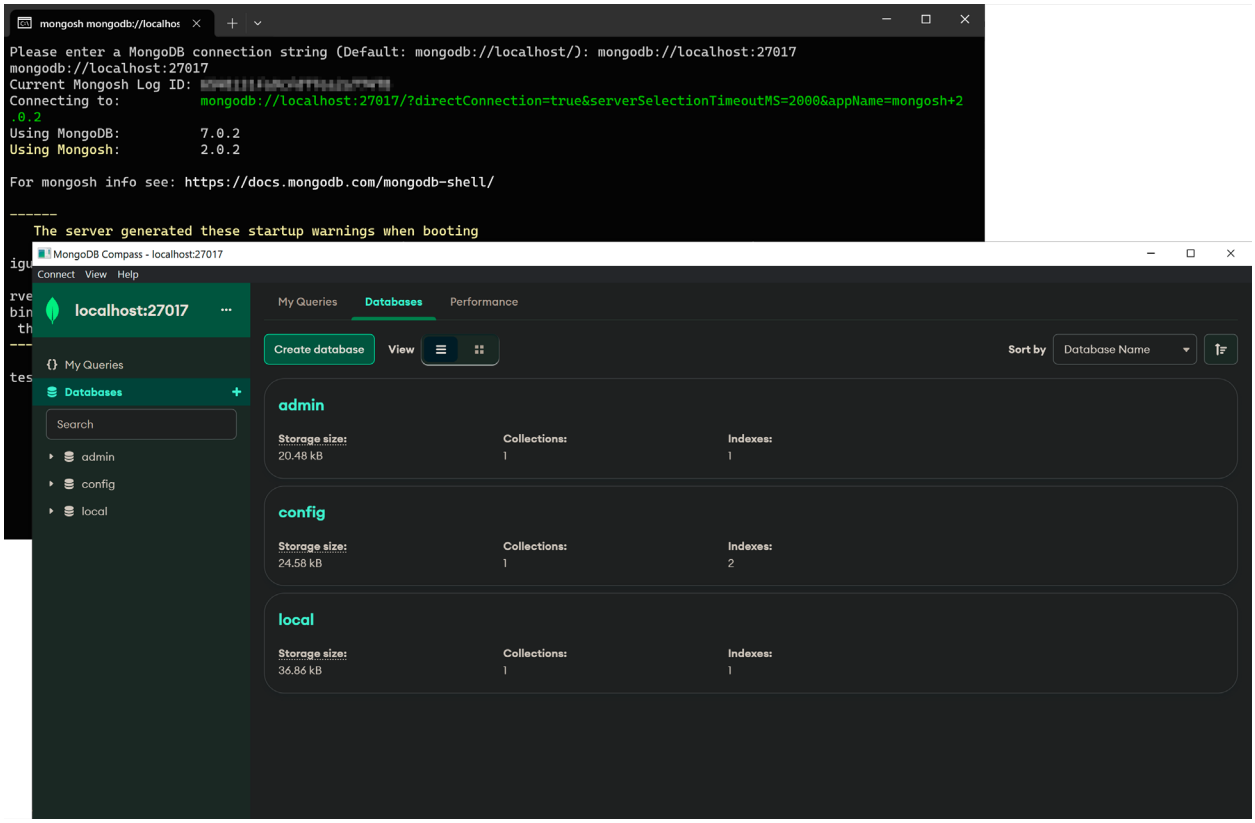

Рисунок 2 — MongoDB Shell та MongoDB Compass успішно з'єдналися з сервером MongoDB

Налаштування реплікації

Хоча виконаних у попередньому розділі дій достатньо для того, щоб почати розробку додатків з використанням MongoDB у якості сховища даних, у production-середовищі застосування одного екземпляру серверу MongoDB вважається поганою практикою. Для того, щоб забезпечити резервування (redundancy) та високу доступність (high availability) виробничих розгортань (production deployments), використовують відразу декілька запущених екземплярів серверу MongoDB одночасно, які називаються репліка-сетами (з англ. replica set — набір копій або реплік).

Репліка-сет може містити кілька вузлів (node), що безпосередньо працюють з даними і, за бажанням, один арбітражний вузол (arbiter). З вузлів, що обробляють дані, один і тільки один вважається основним (primary) вузлом, тоді як інші вважаються вторинними (secondary) вузлами.

Основний вузол отримує усі операції на запис, які він також заносить в журнал операцій (operation log, або скорочено oplog). Цей журнал асинхронно копіюється вторинними вузлами, які застосовують виконані операції на основному вузлі до своїх даних таким чином, щоб дані, збережені на вторинних вузлах, були ідентичними даним основного. Завдяки цьому репліка-сет може продовжувати функціонувати, незважаючи на відмову одного або кількох членів: наприклад, якщо зв'язку з основним вузлом немає довше, ніж визначається параметром `electionTimeoutMillis` (за замовчуванням 10 секунд), один із працюючих вторинних вузлів призначається у якості нового основного.

За замовчуванням операції читання також відбуваються з основного вузла, хоча клієнти можуть налаштувати читання і з вторинних вузлів. Тут варто пам'ятати, що синхронізація даних між вузлами займає певний час, а тому існує шанс, що клієнт прочитає з вторинного вузла застарілі дані.

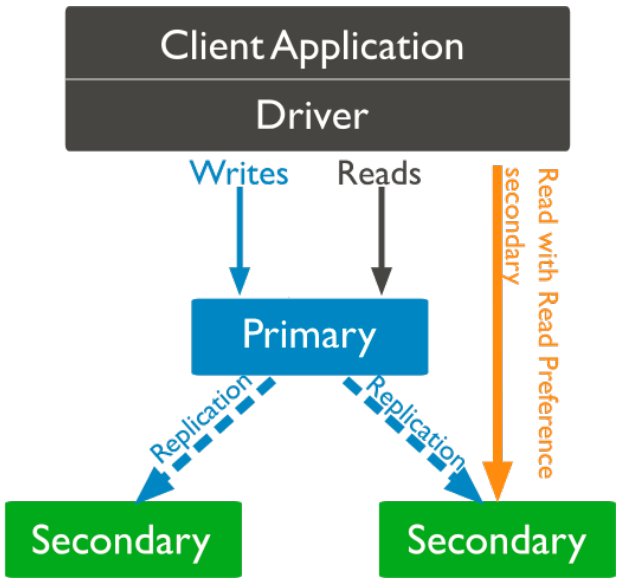

Рисунок 3 — Репліка-сет з трьома вузлами

Більш детальну інформацію про реплікацію у MongoDB можна прочитати тут: https://www.mongodb.com/docs/manual/replication/.

Налаштуємо простий репліка-сет з трьома вузлами. (Примітка. Для production-систем рекомендується «піднімати» вузли репліка-сету на різних серверах, щоб по-справжньому забезпечити високу доступність, у разі якщо якісь із них вийдуть з ладу. У рамках лабораторної роботи ми запустимо усі екземпляри MongoDB на одній машині, щоб продемонструвати принцип. Детальні відомості щодо налаштування та розгортання MongoDB у production-середовищі знаходяться тут: https://www.mongodb.com/docs/manual/administration/production-checklist-operations/.)

1. Запустимо три екземпляри MongoDB. Для цього створимо три директорії для збереження даних та оберемо назву репліка-сету. Наприклад, дамо репліка-сету назву `rs0`, створимо директорії `data0`, `data1`, `data2` на диску `C`, та запустимо інстанси на трьох послідовних портах починаючи з `27017`. У різних вікнах командної строки послідовно виконаємо наступні команди:
   ```
   .\mongod.exe --port 27017 --dbpath C:/data0/db --replSet rs0
   .\mongod.exe --port 27018 --dbpath C:/data1/db --replSet rs0
   .\mongod.exe --port 27019 --dbpath C:/data2/db --replSet rs0
   ```
   Тут параметр `--replSet` дозволяє задати назву репліка-сет кластера.
2. Запустимо MongoDB Shell, приєднаємося до інстансу, який бажаємо зробити основним (наприклад до того, що запущений на порту `27017`), та ініціюємо репліка-сет командою `rs.initiate()`.
3. Після цього додамо інші інстанси у даний репліка-сет:
   ```
   rs.add("localhost:27018")
   rs.add("localhost:27019")
   ```
4. Перевірити працездатність репліка-сету та переглянути детальну інформацію про нього можна командою `rs.status()`. Переглянути конфігурацію репліка-сету можна за допомогою команди `rs.conf()`. Переглянути вже наявні у кластері бази даних можна виконавши команду `show dbs`, або подивившись список, що відображається у MongoDB Compass.

CRUD-операції

Після того, як репліка-сет успішно налаштовано, можна почати заповнювати MongoDB даними. Структура зберігання даних у MongoDB є наступною: на самому сервері сховища зберігаються бази даних (з самого початку там вже є `admin`, `local` та `config`) в яких, в свою чергу, зберігаються колекції в яких зберігаються документи.

Розглянемо як можна створити базу даних та колекцію, та записати в неї документ. (Примітка. Команди, що розглядатимуться далі застосовуються саме для MongoDB Shell. Варто зауважити, що для MongoDB існує велика кількість

драйверів для різних мов програмування і точний синтаксис цих команд для кожної мови може відрізнятися. Крім того, ці операції можна виконати через графічний інтерфейс у MongoDB Compass, проте наразі буде корисно ознайомитися з тим, як вони виконуються саме у MongoDB Shell, так як основний принцип зберігається усюди.)

1. Створимо базу даних. Як такої команди для створення бази даних у MongoDB Shell немає, проте є команда переключення на базу даних: `use mynewdatabase`. Якщо після цього переглянути наявні бази даних командою `show dbs`, то новоствореної бази у списку поки що не буде. Це відбувається тому, що остаточно її буде створено лише з записом першого документу в її колекцію. Тим не менше, перевірити, що активною базою даних в даний момент є саме `mynewdatabase` можна виконавши команду `db`.

2. Створення колекції також відбувається з першим записом даних у неї. Звернутися до колекції можна наступним чином. Так як `db` є не лише командою, а й фактично змінною (на кшталт того як у JavaScript до прив'язки (binding), що містить функцію, можна звернутися двома способами: власне викликати функцію або повернути її як значення; взагалі система команд MongoDB фактично побудована навколо інтерпретатору JavaScript), логічно припустити, що `db` посилається на активну базу даних (в даному випадку, `mynewdatabase`). Так як колекція є сутністю, що належить базі даних, до неї можна звернутися як `db.collection`, і власне на колекції тепер можна викликати доступні для неї функції — наприклад, функцію вставки документу (пам'ятаємо, що він JSON-подібний). Таким чином, утворимо у базі даних `mynewdatabase` колекцію з назвою `user`, та вставимо в неї простий документ, викликавши команду `insert`: `db.user.insert({name: "John", age: 25})`. Команда `insert` також дозволяє вставляти декілька об'єктів одночасно; для цього їх треба заключити у квадратні дужки, що є синтаксичним позначенням масиву: `db.user.insert([{…}, {…}])`. У якості альтернативи можна також скористатися командою `insertMany`, яка працює лише з масивом об'єктів. Також варто зауважити, що MongoDB автоматично створює унікальний дванадцятибайтовий ідентифікатор (`ObjectId`) для кожного документу, який складається з 4-байтової позначки часу, що вимірюється в секундах з епохи Unix, 5-байтового випадкового значення та 3-байтового інкрементного лічильника, що ініціалізується випадковим значенням.

3. Запити на читання даних виконуються за допомогою команди `find`, яка може приймати до трьох параметрів: фільтр, проекцію та опції пошуку. Якщо застосувати її без жодного з них, команда поверне всю колекцію. Для того, щоб отримати лише частину записів, застосовується об'єкт фільтрації у якості першого аргументу. Наприклад, якщо необхідно вибрати всі документи, в яких ім'я дорівнювало б значенню `"John"`, це можна зробити так: `db.user.find({name: "John"})`. Мовою SQL це можна було б записати наступним

чином: `SELECT * FROM user WHERE name = "John"`. Також фільтри підтримують низку операторів, які дозволяють робити більш складні запити. Наприклад, якщо необхідно вибрати всі документи з іменами `"John"` та `"Bill"`, це можна зробити за допомогою параметру `$in`: `db.user.find({name: { $in: [ "John", "Bill" ] }})`. Як можна побачити, синтаксис тут дещо складніший тому що ми маємо справу з вкладеними один в одного документами: спочатку ми вказуємо, що будемо фільтрувати за полем `name`, але потім, замість звичайного значення вказуємо об'єкт з оператором `$in`, який говорить про те, що повернути треба лише ті документи, в яких `name` приймає одне із значень, наведених далі у масиві. Фактично, це аналог наступної команди SQL: `SELECT * FROM user WHERE name IN ("John", "Bill")`. Фільтрацію можна проводити за декількома полями одночасно; для цього вони відокремлюються комами. Наприклад: `db.user.find({name: "John", age: { $gte: 18 }})`. Цією командою ми шукаємо усіх користувачів з ім'ям, вік яких більше або дорівнює `18` (`gte` є скороченням від «**g**rea**t**er or **e**qual» — «більше, або дорівнює»).

Другим параметром команди `find` є проекція, за допомогою якої можна залишити в документах, що повертаються в результаті запиту, лише частину полів. Цей параметр також представляє собою об'єкт, в якому у якості ключів зазначаються назви полів, а у якості значень цих ключів — булевий флаг, що вказує необхідність їх включення або виключення. Цей флаг можна вказувати значеннями `false` або `true`, чи, відповідно, цифрами `0` або `1`. Наприклад, команда `db.user.find({ }, {_id: 0, "name": 1})` поверне всі документи колекції користувачів (так як об'єкт фільтрації пустий), але з полів кожного об'єкту виключить унікальний ідентифікатор (його треба виключати у явному вигляді) та залишить лише поле `"name"`.

Третім параметром є об'єкт опцій (або параметрів) пошуку (`FindOptions`). Він дозволяє застосувати низку параметрів пошуку, починаючи з простих, таких як сортування (`sort`), обмеження результуючої вибірки (`limit`), пропуск деякої кількості документів результуючої вибірки (`skip`) тощо, і закінчуючи заданням правил порівняння строк (`collation`) та властивостями узгодженості та ізоляції даних на читання або запис (`readConcern` та `writeConcern`). У якості прикладу наведемо запит, який знаходить користувачів не молодше 18 років, прибирає унікальний ідентифікатор та сортує результат за віком: `db.user.find({"age": {$gte: 18}}, {_id: 0}, {sort: "age"})`.

Більш детальну інформацію про запити у MongoDB, а також доступні для них оператори, можна знайти на сайті документації:

https://www.mongodb.com/docs/manual/tutorial/query-documents/
https://www.mongodb.com/docs/manual/reference/method/db.collection.find/
https://www.mongodb.com/docs/manual/reference/operator/query/

4. Оновлення даних у MongoDB виконується за допомогою наступних команд:

```
db.collection.updateOne(<filter>, <update>, <options>)
db.collection.updateMany(<filter>, <update>, <options>)
db.collection.replaceOne(<filter>, <update>, <options>)
```

Як можна побачити по сигнатурах, принцип застосування даних команд подібний до команд читання. Наведемо приклад використання однієї з них:

```
db.user.updateMany(
   { "age": { $lt: 50 } },
   {
     $set: { "name": "John", status: "User" },
     $currentDate: { lastModified: true }
   }
)
```

Перший аргумент задає фільтр пошуку документів, які потрібно оновити (усі, де поле `age` менше ніж `50`), а другий визначає поля, які будуть оновлені (оператор `$set`), а також оновлює (або створює, якщо його немає) поле `lastModified` оператором `$currentDate`, куди записується дата оновлення. Таким чином, усім користувачам молодше 50 років буде присвоєно ім'я `John` та проставлено статус `User`.

Більш детально з операцією оновлення можна ознайомитися на сайті документації:

https://www.mongodb.com/docs/manual/tutorial/update-documents/
https://www.mongodb.com/docs/manual/reference/operator/update/

5. Основними операціями видалення є `deleteMany()` та `deleteOne()`. Вони так само приймають об'єкт фільтру документів на видалення у якості аргументу. Наприклад, команда

```
db.user.deleteOne({status: "User"})
```

видалить першого користувача, у якого проставлено поле `status` має значення `User`. Зауважимо, що, як і у випадку оновлення, команди, що працюють з одним документом (`updateOne`, `deleteOne`) оновлюють або видаляють лише перший документ, який буде знайдено MongoDB, а команди, що працюють з багатьма (`updateMany`, `deleteMany`) — усі, що будуть знайдені за даним фільтром.

Більш детальну інформацію про команду видалення можна знайти на сайті документації:

https://www.mongodb.com/docs/manual/tutorial/remove-documents/

Aggregation Pipeline

У MongoDB, як і у звичайних реляційних базах даних, існують операції агрегації: наприклад, `estimatedDocumentCount()`, що повертає приблизну кількість документів у колекції, `count()`, що повертає точку кількість, але працює повільніше, та `distinct()`, що повертає масив документів, унікальних за деяким полем. Проте MongoDB має ще один потужний інструмент, який дозволяє створювати складні запити даних, виконувати з ними різні трансформації, додавати нові поля з обчисленими значеннями і так інше, який називається Aggregation Pipeline (конвеєр агрегації).

Aggregation Pipeline складається з одного або кількох етапів обробки документів. Кожен етап виконує операцію над вхідними документами. Наприклад, етап може фільтрувати документи, групувати документи та обчислювати значення. Вихідні документи кожного етапу передаються на наступний.

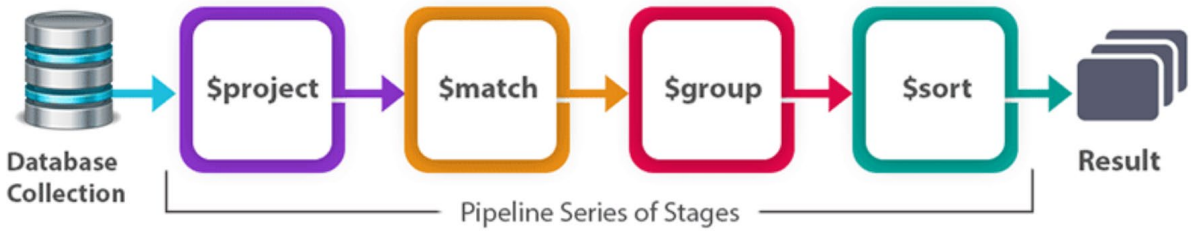

Рисунок 4 — Aggregation Pipeline з чотирма етапами обробки

Доступ до Aggregation Pipeline надається через команду `aggregate()`. Вона, у якості першого аргументу, приймає масив агрегаційних операторів, що утворюють вищезазначені етапи (у загальному вигляді це вигадає так: `db.collection.aggregate([{stage1}, {stage2}, {stage3}, …])`), а у якості другого приймає необов'язковий об'єкт `options`. Операторів Aggregation Pipeline існує велика кількість та кожен з них має свої синтаксичні особливості, а тому для успішного застосування даного функціоналу варто користуватися офіційною документацією.

Наведемо список основних операцій Aggregation Pipeline:

- `$match`: фільтрує документи, щоб передати на наступний етап конвеєра лише ті, що відповідають зазначеним умовам (аналог `WHERE` у мові `SQL`);
- `$project`: передає до наступного етапу конвеєра документи лише з зазначеними полями (вони можуть бути існуючими або обчисленими; аналог переліку атрибутів після команди `SELECT` у `SQL`);
- `$group`: розділяє документи на групи відповідно до «ключу групування» так, щоб на виході був один документ для кожного унікального ключа (аналог `GROUP BY` у `SQL`);

- `$sort`: сортує всі вхідні документи та передає їх до наступного етапу у відсортованому порядку;
- `$skip`: пропускає вказану кількість документів, які надходять у етап, і передає решту документів до наступного етапу в конвеєрі;
- `$limit`: обмежує кількість документів, що передаються на наступний етап конвеєра;
- `$unwind`: деконструює поле масиву вхідних документів, перетворюючи його на документ для кожного елемента. Кожен вихідний документ є вхідним документом із значенням поля масиву, заміненим елементом;
- `$out`: бере документи, повернуті конвеєром агрегації, і записує їх у вказану колекцію (даний оператор має бути останнім етапом у конвеєрі).

Повний перелік операторів Aggregation Pipeline можна знайти на сайті документації: https://www.mongodb.com/docs/manual/reference/operator/aggregation-pipeline/.

Для створення потужних конвеєрів у Aggregation Pipeline також використовуються Expression Operators (операції виразів). Їх можна додавати в код операторів Aggregation Pipeline для виконання обчислень, задання умов, звернення до полів тощо. В загальному випадку ці вирази приймають масив аргументів і мають такий вигляд: `{ <operator>: [ <argument1>, <argument2>, ... ] }`, хоча якщо оператор приймає один аргумент, позначення масиву можна опустити: `{ <operator>: <argument> }`.

Повний перелік Expression Operators можна знайти на сайті документації: https://www.mongodb.com/docs/manual/reference/operator/aggregation/#expression-operators.

Наведемо декілька прикладів використання Aggregation Pipeline.

1. Застосування `$match` та `$project`.

   ```
   db.employees.aggregate([
     { $match: { dept: "Admin" } },
     { $project: { _id: 0, name: 1, dept: 1 } },
   ]);
   ```

   У даному запиті ми залишаємо лише працівників, що працюють в адміністративному департаменті, а потім для кожного документу залишаємо лише поля `name` та `dept` (поле `_id` необхідно прибирати у явному вигляді).

2. Застосування `$group`. Спочатку розглянемо більш простий приклад:

   ```
   db.employees.aggregate([{ $group: { _id: "$dept" } }])
   ```

   У даному запиті ми групуємо документи за полем `dept`. Зверніть увагу, що поле `_id` тут використовується для задання «ключу групування», яке замінить собою

первинні ідентифікатори документів. Також варто зазначити, що поле, яке виступає у якості значення «ключа групування» (dept), записується зі знаком долара і в лапках. Це є одним з прикладів використання Expression Operators, а саме, в даному випадку, ми таким чином звертаємося до існуючого поля документу dept і передаємо його значення полю _id, що на виході нам дасть множину об'єктів наступного вигляду: {"_id": "Admin"}, {"_id": "…"}.

Розглянемо більш детальний приклад.

```
db.sales.aggregate([
  {
    $group: {
      _id: "$item",
      totalSaleAmount: {$sum: {$multiply: ["$price", "$quantity"]}},
    },
  },
  {
    $match: { totalSaleAmount: { $gte: 100 } },
  },
]);
```

Тут ми групуємо за полем item, але ще й додаємо нове агрегаційне поле totalSaleAmount, яке у якості значення отримує суму добутків ціни та кількості. Ми знову тут бачимо використання Expression Operators: $sum та $multiply є операціями, що дозволяють виконати відповідні обчислення, а "$price" та "$quantity" є зверненнями до значень відповідних полів документів (і аргументами оператору $multiply, результат якого передається в $sum).

Після групування застосовується $match для того, щоб на виході залишити тільки ті документи, у яких загальна сума продажу перевищує або рівна 100. Результат виглядатиме наступним чином:

```
{ "_id" : "abc", "totalSaleAmount" : Decimal128("170") }
{ "_id" : "xyz", "totalSaleAmount" : Decimal128("150") }
{ "_id" : "def", "totalSaleAmount" : Decimal128("112.5") }
```

Тут у якості поля _id записані деякі товари (вище вони містилися у полі item), а у якості поля totalSaleAmount розрахована загальна сума продажу.

3. Застосування $unwind.

Уявімо, що ми виконали вставку наступного документу у колекцію inventory:

```
db.inventory.insertOne({ "_id": 1, "item": "ABC1", sizes: ["S", "M",
"L"] })
```

Тепер застосуємо оператор $unwind до масиву з трьох елементів sizes:

```
db.inventory.aggregate([{ $unwind : "$sizes" }])
```

На виході отримаємо три документи наступного вигляду:

```
{ "_id" : 1, "item" : "ABC1", "sizes" : "S" }
{ "_id" : 1, "item" : "ABC1", "sizes" : "M" }
{ "_id" : 1, "item" : "ABC1", "sizes" : "L" }
```

Як бачимо, для кожного елементу масиву `sizes` тепер створено окремий документ з однойменним полем, яке у якості значення має не масив розмірів, а один із розмірів.

Одним із зручних та наочних способів побудови конвеєрів агрегації є використання MongoDB Compass — клієнту для MongoDB з графічним інтерфейсом користувача.

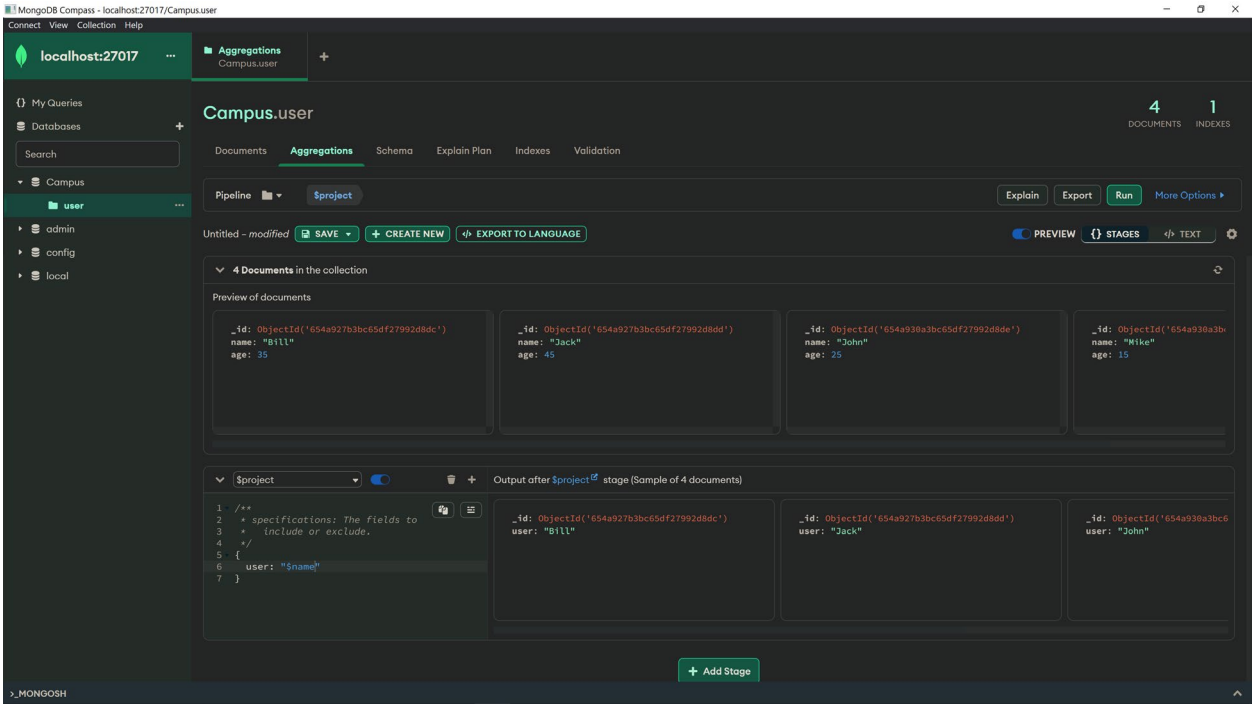

Рисунок 5 — Побудова конвеєру Aggregation Pipeline у MongoDB Compass

Все необхідне для роботи з Aggregation Pipeline у MongoDB Compass знаходиться у вкладці `Aggregations`. Перемикач `STAGES/TEXT` дозволяє переключатись між графічним та текстовим виглядом конвеєру. У графічному режимі зверху зображується рядок з первинними документами колекції, після якого можна додавати агрегаційні етапи конвеєру за допомогою кнопки `Add Stage`. Кожен рядок етапу ліворуч містить випадаючий список операцій та поле для вводу коду для них, а праворуч результат застосування даної операції. Етапи можна «вимикати» перемикачем праворуч від випадаючого списку операцій. Також можна налаштувати кількість документів для показу в попередньому перегляді у вікні налаштувань, що відкривається по натисканню на іконку шестерні, що розташована праворуч від перемикача `STAGES/TEXT`. Більш детальну довідку щодо роботи з MongoDB Compass можна отримати на офіційному сайті документації: https://www.mongodb.com/docs/compass/current/.

Завдання на лабораторну роботу

1. Встановити сервер та клієнти MongoDB та налаштувати репліка-сет.
2. Створити у MongoDB базу даних та внести в неї документи відповідно до варіанту лабораторної роботи №1.
3. Створити запити, що повертають дані згідно індивідуального завдання варіанту лабораторної роботи №1. Щонайменше одне завдання має бути виконане із застосуванням Aggregation Pipeline.

Вимоги до звітності

Звіт має містити наступні розділи:

1. Опис налаштування репліка-сету.
2. Створення бази даних та внесення у неї документів згідно завдання варіанту лабораторної роботи №1.
3. Написання запитів згідно індивідуального завдання варіанту лабораторної роботи №1.
4. Висновки.

Звіт оформити окремим файлом у форматі Word або Markdown з лістингом коду та власними коментарями, супроводжуючими етапи виконання роботи.