МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
імені ВАДИМА ГЕТЬМАНА

Н.В. СИТНИК, І.С. ЗІНОВ'ЄВА

ОРГАНІЗАЦІЯ БАЗ ДАНИХ NoSQL

ПРАКТИКУМ

КНЕУ 115 РОКІВ КИЇВ 2022

УДК 004.655.3(076)
С41

Рецензенти
К. В. Молодецька, д.т.н., проф.
(Поліський національний університет)
С. П. Ріппа, д.е.н., проф.
(Київський національний економічний університет
імені Вадима Гетьмана)

Рекомендовано до друку Вченою радою КНЕУ
Протокол № 3 від 25.11.2021 р.

Ситник Н.В., Зінов'єва І.С.
С41 Організація баз даних NoSQL [Електронний ресурс] : практикум / Н.В. Ситник, І.С. Зінов'єва. — К. КНЕУ, 2022. — 167, [1] с.

ISBN 978–966–926–406-0

У практикумі наведено матеріали, необхідні для практичної апробації теоретичних знань з організації баз даних NoSQL і використання їх у сучасних інформаційних системах. Подано блок навчально-методичного забезпечення, який включає по кожній темі перелік знань і вмінь, термінологічний словник, плани практичних занять, навчальні завдання, інструктивні та методичні вказівки для проведення лабораторних робіт і питання для самоконтролю засвоєння знань.

Практикум призначено для студентів вищих навчальних закладів, що навчаються за спеціальністю «Комп'ютерні науки» галузі знань «Інформаційні управляючі системи і технології». Може бути використаний аспірантами, викладачами та спеціалістами, які зацікавлені в набутті практичних знань і навичок з організації баз і сховищ даних..

УДК 004.655.3(076)

ISBN 978–966–926–406-0

ЗМІСТ

ПЕРЕДМОВА

Практикум «Організація баз даних NoSQL» призначений для підготовки бакалаврів за спеціальністю 122 «Комп'ютерні науки», галузь знань «Інформаційні технології». Останнє десятиліття характеризується появою різного роду WEB-сервісів, мобільних додатків, гепросторових інформаційних систем та інтернет речей, що призвело до вибухового зростання обсягів даних. Крім обсягів даних змінилися їх характеристики, виникла потреба в зберіганні та опрацюванні неструктурованої або слабоструктурованої інформації, яка є проблемною для реляційних баз даних. Новітнім напрямом розвитку баз даних є бази даних NoSQL, які призначені для вирішення зазначеної проблеми. NoSQL - це спільна назва всіх СКБД, що можуть опрацьовувати великі обсяги розподілених неструктурованих чи слабоструктурованої даних. Назва NoSQL не заперечує реляційні СКБД, вона трактується як «Not Only SQL» тобто «не лише SQL». Базами даних NoSQL не підтримується реляційна модель і не використовується мова запитів SQL. Характерним для баз даних NoSQL є відсутність регламентованої структури та жорстко заданої схеми, але при цьому підтримка гнучких динамічних схем з можливістю опрацювання агрегатів і ненормалізованих даних. Бази даних NoSQL підтримують горизонтальне масштабування на кластерах та роботу з Big Data. Ці бази даних на сьогодення є найпоширенішим і найоптимальнішим засобом зберігання неструктурованих даних, які широко застосовуються в інтернет і мобільних додатках.

Метою практикуму є формування фундаментальних теоретичних знань і практичних навичок з організації нереляційних баз даних та оволодіння сучасною технологією роботи у середовищі сучасних систем керування базами даних NoSQL.

І. НАВЧАЛЬНО-МЕТОДИЧНЕ ЗАБЕЗПЕЧЕННЯ ДО ТЕМ ДИСЦИПЛІНИ

Тема 1.

ОСНОВИ КОНЦЕПЦІЇ НЕРЕЛЯЦІЙНИХ БАЗ ДАНИХ NoSQL (Not Only SQL)

1.1. Перелік знань і вмінь

Вивчення матеріалу даної теми дасть змогу студентам опанувати основні терміни та визначення, які пов'язані з базами даних NoSQL, що створить основу для вивчення наступних тем.

Вивчивши матеріал теми, ви будете знати:

— класифікацію баз даних;

— проблеми практичного використання реляційних баз даних при роботі з WEB-додатками та великими даними (BigData);

— поняття великих даних (BigData) і проблеми їх представлення в реляційних базах даних;

— передумови та причини виникнення концепції нереляційних баз даних;

— порівняльні характеристики реляційних і нереляційних баз даних;

— призначення масштабування даних і його практичне застосування;

— особливості вертикального та горизонтального масштабування даних і їх порівняльні характеристики;

— теоретичні засади баз даних NoSQL (теорема CAP);

— переваги та недоліки баз даних NoSQL;

— практичне застосування баз даних NoSQL.

1.2. Термінологічний словник

Агрегат **—** це сукупність атрибутів, які інтерпретуються СКБД як єдине ціле. У таблицях реляційної бази дані зберігаються у вигляді атомарних атрибутів. Реляційні бази даних не підтримують агрегати, тому їх називають безагрегатними (aggregate-ignore). Підтримку агрегатних даних забезпечують бази даних NoSQL. Переважна більшість баз даних NoSQL (за виключенням

графових) є агрегатно-орієнтованими. Агрегати зручні для роботи у кластерах, так як агрегат представляє собою одиницю реплікації і фрагментації.

Атомарні атрибути — це атрибути, які зберігають неподільне значення, яке не може бути списком чи певною множиною значень. Тобто це такі дані, розподіл яких на кілька елементів призводить до порушення їх семантики.

Атомарність транзакції — трактується як «все або нічого», тобто успішно виконана транзакція вносить зміни до бази даних, у протилежному випадку змін не відбувається.

Багатоваріантна персистентність (Polyglot Persistence) — використання кількох СКБД для вирішення різних за технологією задач. Багатоваріантна персистентність може бути реалізована на різних рівнях, наприклад, по підприємству в цілому, або в рамках окремого IT-проекту.

База даних (БД) — це поіменована, структурована сукупність даних, що характеризує окрему предметну область і перебуває під управлінням системи керування базами даних (СКБД).

База даних NoSQL — спільна назва для цілого класу баз даних відмінних від традиційних реляційних моделей, що підтримують неструктуровані та слабоструктуровані агрегатні дані без схем, не підтримують трансакції і не мають засобів мови SQL. Термін NoSQL («not only SQL» або «не лише SQL») був уперше запропонований у 2009 році і застосовується до баз даних, у яких вирішуються проблема горизонтального масштабування та роботи з Big Data.

База даних NewSQL — клас сучасних реляційних баз даних, які об'єднують у собі переваги NoSQL, підтримують трансакції та мову SQL. Термін був запропонований у 2011 році Метом Аслетом. Необхідність у базах даних NewSQL пояснюється потребою розподіленого, швидкого і надійного оброблення великих даних, у той час як бази NoSQL, не маючи механізму підтримки транзакцій, не відповідають вимогам надійності збереження та забезпечення цілісності й узгодженості даних.

Вертикальне масштабування (Vertical scaling) — спосіб збільшення обсягу БД шляхом нарощення кількості комп'ютерів і серверів БД, тобто за рахунок зростання кількості процесорів, дискової та оперативної пам'яті тощо. Цей тип масштабування, з одного боку, є найпростішим і не вимагає кардинальних змін на прикладному рівні (в існуючій системі усі слабкі компоненти можна замінити на потужніші). З іншого боку, він є досить дорогим і має певну суто технічну межу, яка обмежена існуючою на

конкретний момент часу максимально можливою потужністю сервера або фінансовими ресурсами.

Горизонтальне масштабування (Horizontal scaling) — спосіб збільшення обсягу БД шляхом розбиття її на окремі складові та переміщення їх на окремі сервери, що паралельно виконують одну і ту ж функцію, тобто має місце автоматичне розподілення даних між кількома серверами та підтримка кількох датацентрів з можливість підключення нових комп'ютерів, які об'єднуються в кластери. Цей тип масштабування можу потребувати змін на прикладному рівні.

BASE-вимоги — вимоги, що підтримуються базами даних NoSQL, зокрема: 1) базова доступність (*Basic Availability*) — вимога передбачає, що збій на деяких вузлах приводить до відмови в обслуговуванні лише незначної кількості звернень при збереженні доступності в переважній більшості випадків, тобто гарантується успішне чи безуспішне завершення кожного запиту; 2) гнучкий стан (*Soft state*) — вимога передбачає, що стан системи може змінюватись для досягнення узгодженості не залежно від того, були введені нові дані чи ні; 3) цілісність/узгодженість у підсумку (*eventual consistency*) — вимога передбачає, що в деяких випадках існує можливість суперечливості даних, але завжди досягається їх узгодженість (наприклад, пізніше під час читання).

Великі дані (Big Data) — набір структурованих і неструктурованих даних великих обсягів і значної різноманітності, що ефективно обробляються горизонтально масштабованими програмними інструментами. Основними характеристиками великих даних є три «V», а саме: *volume* — фізичний об'єм, *velocity* — швидкість зростання обсягів даних і швидкість їх оброблення, *variety* — різноманітність і можливість одночасного оброблення різних типів даних (структурованих і неструктурованих).

Ізольованість транзакції — властивість, яка характеризує локальне виконання кожної окремої транзакції у БД. Ізольовані транзакції виконуються паралельно та не впливають одна на одну.

Комп'ютерний кластер або просто ***кластер —*** це кілька незалежних обчислювальних машин, що використовуються спільно і працюють як одна система для вирішення тих чи тих задач, наприклад, для підвищення продуктивності, забезпечення надійності, спрощення адміністрування тощо. Обчислювальний кластер потрібен для збільшення швидкості обрахунків за допомогою паралельних обчислень.

Консистентність даних (***data consistency*** чи ***data validity***) — узгодженість і цілісність даних, їх внутрішня несуперечливість.

Кластер серверів (Server Cluster) — певна кількість серверів, об'єднаних у групу, які утворюють єдиний уніфікований обчислювальний ресурс. Кластери можуть використовувати просте апаратне забезпечення за рахунок цього вартість горизонтального масштабування є набагато меншою за вертикальне масштабування. Крім того, кластери є надійнішими, у разі виходу з ладу одного з комп'ютерів кластер продовжує функціонувати.

Масштабування (scalability) — спосіб подолати збільшення навантаження на БД. Як правило, масштабування потрібне, коли не вистачає потужностей наявного сервера (або серверів), тоді дані розділяються на певні групи для розміщення їх на окремих серверах. Види масштабування — вертикальне та горизонтальне. Стратегії масштабування — партиціювання, шардинг, реплікації.

Модель даних — формальна теорія представлення та оброблення даних, яка визначає методи: опису типів даних і логічних структур, маніпулювання даними, опису та підтримки цілісності бази даних.

Надійність транзакції — властивість, яка означає, що зміни внесені до бази даних після успішного завершення транзакції гарантовано зберігаються, навіть у випадку збоїв обладнання чи програмного забезпечення.

Партиціювання (partitioning) — це функціональний розподіл даних на логічно завершені частини згідно певних критеріїв (функцій). Такий розподіл поділяє всі операції з оброблення на кілька незалежних і паралельних процесів, що суттєво прискорює роботу СКБД. Наприклад, на сайтах новин усі дані розподіляються за датою їх публікації, так як до свіжих новин більше звернень ніж до архівних.

Прозорі кластери (Transparent cluster) — системи NewSQL для масштабування поверх традиційних СКБД, які забезпечують проміжний шар (middleware), для автоматичного розподілу баз даних на кілька вузлів. При цьому база даних зберігається у своєму початковому форматі та забезпечується можливість прозорого її підключення до кластера, щоб забезпечити масштабованість.

Персистентні структури даних (persistent data structure) — це структури даних, які при внесенні до них змін зберігають усі свої попередні стани та доступ до них.

Реляційна база даних (Relational Data Base Management System — RDBMS) — це модель бази даних, в основу якої покладено математичне поняття відношення, фізичним представленням якого є поіменована двовимірна плоска таблиця, що складається з рядків (кортежів) і стовпчиків (атрибутів). Реляційні бази

даних — це чітко структуровані дані, що характеризують певну предметну область і представляються у вигляді логічно пов'язаних поіменованих двовимірних плоских таблиць, що знаходяться під управлінням системи керування базами даних (СКБД).

Реплікація — синхроне чи асинхроне копіювання даних між кількома серверами. Ведучий сервер називають майстром (master), а підпорядковані сервери — слейвами (slave). Майстри використовуються для зміни даних, а слейви — для читання. У класичній схемі реплікацій, як правило, використовується один майстер і кілька слейвів, так як у переважній більшості вебпроектів привалюють операції читання над операціями запису. Проте є системи, що підтримують кілька майстерів. Репліка може включати всю базу даних (повна репліка), чи певну її підмножину.

Теорема CAP (Consistency, Availability, Partition tolerance), відома як **теорема Брюєра** — евристичне твердження про те, що у будь-якій розподіленій системі (це стосується і розподіленої БД) можливо забезпечити не більше двох з трьох таких властивостей:

- ***узгодженість*** (*consistency*) — в усіх вузлах мережі в один і той же момент часу дані не протирічать один одному;
- ***доступність*** (*availability*) — будь-який запит до розподіленої БД завершується коректною відповіддю;
- ***толерантність до розподілу*** (*partition tolerance*) — розщеплення розподіленої системи на кілька ізольованих секцій не приводить до некорректного відгуку кожної секції.

Транзакції — основа забезпечення несуперечливості даних у реляційних БД. Транзакція гарантує виконання всіх логічно пов'язаних команд, що входять до однієї трансакції. Якщо хоч одна команда завершується помилкою, то всі команди відкочуються назад, так, ніби вони і не виконувалися. Трансакції характеризуються такими властивостями ACID: атомарність (Atomicy), узгодженість (Consistency), ізольованість (Isolation), надійність (Durability).

Узгодженість транзакції — властивість, яка означає, що транзакція буде успішно завершена лише у випадку, якщо зміни не порушують цілісності бази даних.

Шардінг (sharding) — це розподіл даних між різними серверами. Шардінг є подальшим розвитком партиціювання і дозволяє виконати розподіл даних на групи, що, в ідеалі, повинно гарантувати, що дані, до яких виконується одночасне звернення, розміщуються на одному сервері. Процес шардингу виконує розподіл

даних між окремими шардами з використанням ключа шардингу. Ключ шардингу — це параметр, згідно якого виконується розподіл даних між окремими серверами, наприклад ідентифікатор клієнта чи організації. На одному вузлі розміщується один шард. При цьому слід враховувати максимальний взаємозв'язок даних в одному шарді.

1.3. Завдання для індивідуальної роботи

Завдання 1. На основі вивчення лекційного матеріалу та знань отриманих при вивчення курсу «Організація баз та сховищ даних» проведіть співставлення характеристик реляційних та NoSQL баз даних.

Завдання 2. За літературними та інтернет-джерелами дослідити та проаналізувати перехід від реляційної до мультимодельної клієнт-серверної СКБД Microsoft SQL Server.

1.4. Практичне заняття

Мета заняття: опанувати основи концепції баз даних NoSQL та ознайомитись із перевагами їх використання при створенні розподілених інформаційних систем. Вивчення основних термінів визначень.

ПЛАН

1. Охарактеризуйте передумови розробки концепції баз даних NoSQL.
2. Порівняйте реляційну модель бази даних з моделлю NoSQL.
3. Дайте визначення теореми CAP.
4. Наведіть поняття масштабування, характеристику вертикального та горизонтального масштабування даних.
5. Здійсніть порівняльну характеристику BASE- та ACID-вимог до бази даних.
6. Охарактеризуйте переваги та недоліки баз даних NoSQL.

1.5. Навчальні завдання

Завдання 1. Побудувати таблицю порівняльних характеристик основних ознак реляційних і баз даних NoSQL.

Завдання 2. Вивчіть матеріали ресурсу https://db-engines.com/en/ranking, проаналізуйте та дослідить тенденції розвитку баз даних NoSQL за останні 3 роки.

1.6. Питання для самоконтролю

1. Характеризуйте причини створення та розвитку концепції баз даних NoSQL.
2. Дайте визначення агрегата та які моделі бази даних можуть їх підтримувати.
3. Які переваги мають бази даних NoSQL при роботі у WEB-середовищі?
4. Сформулюйте теорему CAP та її застосування в розподілених системах.
5. Характеризуйте основні підходи до розподілу даних.
6. Що представляє собою масштабування та які є його види?
7. Що представляє собою реплікація та її види в базах даних NoSQL?
8. Охарактеризуйте BASE-вимоги до баз даних NoSQL.
9. Охарактеризуйте клас баз даних NewSQL.
10. Охарактеризуйте ACID-властивості транзакцій. Чому бази даних NoSQL не мають механізму підтримки транзакцій?
11. Охарактеризуйте основні сфери застосування баз даних NoSQL.

1.7. Література до теми

1. Кайл Бэнкер MongoDB в действии / пер. с англ. Слинкина А.А. М.: ДКМ Пресс, 2010. 394 с.
2. Редмонд Уилсон: Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL / пер. с англ. М.: ДМК-Пресс, 2017. 384 с.
3. Робинсон Ян, Вебер Джим, Эифрем Эмиль Графовые базы данных: новые возможности для работы со связанными данными / пер. с англ. Р.Н. Рагимова; науч. ред. А. Н. Кисилев. 2-е изд. М.: ДМК Пресс, 2016. 256 с.
4. Фаулер Мартин, Садаладж Прамодкумар Дж. NoSQL: новая методология разработки нереляционных баз данных. Пер. с англ. М.: ООО «И.Д. Вильямс», 2013. 192 с.

Тема 2.

ДОКУМЕНТНО-ОРІЄНТОВАНІ БАЗА ДАНИХ NoSQL ТА ОСНОВИ ЇЇ СТВОРЕННЯ В СКБД MongoDB

2.1. Перелік знань і вмінь

При вивченні даної теми студенти знайомляться з документно-орієнтованою СКБД MongoDB та отримають практичні навички з інсталяції нереляційної БД на робочу станцію, опанують навики первинного налаштування операційної системи для роботи із СКБД MongoDB. Ознайомляться з JSON форматом та опанують синтаксис опису документів у цьому форматі. Отримають практичні навички моделювання документів, а саме: моделювання вкладених документів і масивів.

*Вивчивши матеріал теми, ви будете **знати**:*

- що представляє собою документно-орієнтована база даних NoSQL під управління СКБД MongoDB;
- основні елементи БД MongoDB і відповідність їх структурним елементам реляційних БД;
- синтаксис JSON формату для опису об'єктів (документів) на логічному рівні.

*Вивчивши матеріал теми, ви будете **вміти**:*

- інсталювати СКБД MongoDB на робочу станцію;
- встановлювати на робочу станцію графічний клієнт Compass;
- виконувати первинне налаштування операційної системи для роботи із СКБД MongoDB;
- описувати документи на JSON форматі;
- створювати вкладені документи та масиви.

2.2. Термінологічний словник

Документ (об'єкт) — невпорядкований набір пар «ключ / значення», що входить до колекції. Об'єкт починається з фігурної дужки `{` і закінчується фігурною дужкою `}`. Кожна назва ключа супроводжується двокрапкою `:`, а пари «ключ / значення» розділяються комою. Наприклад:

```
{"name":"Honda", "model":"CRV"}
```

Документ ідентифікується первинним ключем, який задається розробником чи автоматично встановлюється СКБД MongoDB.

Вкладені документи в MongoDB — використовуються для об'єднання документів однієї колекції між собою, можна вважати це своєрідним аналогом команди `JOIN` у реляційних моделях. Вкладені документи можна використовувати для моделювання зв'язку 1:1. Наприклад:

```
{"Name": "L920",
   "Model": "Lumia 920",
   "OSFamily": "Windows",
   "OSVersion": "8",
   "Brand": {"BrandName": "Nokia",
        "BrandCountry": "Finland"}}
```

Вкладений документ «Brand»

Для реалізації зв'язку 1:Б використовується масив вкладених документів.

Колекція MongoDB — складова бази даних, що містить сукупність структурно чи семантично подібних документів. Колекція може вміщувати документи різної структури, з різними типами значень і назвами полів. Спільним для об'єднання документів в одну колекцію може бути предметна область та їх семантична спорідненість.

Масив JSON (одновимірний) — множина значень, що мають порядкові номери (індекси). Документ у середовищі MongoDB може містити масиви. Масив береться у квадратні дужки `[ ]`. Значення розділяються комами. Значення може бути: рядком у подвійних лапках, числом, значеннням true чи false, об'єктом, масивом, чи значенням null. Наприклад:

```
{"name":"BMW",
model":"X64","engine":2,"color":["grey","black"]}
```

Розподілена БД (DDB — distributed database) — це сукупність взаємопов'язаних БД, розподілених у комп'ютерній мережі.

Система GridFS призначена для зберігання файлів, розмір яких перевищує 16 МБ. Для роботи з нею необхідно встановити через Nuget пакет MongoDB.Driver.GridFS.

Додаток «mongod.exe» — сервер бази даних, який обробляє запити, управляє форматами даних і т.д.

Додаток «mongos.exe» — виконує шардинг даних, надає сервіс маршрутизації для конфігурування шардів і визначає розташування даних у кластері.

Додаток «mongo.exe» — консольний клієнт для «mongod.exe». Представляє собою інтерфейс для роботи з базою даних.

СКБД MongoDB — NoSQL-система керування документно-орієнтованою, розподіленою БД, яка орієнтована на WEB-додатки й інфраструктуру Інтернету. MongoDB — яке підтримує горизонтальне масштабування з використанням шардінгу (sharding), завдяки чому згідно вибраного ключа шардування дані поділяються на окремі частини для зберігання у різних кластерах. СКБД MongoDB не підтримує транзакційну цілістність, тобто не забезпечує виконання ACID-вимог. MongoDB є кросплатформною системою і може працювати у різних операційних середовищах, зокрема: Windows, Linux, MacOS, Solaris. MongoDB має драйвери для багатьох мов програмування: C, C#, C++, Erlang, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby та Scala. MongoDB не має графічного інтерфейсу адміністрування. Переважна більшість функцій адміністрування виконується з командного рядка.

У середовищі СКБД MongoDB прийнято оперувати поняттями: база даних, колекція, документ (рис. 1).

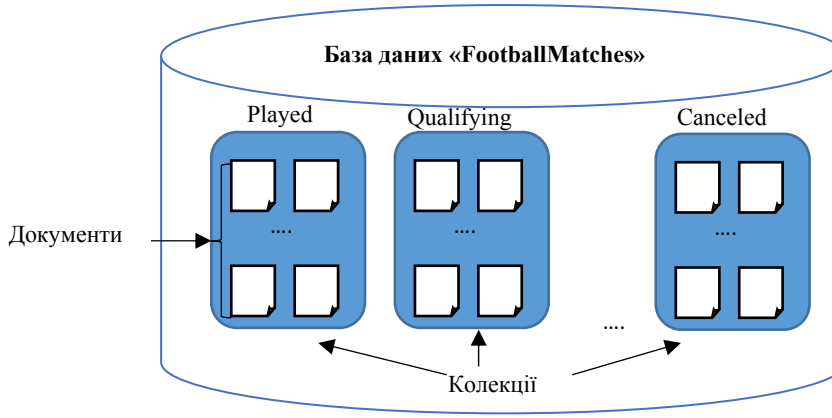


Рис. 1. Графічна схема структури нереляційної бази даних у СКБД MongoDB

Відповідно, якщо бачимо конструкцію типу: FootballMatches.Played, то можемо говорити, що мова іде про колекцію «Played», яка належить базі «FootballMatches».

Формат BSON — це бінарний двійковий формат для зберігання даних MongoDB на фізичному рівні.

Формат JSON (JavaScript Object Notation) — використовується для опису на логічному рівні даних документів у СКБД

MongoDB. Він базується на підмножині мови програмування JavaScript. JSON побудований на базі набору пар «ключ/значення». Цей формат підтримує всі основні типи даних: числа, рядки, булеві значення, а також масиви.

Валідність JSON формату — перевірка на правильність синтаксису об'єктів, описаних у JSON-форматі.

2.3. Інструктивні та методичні вказівки для виконання лабораторної роботи №1 «Установка та перше знайомство з СКБД MongoDB»

Мета роботи: знайомство із СКБД MongoDB, інсталяція нереляційної СКБД на робочу станцію, отримання навиків первинного налаштування операційної системи для роботи в середовищі MongoDB.

Завдання роботи

1. Ознайомитись із загальною інформацією про СКБД MongoDB, яку подано в лекційному матеріалі, а також у джерелах [1, 2, 3].
2. Прочитати інструкцію з інсталяції СКБД MongoDB, яка наводиться нижче.
3. Встановити на робочу станцію СКБД MongoDB та її графічний клієнт Compass.
4. Виконати налаштування операційної системи для роботи із СКБД MongoDB на робочій станції.
5. Оформити звіт з лабораторної роботи.

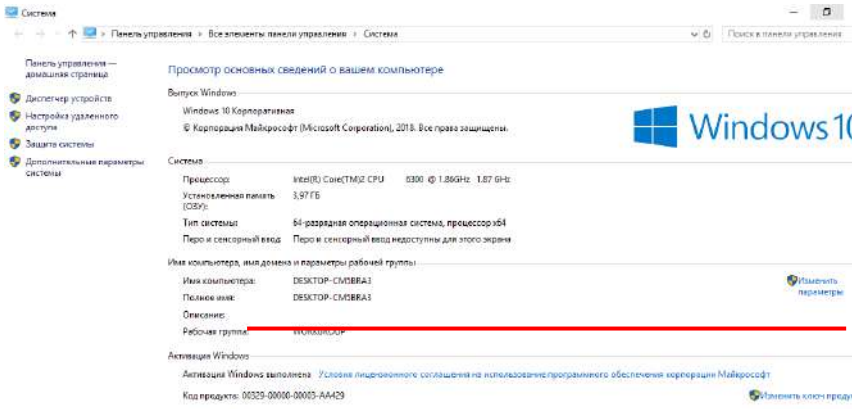
Інструкція з установки СКБД MongoDB

Перш за все потрібно визначитися з версією СКБД. Є 64-бітні і 32-бітні версії MongoDB. 64-бітні версії MongoDB працюють лише на Windows Server 2008 R2, Windows 7 64-bit і з новішими версіями ОС. MongoDB для 32-бітних систем працюють лише на 32-бітних ОС, починаючи з Windows Vista і пізніших версій.

Для з'ясування версії Windows на конкретному сервері, можна командами в командному рядку чи консолі Powershell:

```
wmic os get caption
wmic os get osarchitecture
```

Або за допомогою Панелі інструментів Windows у розділі Система:



Після того, як визначено розрядність системи, можна переходити до безпосереднього завантаження виконавчих файлів із мережі Інтернет за визначеними параметрами операційної системи робочої станції.

Відзначимо, що оскільки в комп'ютерних класах ННІ «Інститут інформаційних технологій в економіці» встановлено операційну систему Windows, то в цій інструкції розглядається механізм інсталяції СКБД MongoDB саме у операційному середовищі Windows. Якщо необхідно встановити нереляційну СКБД на інші операційні системи, зверніться до цих ресурсів [4, 5].

Інсталяція MongoDB (для Windows)

Перейти на офіційний сайт MongoDB. Там для загального доступу пропонується два варіанти СКБД MongoDB:

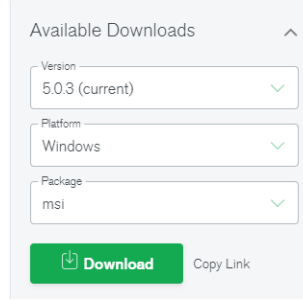
1. Це версія Community version (https://www.mongodb.com/download-center/community).
2. Це версія Enterprise version (https://www.mongodb.com/download-center/enterprise).

Обидві доступні до завантаження. Різниця між ними полягає у тому, що корпоративна версія має ширший функціонал (Ops Manager, Compass і Connector for BI). Окрім того, MongoDB Enterprise version має вищій рівень захисту даних завдяки Encrypted Storage Engine, LDAP і Kerberos.

 У межах виконання циклу лабораторних робіт із СКБД MongoDB студентам пропонується завантажити Enterprise version.

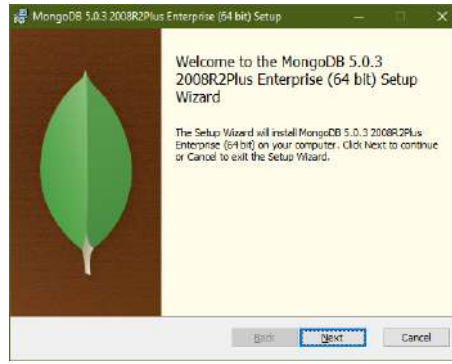
Визначившись із версією СКБД MongoDB варто перейти до завантаження. Для цього необхідно у вікні завантаження вказати:

- у полі Version: 5.0.3 (current release). Зверніть увагу, що номер версії може змінюватись на момент виконання лабораторної роботи. У даному випадку ключовим орієнтиром для студента є напис актуальності релізу (current). Однак, враховуючи різні обставини, студентом може бути прийнято рішення завантаження більш ранньої версії СКБД, наприклад: 4.2, 3.6 та ін.;
- у полі OS: обрати версію та розрядність ОС актуальну для вашої робочої станції;
- у полі Package: можна обрати форму завантаження: або MSI або ZIP. Студентам рекомендується обрати варіант msi:

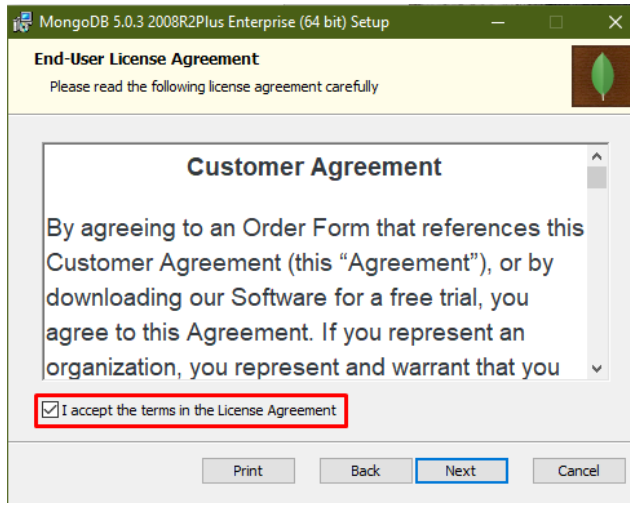


Після, варто натиснути на кнопку «Download». Для завантаження MongoDB Enterprise version необхідно заповнити невелику реєстраційну форму.

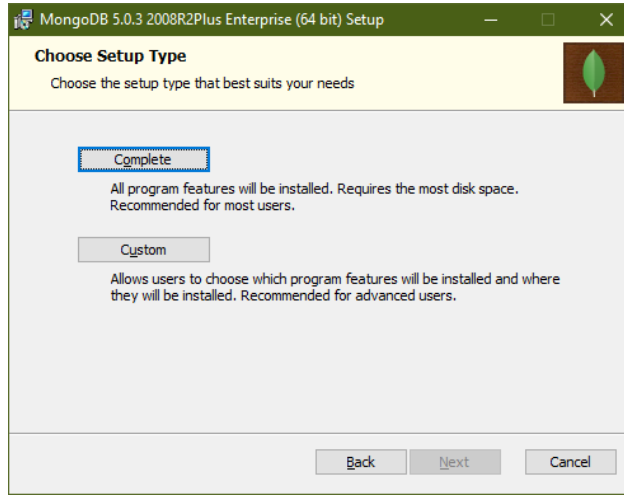
Установка MongoDB відбувається за допомогою майстра інсталяції — Setup Wizard:



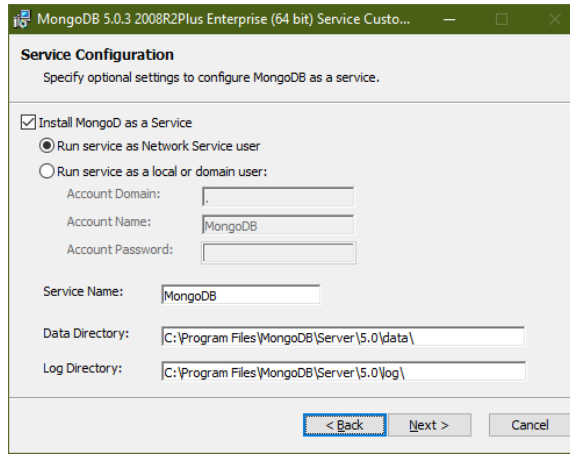
Для того, щоб інсталювати програму на робочу станцію важливо прочитати та прийняти умови ліцензії.



Пропонується установити пакет даних, тому варто обрати Тип установки — Complet:



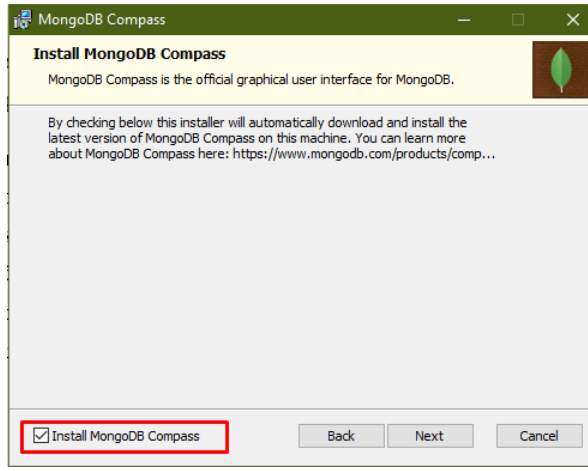
Якщо було обрано Enterprise version для інсталяції, тоді Setup Wizard запропонує здійснити налаштування конфігурації:



Для виконання задач лабраторних робіт достатньо залишити налаштування запуску СКБД MongoDB як мережевого користувача.

Поле «Data Directory» зазначає шлях до каталогу даних користувача. Для виконання лабораторних робіт директорію змінювати не рекомендується. Поле «Log Directory» вказує директорію, де будуть зберігатись дані про роботу користувацьких служб. Для виконання лабораторних робіт директорію змінювати не рекомендується.

У Enterprise version можна під час інсталяції разом виконати завантаження графічного клієнта Compass:



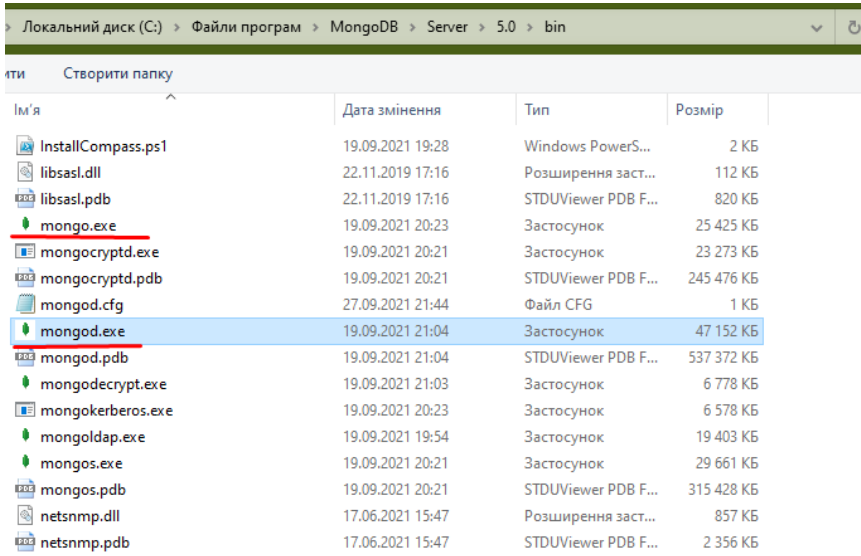
Графічний клієнт Compass можна завантажити окремо. Далі буде опис дій.

Фактично установка СКБД MongoDB виконана, після натискання спочатку на кнопку «Install», а потім — «Finish» у вікні Setup Wizard.

Як зазначалось вище, виконавчі файли СКБД MongoDB на робочій станції інстальовані у директорії: C:\Program Files\MongoDB\Server\5.0. Виконавчі файли знаходяться у каталозі bin.

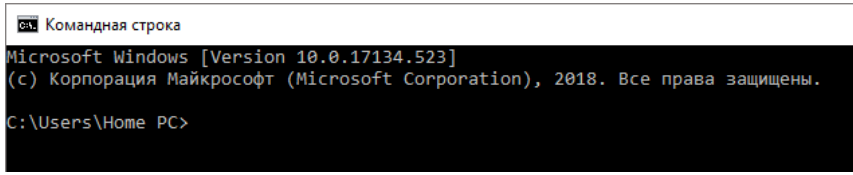
Серед іншого, там знаходяться два виконавчі файли:

- mongo.exe — додаток запуску клієнтської частини СКБД MongoDB;
- mongod.exe — додаток запуску серверної частини СКБД MongoDB.



Робота із СКБД MongoDB починається із запуску: 1) серверної частини, 2) клієнтської частини.

Користивацькі файли СКБД MongoDB зберігає у каталозі Користувачі — Назва користувача: C:\User\Назва користувача:



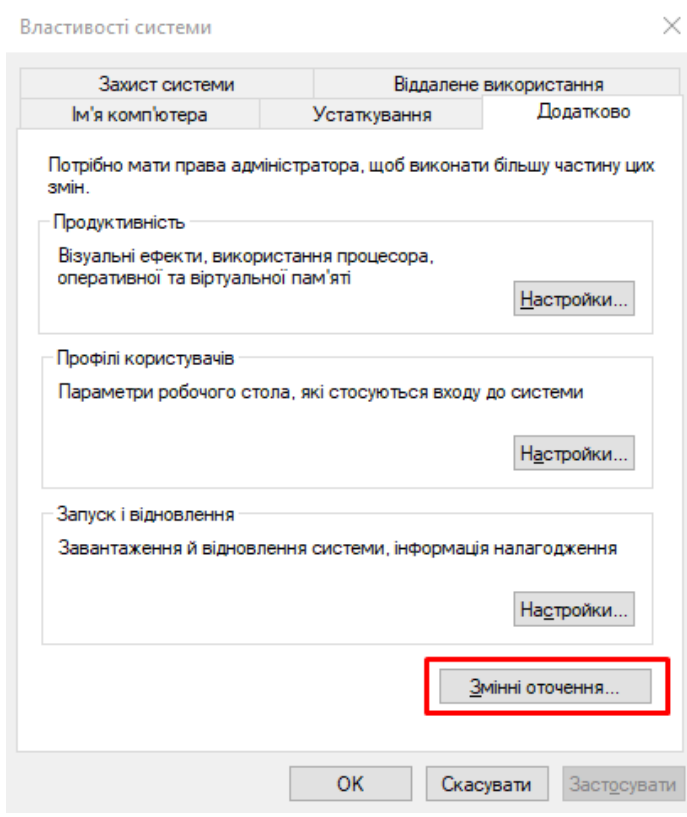
Для коректної роботи СКБД, особливо, на робочих станціях під ОС Windows, необхідно налаштувати ряд системних параметрів, а саме «Змінні середовища». Для цього потрібно зайти у налаштування Паналі управління — «Додаткові параметри системи».

Є кілька віаріантів виклику діалогового вікна Додаткові параметри системи. Найпоширеніші:

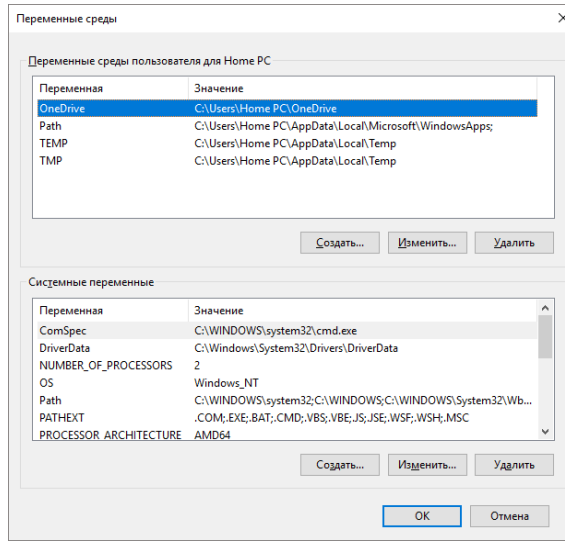
1) Windows logo key + R → надрукувати або sysdm.cpl або SystemPropertiesAdvanced;

2) Робочий стіл (на іконці Мій комп'ютер клікнути правою клавішею миші) → Властивості → Додаткові налаштування системи.

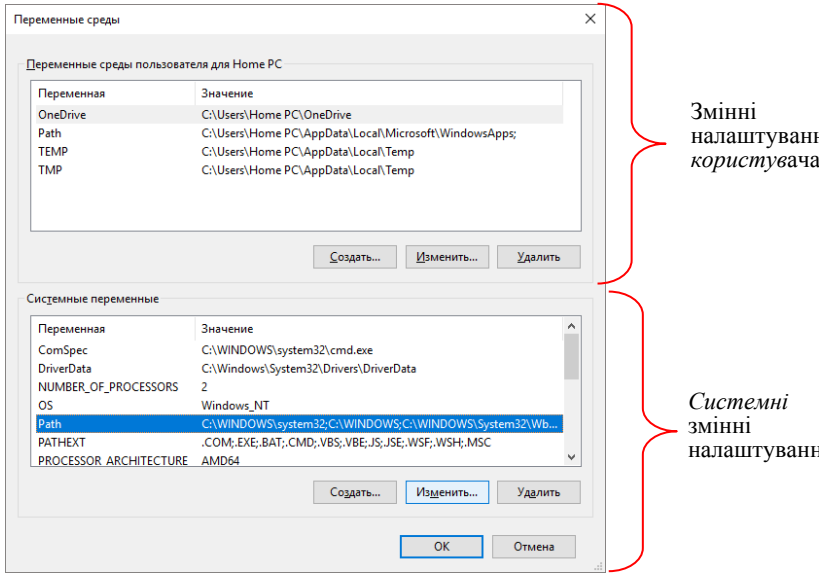
У діалоговому вікні обрати закладку «Додатково» (Advanced) натиснути на пункт «Змінні середовища» (Environment variables).



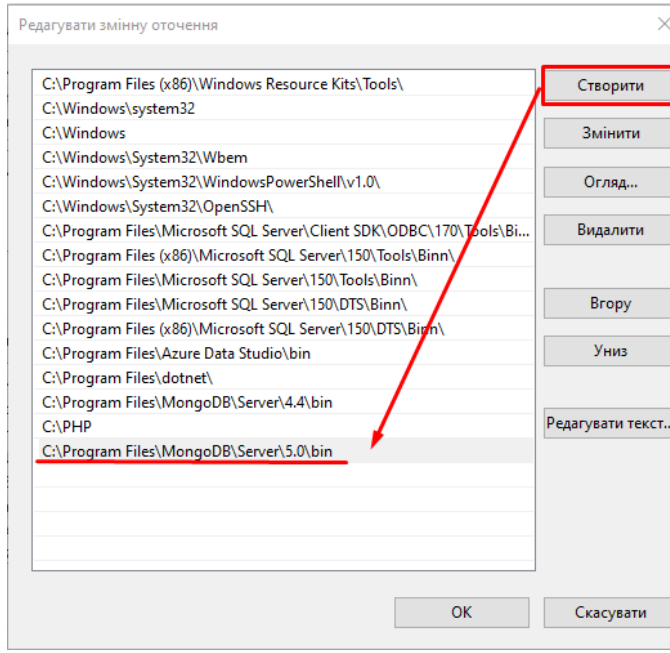
Саме вікно «Змінні середовища» складається з двох типів змінних: змінні користувача та системні змінні:



Внесення змін необхідно здійснити у середовищі системних змінних. Для цього треба знайти параметр Path, виділити його і натиснути кнопку Змінити:



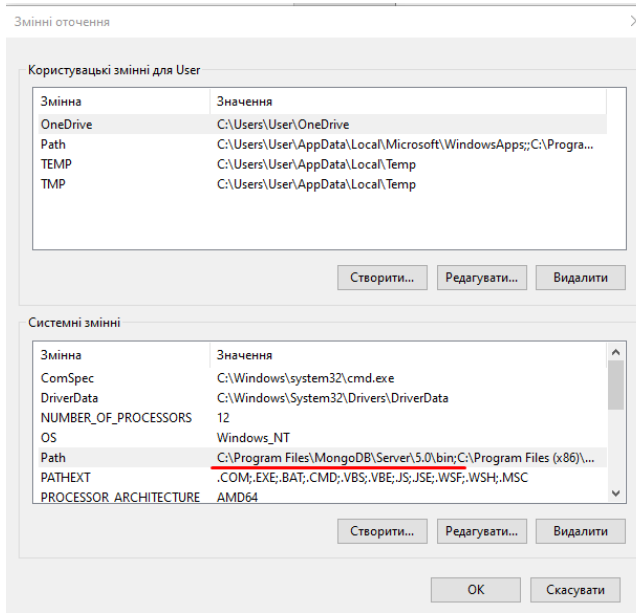
З’явиться нове діалогове вікно редагування змінних середовища.



У діалоговому вікні редагування змінних середовища необхідно натиснути кнопку «Створити» та вказати шлях, до теки із виконавчими файлами MondoDB. Фактично указати шлях до директорії «bin» СКБД MongoDB (наприклад, C:\Program Files\MongoDB\Server\5.0\bin), куди встановлено виконавчі файли СКБД MongoDB на робочій станції.

Рекомендується перемістити створене змінне середовище MondoDB на перше місце у переліку (за допомогою кнопки «Вгору». Це необхідно зробити для того, що під час роботи із консоллю введені команди звіряються із наявним переліком змінних середовища, і якщо на комп’ютері були раніше встановлені інші версії MongoDB, то запуститься та версія СКБД, яка у переліку буде першою.

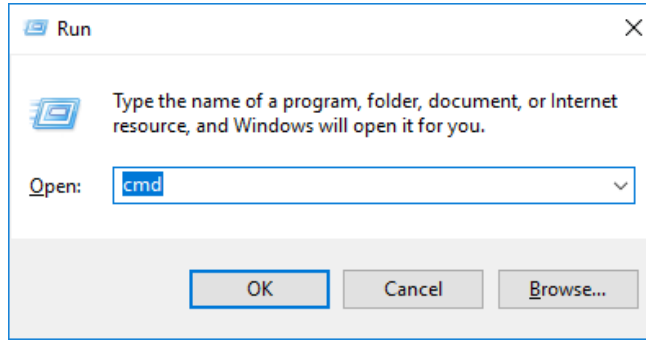
Після виконаних змін, у вікні «Параметри середовища» можна побачити, що параметр Path отримав додаткові налаштування:



Запустимо консоль або командну строку, для того, щоб переконатись, що СКБД MongoDB інстальована на робочу станцію коректо.

Для перевірки коректності встановлення MongoDB, необхідно відкрити командний рядок:

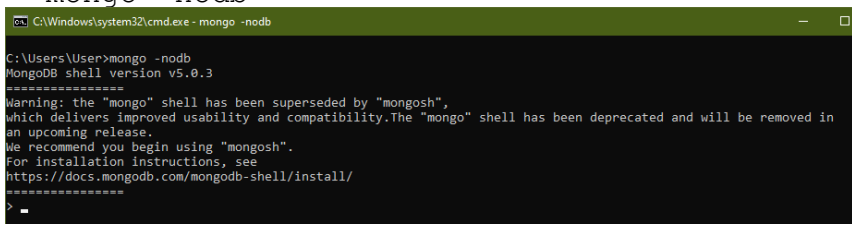
– або за допомогою клавіш Windows + R,у поле вводу написати «cmd» і натиснути «OK»):



– або у пункті мені Пуск знаходимо консоль під назвою «cmd» або «командна строка» і запускаємо її.

У результаті цих дій відкривається вікно консолі з, як уже зазначалось, почтаковою директорією. Для того, щоб перевірити, чи внесені зміни відбулись і чи коректно встановлено СКБД, введемо у командний рядок таку команду (що означає: запуск утіліти Mongo без завантаження бази даних):

```
mongo –nodb
```



У відповідь на введену команду на консолі має з’явитись системна відповідь щодо запуску утілити Mongo та зазначення її поточної версії.

Для того, щоб припинити роботу із утилітою Mongo у командному рядку треба ввести команду:

```
quit()
```

Для того, щоб припинити роботу у консолі треба ввести команду:

```
exit
```

Звернемо увагу на системне повідомлення, що виводиться на екран для версії СКБД MongoDB 5.0.3. Це попередження, про те, що найближчим часом буде замінено клієнтську частину, за яку відповідає виконавчий файл mongo.exe та командна у консолі `mongo`, на прогресивнішу оболонку (MongoDB Shell) — `mongosh`. Очікується, що у наступних релізах MongoDB, клієнтська частина СКБД `mongosh` повністю замінить `mongo`.

Основні переваги `mongosh`:

– вища швидкість взаємодії з серверною частиною СКБД MongoDB;

– підсвічування різних синтаксичних конструкцій у командному рядку (;

– наявність контекстної довідки;

– чітка локалізація помилок.

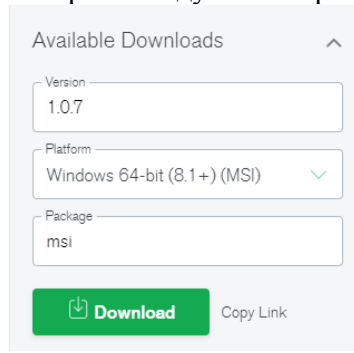
Клієнтська оболонка `mongosh`, на даному етапі, є автономним продуктом, що розроблений окремо від СКБД MongoDB, має відкритий код за ліцензією Apache 2 [7].

Отже, наразі, `mongosh` потрібно встановити як окрему складову СКБД MongoDB.

Для завантаження оновленої клієнтської оболонки рекомендується перейти у розділ Tools на сторінці завантаження СКБД MongoDB, де зібрано доступні додаткові інструменти роботи з даною СКБД і обрати пункт MongoDB Shell (або перейти за посиланням: https://www.mongodb.com/try/download/shell).

Необхідно у вікні завантаження вказати:

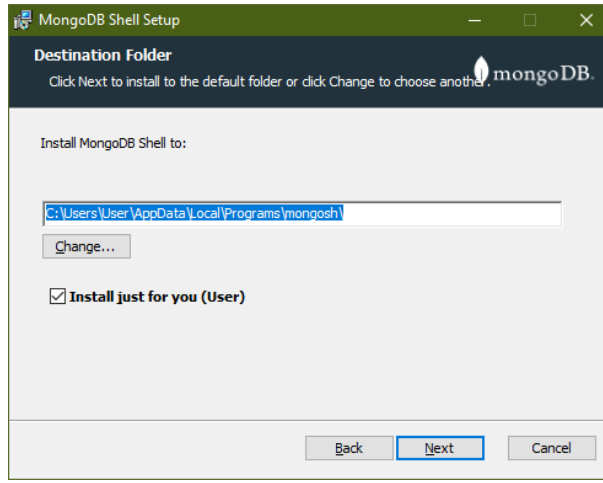
- у полі Version: доступну версію оболонки. На момент написання інструкції — це 1.0.7;
- у полі OS: обрати версію та розрядність ОС актуальну для вашої робочої станції;
- у полі Package: можна обрати форму завантаження: або MSI, або ZIP. Студентам рекомендується обрати варіант msi:



Після, варто натиснути на кнопку «Download».

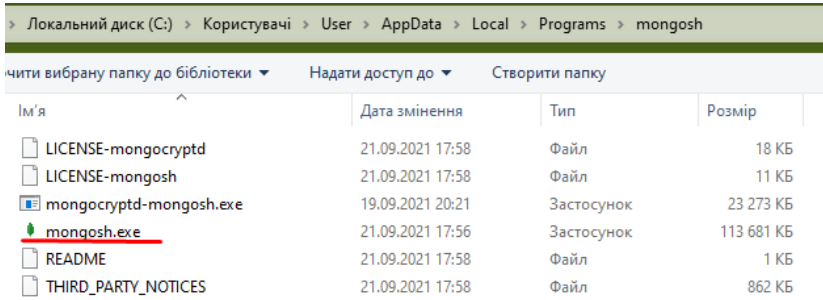
Установка MongoDB відбувається за допомогою майстра інсталяції — MongoDB Shell Setup Wizard:

Для того, щоб інсталювати оновлену клієнтську оболонку на робочу станцію важливо потрібно вказати директорію, куди буде встановлено `mongosh`. За замовчуванням, рекомендується директорія: C:\Users\User\AppData\Local\Programs\mongosh\. Також, майстра інсталяції рекомендує встановити оболонку тільки для поточного користувача на ПК (Install just for you (Назва поточного облікового запису)). Студентам рекомендується не змінювати рекомендовані параметри.



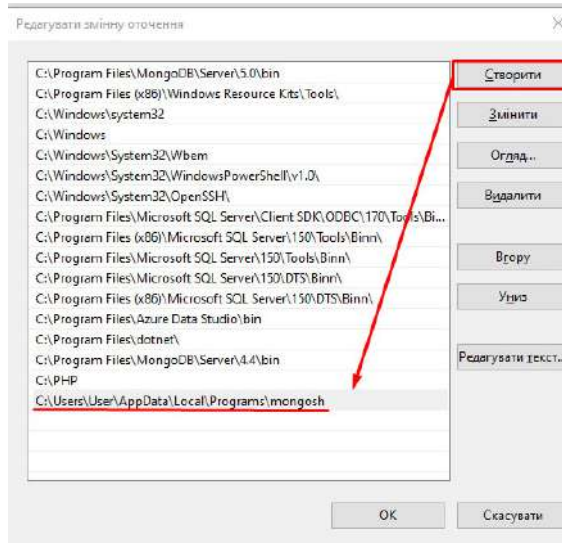
І натиснути кнопку Install, а, згодом — Finish.

Як зазначалось вище, виконавчі файли оновленої клієнтської частини на робочій станції інстальовані у директорії: C:\Users\User\AppData\Local\Programs\mongosh\. Виконавчий файл — mongosh.exe — оновлений додаток запуску клієнтської частини СКБД MongoDB:



Далі, аналогічно, як і зі встановленням СКБД MongoDB, потрібно налаштувати змінні середовища для `mongosh`.

Внесення змін необхідно здійснити у середовищі системних змінних:



У діалоговому вікні редагування змінних середовища необхідно натиснути кнопку «Створити» та вказати шлях, до теки із виконавчими файлами оновленої клієнтської оболонки MongoDB Shell. Фактично указати шлях до директорії «mongosh» (наприклад, C:\Users\User\AppData\Local\Programs\mongosh\), куди встановлено виконавчі файли `mongosh` на робочій станції.

Запустимо консоль або командну строку, для того, щоб переконатись, що оновлена MongoDB Shell інстальована на робочу станцію коректо.

Для перевірки коректності встановлення MongoDB, необхідно відкрити командний рядок:

```
mongosh -nodb
```

У відповідь на введену команду на консолі має з'явитись системна відповідь щодо запуску утілити `mongosh` та зазначення її поточної версії.

Для того, щоб припинити роботу із утилітою `mongosh` у командному рядку треба ввести команду:

```
quit()
```

Для того, щоб припинити роботу у консолі треба ввести команду:

```
exit
```

Перейдемо до завантаження графічного клієнта MongoDB — Compass (якщо він не був встановлений під час інсталяції СКБД MongoDB). Для цього варто перейти за посиланням: https://www.mongodb.com/download-center/compass?jmp=hero .

Принцип вибору версії для завантаження і сам процес, аналогічний до описаного вище при інсталяції СКБД MongoDB. При виборі версій пропонується кілька різних типів:

- Stable — стабільна версія MongoDB Compass, з усіма функціями і можливостями;
- Community Edition Stable — базова безкоштовна версія для розробки з MongoDB і включає в себе базові функції Compass. Спільне видання вільно доступне для всіх користувачів;
- Readonly Edition — версія обмежена лише операціями читання (можливості запису та видалення відсутні).
- Isolated Edition — версія підтримує роботу з локальною версію бази MongoDB, без мережевих підключень;
- Beta — тестова бета-версія графічного клієнта.

Для виконання лабораторних робіт варто завантажити версію типу Stable для відповідної ОС та її розрядності.

Перевірити коректність інсталювання MongoDB Compass можна шляхом запуску відповідного ярлика на робочому столі:

Детальнішу інформацію щодо інсталяції MongoDB можна знайти на офіційному сайті компанії розробника [4].

Для користувачів OSX / Linux корисною може бути відеоінструкція з налаштування MongoDB: https://youtu.be/-uo2Jhlj318.

Ознайомитись із матеріалами по MongoDB Compass можна на офіційному сайті компанії розробника: https://docs.mongodb.com/compass/current/.

2.4. Рекомендовані джерела до лабораторної роботи №1

1. What is MongoDB? / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://www.mongodb.com/what-is-mongodb
2. MongoDB: дані Вікіпедії [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/MongoDB
3. Он-лайн керівництво із MongoDB [Електронний ресурс]. URL: https://metanit.com/nosql/mongodb/
4. Install MongoDB / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/installation/
5. Install MongoDB Community Edition on Linux / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/administration/install-on-linux/
6. Install MongoDB Community Edition on macOS / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/
7. MongoDB Shell / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://www.mongodb.com/try/download/shell

2.5. Інструктивні та методичні вказівки для виконання лабораторної роботи №2 «Знайомство з форматом даних JSON»

Мета роботи: ознайомитись із форматом представлення даних JSON, зрозуміти специфіку представлення даних у форматі JSON, створювати власні об’єкти JSON.

Завдання роботи:

1. Ознайомитись із загальною інформацією про JSON [1-3].
2. Прочитати інструкцію до виконання лабораторної роботи, яка представлена нижче.

3. Засвоїти специфіку синтаксису представлення даних у форматі JSON.

4. Завантажити відкриті дані з доступного ресурсу у форматі *.json і проглянути їх у сервісі on-line перегляду JSON-об'єктів.

5. Створити опис об'єктів у форматі JSON відповідно до задачі обраної предметної області.

6. Перевірити правильність синтаксису створеного запису.

7. Дати відповіді на самостійні питання, поставлені в роботі.

8. Оформити звіт з лабораторної роботи.

Інструкція з виконання роботи

JSON (JavaScript Object Notation) — це загальний формат для представлення значень і об'єктів [1]. Його опис задокументовано в кількох стандартах [2], актуальний — RFC 7159 [3].

У стандарті RFC 7159 зазначено, що створено формат JSON для легкого обміну даними між програмними продуктами, інформаційними системами, веб-додатками незалежно від мови.

Початково JSON було створено для JavaScript, але згодом він набув популярності і наразі багато мов програмування мають бібліотеки, які можуть працювати з цим форматом. Таким чином, JSON легко використовувати для обміну даними, коли клієнт використовує JavaScript, а сервер написаний на Ruby / PHP / Java / будь-якій іншій мові програмування.

Про те, що файл збережено (створено, згенеровано) у форматі JSON можна пересвідчить якщо файл має розширення *.json.

Особливості JSON:

– формат представляє собою текстовий опис даних на основі синтаксису JavaScript;
– форма структурування — рядок;
– зрозумілий, але суворий синтаксис;
– компактне представлення даних.

JSON підтримує такі типи даних:

object	об'єкти	{ ... }
array	масиви	[...]
number	числа	цілі або дрібні (через кому)
string	рядки	" … "
	логічні значення	true / false
	нульові значення	null

JSON вимагає використання подвійних лапок " … ", які необхідно використовувати у парах «ключ / значення», де вони описані текстом.

JSON побудований на двох типових структурах даних:

1) колекція пар «ключ / значення»;

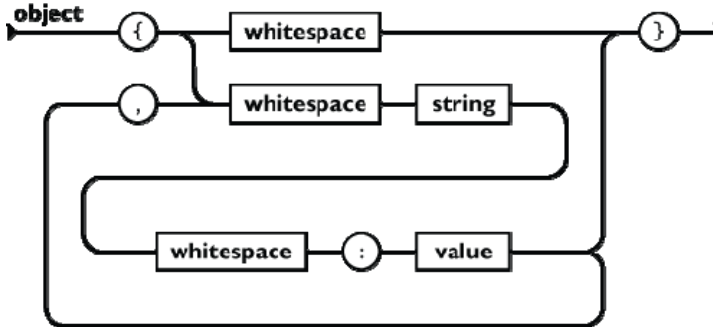
2) упорядкований перелік значень.

У нотації JSON зазначені структури даних мають вигляд:

– об’єкт — невпорядкований набір пар «ключ / значення». Об'єкт починається з фігурної дужки `{` і закінчується фігурною дужкою `}`. Кожна назва ключа супроводжується двокрапкою `:` , а пари «ключ / значення» розділяються комою `,` . Наприклад:

```
{ "name" : "John" }
```

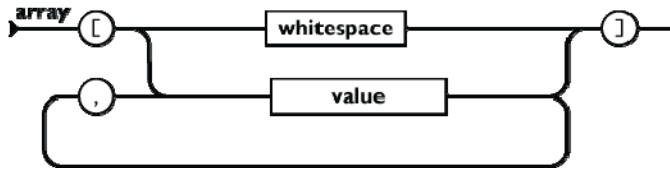
```
{ "name" : "John", "family name" : "Gorland" }
```



Джерело даних: [1]

– масив — впорядкований набір (колекція) значень. Масив починається з квадратної дужки `[` і закінчується такою ж дужкою `]` . Значення в середині масиву розділяються комами `,` . Наприклад:

```
{ "name" : [ "John", "Alice", "Donna" ] }
```



Джерело даних: [1]

У парах «ключ / значення», саме значення може бути представлене: *string* рядком (текстове пояснення), *number* числом, логічним значенням (*true, false*), нульовим значенням *null*, *object* об'єктом або *array* масивом.

Рядок — текстове поле, яке нічого не містити, або містити символи Unicode, значення представляється у подвійних лапках " … " . Наприклад:

```
{ "shopperName": "John Smith", "shopperEmail":
          "johnsmith@example.com" }
```

Число — числове поле, використовується десяткова система числення. При використанні числових значень подвійні лапки не використовують:

```
{ "name" : "John", "age" : 39 }
```

Розглянемо приклад представлення даних у форматі JSON для задачі «Облік замовлень продукції в інтернет-магазині»:

Звичайне представлення	Ієрархічне представлення
{ "orderID": 145, "shopperName": "Iryna Zinovyeva", "shopperEmail": "ira.zinovyeva@gmail.com", "contents": [{ "productID": 34, "product-Name": "InterectiveDollSurpriceLOL", "quantity": 2 }, { "productID": 56, "product-Name": "EcoHouseToyMaliby", "quantity": 1 }], "orderCompleted": true }	{ "orderID": 145, "shopperName": "Iryna Zinovyeva", "shopperE-mail": "ira.zinovyeva@gmail.com", "contents": [{ "productID": 34, "product-Name": "InterectiveDollSurpriceLOL", "quantity": 2 }, { "productID": 56, "product-Name": "EcoHouseToyMaliby", "quantity": 1 }], "orderCompleted": true }

Вище представлено два способи представлення даних у форматі JSON: звичайний — у рядку, ієрархічний — більш зручний для сприйняття. Спосіб представлення даних залежить від редактора, із яким працює розробник.

Розглянемо структуру синтаксису представлення даних для задачі «Облік замовлень продукції в інтернет-магазині»:

1. На початку і наприкінці використано фігурні дужки { … }, які дають зрозуміти, що це об’єкт.

2. Усередині об’єкта представлено кілька пар «ключ / значення». Розглянемо їх:

"orderID": 145 — поле з ключем orderID. Цьому полю відповідає значення 145. Можна його інтерпретувати як «Замовлення №145»;

"shopperName": "Iryna Zinovyeva" — поле з ключем shopperName і значенням Iryna Zinovyeva. Можна його інтерпретувати як «Ім’я замовника Iryna Zinovyeva»;

"shopperEmail": "ira.zinovyeva@gmail.com" — використано для збереження електронної адреси покупця;

"contents": [...] — поле з ключем content, значення якого масив. Можна його інтерпретувати як «Вміст кошика покупки». Масив contents містить два об'єкти, які відображають вміст кошика. Кожен об'єкт має три параметри (ключі): productID, productName, quantity;

"orderCompleted": true — поле з ім'ям orderCompleted, значення якого true. Можна його інтерпретувати як «Замовлення сформовано».

Загалом, представлення даних у форматі JSON може бути настільки складним, наскільки це необхідно, об'єкти і масиви можуть включати інші об'єкти і масиви, як це показано вище.

Наразі, формат JSON усе частіше використовується в якості стандарту представлення відкритих даних в Україні. Зокрема, такі державні та недержавні портали, як:

1) портал Відкритих даних України: https://data.gov.ua;
2) Єдиний державний реєстр декларацій: https://public.nazk.gov.ua;
3) Національний банк України: https://bank.gov.ua/ua/;
4) відкриті дані Київської міської державної адміністрації: https://data.kyivcity.gov.ua;

та інші представляють можливість користувачам генерувати за отримувати дані у форматі JSON.

Наприклад, на сайті Національного банку України публікуються відкриті набори даних щодо курсів валют, доходи і витра-

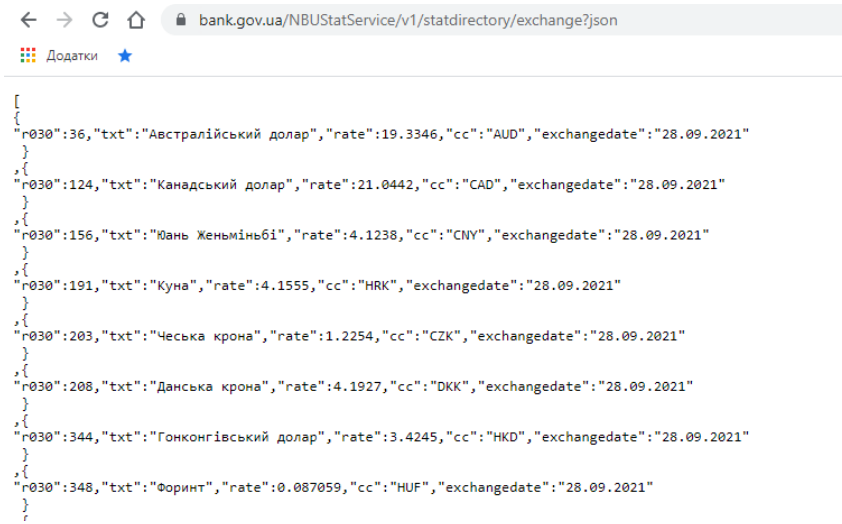
ти банків, макроекономічних показники та інші фінансові показники. За посиланням:

– https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json можна дізнатись актуальний курс валют на поточний день;

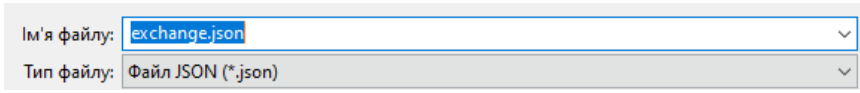
– https://bank.gov.ua/NBUStatService/v1/statdirectory/basindbank?date=20160101&period=m&json можна ознайомитись з основними показниками діяльності банків в Україні (помісячно);

– https://bank.gov.ua/NBUStatService/v1/statdirectory/inflation?period=m&date=201501&json дізнатись про індекси споживчих цін за різними товарними групами та ін.

Якщо перейти за представленим посиланням щодо актуального курсу валют, то у вікні браузера можна побачити таке:



Ці дані можна зберігати на робочу станцію за допомогою команди Зберегти як… Як видно, файл, що пропонується завантажити має формат JSON:



Для зручності роботи із JSON можна скористуватись сервісом on-line перегляду JSON об’єктів, який доступний за посиланням: https://codebeautify.org/. Даний ресурс може бути корисний для

роботи з описами в форматі JSON, оскільки тут підтримується функціонал:

– перегляду — JSON Viewer (https://codebeautify.org/jsonviewer);

– перевірки синтаксису — JSON Validator (https://codebeautify.org/jsonvalidator);

– редактор — JSON Editor (https://codebeautify.org/online-json-editor);

– конвертор з JSON в інші формати (XML, CVS, YAML, тощо) та навпаки.

Для перегляду даних НБУ щодо курсу валют, скористаємось JSON Viewer, для цього, перейдемо за вказаним посиланням: https://codebeautify.org/jsonviewer і натиснемо кнопку Load Url. Вставимо посилання на набор відкритих даних НБУ та натиснемо кнопку Load:

Вікно JSON Viewer розділено на три секції:

– з лівого боку — вікно з описом даних у JSON (включає код);

– посередині — функціональна панель;

– з правого боку — вікно представлення даних (без коду).

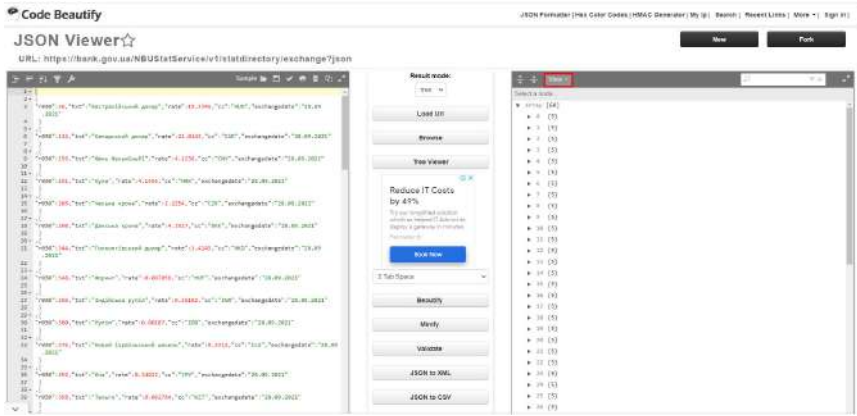
Завантаживши таким чином дані, можна одночасно бачити код об’єкта JSON та його зрозумілу інтерпретацію.

Для того, щоб завантажити дані у форматі *.json, можна скористатись кнопкою Browse на функціональній панелі JSON Viewer.

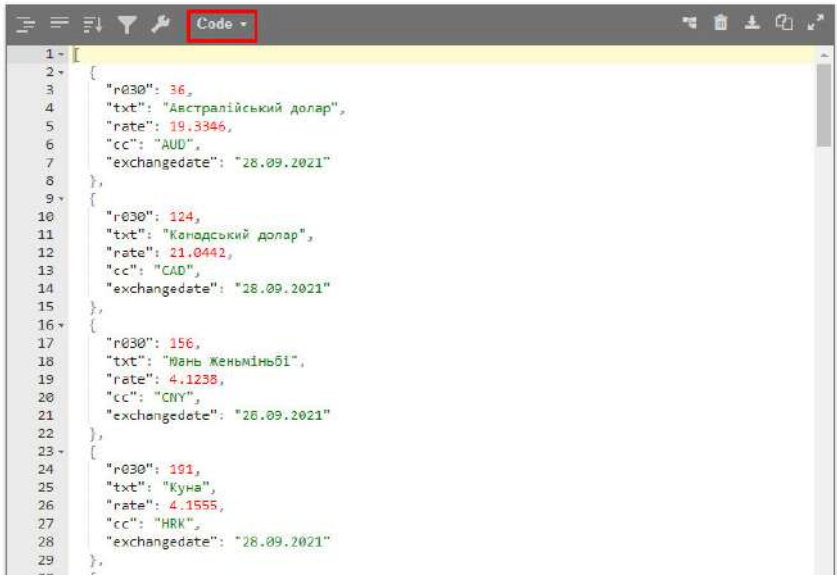
Закладка «Viewer» передбачає роботу із форматом JSON у ієрархічному простішому для сприйняття представленні.

Завантажені дані матимуть вигляд:

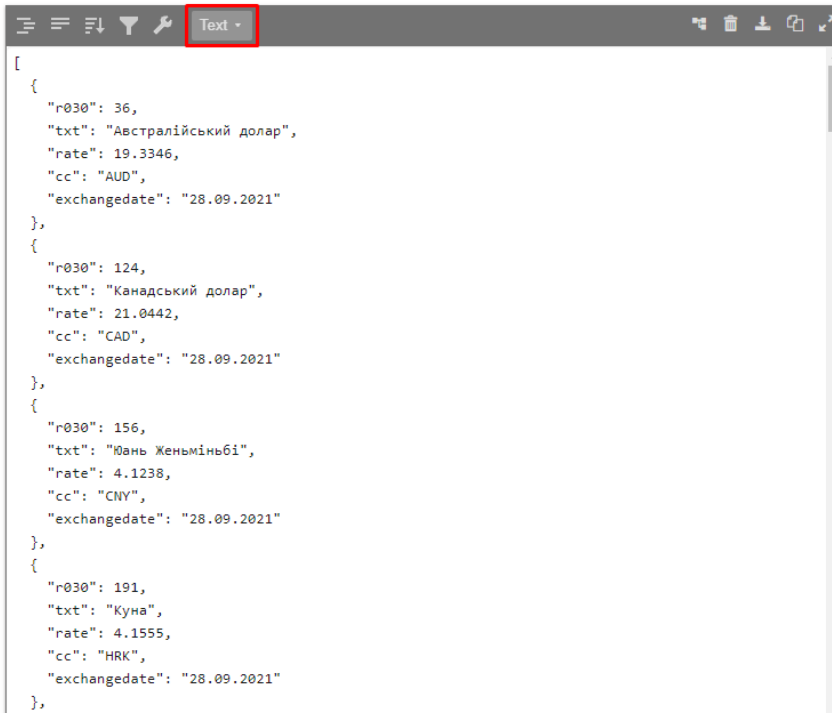
– за замовченням:



– у ієрархічному представленні. У вікні представлення даних у спадаючому списку замість пункту «View» обрати пункт «Code»:



– у вигляді текстового рядка. У вікні представлення даних у спадаючому списку замість пункту «Code» обрати пункт «Text»:



Розберемо найпоширеніші помилки синтаксису під час роботи із JSON:

`{ name: "John", //` Помилка: ім'я ключа без лапок

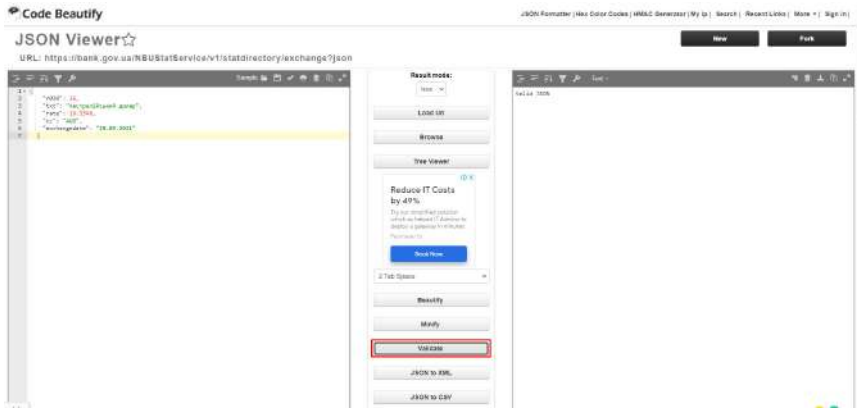
`"surname": 'Smith', //` Помилка: одинарні лапки в значенні, замість подвійних

`'isAdmin': false //` Помилка: одинарні лапки в ключі, замість подвійних

`"friends": [0,1,2,3] //` Відсутні помилки `}`

Для перевірки правильності синтаксису формату JSON доцільно використовувати онлайн-валідатори, наприклад: https://codebeautify.org/jsonvalidator.

Для того, щоб перевірити дані у форматі *.json, можна скористатись кнопкою Validat на функціональній панелі JSON Viewer:



Отже, JSON — це текстовий тип формату даних, який має власний незалежний стандарт і бібліотеки для більшості мов програмування. JSON підтримує прості об'єкти, масиви, рядки, числа, логічні значення і категорію «null». Синтаксис JSON суворий не тому, що його розробники бажали ускладнити роботу іншим, а тому, що дозволяє легко, надійно і швидко реалізовувати алгоритм кодування і читання, зрозумілий не тільки комп'ютерній техніці, але й людині.

2.6. Навчальні завдання

1. По темі курсового проекту з дисципліни «Організація баз та сховищ даних», який Ви виконували на 2 курсі, запроектуйте документарну БД, яка б складалась із 2-х колекцій.

1.1. У кожній колекції опишіть у JSON форматі по 5 документів.

1.2. Один з документів має містити масив.

1.3. Один з документів повинен містити вкладений документ.

2. Перевірити правильність синтаксису через онлайн-валідатор;

3. Результати показати викладачу.

2.7. Питання для самоконтролю

1. Яку модель бази даних NoSQL підтримує СКБД MongoDB?

2. Які є версії СКБД MongoDB і в якому операційному середовищі СКБД може працювати?

3. Чим відрізняються між собою Community version та Enterprise version?

4. Яке призначення виконавчих файлів: mongo.exe та mongod.exe?

5. За допомогою якої консольної команди можна перевірити коректність встановлення MongoDB на робочій станції?

6. Яким чином можна одночасно інсталювати СКБД MongoDB і графічний клієнт MongoDB — Compass?

7. Які функції при роботі з БД MongoDB підтримує Compass?

8. Який синтаксис представлення даних JSON форматі.

9. Поняття масиву (array) їх види та синтаксис опису в JSON-форматі.

2.8. Рекомендовані джерела до лабораторної роботи №2

1. Офіційна сторінка JSON [Електронний ресурс]. URL: http://www.json.org

2. Стандарт RFC 4627 [Електронний ресурс]. URL: https://tools.ietf.org/html/rfc4627

3. Стандарт RFC 7159 [Електронний ресурс]. URL: https://datatracker.ietf.org/doc/html/rfc7159

Тема 3.

МОДЕЛЮВАННЯ ДАНИХ В СЕРЕДОВИЩІ СКБД MongoDB

3.1. Перелік знань і вмінь

При вивченні даної теми студенти знайомляться з роботою у середовищі СКБД MongoDB за архітектурою «клієнт-сервер» та отримають базові практичні навички роботи із нереляційною базою даних через консоль, зокрема, щодо створення: бази даних, колекцій та документів; відпрацюють їх на прикладі конкретної предметної області.

*Вивчивши матеріал теми, ви будете **знати**:*

- команди для запуску СКБД MongoDB;
- команди створення бази даних і колекцій;
- поняття обмеженої колекції;
- команди перевірки створення бази даних і колекцій;
- основні елементи для опису документи у форматі JSON.

*Вивчивши матеріал теми, ви будете **вміти**:*

- запускати на робочій станції серверну та клієнтську частину СКБД MongoDB;
- створювати та вилучати базу даних;
- створювати та вилучати колекції;
- виконувати перевірку створення бази даних і колекцій;
- описувати документи колекції у форматі JSON;
- створювати, вилучати та вносити зміни до документів.

3.2. Термінологічний словник

БД MongoDB — документно-орієнтована маштабована база даних, яка представляє собою набір колекцій, що, в свою чергу, складаються з певного набору документів. Нову БД MongoDB створює команда:

```
use ім'я_БД
```

Якщо така БД уже існує, то за допомогою зазначеної команди виконується її підключення. Ім'я бази даних обмежено 64 байтами і не може вміщувати символів: /, \, ., ", *, <, >, :, |, ?, $. Також у назві не можна використовувати зарезервовані імена: local, admin, config.

Вилучення БД виконується командою:

```
db.dropDatabase()
```

Перегляд створених і збережених на робочій станції БД виконує команда:

```
show dbs
```

Документ в MongoDB — це ідентифікований первинним ключем набір полів довільної структури. Документ представляється набором пар «ключ / значення». Документ обов'язково повинен мати первинний ключ, який має бути унікальним у межах колекції і записується у полі _id. Первинний ключ може задаватися розробником чи генеруватися системою, як ідентифікатор об'єкта. Документ може мати вкладені документи чи масиви. Розмір документа не може перевищувати 16 МБ. Це обмеження гарантує, що жоден документ не зможе використовувати надмірну кількість оперативної пам'яті та не спричинятиме зростання трафіку при передачі. Для зберігання документів більше ніж 16 МБ у MongoDB використовується GridFS. Документ є еквівалентом рядка (кортежу) таблиць реляційної бази даних. Додавати документи в колекцію можна за допомогою команд:

- `insertOne()` — добавляє один документ;
- `insertMany()` — добавляє кілька документів;
- `insert()` — можна добавляти як один, так и кілька документів.

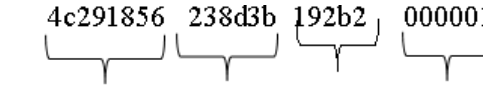
Основний синтаксис команди вставки документа в колекцію:

```
db.COLLECTION_NAME.insert(document)
```

Наприклад:

```
db.car.insert({ "name": "Honda", "model": "CR - V", "color": ["black", "white", "silwer"], "engine": "2,4", "price": 35000})
```

Ідентифікатор документа — це унікальний 12-байтовий, 16-значний бінарний ідентифікатор, який вміщує 4-х байтну мітку часу типу timestamp, 3-х байтний ідентифікатор комп'ютера, 2-байтний ідентифікатор процесу і 3-х байтне поле для лічильника. Це поле автоматично створює MongoDB і воно є первинним ключем документ.



Первинний ключ може задаватися розробником бази даних. При цьому слід пам'ятати про забезпечення його унікальності.

Колекція в MongoDB — це сукупність структурно чи семантично подібних документів. Колекція може вміщувати документи різної структури з різними типами значень та назв полів. Спільним для об'єднання документів у колекцію може бути одна предметна область. Колекція ідентифікується ім'ям, яке не повинно бути більшим ніж 128 різних алфавітно-цифрових символів та знаку підкреслювання. У той же час, ім'я колекції не може починатися з префікса system, так як він є зарезервованим і не може містити знак долара $. Колекція є еквівалентом таблиці реляційної бази даних. Колекцію створює команда:

```
db.createCollection("ім'я_колекції")
```

Переіменування колекції виконує команда:

```
db.collection.renameCollection("нове_ ім'я")
```

Вилучення колекції виконується командою:

```
db.collection.drop()
```

Переглянути перелік колекцій, створених у БД можна командою:

```
show collections
```

Метод insert — відбувається вставка документа чи масиву в колекцію. Синтаксис метода:

```
db.< "ім'я колекції">.insert(<"документ"> чи <"масив">)
```

Об'єкт (документ) у форматі JSON — це неупорядкована множина пар «ім'я/значення», взятих у фігурні дужки { }. Між іменем і значенням стоїть символ ':' (двокрапка), а пари «ім'я/значення» розділяються комою. Приклад документа, що містить характеристики телефону «Nokia Lumia 920» в JSON, буде має вид:

```
{"Name": "NL920",
 "Brand": "Nokia",
 "Model": "Lumia 920",
 "OSFamily": "Windows",
 "OSVersion": "8"}
```

Обмежена колекція (capped collection) — колекція, яка має фіксований розмір за обсягом пам'яті в байтах, який вказується в параметрі **size** та за кількістю документів — параметр **max.** Обмежена колекція гарантує, що документи будуть розташованими в тому порядку, в якому вони добавлялись в колекцію. Така ко-

лекція має фіксований розмір. Коли в колекції уже немає місця, старіші документи вилучаються, і в кінець дозаписуються нові. Параметр **size** має пріоритет перед параметром **max.** Якщо колекція досягла максимально можливого розміру пам'яті при не перевищеному ліміті кількості документів, то СКБД MongoDB все рівно вилучить певні документи, щоб задовільнити вимоги **size** параметра. Команда створення обмеженої колекції:

```
db.createCollection("Ім'я колекції",  {capped: true, size: максимальна пам'ять байт, max: максимальна кількість документів})
```

Для перевірки чи колекція є обмеженою чи ні використовується команда isCapped:

```
db.("ім'я_колекції").isCapped()
```

При створенні обмежених колекцій слід враховувати таке:

- з обмеженої колекції не можна видалити документи. Можливо лише видалити усю колекцію;
- у процесі додавання нового документу не потрібно шукати місце для його розташування на диску. Він включається в кінець колекції, ця операція в обмежених колекціях виконується дуже швидко;
- під час зчитування документів MongoDB повертає документи в тому ж порядку, що і на диску. Це робить також операцію читання дуже швидкою.

Формат BSON (Binary JavaScript Object Notation) — бінарна версія JSON для фізичного зберігання даних у двійковому в формате.

3.3. Інструктивні та методичні вказівки для виконання лабораторної роботи №3 «Робота в середовищі MongoDB зі створення бази даних, колекцій та документів»

Мета лабораторної роботи: ознайомлення з роботою MongoDB у консолі. Отримання навичок створення бази даних, колекцій і документів, їх вилучення та перейменування.

Завдання роботи

1. Ознайомитись із поняттями база даних, колекція і документ у середовищі СКБД MongoDB за матеріалами лекції та джерелами [1, 2, 3].
2. Прочитати інструкцію, яка представлена нижче.
3. Ознайомитись із базовими командами роботи з СКБД MongoDB через консоль (командний рядок).

4. Створити БД і дві колекції, кожна з яких буде вміщувати 5-10 документів.
5. Відпрацювати моделювання зв’язків між документами однієї колекції.
6. Відповісти на питання, поставлені в роботі.
7. Оформити звіт з лабораторної роботи.

Інструкція з виконання роботи

Робота у середовищі СКБД MongoDB відбувається за кількома схемами:

– через архітектуру «клієнт-сервер» (у якості тонкого клієнта виступає: екземпляр mongo, запущений через консоль; графічний клієнт Compass MongoDB);

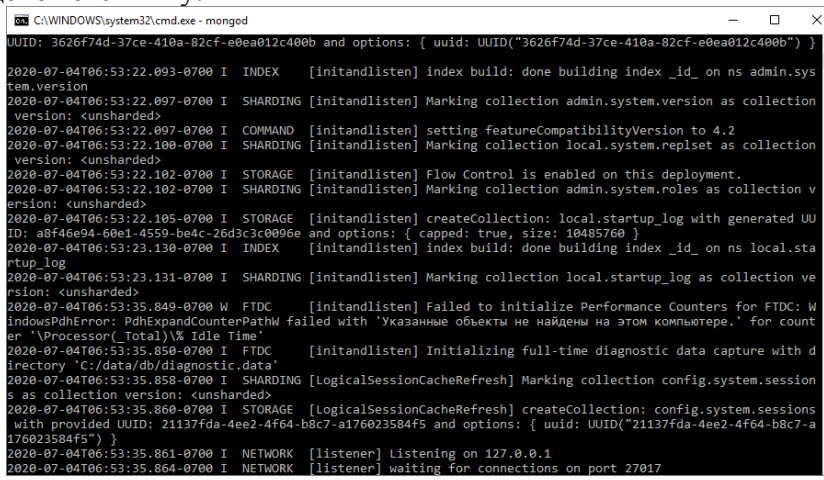
– через роботу у хмарі (за допомогою підключення до MongoDB Atlas Cluster).

У межах цього курсу ознайомимось, як відбувається робота у середовищі СКБД MongoDB через архітектуру «клієнт-сервер».

Робота із СКБД MongoDB починається із виклику командного рядка та введенням першої команди:

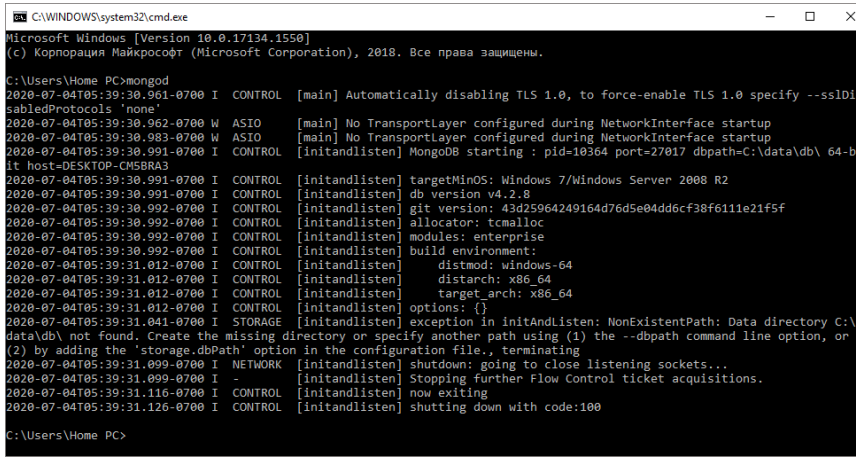
```
mongod
```

У відповідь система повинна видати інформацію про завантаження серверної частини MongoDB. На екрані консолі буде виведено текст типу:

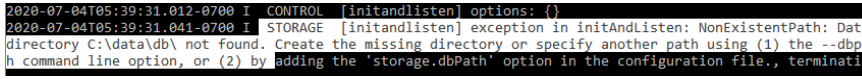


Означене свідчить, що серверна частина запущена і чекає на з'єднання із клієнтом на localhost (IP: 127.0.0.1, порт: 27017) за замовчуванням.

В іншому випадку, на екран буде виведено інформацію про помилку. Наприклад:



Як видно із даних консолі, MongoDB було запущено, але потім робота була припинена, про що свідчить останній рядок: «shutting down with code:100». Причину помилки запуску можна також побачити у вікні консолі:



Тобто, для запуску серверної частини MongoDB потрібен каталог db у директорії C:\data\db\. Його можна створити кількома шляхами:

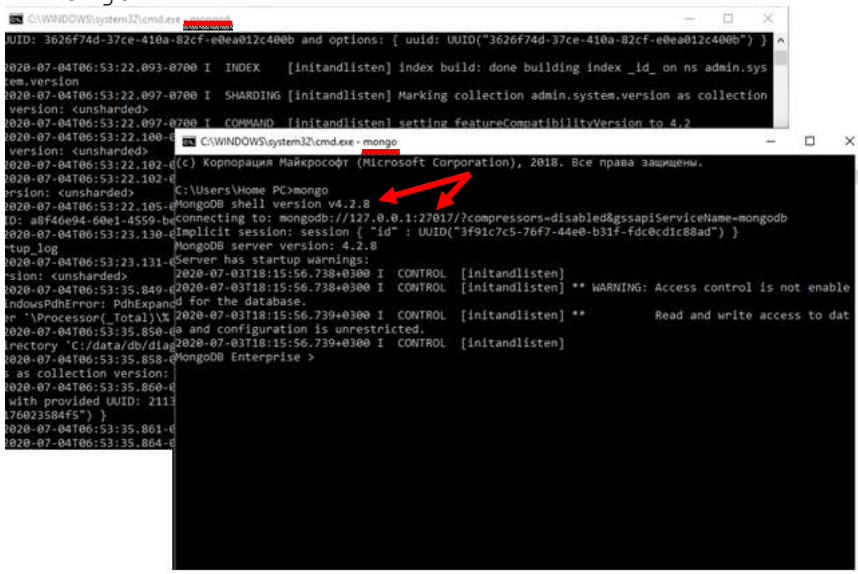
1) у вікні Провідника за вказаною директорію створити папку db;

2) у командному рядку виконати команду:

```
mkdir C:\data\db
```

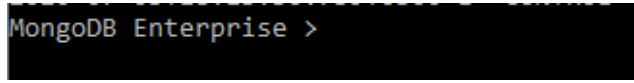
Після того, як серверна частина успішно запущена, потрібно відкрити нове вікно консолі (при цьому не закриваючи попереднє вікно) та запустити у ньому клієнтську частину:

```
mongo
```



Натиснувши Enter, отримаємо інформацію про версію MongoDB та підключення до серверної частини через IP: 127.0.0.1, порт: 27017.

Про роботу в середовищі MongoDB свідчить зміна поточного розміщення:



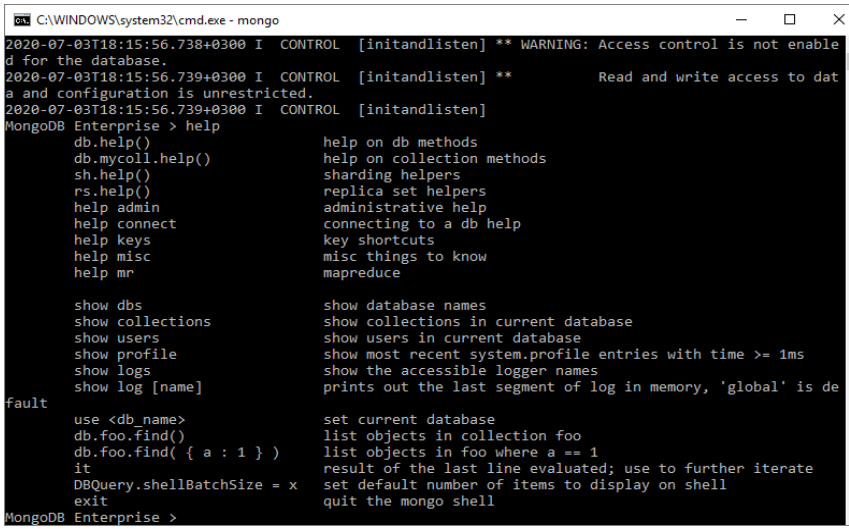
Таким чином, запущено на робочій станції серверну та клієнтську частину СКБД MongoDB, що дає змогу переходити до роботи безпосередньо у цьому середовищі.

Тепер ознайомимось із базовими командами роботи із СКБД MongoDB у консолі.

Подальша робота буде відбуватись у вікні консолі із запущеною клієнтською частиною. При цьому, вікно консолі із серверною частиною (mondgod) не потрібно закривати.

Для перевірки можливих команд доступних команд слід ввести команду

```
help
```



Розберемо ці команди:

`db.help()`	довідка щодо методів роботи із БД;
`db.mycoll.help()`	довідка щодо методів роботи із колекціями БД;
`sh.help()`	довідка щодо роботи із колекціями, що збережені на кількох серверах (розподілено) — шардинг;
`rs.help()`	довідка щодо синхронізації колекцій, що збережені на кількох серверах (розподілено) — реплікація;
`help admin`	довідка з адміністрування БД;
`help connect`	довідка щодо підключення клієнта до сервера;
`help keys`	довідка щодо гарячих клавіш;
`help misc`	інші довідкові дані;
`help mr`	довідка щодо фреймворка Map-Reduce;
`show dbs`	виводить на екран консолі перелік існуючих (створених) БД;
`show collections`	виводить на екран консолі існуючі колекції у поточній БД;
`show users`	виводить на екран консолі список користувачів, підключених до поточної БД;

`show profile`	виводить на екран консолі останні записи system.profile entries із інтервалом часу >= 1ms;
`show logs`	виводить на екран консолі доступні журнали подій;
`show log [name]`	виводить останній сегмент із журналу подій;
`use <db_name>`	підключає існуючу або створює нову БД;
`db.foo.find()`	виводить перелік об'єктів у колекції «foo»;
`db.foo.find( { a : 1 })`	виводить перелік об'єктів у колекції «foo», де a = 1;
`it`	результат оцінки останнього рядка; використовувати для подальшого перебору;
`DBQuery.shellBatchSize = x`	задає певну кількість елементів для відображення
`exit`	вийти із MongoDB

Для прикладу, напишемо команду для виведення списку наявних баз даних:

```
show dbs:
```

```
MongoDB Enterprise > show dbs
admin   0.032GB
config  0.000GB
local   0.000GB
ships   0.031GB
MongoDB Enterprise >
```

У вікні зявилось повідомлення, що на сервері збережено 4 БД, із яких: admin — містить дані (0,032GB); ships — містить дані (0,031GB); config — пуста БД; local — пуста БД.

У випадку, коли на сервері MongoDB не має створених користувачем БД, вікно консолі буде мати інший вигляд:

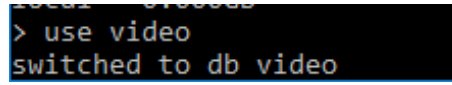
```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Виведені на екран БД створені MongoDB автоматично.

Для того, щоб переключитися на іншу базу даних, необхідно виконати команду:

`use`

та написати назву необхідної БД, наприклад «use video». Якщо такої БД не було серед списку можливих, то вона створиться сама автоматично:



Наповнимо БД «Video» колекціями. Наприклад, пропонуємо створити колекцію «Cartoons», де будуть зберігатись документи про мультфільми.

Прототип команди такий:

```
db.createCollection(<name>, { capped: <boolean>,
autoIndexId: <boolean>,
size: <number>,
max: <number>,
storageEngine: <document>,
validator: <document>,
validationLevel: <string>,
validationAction: <string>,
indexOptionDefaults: <document>,
viewOn: <string>,
pipeline: <pipeline>,
collation: <document>,
writeConcern: <document>} )
```

Більше про значення кожного із полів команди можна почитати тут [6].

У більшості випадків дана команда використовується у СКБД MongoDB для створення обмеженої колекції.

Наприклад, створимо колекцію «Cartoons» із такими параметрами:

capped: true — означає, що колекція створюється з обмеженнями (в іншому випадку, треба вказувати замість «true», «false» чи взагалі це поле пропустити);

size: 5242880 — визначає максимальне обмеження розміру колекції у байтах. Після досягнення максимального розміру колекції СКБД MongoDB видаляє старі документи, щоб звільнити місце для нових документів;

max: 5000 — визначає максимальну кількість документів у колекції. Обмеження розміру (поле «size») має перевагу над цим обмеженням (поле «max»). Тобто, якщо розмір колекції переви-

щує 5242880 байт (або 5 МБ), тоді додаткові документи створюються тільки через видалення старих, навіть, якщо їх кількість є меншою за 5000.

Отже, для створення колекцію «cartoons» із заданими налаштуваннями необхідно виконати команду:

```
db.createCollection("cartoons", {capped: true, size: 5242880, max: 5000})
```

```
MongoDB Enterprise > use video
switched to db video
MongoDB Enterprise > db.createCollection("cartoons", { capped: true, size: 5242880, max: 5000} )
{ "ok" : 1 }
MongoDB Enterprise >
```

Система зреагувала написом `{ "ok" : 1 }`, що означає, що у підключеній БД створено одну колекцію.

Можемо перевірити факт створення колекції через команду:

```
show collections
```

```
MongoDB Enterprise > show collections
cartoons
MongoDB Enterprise >
```

Тепер створимо у цій колекції документ (запис). Це можна зробити за допомогою команд `insertOne()`, `insertMany()` або `insert()`. Наприклад:

```
db.cartoons.insertOne({"title": "Tom and Jerry", "year": 1940, "genre": "comedy", "country": "USA", type: "serial"})
```

```
cartoons
MongoDB Enterprise > db.cartoons.insertOne({"title": "Tom and Jerry", "year": 1940, "genre": "comedy", "coutry": "USA",
"type": "serial"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5c5dd586ca95bdcfd33eb530")
```

Для бази даних «video» до колекції «cartoons» додано документ з записом про мультфільм «Том і Джері».

Запис «acknowledged»: «true» означає, що запит було успішно виконано, а поле «insertedId» показує який унікальний ідентифікатор було присвоєно даному запису.

Кожен запис в базі даних повинен мати поле «_id», яке має унікальне значення. Якщо при створенні нового запису поле «_id» не було вказано, то MongoDB автоматично присвоїть унікальний номер.

За допомогою команди «insertOne» можна також створювати нові колекції. Це можна зробити за допомогою запиту:

```
db.movies.insertOne({"title": "Tom and Jerry", "year": 1940, "genre": "comedy", "country": "USA", type: "serial"})
```

Така команда буде ініціювати створення нової колекції під назвою «movies» та внесення до неї документу з записом про мультфільм «Том і Джері».

Перевіримо:

```
show collections
```

```
MongoDB Enterprise > show collections
cartoons
movies
MongoDB Enterprise >
```

Тепер у ДБ «video» створено дві колекції: «cartoons», «movies». Обидві колекції містять оди і той самий документ із записом про мультфільм «Том і Джері».

Знайдемо документи, що створені у колекції «movies». Скористаємось командою:

```
db.movies.find()
```

```
MongoDB Enterprise > db.movies.find()
{ "_id" : ObjectId("5c5dd782ca95bdcfd33eb531"), "title" : "Tom and Jerry", "year" : 1940, "genre" : "comedy", "coutry" :
 "USA", "type" : "serial" }
MongoDB Enterprise >
```

Для того, щоб було зручніше переглядати записи (записи подавались структуровано), можна до попередньої команди додати «pretty()»:

```
db.movies.find().pretty()
```

```
MongoDB Enterprise > db.movies.find().pretty()
{
        "_id" : ObjectId("5c5dd782ca95bdcfd33eb531"),
        "title" : "Tom and Jerry",
        "year" : 1940,
        "genre" : "comedy",
        "coutry" : "USA",
        "type" : "serial"
}
MongoDB Enterprise >
```

Звернемо увагу, що окрім інформації, яку ми вводили, з'явилось ще одне поле «_id», яке має значення виду ObjectId («унікальний_номер»).

За допомогою запиту «find» можна знаходити не тільки колекції, але і поля у них. Наприклад, знайдемо у колекції «cartoons» БД «video» запис із назвою «Moana»:

```
db.cartoons.find({"title": "Moana"}).pretty()
```

```
MongoDB Enterprise > db.cartoons.find({ "title": "Tom and Jerry" }).pretty()
{
        "_id" : ObjectId("5c5dd586ca95bdcfd33eb530"),
        "title" : "Tom and Jerry",
        "year" : 1940,
        "genre" : "comedy",
        "coutry" : "USA",
        "type" : "serial"
}
MongoDB Enterprise > db.cartoons.find({ "title": "Moana" }).pretty()
MongoDB Enterprise >
```

СКБД MongoDB команду виконала, але не вивела на екран цей документ, бо його в колекції «cartoons» не існує.

Припустимо, що документ про мультфільм «Том і Джері» у колекції «cartoons» буде додано випадково і його треба видалити.

```
db.cartoons.remove({"title": "Tom and Jerry"})
```

```
MongoDB Enterprise > db.cartoons.remove( { "title": "Tom and Jerry"} )
WriteResult({
        "nRemoved" : 0,
        "writeError" : {
                "code" : 20,
                "errmsg" : "cannot remove from a capped collection: video.cartoons"
        }
})
MongoDB Enterprise >
```

Звернемо увагу, що СКБД MongoDB команду виконала, але нічого не видалила (що видно із поля "nRemoved" : 0), оскільки колекція «Cartoons» створена як обмежена.

Якщо ж виконати ту саму команду по відношенню до колекції «movies», то документ буде видалений, оскільки дана колекція не є обмеженою:

```
db.movies.remove({"title": "Tom and Jerry"})
```

```
MongoDB Enterprise > db.movies.remove({"title": "Tom and Jerry"})
WriteResult({ "nRemoved" : 1 })
MongoDB Enterprise >
```

Перевірити наявність обмежень у колекції можна за допомогою команди:

```
db.capped("ім'я_колекції").isCapped()
```

Для колекції «cartoons» команда буде мати вигляд:

```
db.cartoons.isCapped()
```

```
MongoDB Enterprise > db.cartoons.isCapped()
true
MongoDB Enterprise >
```

Переіменування колекції виконує команда:

```
db.collection.renameCollection("нове_ ім'я")
```

Наприклад, потрібно, щоб назви колекцій починались із великої літери. Так, щоб змінити назву колекції «movies» виконаємо команду:

```
db.movies.renameCollection("Movies")
```

```
MongoDB Enterprise > db.movies.renameCollection("Movies")
{ "ok" : 1 }
MongoDB Enterprise >
```

Оскільки у колекції «Movies» немає жодного документу, видалимо усю колекцію із БД, для цього скористаємось командою:

```
db.Movies.drop()
```

```
MongoDB Enterprise > db.Movies.drop()
true
MongoDB Enterprise >
```

У разі необхідності видалення усієї БД, можна використати команду, яка видаляє підключену БД:

```
db.dropDatabase()
```

```
MongoDB Enterprise > db.dropDatabase()
{ "dropped" : "video", "ok" : 1 }
MongoDB Enterprise >
```

3.4. Навчальні завдання

У додатку 1 вибрати тему для виконання циклу лабораторних робіт у середовищі СКБД MongoDB. Також можна залишити тему курсового проекту з дисципліни «Організація баз та сховищ даних», який виконувався на 2 курсі. По обраній темі виконайте в середовищі СКБД MongoDB такі завдання:

1. Створення та вилучення БД.
2. Створення, переіменування та вилучення колекцій.
3. Створення та внесення змін до документів з використанням результатів лабораторної роботи №2.
4. Створити вкладені документи, що моделюють зв’язок 1:1 та 1:Б.
5. Доповніть колекції документами, що створені при виконанні лабораторної роботи №2.

3.5. Питання для самоконтролю

1. Що таке у СКБД MongoDB «колекція»?
2. У чому суть «обмеженої колекції»?

3. Поясність, що відбудеться при виконанні у консолі такої команди: db.createCollection("profile", {capped: true, size: 9500, max: 150}).

4. Який синтаксис команди, для відображення усіх створених колекції у межах поточної БД?

5. Для яких цілей можна використовують команду db.collection.isCapped()?

6. У чому різниця між командами: db.cartoons.insertOne({“title”: “Tom and Jerry”, “year”: 1940}) та db.movies.insertOne({“title”: “Tom and Jerry”, “year”: 1940, type: “serial” }) ?

7. На якому порті за замовчуванням працює СКБД MongoDB на localhost?

8. Яка різниця між командами: insertOne() та insert()?

9. Створення нової обмеженої колекції за допомогою команди db.createCollection(), передбачає пріоритет поля size чи max?

10. За допомогою якої із команд (db.collection.remove() чи db.collection.drop()) можна видалити колекцію. Чому?

3.6. Рекомендовані джерела до лабораторної роботи №3

1. What is MongoDB? / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://www.mongodb.com/what-is-mongodb

2. MongoDB: дані Вікіпедії [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/MongoDB

3. Он-лайн керівництво із MongoDB [Електронний ресурс]. URL: https://metanit.com/nosql/mongodb/

4. Install MongoDB Community Edition on Linux / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/administration/install-on-linux/

5. Install MongoDB Community Edition on macOS / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/

6. db.createCollection() / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/reference/method/db.createCollection/

Тема 4.

РОБОТА З ДОКУМЕНТАМИ БАЗИ ДАНИХ В СЕРЕДОВИЩІ СКБД MongoDB

4.1. Перелік знань і умінь

При вивченні даної теми студенти знайомляться з процесом роботи з документами та колекціями у середовищі СКБД MongoDB на основі концепції CRUD, отримають базові практичні навички створення запитів. Відпрацюють створення запитів до документів з командного рядка та з використанням графічного інтерфейсу Compass.

Вивчивши матеріал теми, ви будете ***знати****:*

- команди вибору документів з бази даних СКБД MongoDB;
- команди створення параметризованих запитів;
- використання логічних операторів у запитах;
- команди управління вибіркою документів з БД;
- команди сортування та групування документів БД.

Вивчивши матеріал теми, ви будете ***вміти****:*

- виконувати відбір документів з бази даних СКБД MongoDB;
- створювати параметризовані запити та проекції;
- управляти вибіркою документів з БД;
- сортувати та агрегувати документи БД.

4.2. Термінологічний словник

Запити агрегування — процес маніпуляції із даними БД, який передбачає групування значень документів колекції з метою виконання певних розрахунків. Аналогом агрегування в SQL Server є команда `group by`.

Запит вибірки — процес маніпуляції із даними БД, який виводить документи вказаної колекції бази даних MongoDB. Якщо документи в БД відсутні — пусту колекцію. Аналогом команди вибірки в SQL Server є `SELECT`.

Операції CRUD — це мнемонічна абревіатура, яка характеризує основні операції з даними: *Create* — створення, *Read* — читання, *Update* — оновлення, *Delete* — вилучення. MongoDB підтримує CRUD операції.

Параметризовані запити — запити з параметрами, які задають умови вибірки, накладаючи певні значення чи обмеження на поля, що вибираються з документів. Параметризовані запити можуть містити кілька умов вибірки, об'єднуючи їх між собою логічними операторами: `$and`, `$or`, `$not`, `$nor`.

Проекція — запит-вибірки з документа не всіх, а лише певних, необхідних значень окремих полів.

Сортування — операція логічного рівня, яка дозволяє впорядковувати вихідні дані запиту по зростанню чи спаданню вказаного поля. Функція сортування `sort` у MongoDB багато в чому подібна до оператора `ORDER BY` у SQL.

Управління вибіркою — можливість регулювання кількості документів, які необхідно видавати в результаті запиту, починаючи з першого документа чи пропускаючи певну їх кількість. MongoDB підтримує управління вибіркою.

MongoDB Compass — графічний клієнт (коннектор) для роботи з СКБД MongoDB, надає можливості роботи з БД, орієнтований на візуалізацію даних і наочність. Може працювати в ОС Linux, Max і Windows.

4.3. Інструктивні та методичні вказівки для виконання лабораторної роботи №4 «Робота з документами в середовищі MongoDB»

***Мета роботи*:** створення запитів до документів з командного рядка в середовищі СКБД MongoDB.

Завдання роботи

1. Ознайомитись із основними видами запитів в середовищі СКБД MongoDB, які викладені в лекційному матеріалі, а також у джерелах [1, 2].
2. Прочитати інструкцію зі створення запитів до бази даних, яка наводиться нижче.
3. Виконати основні типи запитів, згідно вказаних завдань.
4. Відповісти на питання, поставлені в роботі.
5. Оформити звіт з лабораторної роботи.

Інструкція з виконання роботи

Реалізація запитів виконується на основі колекцій, які були створені при виконанні лабораторної роботи №2 і 3, де потрібно було створити 2 колекції, та наповнити їх відповідними документами, що характеризують вибрану предметну область.

1. Запити вибірки документів з бази даних

Вибір усіх документів колекції виконує функція `find()`.

Синтаксис:

`db.collection.find()` – для відображення у неструктурованому вигляді;

`db.collection.find().pretty()` – для відображення у структурованому вигляді

Для того, щоб знайти усі документи в БД «Nobel», виконаємо команду:

```
db.nobel_prize.find().pretty()
```

Результатом даного запиту будуть видані всі документи колекції «nobel_prize» в структурованому вигляді.

2. Параметризовані запити

При виконанні параметризованих запитів використовуються логічні оператори, що використовуються для задання умов вибірки:

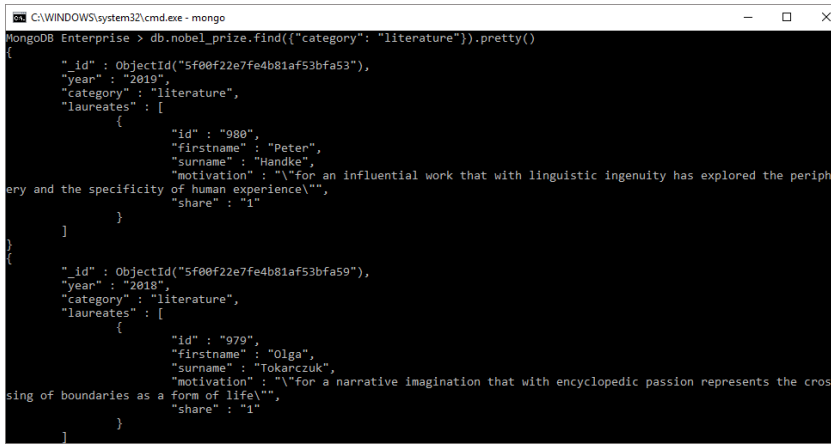
SQL	MongoDB	Опис
<	$lt	менше
<=	$lte	менше або дорівнює
>	$gt	більше
>=	$gte	більше або дорівнює
<>	$ne	не дорівнює

SQL	MongoDB	Опис
NOT	$not	заперечення
EXISTS	$exists	перевірка існування поля
OR	$or	або
NOT OR	$nor	не або
RLIKE, REGEXP	$regex	відповідність регулярному виразу
LIKE	$elemMatch	відповідність усіх полів вкладеного документа
-	$size	відповідність розміру масиву
-	$type	відповідність, якщо поле має зазначений тип

Приклад запиту вибірки з параметрами:

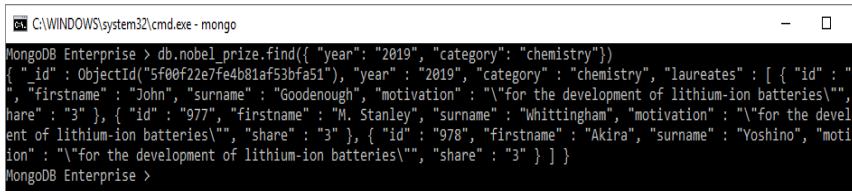
```
db.nobel_prize.find({"category": "literature"}).pretty()
```

Цей запит з параметрами (умовами) вибірки відбирає документи колекції «nobel_prize» за умови, що поле "category" відповідає значенню "literature" і видає результат в структурованому ієрархічному виді.



В умовах вибірки можна вказувати кілька параметрів, наприклад:

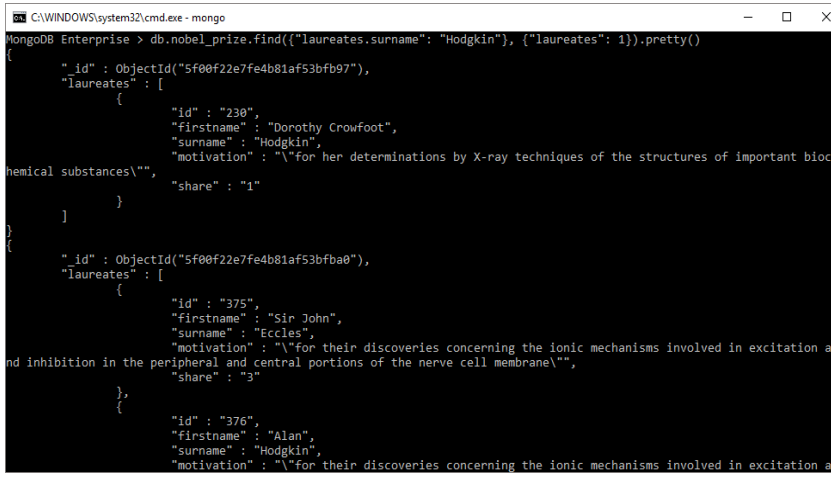
```
db.nobel_prize.find({"year": "2019", "category": "chemistry"})
```



Цей запит відбирає документи колекції «nobel_prize» за двома умовами: отримання ненобелевської премії відбулось у 2019 році, у категорії «хімія». Результат представлено у неструктурованому ієрархічному виді.

Іноді буває потрібно виконати вибірку деяких конкретних полів з документу. Наприклад:

```
db.nobel prize.find({"laureates.surname":
"Hodgkin"}, {"laureates": 1}).pretty()
```



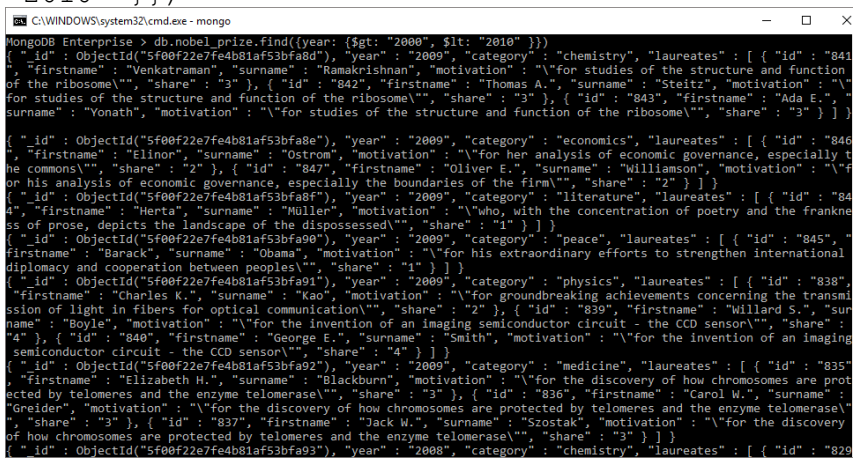
У результаті отримаємо вибірку документів, де лауреат Нобелевської премії (поле "laureates.surname") мав прізвище Hodgkin. У даному запиті обов'язковим полем для виведення на екран є поле "laureates". За замовченням, у MongoDb у подібних параметричних запитах виводиться обов'язково поле " id".

Модифікуємо запит таким чином, щоб поле "_id" на екран не виводилось:

```
db.nobel prize.find({"laureates.surname":
"Hodgkin"}, {"laureates": 1, "_id": 0}).pretty()
```

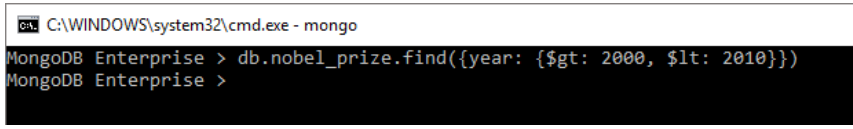
У наступному запиті знайдемо документи, в яких міститься інформація про отримані Нобелевські премії у період 2000 до 2010 рр.:

```
db.nobel prize.find({year: {$gt: "2000", $lt: "2010" }})
```



У наведеному прикладі та у створеній БД у полі «year» значення взяті у подвійні лапки, наприклад: "year" : "2009", тобто це поле string, а не number. Тому, якщо для даної БД виконати запит типу:

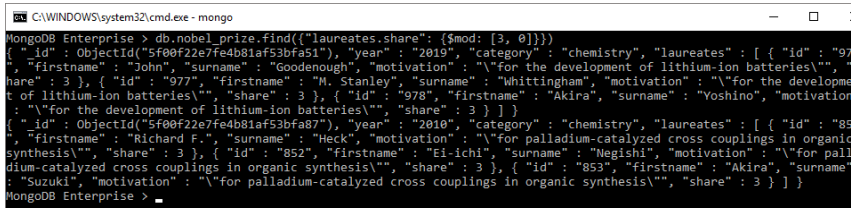
```
db.nobel_prize.find({year: {$gt: 2000, $lt: 2010}})
```



то на екран не буде виведено жодного документа, оскільки у колекції «nobel_prize» не буде документів, що задовольняють параметрам вибірки.

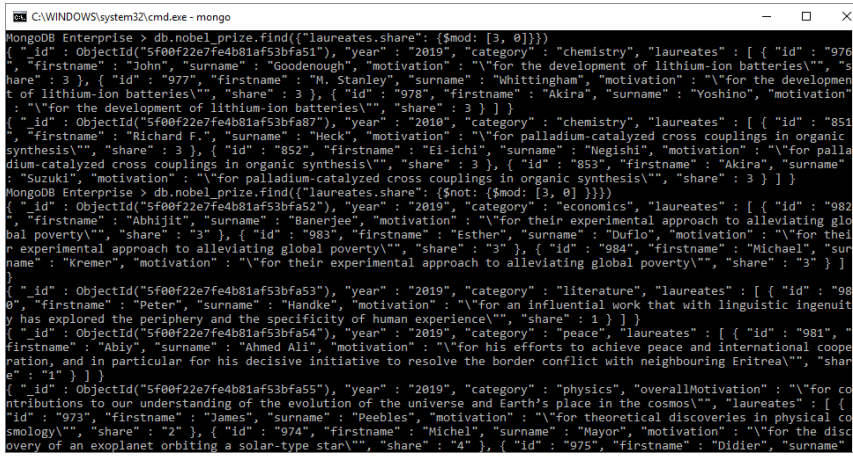
Окремо виділимо оператор $mod, який містить два значення: дільник і залишок.

```
db.nobel_prize.find({"laureates.share": {$mod: [3, 0]}})
```



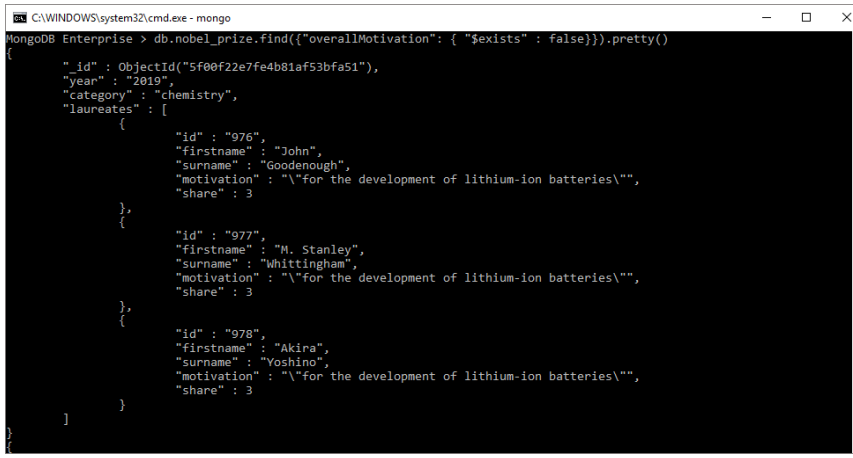
Виконання цього запиту на екран видає документи колекції «nobel_prize», поле "laureates.share" ділиться на дільник 3 з залишком — 0. Щоб отримати всі інші документи колекції, що не відповідають даній умові, використовується оператор $not:

```
db.nobel_prize.find({"laureates.share": {$not:
                {$mod: [3, 0]}}})
```



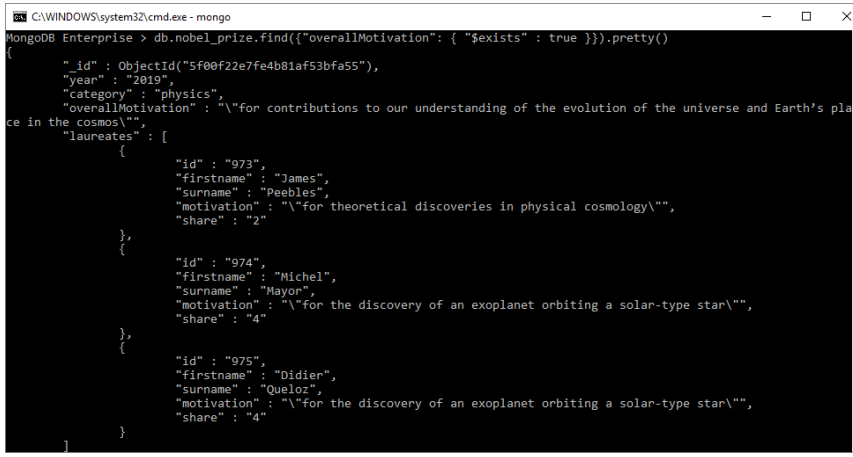
Враховуючи те, що база даних MongoDB може містити документи різної структури, то для відбору певних документів використовується оператор — $exists. Так, наприклад, наступний запити виконує відбір документів колекції, що не містять поле "overallMotivation":

```
db.nobel_prize.find({"overallMotivation":
{"$exists" : false}}).pretty()
```



Протилежним наведеному запиту буде:

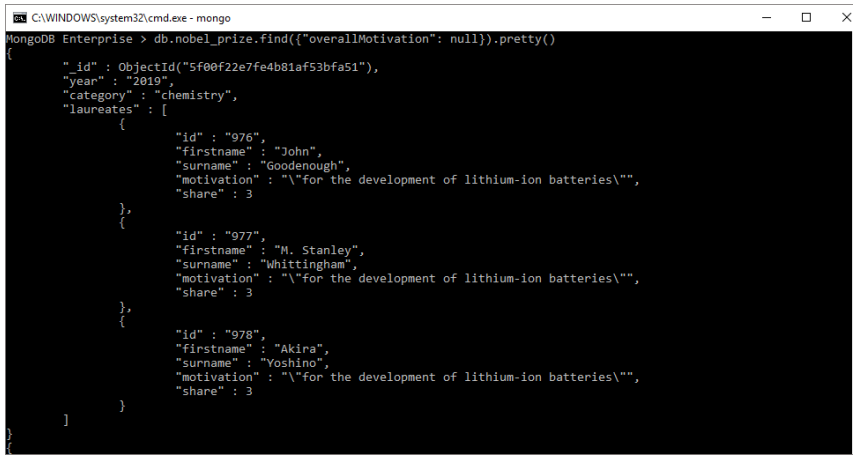
```
db.nobel_prize.find({"overallMotivation": {"$exists" : true }}).pretty()
```



Таким чином, виконується перевірка існування певного поля в документах.

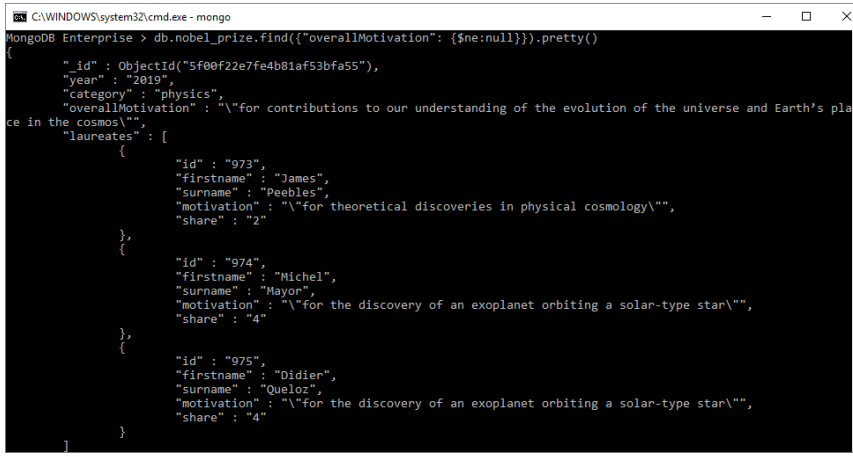
Альтернативою перевірки існування з використанням $exists є запит порівняння значення поля зі значенням null. Наприклад, вибірка документів, у яких поле "overallMotivation" дорівнює нулю, тобто відсутнє:

```
db.nobel_prize.find({"overallMotivation": null}).pretty()
```



Протилежним запитом буде відбір документів, у яких є поле "overallMotivation":

```
db.nobel_prize.find({"overallMotivation": {$ne:null}}).pretty()
```



Логічні оператори в параметризованих запитах

У вибірці можна задавати кілька параметрів, що об'єднуються між собою логічними операторами:

$and — логічне і;

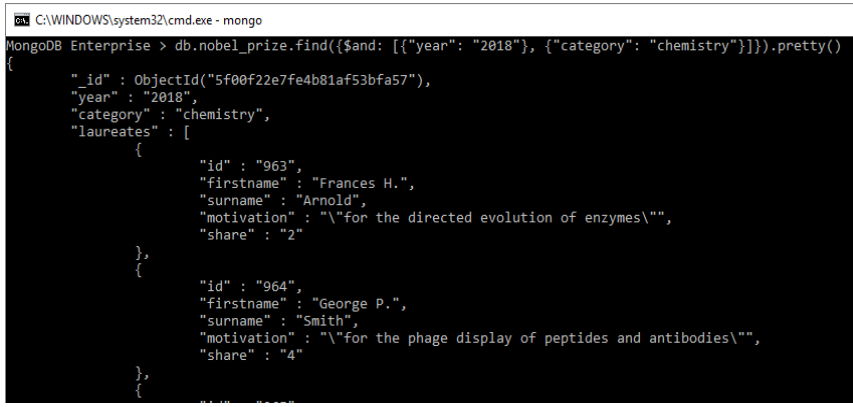
$or — логічне або;

$not — заперечення;

$nor — ні-або (істина, коли всі умови хибні).

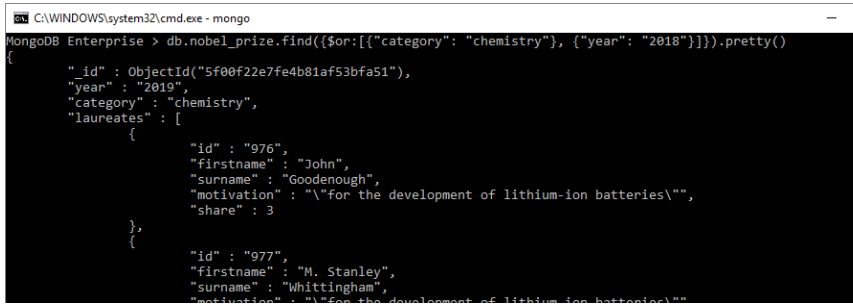
Приклад запиту вибірки з двома параметрами, що об'єднані — $and, у результаті вибираються документи, що відповідають двом умовам:

```
db.nobel_prize.find({$and: [{"year": "2018"}, {"category": "chemistry"}]}).pretty()
```



Приклад запиту вибірки з двома параметрами, що об'єднані — $or для вибірки документів, що відповідають хоча б одній умові:

```
db.nobel_prize.find({$or:[{"category": "chemistry"}, {"year": "2018"}]}).pretty()
```



3. Проекції

Проекція — це вибірка з документа значень окремих визначених полів. Щоб вказати поля, які потрібно отримати в результаті виконання запиту необхідно задати список атрибутів документа з параметром 1чи 0. Параметр 1 — означає, що даний атрибут буде видано, 0 — ні. Проекція має такий синтаксис:

```
db.collection.find({},{field: 1/0} )
```

Вище уже було розглянуто приклад вибірки документів, де лауреат Нобелевської премії (поле "laureates.surname") мав прізвище Hodgkin.

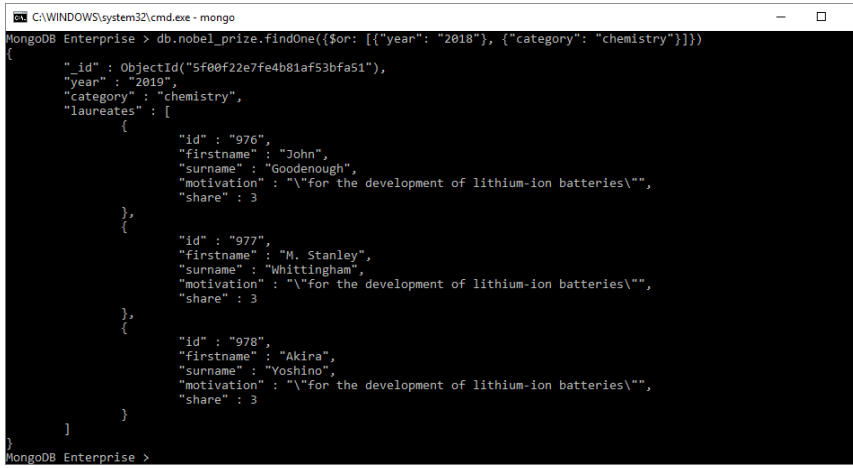
4. Управління вибіркою

Для вибірки одного документа можна скористатися оператором:

```
findOne()
```

Наприклад:

```
db.nobel_prize.findOne({$or: [{"year": "2018"},
{"category": "chemistry"}]})
```

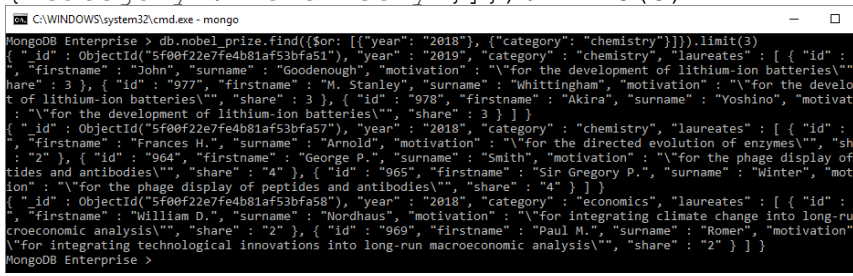


Для вибірки кількох потрібних документів використовується функція `limit()`. Обмеження кількості документів, які необхідно отримати в результаті запиту, в `find()` функція `limit()` має такий синтаксис:

```
db.collection.find().limit ()
```

Наприклад:

```
db.nobel_prize.find({$or: [{"year": "2018"},
{"category": "chemistry"}]}).limit(3)
```



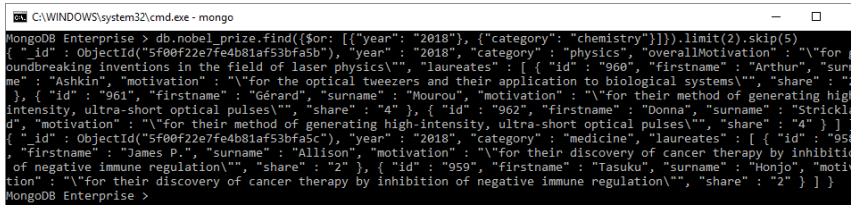
Для пропуску певної кількості документів при виборці в `find()` використовується функція `skip()`, в якій вказується скільки документів потрібно пропустити.

Синтаксис:

```
db.collection.find().skip ()
```

Приклад:

```
db.nobel_prize.find({$or: [{"year": "2018"}, {"category": "chemistry"}]}).limit(2).skip(5)
```



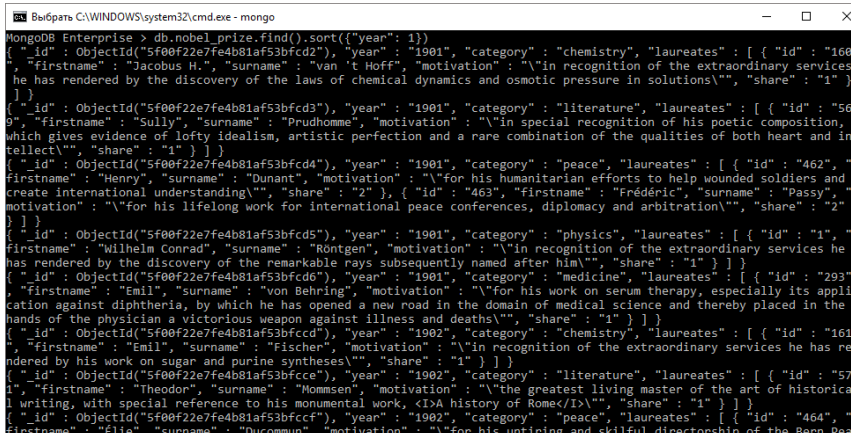
Цей запит вибирає лише два документи з колекції, не враховуючи перші п’ять.

6. Сортування

У MongoDB можна сортувати дані використовуючи функцію `sort()`. Параметр 1 задає сортування за зростанням, параметр -1 задає сортування за спаданням.

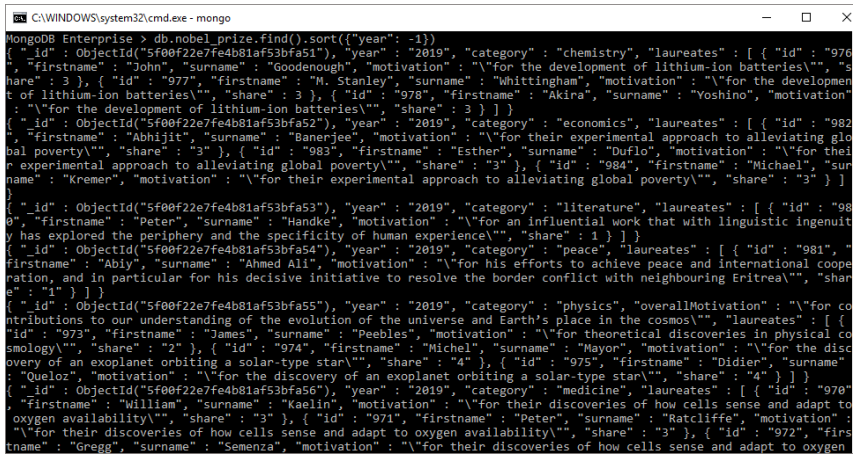
Наприклад, сортування документів колекції «nobel_prize» за роками має вигляд:

```
db.nobel_prize.find().sort({"year": 1})
```



Приклад сортування документів у порядку зворотному порядку:

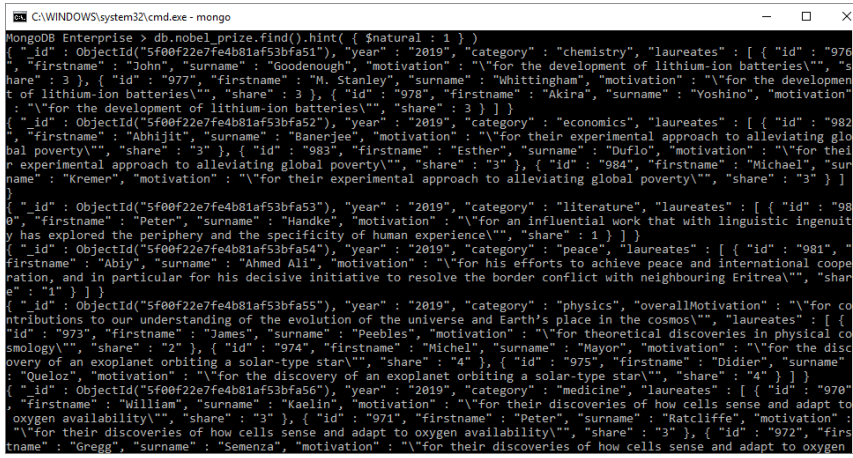
```
db.nobel_prize.find().sort({"year": -1})
```



При сортуванні обмеженої колекції необхідно використовувати параметр `$natural`, за допомогою якого можна побачити документи в порядку їх завантаження у колекцію чи у зворотному порядку.

Приклад сортування документів в порядку зворотному до їх завантаження в колекцію «nobel_prize» наведено нижче:

```
db.nobel_prize.find().hint({$natural : 1})
```



7. Агрегування (групування)

Агрегування — це операція групування значень документів колекції з метою виконання певних розрахунків. Аналогом агрегування в SQL є команда `group by`.

Узагальнений синтаксис агрегування має вид:
`db.collection.aggregate()`
Список операцій агрегування представлено у табл. 1.

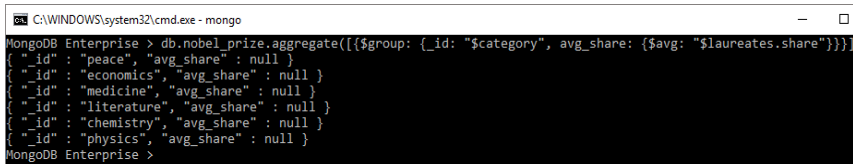
Таблиця 1

Список операцій агрегування у середовищі MongoDB

Вираз	Опис	Приклад
$sum	Підсумок указаних значень всіх документів в колекції.	db.ІМ'Я_КОЛЕКЦІЇ.aggregate([{$group : {_id : "$title", total_salary : {$sum : "$salary"}}}])
$avg	Підрахунок середнього значення указаного поля для всіх документів колекції.	db.ІМ'Я_КОЛЕКЦІЇ.aggregate([{$group : {_id : "$title", avg_salary : {$avg : "$salary"}}}])
$min	Мінімальне значення указаного поля для всіх документів колекції.	db. ІМ'Я_КОЛЕКЦІЇ.aggregate([{$group : {_id : "$title", min_salary : {$min : "$salary"}}}])
$max	Максимальне значення указаного поля для всіх документів колекції.	db. ІМ'Я КОЛЕКЦІЇ.aggregate([{$group : {_id : "$title", max_salary : {$max : "$salary"}}}])
$push	Вставка значення в масив в результатному документі.	db. ІМ'Я_КОЛЕКЦІЇ.aggregate([{$group : {_id : "$title", skills : {$push: "$skills"}}}])
$addToSet	Вставка значення в масив в результатному документі, але не створює дублікати	db. ІМ'Я_КОЛЕКЦІЇ.aggregate([{$group : {_id : "$title", skills : {$addToSet : "$skills"}}}])
$first	Отримує перший документ зі згрупованих. Як правило використовується разом з сортуванням.	db. ІМ'Я_КОЛЕКЦІЇ.aggregate([{$group : {_id : "$title", first_skill : {$first : "$skills"}}}])
$last	Отримує крайній документ зі згрупованих. Використовується разом з сортуванням.	db. ІМ'Я_КОЛЕКЦІЇ.aggregate([{$group : {_id : "$title", last_skill : {$last : "$skills"}}}])

Приклад:

```
db.nobel prize.aggregate([{$group: { id: "$category", avg share: {$avg: "$laureates.share"}}}])
```



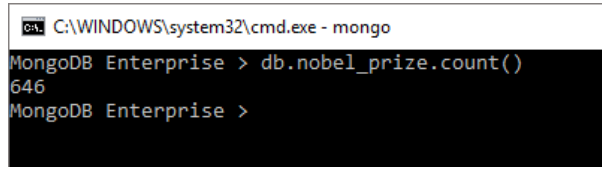
Цей запит розраховує середнє значення поля "laureates.share" по всіх документах колекції «nobel_prize».

У наведеному запиті середнє значення поля "laureates.share" визначено як null, це пов'язано із тим, що у проектованій БД, значення поля мають string і number, отже визначити середнє між різнорідними категоріями не можна.

До агрегатних функцій відноситься також функція `count()`.

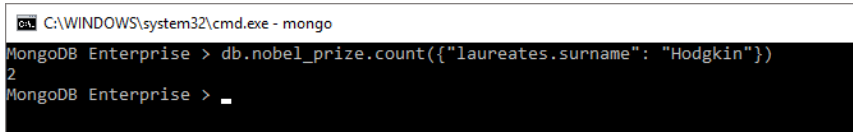
За допомогою функції count() можна підрахувати кількість документів у колекції:

```
db.nobel_prize.count()
```



Функція count, може підрахувати, кількість документів, які відповідають певним умовам. Приклад підрахунку кількості документів, у яких прізвище лауреата премії — Hodgkin:

```
db.nobel prize.count({"laureates.surname":
"Hodgkin"})
```



4.4. Завдання для опрацювання в лабораторній роботі №4

На прикладі створених колекцій виконайте такі завдання:

1. Реалізуйте два параметризованих запити з різними умовами вибірки.
2. Виконайте два запити проекції для вибірки значень різних полів.

3. Виконайте два запити управління вибіркою.
4. Виконайте запит сортування.
5. Реалізуйте два групувальних запита з різними операціями агрегування.

4.5. Інструктивні та методичні вказівки для виконання лабораторної роботи №5 «Робота із колекціями та документами у середовищі Compass»

Мета роботи: поглиблене вивчення особливостей створення, завантаження та видалення колекцій і документів у середовищі СКБД MongoDB через графічний клієнт Compass. Ознайомлення із типами даних СКБД MongoDB і форматом BSON.

Завдання роботи

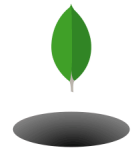
1. Ознайомитись із базовими принципами роботи в графічному клієнті Compass;
2. Доповнити власну БД (згідно узгодженої з викладачем предметної області) документами та колекціями згідно таких параметрів:
кіл-ть колекцій у БД — не менше 2;
кіл-ть документів у першій колекції — не менше 15;
кіл-ть документів у другій колекції — за бажанням автора.
3. Виконати дії над додаванням / редагуванням / видаленням об'єктів (документів, колекцій) у БД через консоль і за допомогою Compass;
4. Самостійно опрацювати реалізацію запитів до БД за допомогою Compass;
5. Відповісти на питання, поставлені в роботі.
6. Оформити звіт з лабораторної роботи.

Інструкція з виконання роботи

Оперувати даними в середовищі СКБД MongoDB можна не тільки через консоль (командний рядок), але і через графічні клієнти. На ринку існує велике різноманіття графічних клієнтів, які працюють із MongoDB: Robo 3T, 3T MongoChef, RockMongo, UMongo, NoSQL Viewer, Mongo Monkey та ін. У цьому курсі студентам пропонується працювати із графічним клієнтом Compass.

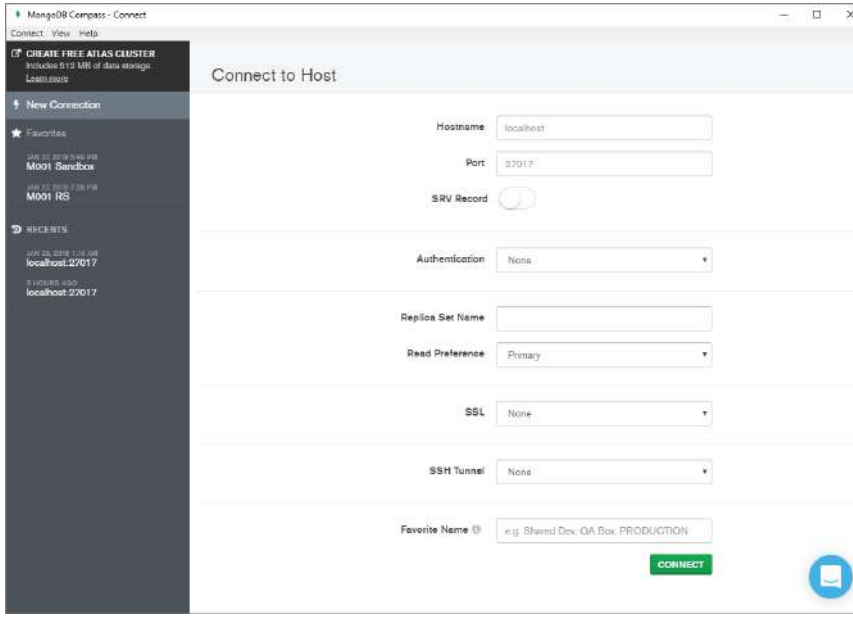
MongoDB Compass — графічний інтерфейс для MongoDB. Він дозволяє працювати із БД без знання синтаксису запитів MongoDB, який потрібен для роботи в консолі, а в зручному візуальному середовищі. Compass можна використовувати для оптимізації продуктивності запитів, управління індексами та роботи з документами, колекціями, базами даних.

Ярлик Compass на робочому столі робочої станції виглядає так:



Для того, щоб мати можливість працювати в середовищі графічного клієнта Compass його потрібно встановити на ваш ПК (якщо це не було виконано раніше, перейдіть до теми 2 даного практикуму, лабораторної роботи №1) та налаштувати підключення до існуючої нереляційної бази, з якою хоче працювати користувач.

Кожен запуск графічного клієнта Compass супроводжується появою діалогового вікна, в якому можна задавати параметри приєднання до БД:



Розберемо зміст основних полів вікна:

Hostname — хост (розміщення) сервера;

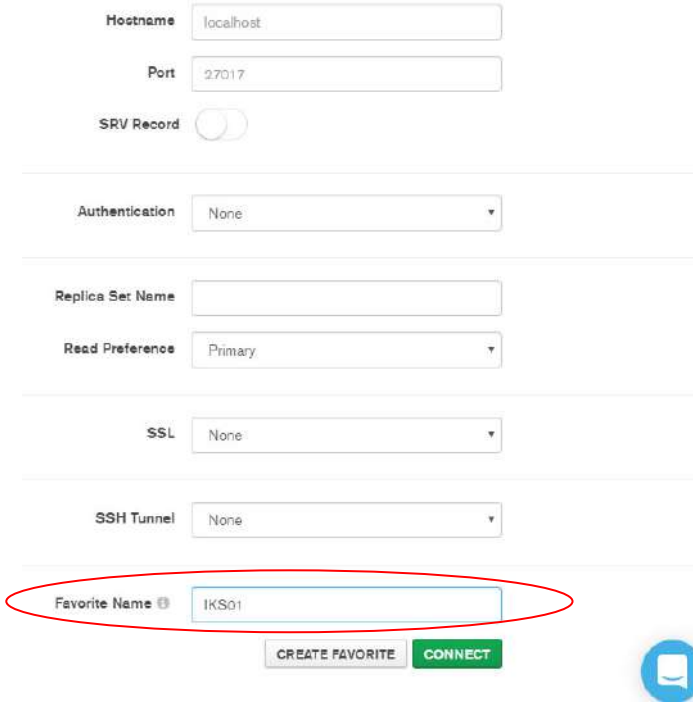
Port — порт протоколу, на якому запущений сервер (за замовченням: 27017);

SRV Record — специфікація, яка визначає розташування сервера (наприклад, адреса і порт);

Authentication	аутентифікація (вибір її типів: None, Username / Password; X.509; Kerberos; LDAP; SCRAM-SHA-256);
Replica Set Name	назва репліки, до якої відбувається підключення;
Read Preference	визначає, як графічний клієнт Compass управляє операціями читання. Може приймати такі опції: Primary, Primary Preferred, Secondary, Secondary Preferred і Nearest;
SSL	визначає тип захищеного підключення;
SSH Tunnel	визначає тип шифрування тунельних даних. Використовується до підключення кластеру MongoDB через тунель.
Favorite Name	ім'я підключення

Більше інформації про налаштування з'єднання Compass і MongoDB можна знайти у [2].

Підключення Compass до сервера, який встановлений на вашій робочій станції називається — підключення до локального сервера (localhost). Для цього не треба змінювати базові налаштування Compass (вони якраз націлені на локальний сервер, який запускається за адресою localhost: 27017). Єдине, що можна зазначити — внести зміни у поле Favorite Name (введіть номер своєї групи):

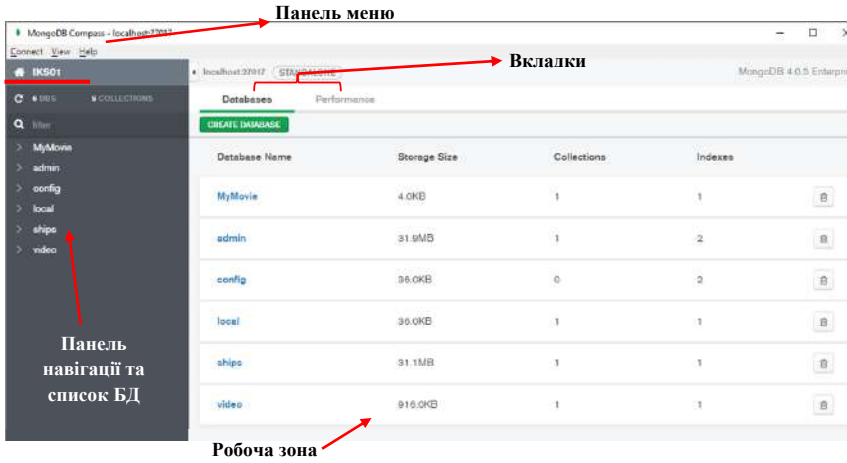


Натиснути кнопку «Create favorite», а потім на кнопку «Connect».

Зверніть увагу, що для вдалого підключення графічного клієнта Compass до середовища MongoDB на localhost, в консолі має бути запущений сервер — mongod, інакше на екран буде виведено повідомлення про помилку, наприклад: `connect ECONNREFUSED 127.0.0.1:27017.`

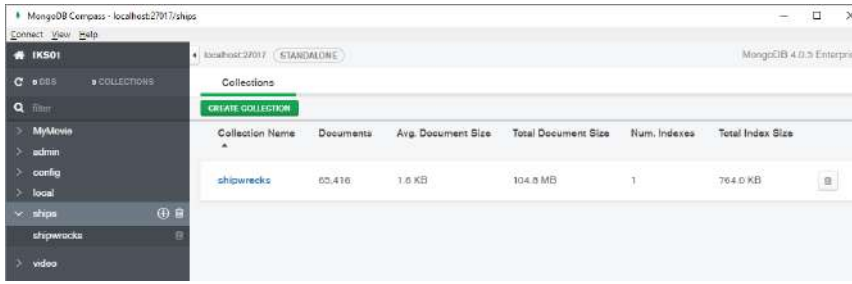
Після вдалого підключення з'явиться робоче вікно Compass, у верхньому куті якого має з'явитись назва створеного підключення.

Розглянемо структуру робочого вікна графічного клієнта Compass.



Панель меню представляє собою три пункти: підключення (Connect), вигляд (View), Допомога (Help).

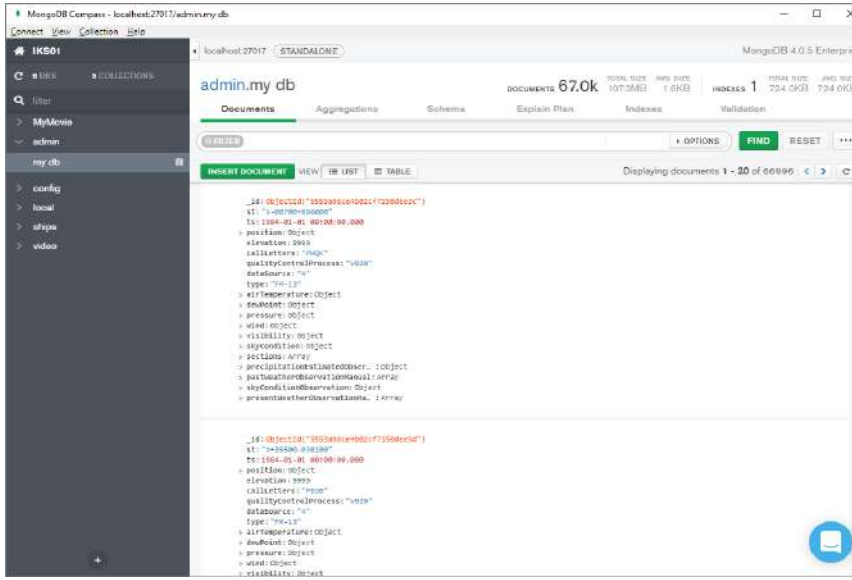
Панель навігації містить перелік підключених ДБ на локальному сервері робочої станції. Шляхом натискання на назви БД, можна бачити перелік колекцій, які створені у цій БД.



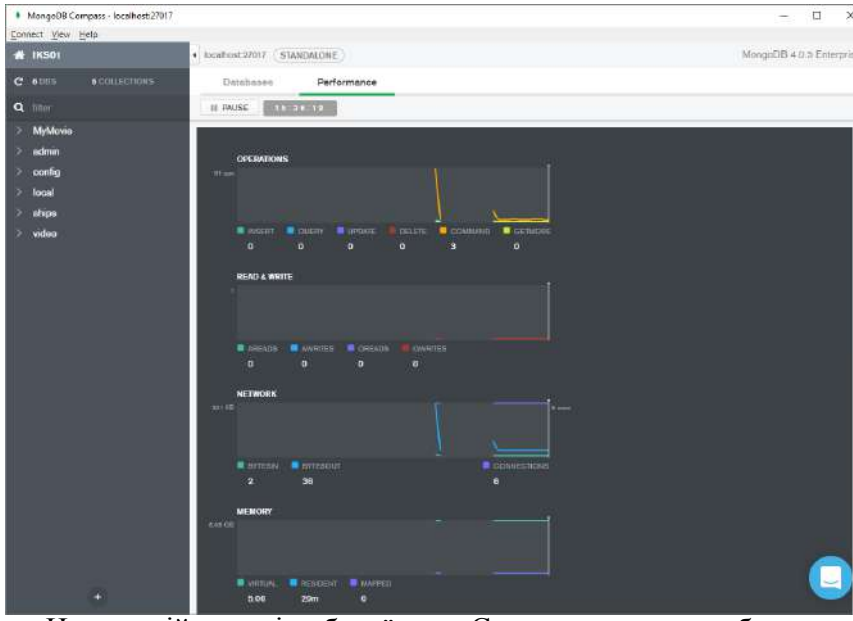
У робочі зоні відображаються дані по обраному об’єкту (БД, колекції, документам). Тут наводиться характеристика об’єкта (наприклад, розмір, кількість документів, назва тощо).

Що стосується вкладок, то у Compass є два режими відображення даних:

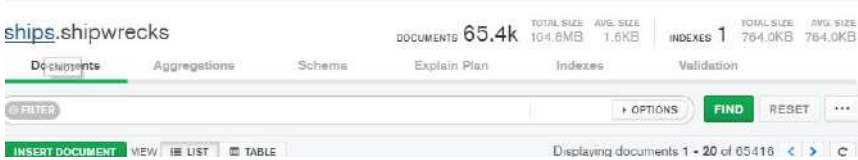
1. База даних (Database) — у цьому режимі відображаються власне об’єкти нереляційної БД (колекції, документи):



2. Продуктивність (Performance) — представляє параметри роботи сервера в режимі реального часу (моніторинг: кількості операцій, зокрема операцій зчитування та запису; роботи в мережі; пам’яті).



На верхній панелі робочої зони Compass можемо побачити таку інформацію:



ships.shipwrecks — робота у БД «ships» із колекцією «shipwrecks»;

DOCUMENTS 65.4k — кількість документів у колекції;

TOTAL SIZE 104.8MB — загальний розмір колекції;

AVG. SIZE 1.6KB — середній розмір документа в колекції;

INDEXES 1 — кількість створених індексів у колекції.

Більше інформації можна знайти у [2].

Через графічний інтерфейс програми Compass можна виконувати ті ж операції над даними (згідно концепції CRUD), що і через консоль.

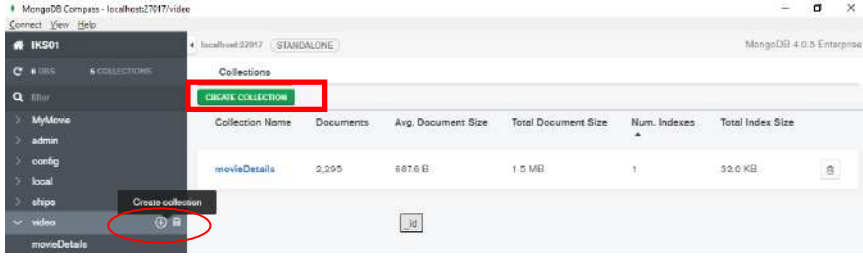
Для створення БД у Compass потрібно натиснути кнопку «Create database» у вкладці даних Database:

Відповідно, для видалення створеної БД доцільно натиснути піктограму із смітником напроти даної БД, або ті самі дії виконати можна на панелі навігації.

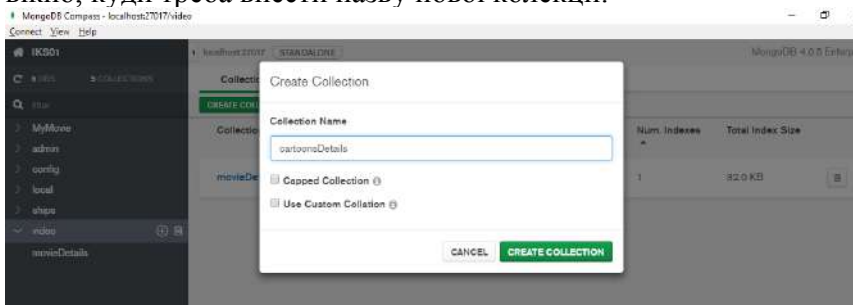
Для створення колекції, потрібно перейти у ту БД, у якій має розміщуватись ця колекція. Далі виконання операції створення можна виконувати двома шляхами:

1). натиснути відповідну кнопку «Create collections» у верхній частині панелі робочої зони;

2) натиснути на «+» біля назви БД на панелі навігації:

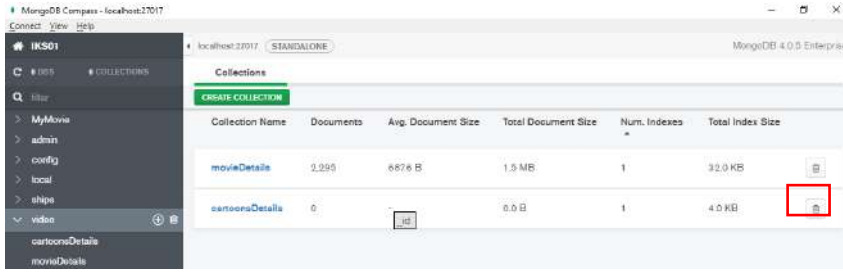


При обранні будь-якого із двох варіантів з’явиться діалогове вікно, куди треба внести назву нової колекції:



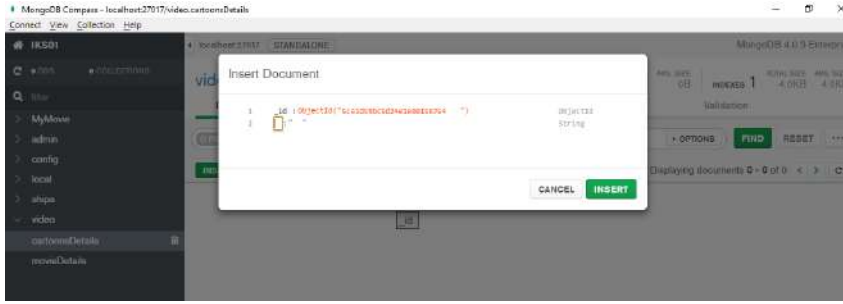
І натиснути відповідну кнопку «Create collections» (або «Cancel» у разі зміни намірів). Як видно, можна також новій колекції задати параметр обмеженої, якщо поставити відмітку навпроти «Capped Collection».

Одразу колекція «cartoonDetails» з'явилась як на панелі навігації, так і в робочій зоні:



Відповідно, видалити створену колекцію можна натисканням робочій області напроти назви колекції піктограми із смітником.

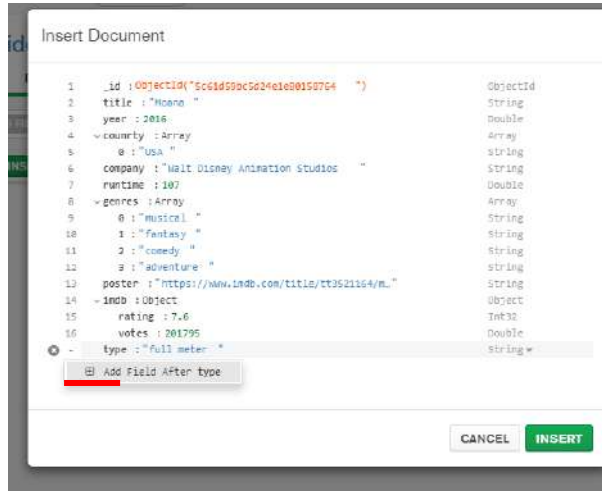
Для створення документа необхідно натиснути кнопку «Add data», яка з'явиться на панелі задач, у разі, якщо ви зайшли у потрібну колекцію. Та із спадаючого списку обрати пункт «Insert document»:



Compass автоматично створює поле _ id, але за бажання користувач може його замінити на власний запис. Головна вимога — цей запис має бути унікальним:

Створення полів документа відбувається шляхом введення пар «ключ : значення» та визначення типу даних, що містяться в парах. Наприклад, строимо документ для мультфільму «Moana» із такими полями:

_ id — objectId;
title (назва) — string (строка, текст);
year (рік випуску) — double (число з плавучею точкою);
country (країна виробник) — array (масив);
company (кіностудія виробник) — string;
runtime (тривалість) — double;
genres (жанр) — array;
poster (постер) — string;
imdb (рейтинг) — object (об’єкт): raiting (сам рейтинг) — int (числове поле); vouts (кількість голосів) — double);
type (тип) — string.



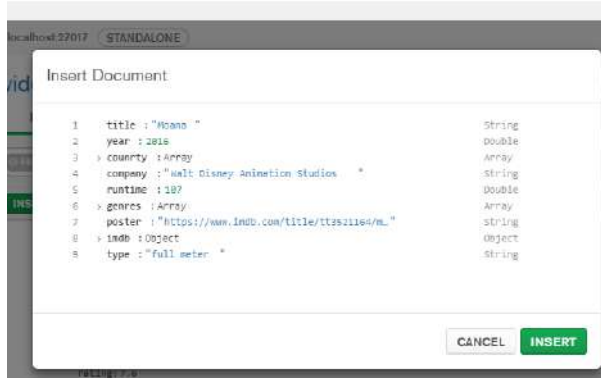
Для того, щоб додавати поля у документ треба натиснути іконку «+», яка з’являється у рядку із яким працювали останнім. Натиснувши на кнопку «Insert» документ буде доданий до колекції.

Наступний документ у колекції можна створити або таким самим чином, або шляхом дублювання (створення копії) від уже створеного документу.

При підведенні курсору миші у робочу зону, де розташований створений документ з’являються додаткові кнопки у верхній частині робочої зони:

які відповідають за створення документа, копіювання, дублювання, видалення.

Якщо обрати кнопку «Clone document», Compass створить копію попереднього документа і запропонує внести у нього правки:

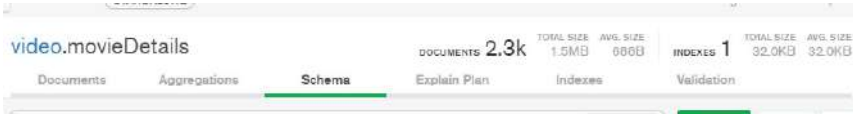


Внесемо відповідні корективи для мультфільму «Mulan».

Якщо уважно придивитись до поля «title», то можна побачити там помилку: замість «Mulan» написано «ulan». Для того, щоб її виправити, потрібно внести відповідні зміни шляхом натискання на кнопку  та потім оновити дані документа (кнопка «Update»).

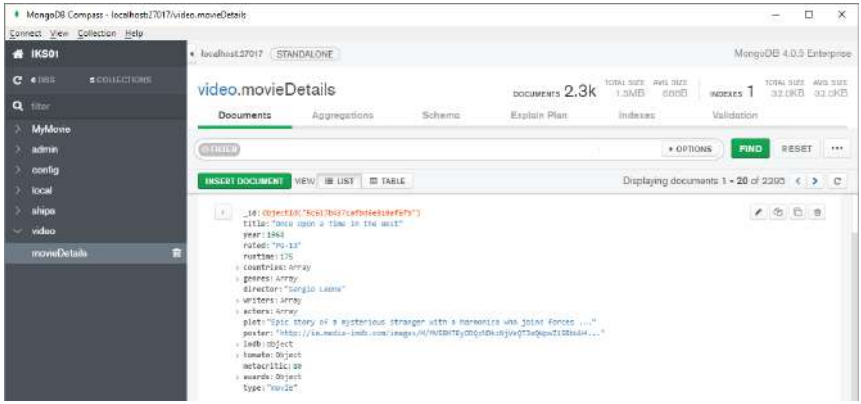
Оскільки графічний клієнт Compass підключений до сервера СКБД MongoDB на локальній робочі станції, то усі дії, які виконуються в межах CRUD можуть виконуватись як через Compass так і через консоль.

Коли корисувач на Панелі навігації Compass обрав необхідну БД, з якою потрібно працювати, то кількість вкладок у робочому вікні збільшилась. З'явились вкладки для роботи з документами, агрегацією (запитами агрегування), схема даних та ін.

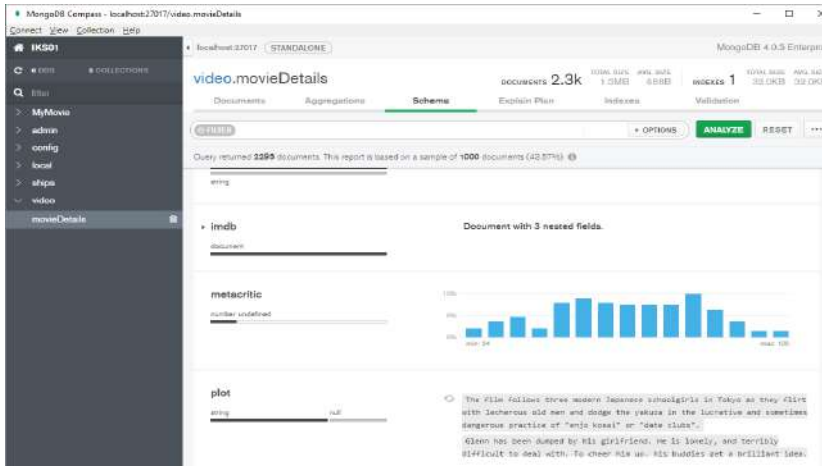


У межах даної роботи студентам пропонується робота із такими вкладками:

- Documents — тут містяться дані документів (перелік полів, їх розмірність, зміст тощо). Використовується для роботи із даними;

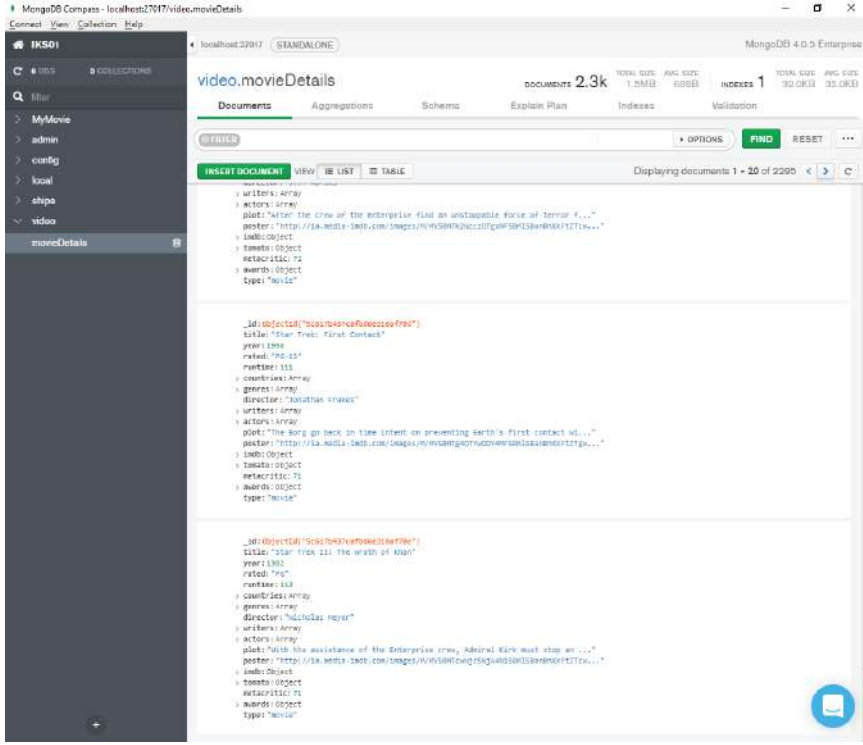


- Schema — візуалізація та різного роду групування даних, що містяться у колекції з метою їх аналізу.



Почнемо роботу із вкладки «Documents».

У робочому полі представлено перелік документів, що входять до колекції «movieDetails», по 20 документів на сторінку.



Кожен документ має більш-менш схожу структуру, але не однакову.

СКБД MongoDB підтримує роботу документів, навіть за умови, що не усі поля документів колекції є тотожними і однаковими. В одному документі може бути присутній параметр, а у іншому документі — його взагалі може не бути чи його значення буде «null» тощо.

Порівняйте:

```
1. _id : 5c617b437cafbd6e310af6fd
2. title : "Wild Wild West"
3. year : 1999
4. rated : "PG-13"
5. runtime : 106
6. imdb : Object
7. tomato : Object
8. metacritic : 38
```

```
1. _id : 5c617b437cafbd6e310af6fe
2. title : "West Side Story"
3. year : 1961
4. rated : "UNRATED"
5. runtime : 152
6. countries : Array
7. genres : Array
8. director : "Jerome Robbins, Robert Wise"
9. imdb : Object
```

Проте, одне поле у документі є все ж таки обов'язковим — поле ключа або _id. Воно є унікальним у межах колекції і БД у цілому. Якщо користувач із якихось причин не задав значення для поля id, СКБД MongoDB створить його автоматично.

Документ представляє набір пар «ключ-значення». Наприклад, у полі:

title : "Wild Wild West"

title — представляє ключ, а «Wild Wild West» — значення.

Ключі представляють собою рядки. Значення можуть відрізнятись за своїм типом.

Якщо звернутись до представленого вище переліку полів документу, то можемо виділити такі типи даних:

ключ (_id) — ObjectId — 12-байтове значення ключа об'єкта, яке складається з: 4-байти — секунди з «епохи Unix», 5-байтів — випадкові числа, 3-байти — лічильник випадкових значень;

title (назва) — string — послідовність символів;

year (рік) — number — послідовність цифр;

countries (країна) — arrey — масив даних;

awards (нагороди) — object — строковий тип даних і т.д.

Нижче представлено таблицю з описом основних типів даних, що можуть утворювати поля документів у колекціях.

Таблиця 1

Типи BSON (бінарного представлення даних для серіалізації JSON-документів)

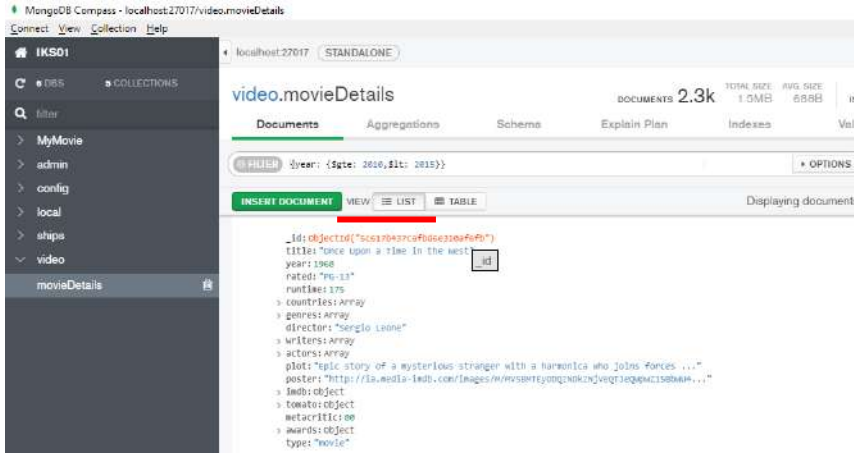
Тип даних	Представлення у MongoDB	Пояснення
1	2	3
Array	«Array»	тип даних для зберігання масивів елементів
Binary data	«Binary»	тип даних для зберігання в бінарному форматі
Boolean	«Boolean»	булевий тип даних, який зберігає логічні вирази типу (true / false)
JavaScript	«Code»	тип даних для зберігання коду Javascript
Date	«Date»	зберігає дату у форматі часу Unix (64-розрядне ціле число, яке представляє кількість мілісекунд з епохи Unix (наприклад, 1 січня 1970 р.)
Decimal128	«Decimal128»	тип даних для зберігання дійсно великих (або дійсно малих) чисел у 128-бітному десятковому поданні (використовується, коли результат округлення повинен бути максимально точним).
Double	«Double»	числовий тип даних для зберігання 64-розрядних чисел з плаваючою комою
Integer32	«Int32»	використовується для зберігання 32-бітових цілих чисел
Integer64	«Int64»	використовується для зберігання 64-бітових цілих чисел
Min key	«MinKey»	використовуються для порівняння значень з найменшим із елементів BSON. MinKey використовується в операціях порівняння та існує, переважно, для внутрішнього використання у MongoDB. Для всіх можливих значень елемента BSON MinKey завжди буде найменшим значенням
Max key	«MaxKey»	використовуються для порівняння значень з найбільшим із елементів BSON. MaxKey використовується в операціях порівняння та існує, переважно, для внутрішнього використання у MongoDB. Для всіх можливих значень елемента BSON MaxKey завжди буде найбільшим значенням
Null	«Null»	тип даних для зберігання значень Null

Продовження табл. 1

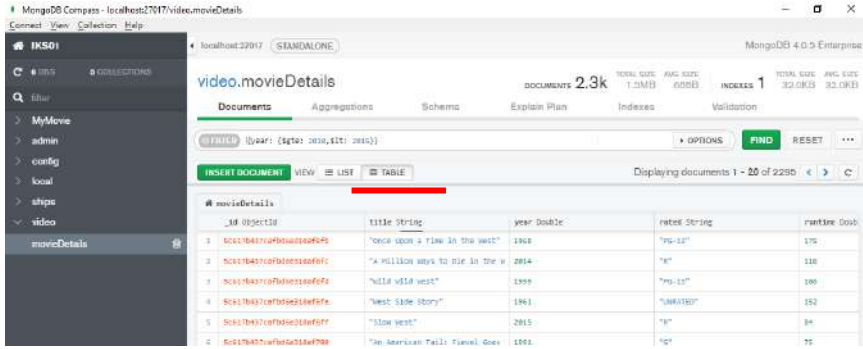
Тип даних	Представлення у MongoDB	Пояснення
1	2	3
Object	«Object»	строковий тип даних, використовується для вкладення нових об'єктів у середину існуючих
ObjectId	«ObjectId»	тип даних для зберігання id документа
BSON Regexp	«BSONRegexp»	представлення даних у вигляді регулярного виразу
String	«String»	строковий тип даних (для строк із кодуванням UTF-8).
BSON Symbol	«BSONSymbol»	тип даних, аналогійний «string». Використовується для мов, в яких є специфічні символи
BSON Map	«BSONMap»	представлення даних у вигляді нового об'єкта — карти, ключову роль в яких має ключове поле
Timestamp	«Timestamp»	використовується для зберігання часу
Undefined	«Undefined»	невизначений тип даних

Переглянути типи даних у документах у Compass можна у режимах:

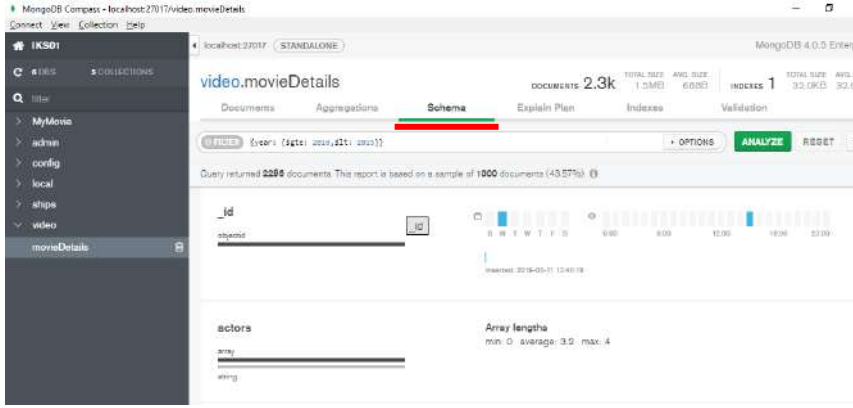
1. View List



2. View Table



3. Вкладка «Schema».



Зупинимось на специфіці представлення даних у вкладці «Schema».

Наприклад, поле «actors» колекції «movieDetails» як видно має одночасно два типи даних:

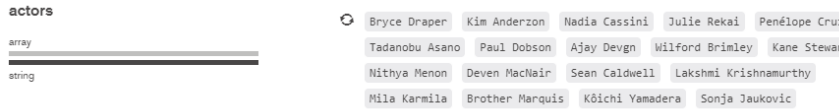
«string» та «arrey»

Для характеристики типу «arrey» можна клікнути курсором миші на смужці під написом і вона зафарбується у чорний колір. З іншого боку будуть представлені загальні характеристики параметрів усіх документів колекції «movieDetails» за цим полем:

- максимальна кількість елементів масиву — 4;
- середня кількість елементів масиву — 3,2;
- мінімальна — 0.

Якщо клікнути курсором миші на смужці над підписом «string»:

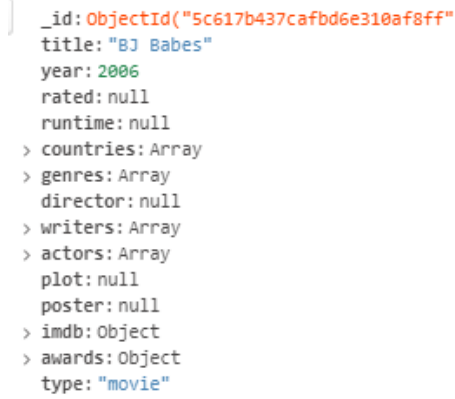
побачимо, що з'явились написи із прізвищами, які збережені як «string». Поруч із написами є поле «оновлення», яке дозволяє переглядати «порціями» перелік усіх полів із типом «string»:



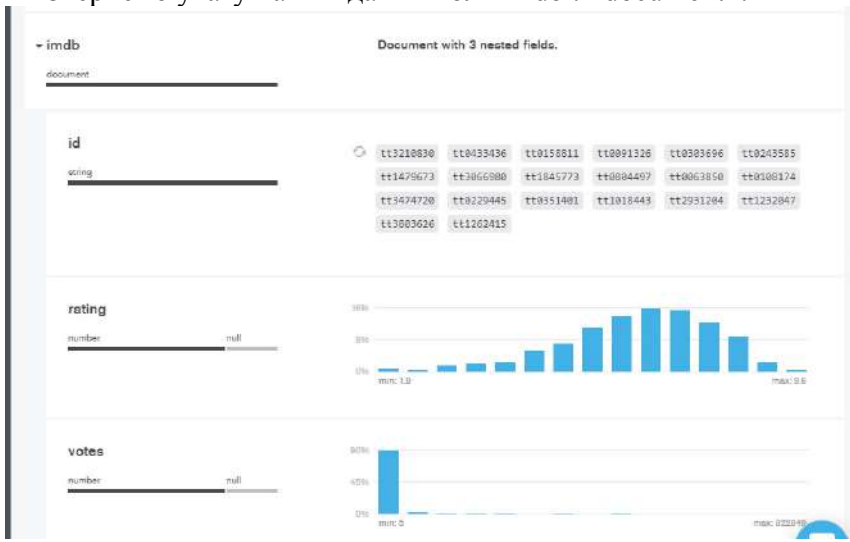
Якщо подивитись на поле «director», то бачимо, що тут є також два типи даних:

«string» і «null»

Наявність поля «null» говорить про те, що у колекції «movieDetails» є документи із полем «director», у яких немає даних. Як от у документа з _id: «5c617b437cafbd6e310af8ff»:



Звернемо увагу на тип даних поля imdb : «document»:

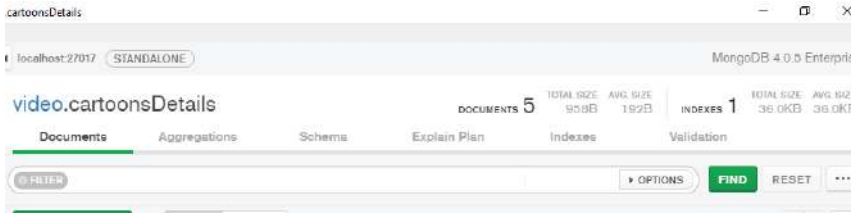


Він передбачає наявність додаткових даних у своїй структурі. Як видно, поле рейтинг «imdb» складається із:

- id (рейтинга) — тип «string»;
- rating — тип «number» та «null»;
- votes (голосування) — тип «number» і «null».

Читання (перегляд) документів, що містяться в колекціях у консолі використовується за допомогою конструкції: db.collection.find ().

Якщо потрібно виконати запит пошуку через Compass, його потрібно виконати у полі запитів «Filter»:

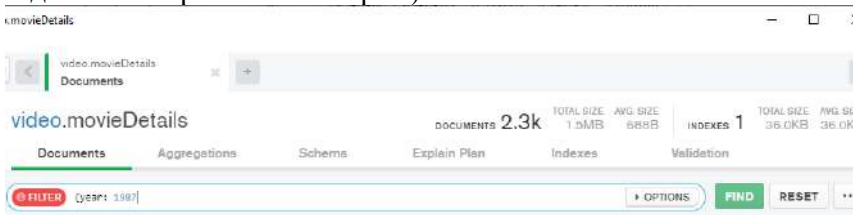


Наприклад, якщо потрібно знайти документи, у міститься інформації про фільми, випущені у прокат в 1987 році, запит буде виглядати таким чином:

```
{year: 1987}
```

Тут треба звернути увагу на такі речі:

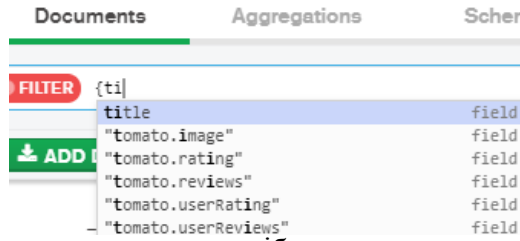
- параметри запиту потрібно писати у фігурних дужках {}, інакше аналізатор буде видавати помилку (Слово «Filter» буде підсвічено червоним кольором):



- пари «ключ/значення» у Compass не потрібно брати у додаткові лапки, лапки доцільно використовувати, тільки для значень, які мають тип даних «string», наприклад:

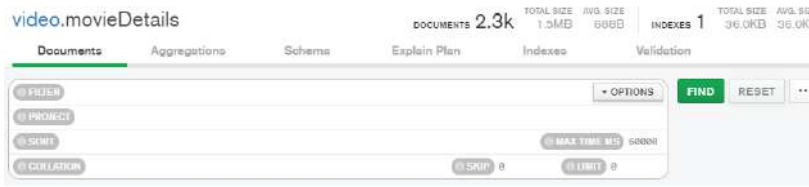
```
{title: "Moana"}
{rated: "PG-13", director: "Sergio Leone", runtime: 175}
{"countries.1": "New Zealand"}
{year: {$gte: 1980}, runtime: 180};
```

- у Compass є вбудований помічник, який при набиранні тексту в рядку «Filter» випадаючи списком пропонує варіанти підстановки:



- для виконання запиту потрібно натиснути на кнопку Find (вона буде мати зелену заливку, якщо запит синтаксично коректний, або сіру — якщо у запиті присутні помилки), для скидання результатів запиту можна натиснути кнопку Reset.

Якщо при роботі із запитами у Compass є необхідність використання проекцій, сортування та інших опцій, доцільно натиснути на кнопку «Options»:



поле Project — використовується для роботи з проекціями;

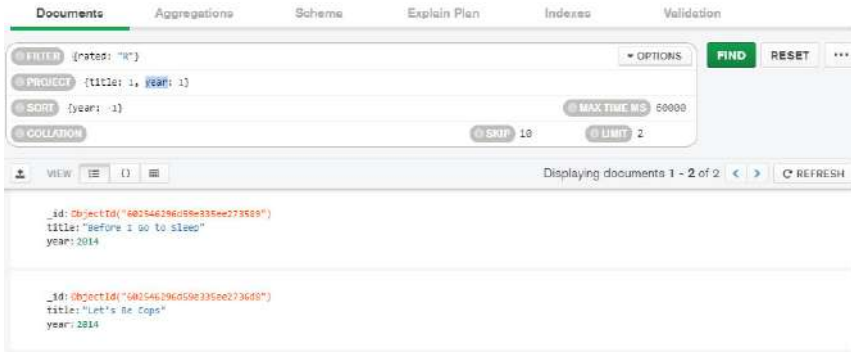
поле Sort — використовується для виконання сортування;

поле Collation — використовується для зіставлення або порівняння;

поле Max time MS — використовується для визначення сукупного ліміту часу (в мілісекундах) для операцій обробки;

поле Skip — використовується для виконання пропуску певних документів за результатами запиту;

поле Limit — використовується для обмеження кількості виведення документів за результатами запиту.



Якщо вам потрібна довідка, щодо роботи із тип чи іншим полем можна натиснути на знак *i* (

), що міститься перед назвою поля і це викличне відкриття посилання в браузері з відповідною супровідною документацією.

4.6. Завдання для опрацювання в лабораторній роботі №5

На прикладі створених колекцій виконайте такі завдання у MongoDB Compass:

1. Реалізуйте два параметризованих запити з різними умовами вибірки.
2. Виконайте два запити проекції для вибірки значень різних полів.
3. Виконайте два запити управління вибіркою.
4. Виконайте запит сортування.

5. **Додаткове завдання** — Реалізуйте два групувальних запита з операціями агрегування: $count, $match (вкладка «Aggregations»).

4.7. Питання для самоконтролю

Дайте відповідь на питання:

1. За допомогою якого оператора можна виконувати вибірку?
2. Який параметр дозволяє представляти вибірку в структурованому ієрархічному вигляді?
3. Які умови вибірки можуть використовуватися в параметризованих запитах?
4. Які логічні оператори можуть використовуватися в параметризованих запитах?
5. Що таке проекція та її призначення в запитах?
6. Які функції використовуються для управління вибіркою?
7. Що виконує функція count()?
8. Що представляє собою запит агрегування?
9. Які операції можна виконувати в запитах агрегування?
10. Для виконання яких функції у Compass використовується пункт меню «Connect»?
11. Для виконання яких функції у Compass використовується пункт меню «View»?
12. Для виконання яких функції у Compass використовується пункт меню «Help»?
13. Яке з перелічених полів є обов'язковим для створення документу у колекції БД «video» (actors, director, year, _id, comments)?
14. Виходячи з даних наведених на зображені нижче, яке з тверджень буде правильним:

а. не всі документи в цій колекції мають однаковий тип значення для цього поля;

б. більшість документів у цій колекції містять значення «int32» для цього поля;

в. деякі документи в цьому полі мають тип даних «Year».

4.8. Література до теми

1. MongoDB CRUD Concepts / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/core/crud/index.html
2. MongoDB Compass / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/compass/current/
3. Field Update Operators / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/reference/operator/update-field/

Тема 5.

РОБОТА З МАСИВАМИ І РЕГУЛЯРНИМИ ВИРАЗАМИ В СЕРЕДОВИЩІ СКБД MongoDB

5.1. Перелік знань і вмінь

При вивченні даної теми студенти ознайомляться зі принципами роботи з масивами та регулярними виразами у MongoDB. Отримають базові навички створення документів, що містять масиви і компетенції роботи з ними. Відпрацюють створення запитів до документів з використанням регулярних виразів.

Вивчивши матеріал теми, ви будете ***знати****:*

- поняття масиву та його створення в MongoDB;
- команди внесення змін до масивів;
- команди вибірки по масивах;
- створення запитів за шаблоном з використанням регулярних виразів.

Вивчивши матеріал теми, ви будете ***вміти****:*

- створювати документи з масивами в MongoDB;
- вносити зміни до масивів;
- створювати запити вибірки по масивах;
- створювати запити з використанням регулярних виразів.

5.2. Термінологічний словник

Асоціативні масиви **—** це масиви MongoDB, які можуть мати будь-яке поєднання ключів і їх значень.

Бекслеш-символ — це спеціальний символ (\) — зворотна коса риска (backslash), який використовується в регулярних виразах.

Ключ індексу (*key*. *index*:) — це порядковий номер елемента в масиві. Індексація елементів масиву починається з 0, 1, 2, ...

«Нормальні» масиви — визначаються як масиви MongoDB зі зростаючими числовими індексами, починаючи з 0 та збільшенням на одиницю для кожного нового елементу.

Масив (одновимірний) — це множина значень, які мають порядкові номери (індекси). Масив береться у квадратні дужки [ ]. Значення розділяються комами. *Значення може бути:* рядком у двійних лапках, числом, значенням true чи false, об'єктом, масивом, чи значенням null. Ці структури можуть бути вкладеними одна в одну. Масив **—** це спосіб моделювання звязків між дани-

ми. Враховуючи відсутність у MongoDB схеми даних і JOIN-ів за допомогою масивів, що вмонтовуються в документи, можна моделювати звязки 1:Б та Б:Б. Масиви вкладаються у документи.

Наприклад:

```
{"name":"BMW",
model":"X64","engine":2,"color":["grey","black"]}
```

Модифікатори масивів — це оператори, що використовуються в команді **update** і вказують на види змін до масивів документів колекції. До основних модифікаторів масивів відносяться такі оператори: **$push** — добавляє новий елемент у кінець масиву без перевірки на його унікальність чи створює новий масив у документі за умови його відсутності; **$ne** або **$addToSet** — добавляють новий елемент у кінець масиву за умови його унікальності; **$pop** — вилучає елементи з масиву. Якщо вказана цифра 1, то вилучається останній елемент, якщо -1, то вилучається перший елемент. **$pull** — вилучає конкретний елемент з масиву.

Регулярні вирази (regular expressions) — це формальна мова вибірки та пошуку текстових даних. Для пошуку використовується рядок-зразок (*pattern*), який складається з символів і метасимволів, які задають правила пошуку. Рядок-зразок для пошуку іноді називають шаблоном. У MongoDB використовуються Perl-сумістні регулярні вирази (*Perl-compatible regular expressions* (PCRE)).

5.3. Інструктивні та методичні вказівки для виконання лабораторної роботи №6 «Робота з масивами та регулярними виразами в середовищі MongoDB»

***Мета роботи*:** створення запитів до масивів з командного рядка в середовищі СКБД MongoDB.

Реалізація запитів виконується на основі колекцій і документів, які були створені при виконанні лабораторної роботи №2 — 4.

Якщо створені раніше документи в базі даних студента не містять масивів, то необхідно доповнити колекції документами з масивами для подальшої роботи.

Для роботи з масивами використовуються такі команди.

$set	записує вказане значення у вказане поле
$unset	вилучає поле

$inc	додає зазначене число до вказаного поля
$pop	вилучає останній чи перший елемент з масиву
$push	додає новий елемент в масив
$pull	вилучає з масиву значення, якщо воно є в масиві
$addToSet	аналог оператора $push, але не додає дублікати
$pullAll	вилучає з масиву всі значення.
$each	розширює можливості операторів $push та та $addToSet шляхом додавання кількох елементів для оновлення масиву
$position	розширює можливості оператора $push, вказує позицію в масиві для додавання елементів
$slice	розширює можливості оператора $push, обмежуючи («зрізаючи») розмір відповідних масивів
$sort	розширює можливості оператора $push, змінюючи порядок документів у масиві

Пошук по довжині

Оператор $size знаходить документи, в яких кількість елементів масиву рівно значенню $size. Наприклад, наступна команда віднайде всі документи колекції car, в яких у массиві "color" міститься два элемента:

```
>db.car.find({"color": {$size:2}})
```

Запит вибірки з масиву

Пошук по одному параметру розглянемо на прикладі пошуку автомобілів чорного кольору:

```
>db.car.find({"color":"black" })
```

Якщо при пошуці треба виконувати порівняння по кількох параметрах, то використовується $all. Наприклад, знайдемо документи, за такими двома параметрами:

```
>db.car.find({color:{$all: ["black", "grey"]}})
```

Результатом запиту будуть три документи, що характеризують автомобілі, які мають "black" і "grey" кольори:

```
{"_id": ObjectId("5e3e81342185efa15ac13cdf"), "name": "Honda", "model": "CR – V", "color": ["black", "white", "silwer", "grey"], "engine": "2,4", "price": 35000}
{"_id" : ObjectId("5e3e8b3d2185efa15ac13ce3"), "name": "BMW", "model": "X-6", "color": ["black", "white", "grey" ], "engine": "4.6", "price" : 55000}
```

```
{"_id" : ObjectId("5e40783fa574db4e298c2f73"),
"name" : "Reno", "model": "Sandero", "color" :
["soul red crystal", "grey", "black", "white",
"silwer"], "engine": "2,4", "price": 14000}
```

Для вибору певного елемента масиву необхідно вказати його індекс (порядковий номер у масиві), використовуючи синтаксис *key*. *Index*. Так. наприклад, наступний запит виведе інформацію про всі автомобілі, в яких першим елементом у масиві color буде "soul red crystal" :

```
db.car.find({"color.0": "soul red crystal"})
```

Управління вибіркою масива

Управління вибіркою елементів масиву виконує оператор $slice, який є своєрідною комбінацією функцій limit і skip, але на відміну від них може працювати з масивами. Оператор $slice має два параметри. Перший вказує на загальну кількість елементів масиву. Другий є необов'язковим, але якщо він використовується, тоді перший параметр вказує на зміщення відносно початку (як це робить функція skip), а другий на кількість елементів, які потрібно видати.

Розглянемо роботу даного оператора на прикладі документа:

```
db.car.insert({"name": "Pezo", "model":"VD-1",
"engine": "4.6", "price": 55000, "balance":
[{"black":2},{"white": 5},{"grey":1}]})
```

Наступний запит вибирає перший елемент з масиву "balance":

```
> db.car.find ({name: "Pezo"}, {balance:
{$slice : 1}})
```

Результат запиту:

```
{"_id" : ObjectId("5e3f13259193bd20d80d5008"),
"name" : "Pezo", "model" : "VD-1", "engine" :
"4.6", "price" : 55000, "balance" : [{"black":
2}]}
```

Зворотній випадок, коли потрібно віднайти останній елемент масиву, то виконуємо такий запит:

```
> db.car.find ({name: "Pezo"}, {balance:
{$slice : -1}})
```

Результат запиту:

```
{"_id" : ObjectId("5e3f13259193bd20d80d5008"),
"name": "Pezo", "model": "VD-1", "engine": "4.6",
"price": 55000, "balance": [{"grey": 1}]}
```

Використовуючи два параметри виконаємо запит, який вибирає один елемент, починає з кінця масиву, пропустивши один елемент:

```
> db.car.find ({name: "Pezo"}, {balance: {$slice : [-1,1]}})
```

Результат запиту:

```
{"_id" : ObjectId("5e3f13259193bd20d80d5008"), "name": "Pezo", "model": "VD-1", "engine": "4.6", "price": 55000, "balance": [{"white": 5}]}
```

Внесення змін до масивів

Доповнення масивів новими елементами виконується за допомогою оператора **$push** . Оператор **$push** добавляє новий елемент у кінець масиву. Синтаксис оператора:

```
>db.collection.update (<запит>, {$ push: {<поле>: <значення>}})
```

Параметри:

Ім'я	Опис
поле	ім'я стовпчика чи поля документа
значення	ці значення повинні бути вказаними для полів чи стовпчиків
запит	запит може бути виразом, умовою чи критерієм

Якщо масив у документі відсутній, то він створюється. Розглянемо документ, який не має масивів:

```
db.car.insert({"name": "AUDI", "model":"A-7", "engine": "4.6", "price": 95000})
```

Наступний запит створить у даному документі масив "color":

```
> db.car.update({"name": "AUDI"},{$push: {"color": "silver "}})
WriteResult({"nMatched" : 1, "nUpserted": 0, "nModified": 1})
```

Виконавши find переглянемо структуру документа після внесення змін.

```
db.car.find({"name": "AUDI"})
```

Документ після внесення змін буде мати таку структуру:

```
{"_id" : ObjectId("5e42616f343ba44733f50ddf"), "name": "AUDI", "model": "A-7", "engine": "4.6", "price": 95000, "color": ["silver"]}
```

Добавимо ще один елемент у масив "color", виконавши такий запит:

```
db.car.update({"name": "AUDI"},{$push: {"color": "black"}})
WriteResult({"nMatched": 1, "nUpserted": 0, "nModified": 1})
```

Модифікатор «$push» додає по одному елементу в масив, не перевіряючи їх на унікальність.

Для того, щоб додати в масив лише унікальні елементи використовуються **$ne** або **$addToSet**.

Так, наприклад, з використанням **$ne** наступний запит добавить кольор "blue", якщо він відсутній у масиві "color":

```
> db.car.updateOne({"name": "AUDI", "color": {"$ne": "blue"}},
... ... {$push: {"color": "blue"}})
{"acknowledged": true, "matchedCount": 1, "modifiedCount": 1}
```

Додамо новий колір з перевіркою унікальності значення з використанням **$addToSet:**

```
> db.car.updateOne({"name": "AUDI"}, {$addToSet: {"color": "grey"}})
{"acknowledged": true, "matchedCount": 1, "modifiedCount": 1}
```

При спробі ще раз добавити уже існуючий колір ніяких змін не відбувається і видається таке повідомлення:

```
{"acknowledged": true, "matchedCount": 1, "modifiedCount": 0}
```

Документ після внесення всіх змін має таку структуру:

```
{"_id" : ObjectId("5e42616f343ba44733f50ddf"), "name": "AUDI", "model": "A-7", "engine": "4.6", "price": 95000, "color": ["silver", "black", "blue"]}
```

Вилучення елементів з масиву

Для вилучення використовується модифікатор **$pop.** Наприклад:

```
> db.car.updateOne({"name": "AUDI"},{$pop: {"color" :-1}})
{"acknowledged": true, "matchedCount": 1, "modifiedCount": 1}
```

Якщо вказана цифра 1, то вилучається останній елемент в масиві, якщо -1, то вилучається перший елемент.

Наприклад, наступний запит виділяє перший колір з документа, що характеризує автомобіль Hyundai:

```
db.awto.update({"name" : "Hyundai"}, {$pop: {color: 1}})
```

Іноді потрібно вилучати елемент масиву по конкретному критерію. Для вилучення конкретного елемента з масиву використовується **$pull.** Наприклад, вилучення з масиву "color" кольору "grey" виконується таким чином:

```
> db.awto.update({"name" : "Hyundai"}, {$pull: {"color": "grey"}})
WriteResult({"nMatched": 1, "nUpserted": 0, "nModified": 1})
```

Зміна значення елементів масиву

Зміну значень елементів масиву розглянемо на прикладі документа, що містить дані про продаж за певний місяць:

```
{"_id" : 3, "month" : 1, "sales": [5,10,4,5]}
```

Наступний запит змінить інформацію про продаж елемента масиву з індексом 0, додавши число 20:

```
> db.car.update({"month": 1}, {"$inc": {"sales.0": 20}})
WriteResult({"nMatched" : 1, "nUpserted": 0, "nModified": 1 })
```

Результат виконання запиту:

```
{"_id": 3, "month": 1, "sales": [25, 10, 4, 5]}
```

Робота з регулярними виразами

Регулярні вирази в MongoDB використовують Perl-синтаксис (Perl-сумістні регулярні вирази). Регулярні вирази — це інструментарій при створенні шаблонних запитів, які являють собою рядок, що складається з метасимволів, які задають правила пошуку. Тобто це формальна мова пошуку при роботі з текстовими даними. Подібним до регулярних виразів є оператор Like в мові запитів SQL.

При створенні регулярних виразів необхідно виконувати такі правила:

1. будь який символ позначає сам самого, якщо це не метасимвол. Якщо потрібно відмінити дію метасимволу, то перед ним необхідно поставити знак '\'.
2. рядок символів означає рядок цих символів;
3. множина можливих символів (клас), взятих у квадратні дужки '[]', значить, що у даному місці може стояти один із вказаних у дужках символів. Якщо перший символ у дужках '^' — це значить, що жоден із вказаних символів не може стояти в даному місці у виразі. У середині класу можна використовувати символ '-', який позначає діапазон символів. Наприклад, a-z — одна з малих літер латинського алфавіту, 0-9 — цифра і т.д.;

4. усі символи, включаючи спеціальні, можна позначати за допомогою '\' як у мові С;

5. альтернативні послідовності розділяють символом '|'. Слід відмітити, що цей символ у квадратних дужках рахується звичайним символом;

6. у середині регулярного виразу можна вказувати "підшаблони" взявши їх в круглі дужки та посилатися на них як '\номер'. Перша дужка позначається як '\1'.

При створенні запитів використовуються такі опції:

i	не розрізняти рядкові і заголовні букви
m	багаторядковий рядок

Таблиця метасимволів шаблону:

\	наступний метасимвол вважається звичайним
^	початок рядка
.	один довільний символ. Окрім символу кінця рядка (\n)
$	кінець рядка
|	або
()	групування
{}	клас символів

Модифікатори, які пишуться після метасимволів:

*	Повторюється 0 чи більше разів
+	Повторюється 1 чи більше разів
?	Повторюється 1 чи 0 разів
{n}	Повторюється не менше n раз
{n,}	Повторюється точно n раз
{n,m}	Повторюється не менше n раз, але не більше m

Фігурні дужки в усіх інших випадках враховуються як звичайні дужки.

За замовчанням, дія метасимволів розповсюджується стільки разів, скільки це можливо, не враховуючи результат дії наступних метасимволів.

При створенні регулярних виразів використовуються бекслеш-символи (backslash). Символи, що найчастіше використовуються, приведено нижче:

\d	одна цифра
\D	одна не цифра
\S	один не space символ
\s	один space символ (табуляція, новий рядок, пробіл)
\U	у верхньому регістрі з наступного символа
\u	верхній регістр наступного символа
\E	обмежувач зміни регістра
\L	усі символи в нижньому регістрі до E
\I	нижній регістр наступного символа

Оператор $regex

Пошук тектових даних можна виконувати з оператором $regex з використанням одного з таких синтаксів:

```
{<field>:{$regex:/pattern/,          $options:
"<options>"}}
{<field>:{$regex:/pattern/"<options>"}}
```

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

Наприклад, наступний запит виведе документи, в яких у назві автомобіля, в полі name, є літера "o":

```
>db.car.find ({name: {$regex:"o"}})
```

Наступний запит видасть усі документи, в яких у полі "name" є літери "Ro":

```
>db.car.find({"name": {$regex:"Ro"}})
```

Результат виконання запиту:

```
{"_id" : ObjectId("5e466749c699a19c3b0eb7ba"),
"name": " Alfa Romeo", "model" : "TV", "color":
["soul red crystal", "grey", "black", "white",
"silwer"], "engine": "2", "price": 34000}
{"_id" : ObjectId("5e46675ec699a19c3b0eb7bb"),
"name": " Land-Rover", "model": "TV", "color":
["soul red crystal", "grey", "black", "white",
"silwer"], "engine": "3", "price": 44000}
```

Пошук за шаблоном також можна виконувати за допомогою find. Так наприклад наступний запит видасть назву автомобіля без врахування регістру:

```
>db.car.find({"name": /^bmw$/i})
```

Результат виконання запиту:

```
  {"_id"  :  ObjectId("5e3e8b192185efa15ac13ce2"),
"name": "BMW", "model": "X-5", "color": ["soul red
crystal", "grey"], "engine": 3, "price": 50000}
  {"_id"  :  ObjectId("5e3e8b3d2185efa15ac13ce3"),
"name": "BMW", "model": "X-6", "color": ["black",
"white", "grey"], "engine": "4.6", "price": 55000}
```

5.4. Завдання для опрацювання в лабораторній роботі №6

На прикладі створених колекцій реалізуйте такі основні запити:

1. Реалізуйте запит вибірки з масиву з різними умовами вибірки.
2. Реалізуйте запити вибірки по довжині масиву.
3. Реалізуйте запити вибірки із масиву з елементами управління вибіркою.
4. Виконайте запити додавання нових даних до масиву з використанням операторів, що перевіряють і не перевіряють на унікальність нових елементів.
5. Виконайте запит вилучення елементів з масиву.
6. Виконайте запит зміни значень масиву.
7. Реалізуйте два запита з використанням регулярних виразів.

5.5. Питання для самоконтролю

Дайте відповідь на питання:

1. Як можна виконати вибірку з масиву?
2. Перерахуйте основні оператори пошуку в масиві та характеризуйте їх функції.
3. Який оператор виконує додавання нових елементів у кінець масиву без перевірки на їх унікальність?
4. Який оператор виконує додавання нових унікальних елементів у масив?
5. Характеризуйте поняття регулярного виразу та його використання в MongoDB.
6. Характеризуйте основні метасимволи та їх використання в регулярних виразах.

5.6. Література до теми

1. MongoDB CRUD Concepts / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/core/crud/index.html
2. Field Update Operators / Офіційний портал компанії MongoDB Inc. [Електронний ресурс]. URL: https://docs.mongodb.com/manual/reference/operator/update-field/

Тема 6.

ГРАФОВА МОДЕЛЬ БАЗИ ДАНИХ NoSQL

6.1. Перелік знань і вмінь

При вивченні даної теми студенти знайомляться з концепцією побудови графових баз даних NoSQL у середовищі СКБД Neo4J. Вивчать основні елементи графової моделі БД: вузли та їх властивості, ребра їх властивості та мітки. Отримають базові навички створення графових баз даних і роботи з ними на прикладі СКБД Neo4J.

Вивчивши матеріал теми, ви будете ***знати****:*

- основи побудови графової бази даних та її використання в середовищі СКБД Neo4j;
- оператори мови Cypher для створення таблиць вузлів і таблиць ребер;
- оператори реалізації запитів до графої бази даних на мові Cypher.

Вивчивши матеріал теми, ви будете ***вміти****:*

- інсталювати СКБД Neo4j;
- створювати таблиці вузлів у середовищі СКБД Neo4j;
- створювати таблиці ребер у середовищі СКБД Neo4j;
- створити запити до графової бази даних з використанням засобів мови Cypher.

6.2. Термінологічний словник

Вузол (node) — це об'єкт у базі даних, що характеризує вершину графа. Вузлам можна присвоювати змінні, а також вузли можуть бути помічені однією чи кількома мітками. Властивості вузлів представляються парами «ключ-значення». На рис. 6.1 представлено два вузли, вузол з міткою Customer містить характеристики покупців, а вузол з міткою Product характеризує продукти.

У Neo4j ключі представляються як String, а значення може бути представлено будь-яким типом даних Neo4j, а також масивами цих типів. Кількість вузлів у Neo4j обмежена $2^{35\sim34}$ білліону.

Приклад двох вузлів надано на рис. 6.1.

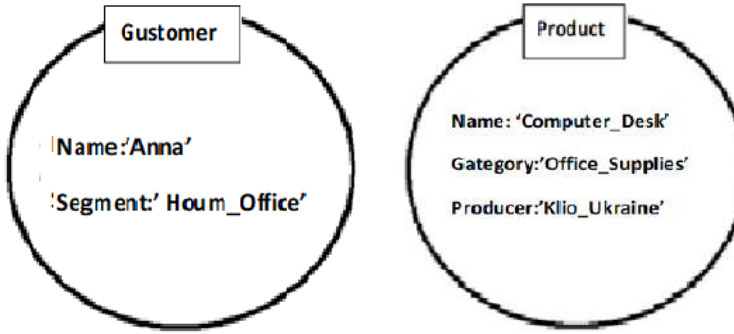


Рис. 6.1. Вузли з мітками Gustomer і Product

Графова база даних (graph database) — у середовищі СКБД Neo4j це безагрегатна база даних NoSQL, що підтримує графову модель с мітками. Графова модель з мітками має такі характеристики: містить вузли та ребра, що характеризують взаємозв'язки; вузли мають властивості (пари ключ-значення); вузли можуть бути помічені однією чи кількома мітками; ребра мають імена і орієнтацію, для них завжди визначений початковий і кінцевий вузол; ребра також мають властивості.

Enterprise Edition — є розширеною версією Community Edition, яка включає функції для підвищенняя продуктивності та масштабування, такі як робота з кластерами та функції онлайн-резервного копіювання. Має розширені функції безпеки, включаючи контроль доступу на основі ролей і підтримку LDAP (Lightweight Directory Access Protocol). Ця версія підходить для масштабних комерційних проектів з підвищеними вимогами до масштабування та доступності.

Запити на Cypher прикріпляють одну чи кілька частин шаблона до заданого певного місця в графі за допомогою предикату, а потім рухають незафіксовані частини, пробуючи найти відповідність.

Мітка (node label) — характеризує тип вузла. Мітки є регістрозалежними, причому Cypher не видає помилку, якщо мітку набрати не в тому регістрі. Таким чином Brand, BRAND і brand — це три різні мітки. У Neo4j мітки використовуються для визначення ролей вузлів у графі. Один вузол у Neo4j може мати кілька міток, так як він може бути зв'язаним з багатьма іншими вузлами, які виконувати різні ролі і мають різне призначення. За допомогою міток можна групувати вузли відповідно до їх призначення і ролі.

Модель бази даних Neo4j — СКБД Neo4j підтримує орієнтований граф з властивостями (Property Graph).

Мова Cypher (Cypher Query Language - COL) — декларативна мова запитів співставлення зі зразком**.** Cypher є не лише мовою запитів, а й мовою маніпулювання даними, тобто виконує функції CRUD для графової бази даних. За синтаксисом ця мова подібна до мови запитів SQL, що використовується у реляційних базах даних.

Community Edition — повнофункціональна версія Neo4j, яка підходить для розгортання в одному екземплярі. Повністю підтримує ключові функції Neo4j, такі як відповідність ACID, мова Cypher та API програмування. Є ідеальною версією для вивчення Neo4j, самостійних проєктів і додатків у невеликих робочих групах.

Паттерни або шаблони (Patterns) — конструкції мови Cypher для створення запитів шляхом співставлення на відповідність певному шаблону, який описує дані, які необхідно отримати в результаті пошуку по графу. Шаблони для опису графових даних і співставлення при їх пошуку використовується в: MATCH, CREATE та MERGE мови Cypher. Шаблони графів представляються ASCII-графікою і являються основою Cypher.

Пошук у глибину (depth-first) — це основна стратегія обходу від початкового вузла до кінцевого. Повторення обходу для пошуку другого шляху завжди починається з одного і того початкового вузла.

Пошук у ширину (breadth-first) — це такий підхід обходу графа, коли він обстежується пластами. Спочатку обстежуються вузли, які находяться від початкового вузла на відстані одного взаємозв'язку, а потім на відстані двох, потім трьох і т.д. Припинення обходу залежить від конкретного алгоритму, який уособлює в собі кінцеву мету пошуку. Прикладом пошуку в ширину є алгоритм Дейкстри для находження найкоротшого шляху між двома вузлами в графі.

Ребро (relation) — відображає зв'язок між двома вузлами графа. Ребра (відношення) повинні бути поіменованими і орієнтованими. У графі не може бути незавершених взаємозв'язків, кожне ребро містить початковий вузол і кінцевий вузол. На рис. 6.2 показано зв'язок між двома вузлами з міткам Customer і Product, який характеризує купівлю і ідентифікований ім'ям Purchase.

Вузол з міткою Customer — початковий, а з міткою Product — кінцевий. У Neo4j ребра можуть бути двонаправленими. Так як і вузли, ребра можуть мати властивості, у вигляді пар «ключ-значення». Кількість вузлів у Neo4j обмежена $2^{35\sim34}$ білону.

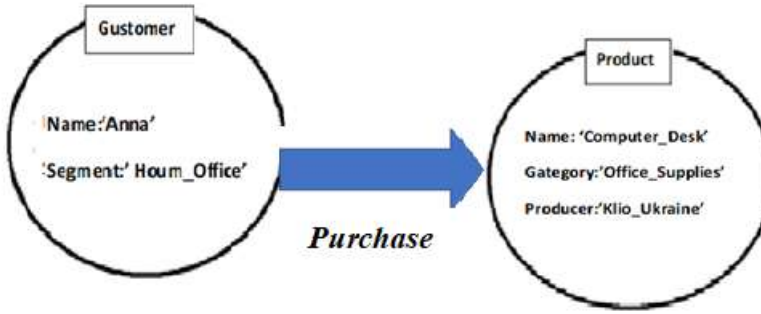


Рис. 6.2. Зв'язок між двома вузлами з міткам Customer і Product ребром Purchase

Сервер Neo4j — це система керування графовою базою даних, яка реалізована на мові Java. Розроблена американською компанією Neo Technology 2003 році. На 2020 рік Neo4j вважається найпоширенішою графовою СКБД (згідно рейтингу сайту DB-Engines). Для роботи з базою даних реалізовані бібліотеки для мов: Java, C++, Perl, PHP, Python, Ruby, Go та ін., а також є REST API. Підтримує ACID (атомарність, узгодженість, ізоляція та довговічність), дозволяє створювати: 32 мільярди вузлів, 32 мільярди ребер і 64 мільярди властивостей. Має засоби візуалізації графа. Це Neo4j Browser, що має графічний інтерфейс на основі веббраузера. Neo4j має дві редакції: комерційну версію Enterprise Edition з додатковими функціями продуктивності та адміністрування, а також версію Community Edition з відкритим програмним кодом.

Тип зв'язку (relation identifier) — відображає тип зв'язку. На відміну від міток конкретний зв'язок може мати лише один тип. Максимальна кількість типів зв'язків у Neo4j — 32767.

Траверс графа — це пошук множини шляхів за заданими умовами.

Шлях — це множина пов'язаних між собою вузлів і ребер.

6.3. Інструктивні та методичні вказівки для виконання лабораторної роботи №6 «Установка та перше знайомство з СКБД Neo4j»

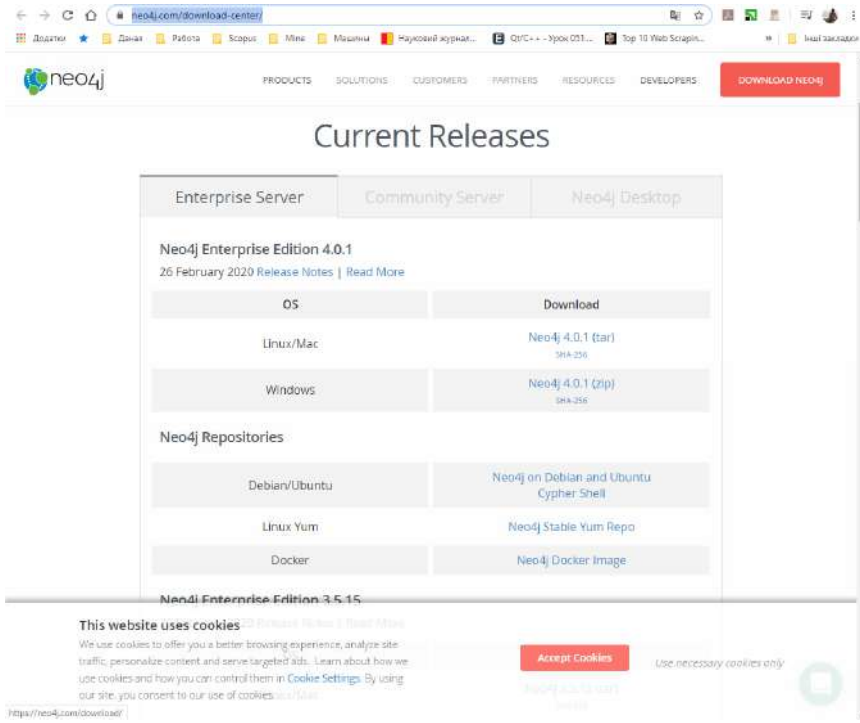
Мета роботи: знайомство із СКБД Neo4j, інсталяція нереляційної графової бази даних на робочу станцію, отримання навиків первинного налаштування операційної системи для роботи із СКБД Neo4j.

Завдання роботи

1. Ознайомитись із загальною інформацією про СКБД Neo4j, яку подано в лекційному матеріалі, а також у джерелах [1–4].
2. Встановити на робочу станцію СКБД Neo4j.
3. Ознайомитись із базовими командами роботи СКБД Neo4j.
4. Відповісти на питання, поставлені в роботі.
5. Оформити звіт з лабораторної роботи.

Інструкція з установки СКБД Neo4j

Для завантаження СКБД на робочу станцію потрібно перейти на сайт вендора за посиланням: https://neo4j.com/download-center/ і обрати необхідну версію релізу, враховуючи встановлену на робочій станції операційну систему та її бітну розрядність.



Ви можете завантажити потрібну версію Neo4j:

- Enterprise server — повнофункціональний дистрибутив, що підтримує кластеризацію, має додаткові функції безпеки, немає обмежень на розмір графів і бази даних, а також кількість одночасних підключень тощо. Однак даний варіант після версії 3.3.0

потребує придбання ліцензії (3.2.8 — остання версія, що включає ліцензію AGPL);

- Community server — робочий дистрибутив, що має певні функціональні обмеження для роботи, порівняно із Enterprise server. Даний варіант є безкоштовним для завантаження та використання;
- Neo4j deskop — графічний дистрибутив, що дозволяє працювати з локальними базами даних Neo4j. Має зручний інтерфейс, проте дещо обмежений функціонал порівняно із Community server.

Порівняльну таблицю відмінностей можна знайти за посиланням: https://neo4j.com/subscriptions/#editions

Умежах даного комплексу робіт студентам пропонується завантажувати та працювати із версією **Community server** (яка відповідає встановленій на ПК операційній системі та її розрядності).

Після завантаження архіву із дистрибутивом на робочу станцію, пропонується розархівувати його у зручну для користувача папку. Для зручності, рекомендується створити папку на диску С (наприклад, С:\neo4j\ ).

Фактично, файли дистрибутива уже збережені на робочій станцій. Для їх запуску можна використати один із двох варіантів:

- запуск СКДБ з консолі;
- запуск СКДБ в якості служби Windows.

Розглянемо перший варіант, тобто запуск Neo4j через консоль.

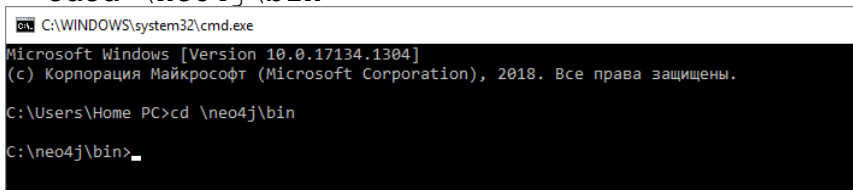
Необхідно запустити консоль:

– або за допомогою клавіш Windows + R, у поле вводу написати «cmd» і натиснути «ОК»);

– або у пункті мені Пуск знаходимо консоль під назвою «cmd» або «командний рядок» і запускаємо її.

У вікні консолі необхідно ввести директорію, де знаходяться файли Neo4j, у нашому випадку це диск С, папка neo4j. Команда має вигляд:

```
cdcd \neo4j\bin
```



Після того, як робоча директорія перемістилась у необхідне місце, де розташовані файли Neo4j, у консолі потрібно запустити виконавчий файл самої СКБД:

```
.\neo4j.bat console
```

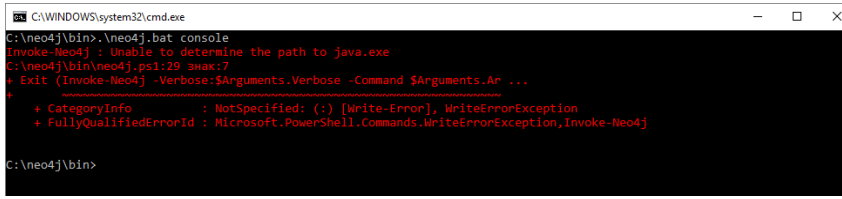
Результатів виконання команди може бути кілька:

– або система запуститься (що буде розглянуто нижче);

– або у вікні консолі буде виведено текст про помилку. Цей варіант розглянемо першим.

Найрозповсюдженішою проблемою запуску Neo4j є відсутність на робочій станції встановленого програмного середовища JAVA Oracle (або наявність застарілої версії).

У випадку, якщо на робочій станції не встановлено попередньо JAVA Oracle, то система виведе у вікно консолі попередження:



Для виправлення ситуації необхідно встановити на робочу станцію Java Development Kit (JDK).

Зверніть увагу, що з середини 2019 року компанія Oracle змінила умови розповсюдження власного програмного забезпечення, і Java Development Kit поставляється у двох варіантах:

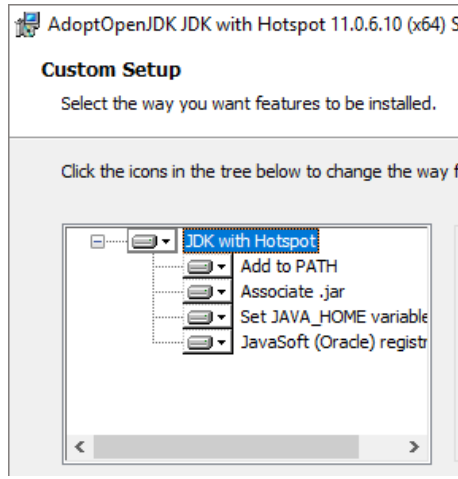
- Oracle JDK (або Java SE) — платна ліцензія, доступна для завантаження на сайті java.com ;
- Open JDK — відкрита (безкоштовна) ліцензія, доступна для завантаження з ресурсів: https://jdk.java.net та http://openjdk.java.net/ .

Open JDK можна завантажити за посиланням:

- набір бінарних файлів у архіві: https://jdk.java.net/archive/;
- отримати файл у форматі MSI у збірці AdoptOpenJDK: https://adoptopenjdk.net/?variant=openjdk11&jvmVariant=hotspot .

У межах даної роботи було обрано завантаження та інсталяцію збірки AdoptOpenJDK. Після завантаження пакету інсталяції на робочу станцію, потрібно запустити, власне, інсталяцію AdoptOpenJDK. У процесі установки варто дотриматись кількох правил:

1) доцільно під час інсталяції в меню обрати завантаження усіх пунктів:



2) вказати директорію для інсталяції збірки AdoptOpenJDK, наприклад: C:\Program Files\Java\ .

Після завершення інсталяції може знадобитись перезавантажити робочу станцію.

Тепер можна знову запустити виконавчий файл запуску СКДБ Neo4j. Якщо усі дії виконнані вірно, то у відповідь на команду:

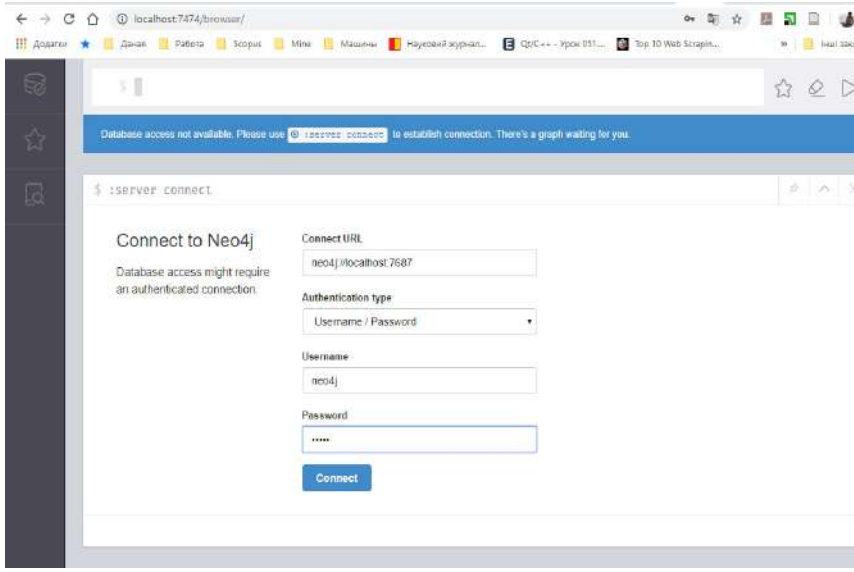
```
.\neo4j.bat console
```

у вікні консолі буде отримано таку відповідь:

```
C:\WINDOWS\system32\cmd.exe - .\neo4j.bat console
Microsoft Windows [Version 10.0.17134.1304]
(c) Корпорация Майкрософт (Microsoft Corporation), 2018. Все права защищены.

C:\Users\Home PC>java -version
openjdk version "11.0.6" 2020-01-14
OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.6+10)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.6+10, mixed mode)

C:\Users\Home PC>cd \neo4j\bin

C:\neo4j\bin>.\neo4j.bat console
2020-03-10 16:19:40.381+0000 INFO  ======== Neo4j 4.0.1 ========
2020-03-10 16:19:40.414+0000 INFO  Starting...
2020-03-10 16:20:24.932+0000 INFO  Bolt enabled on localhost:7687.
2020-03-10 16:20:24.936+0000 INFO  Started.
2020-03-10 16:20:31.217+0000 INFO  Remote interface available at http://localhost:7474/
```

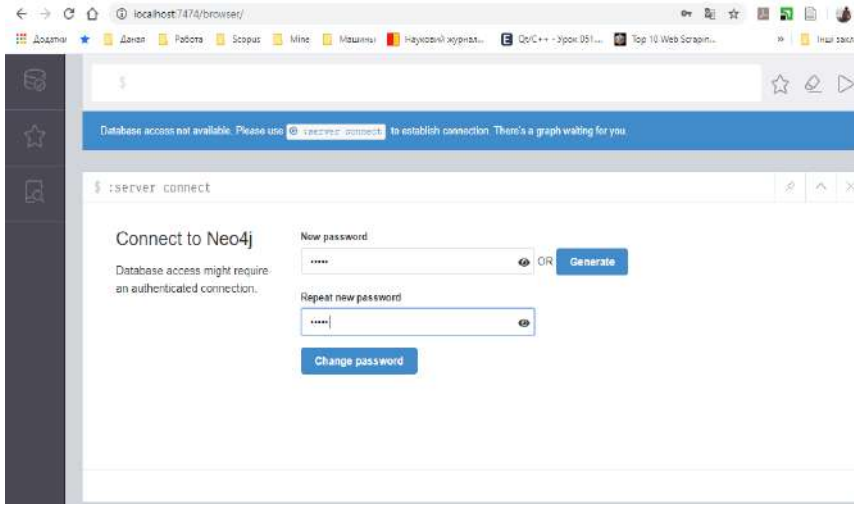
Після запуску Neo4j можна переходити до роботи у веб-інтерфейсі браузера. Для цього у вікні вашого браузера треба вказати адресу, яка за замовчуванням налаштована у сервера Neo4j: http://localhost:7474/ .

У разі підключення з’явиться діалогове вікно:

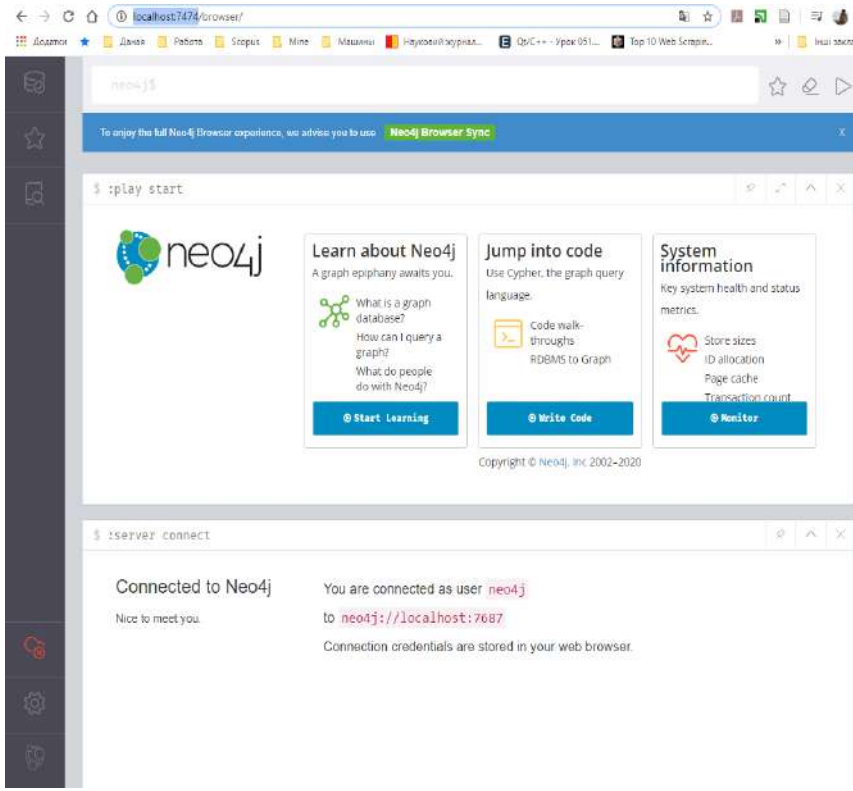


При першому підключенні в якості пароля Ім’я користувача та пароля вкажіть: **neo4j**

Система запропонує змінити пароль доступу на новий:



З’єднання встановлене, тепер можна працювати із СКБД.



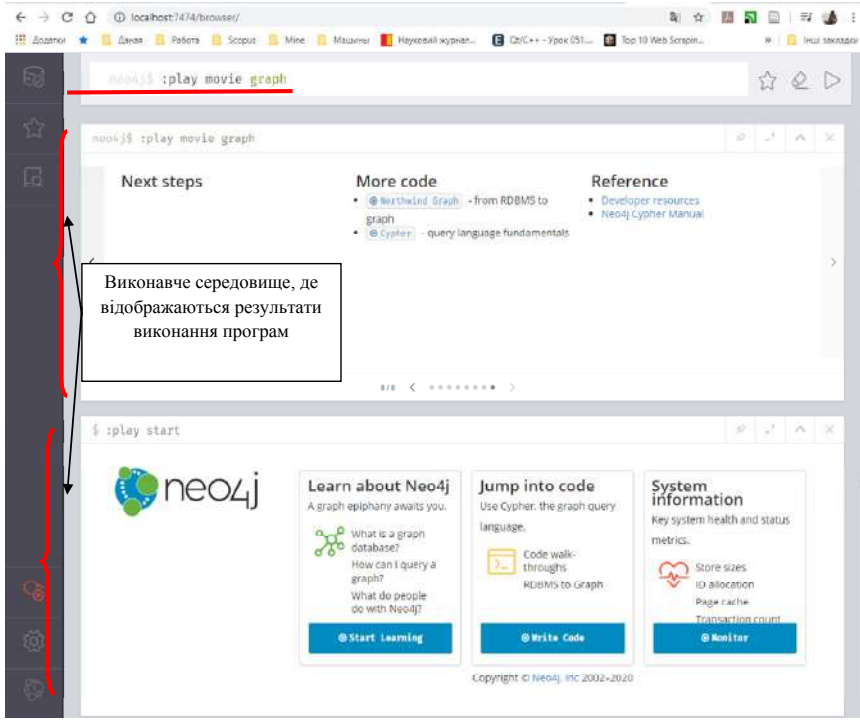
Для зупинки сервера у консолі потрібно вказати команду:

```
Cntr-C
```

Для ознайомлення із роботою Neo4j, розробниками пропонується два тестові набори даних, що вбудовані у Neo4j. Вони викликаються за допомогою команд:

```
:play movie graph
:play northwind graph
```

Запустимо перший набір, для цього у вікні браузера Neo4j треба ввести відповідну команду:



Завантажиться інтерактивний режим побудови графу «Movie», завдяки якому можна встановити зв'язки між акторами та фільмами, в яких вони знімались.

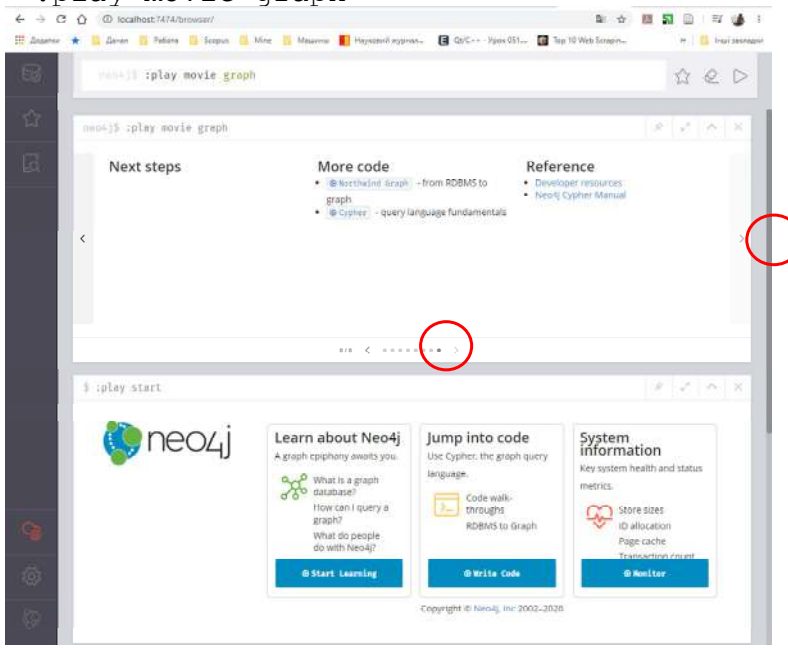
По-суті, Movie Graph — це міні-графічний додаток, що містить дані про акторів, режисерів і пов'язує їх із фільмами, у яких вони співпрацювали. За допомогою тренувального набору даних можна:

- ознайомитись із процедурою додавання даних фільму у графік (Create);
- здійснювати завантаження даних про окремі фільми та акторів (Find);
- створювати запити про зв'язок акторів і режисерів у рамках кінематографічної співпраці (Query);
- вирішувати пошукову задачу (Solve);
- сформувати певні рекомендації (Recommend);
- видалити дані (Clean up).

Розглянемо поетапно, як будується граф на прикладі задачі Movie Graph.

Для того, щоб перейти до навчання роботи із графом і тестовим набором даних, необхідно натисниснути на піктограму зі стрілочкою у право «Далі», що розміщена у виконавчому середовищі виконання команди:

```
:play movie graph
```



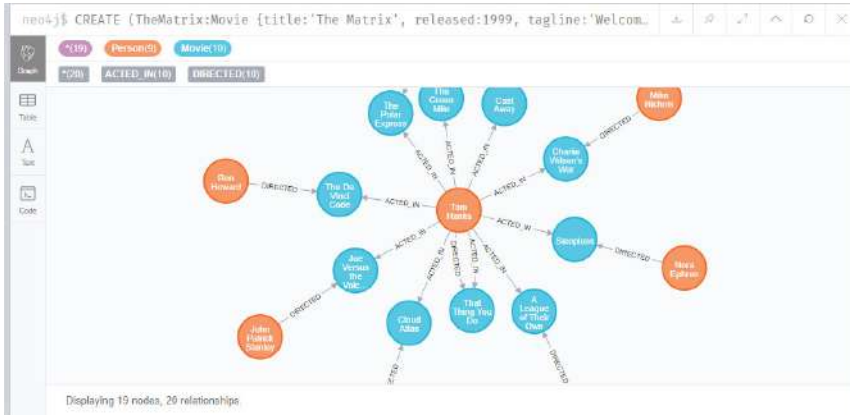
CREATE

У правій частині вікна представлено блок коду, що створює масив даних про фільми. На основі цих даних буде створено граф.

Для того, щоб запустити симуляцію побудови, необхідно виконати кілька дій:

- клікнути на блоці коду. Він копіювався у поле редактора команд у верхній частині вікна браузера;
- натиснути кнопку «Play» для виконання побудови браузера.

У результаті виконання команди отримали граф:



FIND

Розглянемо приклад для запитів пошуку окремих вузлів. Для цього необхідно:

- обрати один із запропонованих на екрані запитів;
- клікнути на ньому, він відобразиться у редакторі;
- натиснути кнопку «Play» для виконання побудови браузера.

Наприклад, у результаті виконання запиту:

```
MATCH (tom {name: «Tom Hanks»}) RETURN tom
```

отримали:

Тобто, вивели на екран дані із графової бази даних, до полю «name» відповідає імя «Tom Hanks». Як видно, одразу дані виведені на екран у вигляді графу. Але їх можна переглянути і в інших режимах:

– таблиці;
– тексту;
– коду.

Таким чином можна шукати дані по акторах, що містяться у базі даних.

QUERY

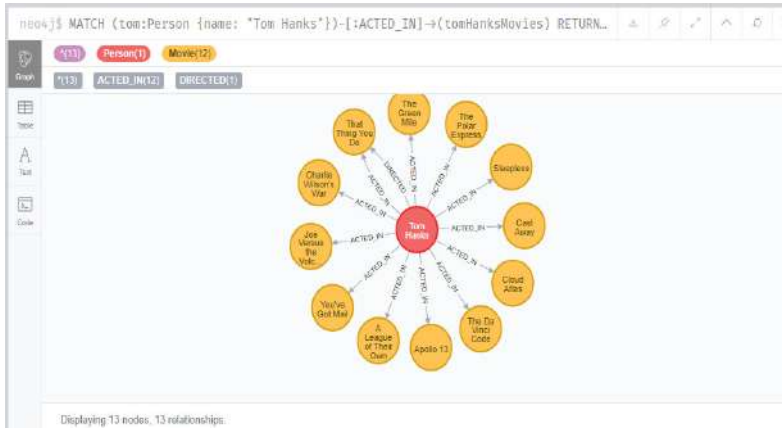
Тепер розглянемо складніші форми запитів.

Наприклад, знайти зв'язки, що об'єднують акторів і режисерів фільмів.

Так, у результаті виконання запиту:

```
MATCH (tom:Person {name: «Tom Hanks»})-[:ACTED_IN]->(tomHanksMovies) RETURN tom,tomHanksMovies
```

СКБД вивела на екран дані у вигляді графу про фільми, у яких знімався актор Tom Hanks, а також, де він виступав у якості режисера:



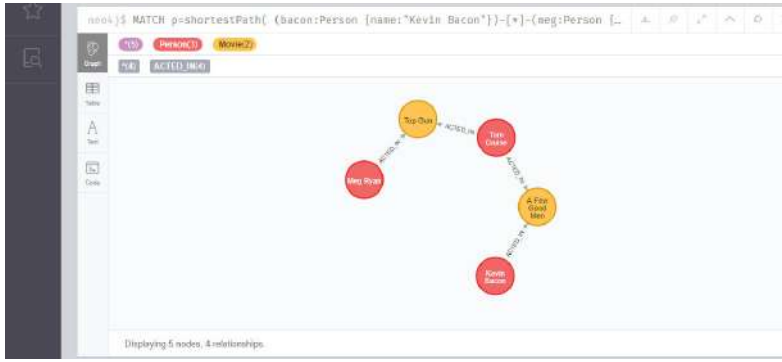
SOLVE

Є розважальна гра, яка має назву «Шість кроків до Кевіна Бейкона» (Six Degrees of Kevin Bacon). Гравці повинні не більше ніж за 6 переходів знайти зв'язок між задуманим актором і Кевіном Бейконом через акторів, разом з якими вони знімалися. У рамках даного навчального прикладу пропонується її спрощена варіація, яку автори назвали «Шлях Бекона» (Bacon Path).

Пропонується кілька запитів:

– запит, який шукає зв'язок між фільмами та акторами на відстані до 4 «ходів» від Кевіна Бекона;

– запит, який шукає зв'язок Кевіном Беконом і Мег Райан у найкоротший спосіб.



Наприклад, у результаті виконання другого запиту отримали граф, що показує зв'язок акторів Кевіна Бекона та Мег Райан через актора Тома Круза.

RECOMMEND

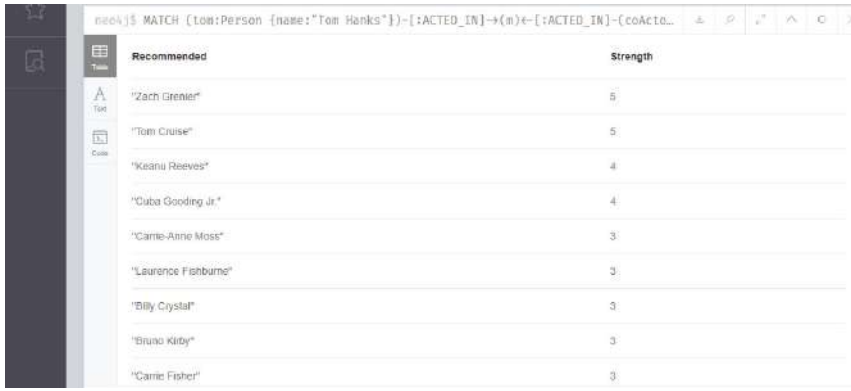
На основі достатньої вибірки можна знаходити зв'язки та формувати рекомендації. Наприклад, вирішити задачу рекомендації нових співавторів для фільмів Тома Хенкса («Tom Hanks»).

Суть підходу — пошук рекомендованих зв'язків, з якими Том не мав досвіду роботи, але із найближчого оточення, із тими особами, у яких у Тома можна простежити зв'язки.

Для Тома Хенкса це означає:

– потрібно знайти акторів, з якими він ще не працював, проте співпрацювали його партнери;

– знайти когось, хто може познайомити Тома зі потенційним співавтором.



Зокрема, в результаті виконання першого запиту, було встановлено, що найбільш щільний зв'язок у Тома Хенкса є з двома потенційними співавторами: Захом Гренієром і Томом Крузом.

CLEAN UP

Після того, як ви попрацювали із тестовими наборами даних, їх можна видалити.

Варто пам'ятати, що для видалення бази даних:

– вузли не можна видалити, якщо існують зв'язки;

– вузли можна видалити тільки разом із зв'язками.

Після закінчення роботи із графовою базою даних, необхідно на серверній частині Neo4j (у консолі) ввести команду:

```
Cntr-C
```

6.4. Завдання для опрацювання в лабораторній роботі

Для предметної області, з якою працювали в попередніх лабораторних роботах, необхідно запроектувати графову базу даних і реалізувати основні типи запитів. У тому разі, якщо специфіка предметної області не дозволяє сформувати графову модель БД, то із наведеного нижче списку оберіть нову предметну область.

Для вибраної предметної області побудуйте графову база даних, створивши вказані вузли та самостійно змоделюйте зв'язки між ними відповідними ребрами.

Перелік рекомендованих предметних областей:

1. **«Мережа аптек»** — має такі вузли: Аптека, Ліки, Бронювання, Виробник, Продажі, Залишки.

2. **«Транспортна компанія» —** має такі вузли: Транспортні_засоби, Клієнти_замовники, Пункти_доставки_вантажів, Вантаж, Замовлення. Компанія надає послуги по доставці вантажів повітряним, річним, морським, залізничним і автомобільним транспортом за різними маршрутам.

3. **«Видавництво»** — має такі вузли: Вид_видання, Автор, Замовник, Видання, Замовлення.

4. **«Мережа пунктів атосервісу»** — має такі вузли: Пункт_автосервісу, Вид_послуг, Співробітник, Клієнти, Послуги, Автомобілі. Кожен пункт автосервісу надає різні послуги з ремонту, фарбування, мийки, хімчистки, тюнінгу й т.п. автомобілів. Кожен співробітник спеціалізується на певних видах робіт. Автомобіль може мати потребувати надання кількох послуг.

5. **«Виробництво»** — має такі вузли: Виріб, Деталь, Матеріал, Станок, Витрати_Матеріал_Деталь. Кожен виріб складається з певної кількості деталей, на виготовлення деталі можуть витрачатися різні матеріали в різній кількості, деталь може проходити оброблення на різних станках з відповідно різними витратами часу.

6. **«Технологія»** — має такі вузли: Цех, Дільниця, Деталь, Станок. Кожен цех має кілька дільниць, які обладнані станками, на яких проходять оброблення деталі. Одна деталь може оброблятися на кількох станках з відповідно різними витратами часу.

7. **«Кафедра»** — має такі вузли: Викладач, Посада, Предмет, Тип_компетентності, Компетентність, Програмні_результати_начання, Спеціальність. Один викладач може викладати кілька предметів. Один предмет може викладатися на кількох спеціальностях, але з різною кількістю кредитів. Компетентності бувають: інтегральні, загальні та спеціальні (фахові). Кожен предмет

характеризується певним набором компетентностей і програмними результатами навчання.

8. **«Оператор мобільного зв'язку»** — має такі вузли: Види_послуг, Клієнт, Договір, Послуги, Тариф.

9. **«Мережа піцерій»** — має такі вузли: Піцерія, Товар, Клієнт, Замовлення, Співробітник. Піцерія приймає замовлення з доставкою товарів клієнту, яке виконують співробітники. Одне замовлення може включати кілька товарів.

10. **«Основні засоби»** — мають такі вузли: Обладнання, Місце_експлуатації, Переміщення, Ремонт, Простої, Амортизація.

11. **«Кінотеатр»** — має такі вузли: Жанр, Фільм, Зал, Сеанси, Продаж_квитків.

12. **«Бібліотека»** — має такі вузли: Вид_видання, Рубрика, Книга, Абонент, Видача_книг.

13. **«Поліклініка» —** має такі вузли: Профіль_лікаря, Лікарі, Пацієнти, Діагноз, Лікування.

14. **«Студентське містечко»** — має такі вузли: Гуртожиток, Особа, Кімната, Поселення, Виселення, Факультет.

15. **«Мережа обмінників валют»** — має такі вузли: Обмінний_пункт, Валюта, Курс, Купівля, Продаж.

16. **«Мережа автосалонів»** — має такі вузли: Регіон, Магазин, Менеджер, Автомобіль, Продажі.

17. **«Система on-line страхування»** — має такі вузли: Програма, Страховий_продукт, Клієнт, Страховий_поліс. Система On-line страхування характеризується наступним фрагментом атрибутів: код виду страхової програми, назва страхової програми (подорожі, житло), кожна страхова програма має певні кількість страхових продуктів, наприклад, страхування житла (квартири, будинку, дачі — від пожежі, затоплення, крадіжки), програма подорожі (медицина за кордоном, тривала робота за кордоном, відпочинок та спорт за кордоном, подорожі Україною) .

18. **«Мережа взуттєвих магазинів» —** має такі вузли: Магазин, Регіон, Відділ, Тип_взуття, Взуття, Продажі. У магазині є відділи взуття — чоловіче, жіноче, дитяче. Взуття поділяється на зимове, літнє та весна-осінь і має розмір, колір і ціну.

19. **«Мережа магазинів парфюмерії» —** має такі вузли: Клас_парфюмерії, Парфум, Клієнт, Тип_аромату. Парфюмерія може бути жіноча та чоловіча таких класів: органічна, нішева, натуральна, професіональна, елітна. Типи ароматів: амброві, зелені, кожані, морські, пряні, квіткові, цитрусові, пудрові.

20. **«Туристичне бюро»** — має такі вузли: Країна, Місто, Путівка, Клієнт, Готель. Клієнтам пропонуються путівки за різ-

ними програмами (відпочинок, екскурсії, туризм і т.п.), путівки різняться за ціною залежно від зірковості готелю та системи обслуговування.

21. **«Мережа On-line аптек»** — має такі вузли: Аптека, Група_препаратів, Препарати, Район, Клієнт, Замовлення. Аптеки приймають замовлення на лікарські препарати, які розбиті по групах: антибіотики, гормональні, гемеопатичні і т.п.

22. **«Каталог музичних треків» —** має такі вузли: Жанри, Групи, Виконавці, Альбоми, Треки.

23. **« Центр зайнятості»** — має такі вузли: Регіон, Роботодавець, Професії, Вакансії, Кандидат_на_працевлаштування, Заявка.

24. **«Авіаперевезення» —** має такі вузли: Аеропорт, Авіакомпанія, Замовник. Замовник замовляє перевезення вантажів, а авіакомпанія виконує авіаперевезання за вказаними маршрутами.

25. **«Управління проєктами»** — має такі вузли: Компанія, Проєкт, Співробітник, Спеціалізація. У кожній компанії є кілька співробітників. Кожен співробітник має певну спеціалізацію і працює над одним чи кількома проєктами. Може бути, що співробітники з різних компаній працюють над одним і тим же проєктом. Виявити співробітників, що мають одинакові спеціалізації. Виявити співтовариства людей, що працюють над однаковими проєктами і мають спільні інтереси.

26. **«Доставка посилок»** — має такі вузли: Посилка, Пункт_Призначення, Поштове_відділення, Вид_доставки. Вибирається найкоротший маршрут і залежно від його протяжності вибирається певний вид доставки. Це може бути доставка велосипедом чи автомобілем.

27. **«Портфіліо студента 6і01»** — має такі вузли: Студент, Знання та навички, Досягнення, Досвід роботи.

Виконуючи лабораторну роботу, слід використовувати приведену нижче коротку довідку по мові Cypher і користуватися наведеними джерелами в списку літератури до теми.

Коротка довідка по мові Cypher

У Cypher ідентифікатори вузлів, ребер та їх параметрів є регістрозалежними.

Ідентифікатори можуть містити букви, числа та символи підкреслювання, але повинні починатися з букви. Якщо ідентифікатор повинен містити інші символи, то його необхідно брати в лапки. Коментар добавляються через подвійний слеш.

Вузли записуються у круглих дужках, у яких вказуються додаткові характеристики, такі як мітка вузла чи його властивостями. Наприклад, (a:Actor) — вузол з міткою Actor.

Ребра записуються у виді пари квадратних дужок, по аналогії з вузлами в дужках задаються додаткові умови, наприклад, [r:Role] — зв'язок типу Role.

Зв'язки відображаються за допомогою двох тире з символами більше та менше, які вказують напрям зв'язку (-- > та < --). Напрям зв'язку задаєтеся стрілками таким чином:

- ()-[]->()
- ()<-[]-()
- ()-[]-() - для випадку, коли напрям зв'язку не важливий

У випадку, коли не потрібно накладати додаткові обмеження на зв'язок між вузлами, квадратні дужки можна пропустити в описі зв'язку: ()--()

Створення вузлів

Вузол створюється командою CREATE.

Синтаксис створення *одного вузла*:

```
CREATE (n)
```

0 рядків, створених вузлів: 1

Синтаксис створення *кількох вузлів*:

```
CREATE (n),(m)
```

0 рядків, створених вузлів: 2

Синтаксис створення *одного вузла з міткою:*

```
CREATE (n:Person)
```

0 рядків, створених вузлів: 1, добавлено міток: 1

Синтаксис створення *одного вузла з кількома мітками:*

```
CREATE (n:Person:German)
```

0 рядків, створених вузлів: 1, добавлено міток: 2.

Створення вузла з мітками і характеристиками (значеннями) полів:

```
CREATE (n:Person { name: 'Anton', specialty: 'Programmer' })
```

0 рядків, створених вузлів: 1, добавлено міток: 1, набір характеристик:2.

Створення ребер (відношень)

Створення зв'язку між двома вузлами. Спочатку створюються вузли, а потім відношення між ними:

```
MATCH (a:Person),(b:Person)
WHERE a.name = 'A' AND b.name = 'B'
CREATE (a)-[r:RELTYPE]->(b)
```

```
RETURN type(r)
```
Результат:

Тип (r)

“RELTYPE”

1 рядок, відношень створено: 1

Створення зв’язку та визначення характеристик (значень) ребра. Ця процедура аналогічна створенню вузлів з характеристиками. Значення характеристик ребра може бути будь-яким виразом. Наприклад:

```
MATCH (a:Person),(b:Person)
WHERE a.name = 'A' AND b.name = 'B'
CREATE (a)-[r:RELTYPE { name: a.name + '<->'
+ b.name }]->(b)
RETURN type(r), r.name
```
Результат:

Тип (r) r.name

1 рядок, створено відношень: 1 Набір характеристик: 1

"RELTYPE"

"A<->B"

Створення повного шляху. При використанні CREATE всі елементи шаблону будуть створені. Наприклад:

```
CREATE p =(andy { name:'Andy' })-
[:WORKS_AT]->(neo)<-[:WORKS_AT]-(michael { 
name: 'Michael' })
RETURN p
```
У результаті виконання буде створено три вузли і два ребра, які будуть присвоєні змінній шляху.

Результат:

1 рядок, створено вузлів: 3 створено відношень: 2 Встановлено характеристик: 2

(2)-[WORKS_AT,0]->(3)<-[WORKS_AT,1]-(4)

Створення шаблонів

Основою Cypher є пошук за заданим шаблоном. Для розуміння поняття шаблон розглянемо основні форми, які можна представити в шаблоні.

Шаблон для одного вузла, який названий за допомогою змінної *a* має такий вид:

```
(a)
```
Шаблон опису вузла з їх властивостями. Властивості, тобто характеристики, вузла описуються у фігурних дужках через кому. Наприклад, опис вузла з двома властивостями:

```
(a {name: 'Anry', sport: 'Football})
```

Шаблони для зв'язаних вузлів: (a)-->(b). Цей шаблон описує два вузли *a* та *b*, а також відношення між ними, що вказує на орієнтоване ребро від *a* до *b*. Опис вузлів і відношень між ними для довільної кількості вузлів і відношень буде мати такий вид:

```
(a)-->(b)<--(c)
```

Слід звернути увагу, що іменування вузлів у таких шаблонах виконується лише у випадках, коли ще будуть звертання в запиті до цього вузла. Якщо такі звертання відсутні, то ім'я можна пропустити, як ще показано далі:

```
(a)-->()<--(c)
```

Шаблони для міток. У шаблонах вузлів можна описувати мітки. Шаблон опису вузла з однією міткою:

```
(a:User)-->(b)
```

Шаблон опису вузла з двома мітками:

```
(a:User:Admin)-->(b)
```

Шаблони для ребер. Ребрам, як і вузлам можуть присвоюватися імена. Ім'я ребра береться у квадратні дужки і стає по середині стрілки, що вказує на зв'язок між вузлами. Наприклад:

```
(a)-[r]->(b)
```

Ребра можуть мати типи. Шаблон опису ребра з певним типом має вид:

```
(a)-[r:REL_TYPE]->(b)
```

Як і з вузлами ім'я ребра може бути пропущене, тоді шаблон представляється так:

```
(a)-[:REL_TYPE]->(b)
```

Слід звернути увагу на той момент, що ребро може мати лише один тип. Проте, якщо є необхідність використання кількох типів, то всі вони мають бути перерахованими в шаблоні з використанням знаку 1 для їх розподілу:

```
(a)-[r:TYPE1|TYPE2]->(b)
```

Така форма представлення шаблону з кількома типами ребра не працює з CREATE та MERGE. Працює з MATCH для опису існуючих даних.

Замість того щоб описувати довгий шлях, перераховуючи в шаблоні всі вузли та ребра, іноді достатньо вказати його довжину.

Наприклад:

```
(a)-[*2]->(b)
```

Це описує три вузла та два ребра (довжина шляху 2), що є еквівалентом такого шаблону:

```
(a)-->()-->(b)
```

Можна, також, вказати діапазон довжин: такі шаблони називають «відношеннями змінної довжини». Наприклад:

```
(a)-[*3..5]->(b)
```

Мінімальна довжина 3 і максимальна 5. Шаблон описує графік з 4 вузлів і 3 взаємозв'язків, 5 вузлів і 4 взаимозв'язків чи 6 вузлів і 5 взаємозв'язків, які об'єднані разом в один шлях.

Будь-яка межа може бути пропущеною. Наприклад, описати шлях довжиною 3 чи більше можна таким чином:

```
(a)-[*3..]->(b)
```

Шлях довжиною 5 чи менше описується:

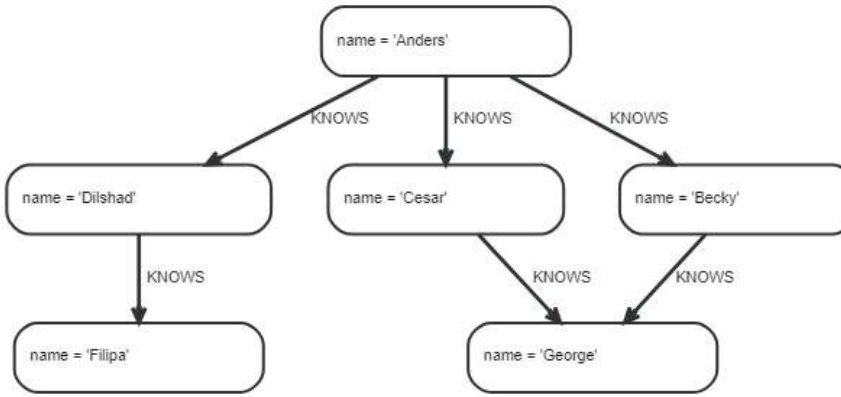
```
(a)-[*..5]->(b)
```

Обидві межі можуть бути пропущені, ч=що дозволяє описувати шляхи будь якої довжини:

```
(a)-[*]->(b)
```

Слід зауважити, що відношення змінної довжини не можна використовувати у CREATE та MERGE.

Приклад запиту з відношенням змінної довжини наведено нижче, він виконується на прикладі графу [5]:




```
MATCH (me)-[:KNOWS*1..2]-(remote_friend)
WHERE me.name = 'Filipa'
RETURN remote_friend.name
```

Результат:

remote_friend.name

2 ряда

"Dilshad"

"Anders"

Основні ключові слова для вибірки:

- MATCH — ключове слово, після якого вказується шаблон, який описує необхідну інформацію для пошуку;
- WHERE — додаткова умова чи обмеження (фільтр), які накладаються на шаблон;
- RETURN — ключове слово, яке визначає результат виконання запиту.

6.5. Питання для самоконтролю

1. Який тип графів підтримує СКБД Neo4j?
2. У чому полягає відмінність графів з властивостями від RDF-графів?
3. Характеризуйте СКБД Neo4j та її специфічні особливості які є відмінними від інших баз даних NoSQL.
4. Які версії підтримує СКБД Neo4j?
5. Дайте визначення вузла та ребра в СКБД Neo4j.
6. Яке призначення в СКБД Neo4j міток?
7. В якому вигляді можна переглянути дані в СКБД Neo4j?
8. Яке призначення патернів (шаблонів) у СКБД Neo4j ?
9. Характеризуйте пошук у глибину.
10. Характеризуйте пошук у ширину.

6.6. Література до теми

1. What is Neo4j? / Офіційний портал компанії Neo4j, Inc [Електронний ресурс]. URL: https://neo4j.com/
2. Graph Databases for Beginners: Why Graph Technology Is the Future / Офіційний портал компанії Neo4j, Inc [Електронний ресурс]. URL: https://neo4j.com/blog/why-graph-databases-are-the-future/
3. What is a Graph Database? Офіційний портал компанії Neo4j, Inc [Електронний ресурс]. URL: https://neo4j.com/developer/graph-database/
4. Robinson Ian, Webber Jim, Eifrem Emil. Graph Databases: New Opportunities for Connected. 2nd ed. O'Reilly Media, Inc., 2015. 236 p. [On-line]. URL: https://www.oreilly.com/library/view/graph-databases-2nd/9781491930885/
5. Patterns / Neo4j Cypher Manual / Офіційний портал компанії Neo4j, Inc [Електронний ресурс]. URL: https://neo4j.com/docs/cypher-manual/current/syntax/patterns/

Тема 7.

РОБОТА З ГРАФОВОЮ БАЗОЮ ДАНИХ В СЕРЕДОВИЩІ СКБД SQL Server 2017

7.1. Перелік знань і вмінь

Під час вивчення даної теми студенти знайомляться з концепцією побудови графових баз даних NoSQL ув середовищі СКБД SQL Server 2017. Вивчать основні елементи графової моделі БД: вузол, ребра та їх типи, відношення як властивості ребер, відношення як властивості вузлів. Отримають базові навички створення графових баз даних.

Вивчивши матеріал теми, ви будете ***знати****:*

- основи побудови графової бази даних та її використання;
- оператори мови T-SQL для створення таблиць вузлів і таблиць ребер;
- оператори реалізації запитів запитів до графої бази даних.

Вивчивши матеріал теми, ви будете ***вміти****:*

- створювати таблиці вузлів у середовищі СКБД Microsoft SQL Server 2017;
- створювати таблиці ребер у середовищі СКБД Microsoft SQL Server 2017;
- створити запити до графової бази даних з використанням засобів мови T-SQL.

7.2. Термінологічний словник

Граф — це мережева структура, що містить набір вузлів (вершин) і ребер (дуг), що відображають певні взаємозв'язки між ними. Вузол — це певна сутність предметної області, яка характеризує, наприклад, якийсь об'єкт чи певну особу та представлена набором атрибутів, а ребро характеризує відношення між сутностями, що їх об'єднує.

Графова база даних у середовищі СКБД Microsoft SQL Server — це користувацька база даних, що повністю інтегрована в основне ядро SQL Server і створюється стандартними способами за допомогою таблиць. Завдячуючи такій інтеграції графова база даних може користуватись широким спектром компонентів SQL Server, як то індекси, служби машинного навчання та ін. Графові бази даних SQL Server підтримує починаючи з версії Microsoft SQL Server 2017.

Таблиці вузли графової бази даних у SQL Server 2017 відображають деякі сутності предметної області, повинні обов'язково мати первинний ключ та параметр **AS NODE** (тобто вузол).

Таблиці ребер містять інформацію про відношення та взаємозв'язки між двома вузлами графа, обов'язково повинні мати параметр **AS EDGE** (тобто ребро) та ідентифікатори таблиць вузлів, які вони об'єднують. При необхідності таблиця ребер може вміщувати додаткові атрибути. Наприклад при моделюванні в транспортних системах, таблиці ребер можуть містити дані про відстань чи час руху між двома вузлами і т.п. Використання таблиць ребер дозволяє ідентифікувати напрямок зв'язку та співвідношення «багато до багатьох» між вузлами.

7.3. Інструктивні та методичні вказівки для виконання лабораторної роботи №8 «Створення та робота з графовою базою даних NoSQL в середовищі СКБД Microsoft SQL Server 2017»

Мета роботи: навчитись створювати графову базу даних NoSQL засобами СКБД Microsoft SQL Server 2017. Набути практичних навичок створення таблиць вузлів і таблиць ребер і реалізації запитів до графової бази даних за допомогою операторів мови Transact-SQL.

Завдання роботи

1. Вивчити основи побудови графових моделей бази даних NoSQL у середовищі реляційної СКБД і характеристики основних її компонент.
2. Вивчити оператори мови T-SQL для створення таблиць вузлів і таблиць ребер і реалізації запитів.
3. Створити таблиці вузлів у середовищі СКБД Microsoft SQL Server 2017.
4. Створити таблиці ребер у середовищі СКБД Microsoft SQL Server 2017;.
5. Створити запити до графової бази даних з використанням засобів мови T-SQL.
6. Відповісти на питання, поставлені в роботі.
7. Оформити звіт з лабораторної роботи.

Інструкція зі створення та роботи з графовою базою даних в середовищі СКБД Microsoft SQL Server 2017»

Створення графової бази даних розглянемо на прикладі графа, що характеризує клуб «Книголюб» і відображений на рис. 7.1.

Даний граф містить 3 вузли (книга, жанр і читач) та 3 ребра. Ребро з іменем belongs характеризує належність книги до певного жанру, ребро — Like1 визначають наступне відношень між вузлами- читачу подобається книга, а ребро - Like2 — читачу подобається певний жанр.

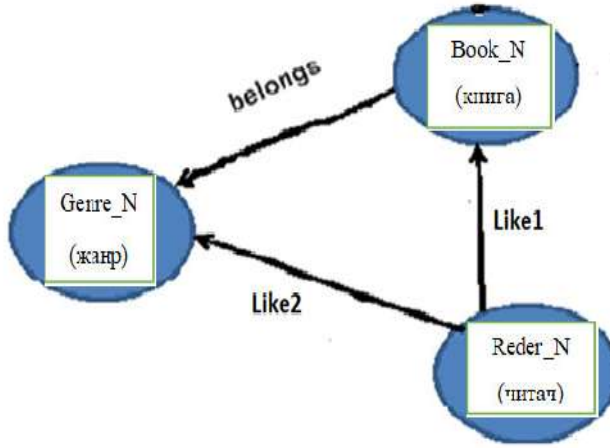


Рис. 7.1. Приклад графа для реалізації у лабораторній роботі

Створення графової бази даних

Для створення графової бази в середовищі СКБД Microsoft SQL Server 2017 не потрібні якісь спеціальні прийоми, чи налаштування для підтримки графа. Графова база даних створюється стандартним способом, так як це робиться і при створення класичної реляційної бази даних. Створення графової бази на мові T-SQL–базовий варіант:

```
USE master
GO
GREATE DATABASE <ім'я графової БД >
GO
```

Синтаксис операторів для створення графової бази **GRAF_Book** даних має такий вид:

```
USE [master]
GO
CREATE DATABASE [GRAF_Book]
GO
```

Таким чином, графова база даних — це просто логічна конструкція, яка визначена в користувацькій БД.

Створення таблиць графової бази даних

Для моделювання графів ву SQL Server 2017 використовуються таблиці двох спеціальних видів. Один вид для зберігання даних вузлів, а другий для відображення ребер (зв'язків).

Таблиця вузлів — відображає певну сутність предметної області, що моделюється графом. Таблиця вузол повинна обов'язково містити первинний ключ та ряд атрибутів, що характеризують сутність. Крім того така таблиця повинна мати параметр **AS NODE** (тобто вузол)**.** Коли вказано цей параметр, то ядро бази даних ідентифікує таблицю як вузол графа і автоматично добавляє у таблицю два псевдостовпчики: **graph_id_ <*hex_string*>** і **$node_id_ <*hex_string*>** і створює унікальний ідентифікатор для стовпчика **$ node_id**. Ядро бази даних використовує перший стовпчик для внутрішніх операцій, а другий стовпчик робить доступним для зовнішнього доступу. Стовпчик **$ node_id** зберігає унікальний ідентифікатор вузла, який представлений ву формате JSON. Шістнадцятирічний суфікс в імені стовпчика робить ім'я стовпчика глобально унікальним. Якщо вибираємо поле $ node id з таблиці, ім'я стовпчика буде мати такий вигляд:

```
$ node id  \ hex string
```

Користувачам рекомендується визначити унікальне обмеження чи індекс для стовпчика $ node_id під час створення таблиці вузла, але якщо це не вказано, то автоматично створюється унікальний некластеризований індекс по замовчуванню.

Таблиця ребер — відображає відношення та взаємозв'язки між двома вузлами графа. Ребра в графі завжди направлені (орієнтовані) і об'єднують два вузли. Таблиця ребер надає можливість моделювання відношень «Б:Б — багато до багатьох» між двома вузлами. Таблиця ребер повинна обов'язково містити параметр **AS EDGE** (тобто ребро) та ідентифікатори таблиць вузлів, які вона об'єднує. При потребі до таблиці ребра може включати інші атрибути, що характеризують та розкривають семантику відношення між сутностями і є необов'язковими. При створенні таблиці ребра автоматично створюється ряд стовпчиків, серед яких слід виділити ті, що представляють інтерес для розробників, а саме такі три стовпчики: **$edge_id, $from_id, $to_id**.

Стовпчик **$ edge_id_ <hex_string>** — однозначно ідентифікує ребро.

Стовпчик **$ from_id_ <hex_string>** — зберігає ідентифікатор **($ node_id)** таблиці-вузла, з якого виходить ребро.

Стовпчик **$to_id_ <hex_string> —** зберігає ідентифікатор **($node_id)** таблиці-вузла, в яку входить ребро.

Ядро бази даних автоматично генерує значення стовпчика **$edge_id**, що зберігає ідентифікатор вузла, аналогічно як це робилося для стовпчика **$ node_id** таблиці-вузла.

Створення таблиць для зберігання даних вузлів

При створенні таблиці-вузла важливим є параметр **AS NODE** (тобто вузол)**.** Коли вказано цей параметр, то ядро бази ідентифікує таблицю, як вузол графа і добавляє у таблицю два псевдостовпчики. Таблиці-вузол завжди повинна містити первинний ключ.

Синтаксис створення таблиці-вузла:

```
USE <ім'я бази даних>
 GO
 CREATE TABLE <ім'я таблиці-вузла>
(<поле- ID> <тип ID > PRIMARY KEY,
 < ім'я поля1> <тип поля1 > ,
 < ім'я поля2> <тип поля2> ,
……
 < ім'я поля N> <тип поляN >)
 AS NODE;
```

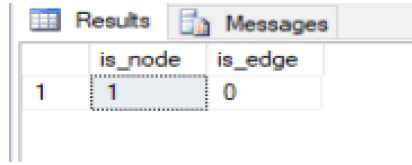
Синтаксис створення таблиці-вузла GENRE_N, яка містить такі атрибути (GenrenID, Name_G) має вид:

```
CREATE TABLE GENRE N
 (GenrenID int identity(1,1) primary key NOT
NULL,
  Name G varchar(50) Not NuLL)
  AS NODE
  GO
```

Перевірити створення таблиці вузла можна за допомогою такого запиту:

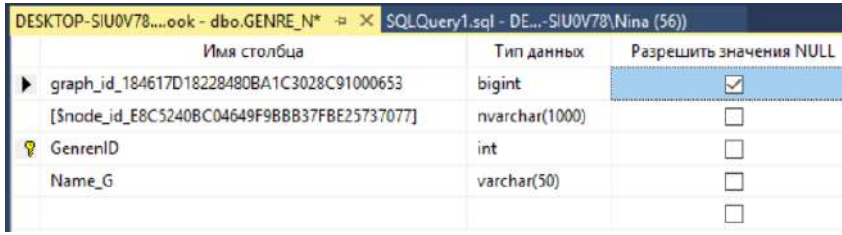
```
SELECT is node, is edge FROM sys.tables
 WHERE name = 'GENRE N'
```

Результатом цього запиту буде таке повідомлення, яке свідчить про створення таблиці — вузла.

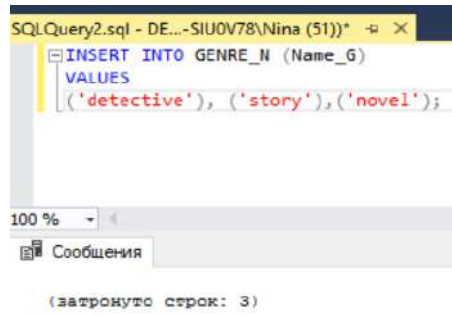


Для таблиць вузлів, значення стовпчика is node встановлюється в 1 , а значення стовпчика is_edge встановлюється в 0. Для таблиць ребер, значення міняються місцями.

Після виконання оператора CREATE в таблицю GENRE_N автоматично добавились два псевдостовпчики, які ідентифікують граф і таблицю вузол:



Завантаження даних до таблиці вузла виконується оператором INSERT. Приклад завантаження даними таблиці вузла GENRE_N і його результат представлено нижче:



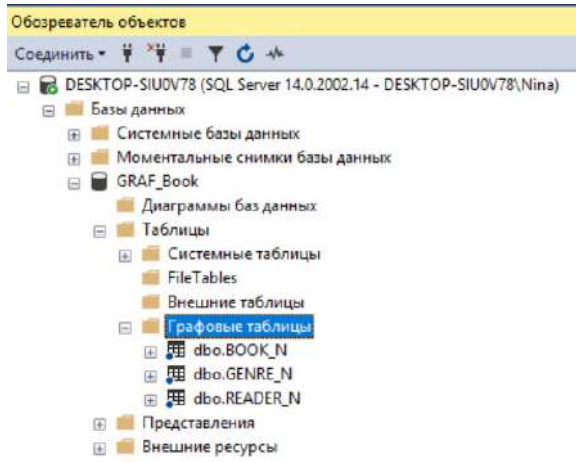
Результат виконання запиту завантаження даних до таблиці вузла GENRE_N:

Аналогічно створюються і завантажуються дані в таблиці вузли READER_N і BOOK_N. Завантажені дані в таблицю вузол READER_N наведені на рисунку нижче:

DESKTOP-SIU0V78....ok - dbo.READER_N | SQLQuery1.sql - lo...-SIU0V78\Nina (54))*

	$node_id_D908D716FC134D27BD16D97AC373B034	ReaderID	Name_R
▶	READER_N","id":0}	1	Krut
	{"type":"node","schema":"dbo","table":"READER_N","id":1}	2	Dmitrov
	{"type":"node","schema":"dbo","table":"READER_N","id":2}	3	Strokov
	{"type":"node","schema":"dbo","table":"READER_N","id":3}	4	Kirpa
*	NULL	NULL	NULL

Створені таблиці вузли відображаються як графові таблиці в оглядачу об'єктів:



Таблиці вузлів Reader і Book завантажуються тестовими даними. Далі наведено тестові дані, що завантажені до таблиці BOOK_N і READER_N.

Запит та тестові дані таблиці вузла BOOK_N:

SQLQuery1.sql - lo...-SIU0V78\Nina (57))* | DESKTOP-SIU0V78....ok - dbo.READER_N

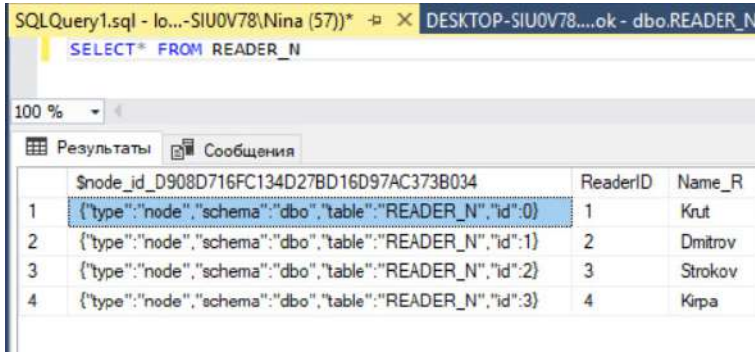
```
SELECT* FROM BOOK_N
```

100 %

Результаты | Сообщения

	$node_id_9AFEA575FAA84D8AB7180D8803F1E434	BookID	Name_B
1	{"type":"node","schema":"dbo","table":"BOOK_N","id":0}	1	Jacgueline Winspire Single Berry Field
2	{"type":"node","schema":"dbo","table":"BOOK_N","id":1}	2	Carol Hart Letters from Houme
3	{"type":"node","schema":"dbo","table":"BOOK_N","id":2}	3	O. Henry Stories
4	{"type":"node","schema":"dbo","table":"BOOK_N","id":3}	4	Tomas Mann Payat
5	{"type":"node","schema":"dbo","table":"BOOK_N","id":4}	5	Lyubko Deresh Cult
6	{"type":"node","schema":"dbo","table":"BOOK_N","id":5}	6	Andrey Kurkov Picnic on the ice
7	{"type":"node","schema":"dbo","table":"BOOK_N","id":6}	7	Gillian Flinn Sharp objects

Запит та тестові дані таблиця вузла READER_N.



	$node_id_D908D716FC134D27BD16D97AC373B034	ReaderID	Name_R
1	{"type":"node","schema":"dbo","table":"READER_N","id":0}	1	Krut
2	{"type":"node","schema":"dbo","table":"READER_N","id":1}	2	Dmitrov
3	{"type":"node","schema":"dbo","table":"READER_N","id":2}	3	Strokov
4	{"type":"node","schema":"dbo","table":"READER_N","id":3}	4	Kirpa

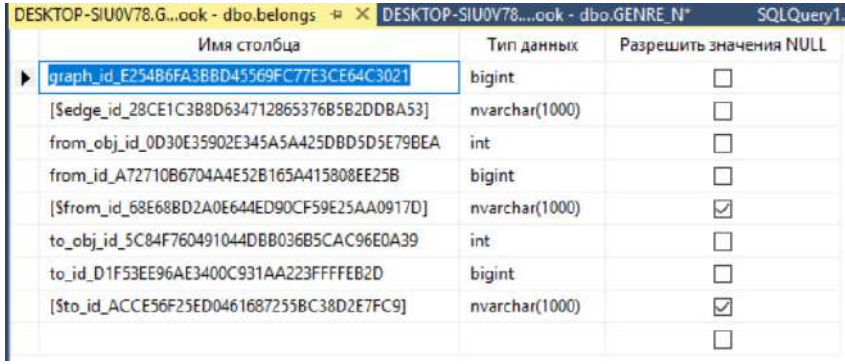
Створення таблиць ребер

Створення таблиці ребер аналогічно створенню таблиці вузлів, але замість **AS NODE** потрібно вказувати параметр **AS EDGE** . Первинний ключ у таблиці ребер є необов'язковим. Обов'язково потрібно вказувати ідентифікатор таблиці вузла, з якого виходить ребро, та ідентифікатор таблиці-вузла, в який ребро входить. Якісь додаткові користувацькі атрибути, що характеризують ребро теж є необов'язковими.

Запит створення таблиці ребра **Belongs** має такий синтаксис :

```
CREATE TABLE belongs
AS EDGE
GO
```

У результаті виконання запиту створюється таблиця ребра **Belongs,** структура якої представлена нижче:



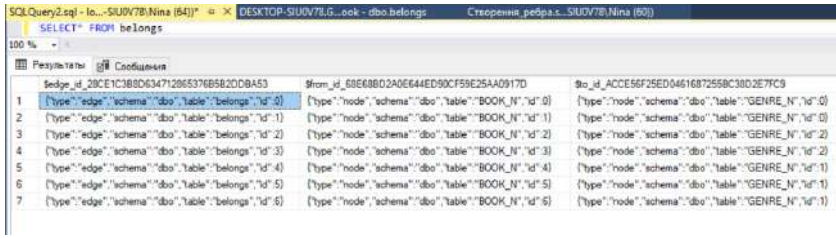
Имя столбца	Тип данных	Разрешить значения NULL
graph_id_E254B6FA3BBD45569FC77E3CE64C3021	bigint	☐
[$edge_id_28CE1C3B8D634712865376B5B2DDBA53]	nvarchar(1000)	☐
from_obj_id_0D30E35902E345A5A425DBD5D5E79BEA	int	☐
from_id_A72710B6704A4E52B165A415808EE25B	bigint	☐
[$from_id_68E68BD2A0E644ED90CF59E25AA0917D]	nvarchar(1000)	☑
to_obj_id_5C84F760491044DBB036B5CAC96E0A39	int	☐
to_id_D1F53EE96AE3400C931AA223FFFFEB2D	bigint	☐
[$to_id_ACCE56F25ED0461687255BC38D2E7FC9]	nvarchar(1000)	☑
		☐

Наступним кроком є створення запиту для завантаження тестових даних до таблиці ребра belongs.

Фрагмент синтаксису запиту INSERT з підзапитами визначає взаємозв'язок між таблицями вузлами та заносить у поле $from_id ідентифікатор вузла BOOK_N, з якого виходить ребро за умови BookID=1, та в поле $to_id — ідентифікатор вузла GENRE_N за умови GenrenID=1:

```
INSERT INTO belongs ($from id,$to id) VALUES
(( SELECT $node id FROM BOOK N WHERE BookID=1),
(SELECT $node_id FROM GENRE_N WHERE
GenrenID=1))
```

Запит вибірки та тестові дані таблиці ребра belongs має вигляд:



Аналогічно створюються і завантажуються тестовими даними наступні таблиці ребра like1 і like2. Далі наведено лістінг завантаження даними таблицю ребро like1:

```
INSERT INTO like1 ($from id,$to id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=1),
(SELECT $node id FROM BOOK N WHERE BookID=1))
INSERT INTO like1 ($from id,$to id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=2),
(SELECT $node id FROM BOOK N WHERE BookID=2))
INSERT INTO like1 ($from id,$to id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=3),
(SELECT $node id FROM BOOK N WHERE BookID=3))
INSERT INTO like1 ($from id,$to id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=4),
(SELECT $node id FROM BOOK N WHERE BookID=4))
INSERT INTO like1 ($from id,$to id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=1),
```

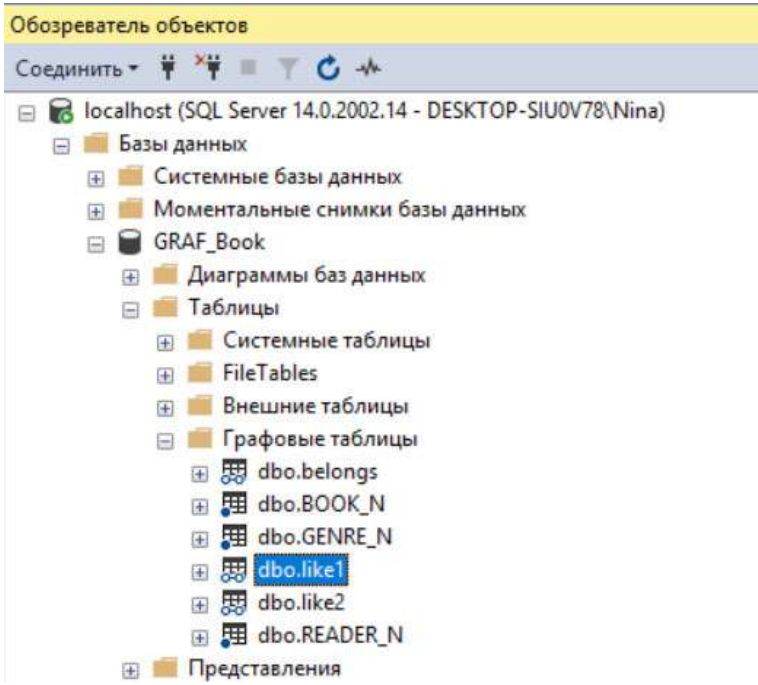
```
(SELECT $node_id FROM BOOK_N WHERE BookID=5))
INSERT INTO like1 ($from_id,$to_id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=2),
(SELECT $node_id FROM BOOK_N WHERE BookID=7))
INSERT INTO like1 ($from_id,$to_id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=3),
(SELECT $node_id FROM BOOK_N WHERE BookID=7))
```

Лістінг завантаження даними таблиці ребро like2 має вид:

```
INSERT INTO like2 ($from_id,$to_id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=1),
(SELECT $node_id FROM GENRE_N WHERE
GenrenID=1))
INSERT INTO like2 ($from_id,$to_id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=2),
(SELECT $node_id FROM GENRE_N WHERE
GenrenID=1))
INSERT INTO like2 ($from_id,$to_id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=3),
(SELECT $node_id FROM GENRE_N WHERE
GenrenID=3))
INSERT INTO like2 ($from_id,$to_id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=4),
(SELECT $node_id FROM GENRE_N WHERE
GenrenID=3))
INSERT INTO like2 ($from_id,$to_id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=1),
(SELECT $node_id FROM GENRE_N WHERE
GenrenID=2))
INSERT INTO like2 ($from_id,$to_id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=2),
(SELECT $node_id FROM GENRE_N WHERE
GenrenID=2))
```

```
INSERT INTO like2 ($from id,$to id) VALUES ((
SELECT $node_id FROM READER_N WHERE
ReaderID=3),
(SELECT $node_id FROM GENRE_N WHERE
GenrenID=2))
```

Усі таблиці ребра та вузли графа будуть відображені в оглядачі об'єктів:



Внесення змін до бази даних графів

Внесення змін до графових баз даних подібно до внесення змін у реляційні бази даних, проте є певні особливості.

Операція вилучення має особливості, коли вона стосується псевдостовпчиків, що автоматично визначаються СКБД і дозаписуються в таблиці вузлів і ребер, а саме стовпчиків: **$node_id, $edge_id, $from_id, $to_id**. Вилучимо дані, що були записані в таблицю ребер **belongs** таким оператором:

```
INSERT INTO belongs ($from id,$to id) VALUES ((
SELECT $node_id FROM BOOK_N WHERE BookID=8),
```

```
(SELECT $node_id FROM GENRE_N WHERE
GenrenID=1))
```

1-й спосіб вилучення — використовується оператор DELETE з використанням підзапиту в умові WHERE:

```
DELETE belongs WHERE $from id= (SELECT $node_id
FROM BOOK N WHERE BookID=8) AND $to id=
(SELECT $node id FROM GENRE N WHERE GenrenID=1)
```

2-й спосіб вилучення — з використанням функції MATCH:

```
DELETE belongs
 FROM BOOK N, belongs, GENRE N
 WHERE MATCH (BOOK N-( belongs)->GENRE N) AND
BOOK_N.BookID=1 AND GENRE_N.GenrenID=1
```

3-й спосіб вилучення — без використання функції MATCH на основі традиційного об'єднання двох таблиць і одного вузла:

```
DELETE belongs FROM belongs INNER JOIN BOOK N
ON belongs.$from id=BOOK N.$node id INNER JOIN
GENRE N ON belongs.$to id=GENRE N.$node id
WHERE BOOK_N.BookID=1 AND GENRE_N.GenrenID=1
```

Вилучення рядка даних з таблиці вузла BOOK_N:

```
DELETE FROM BOOK_N WHERE BookID=8
```

Операція оновлення в графовій базі даних має деякі специфічні особливості. А саме, не можна вносити зміни до стовпчиків, що автоматично формуються СКБД. Тобто не можна змінювати значення стовпчиків: **$node_id, $edge_id, $from_id, $to_id**. Вносити зміни до інших стовпчиків можна з використанням оператора UPDATE.

Базовий синтаксис команди UPDATE має такий вид:

```
UPDATE ім'я таблиці
SET поле= вираз1 [, …, полеN = виразN]
[WHERE умова]
```

Приклад внесення змін до таблиці вузла `GENRE_N` наведений нижче:

```
UPDATE GENRE N
SET Name G='phantasy'
WHERE GenrenID=3
```

Створення запитів до графової бази даних

Запити до графової бази даних можна створювати з використанням оператора SELECT і функції MATCH. Синтаксис функції MATCH має такий вигляд:

```
MATCH (<graph_search_pattern>)
<graph_search_pattern>::=
 {
 <simple_match_pattern>
 | <arbitrary_length_match_pattern>
 |
<arbitrary_length_match_last_node_predicate>
 }
<simple_match_pattern>::=
 {
 LAST_NODE(<node_alias>) | <node_alias> {
 { <-( <edge_alias> )- }
 | { -( <edge_alias> )-> }
 <node_alias> | LAST_NODE(<node_alias>)
 }
 }
 [ { AND } { ( <simple_match_pattern> ) } ]
 [ ,...n ]
<node_alias> ::=
 node_table_name | node_table_alias
<edge_alias> ::=
 edge_table_name | edge_table_alias
<arbitrary_length_match_pattern> ::=
 {
 SHORTEST_PATH(
 <arbitrary_length_pattern>
 [ { AND } { <arbitrary_length_pattern> } ]
 [ ,…n]
 )
 }

<arbitrary_length_match_last_node_predicate>
::=
 { LAST_NODE( <node_alias> ) = LAST_NODE(
<node_alias> ) }
<arbitrary_length_pattern> ::=
 { LAST_NODE( <node_alias> ) | <node_alias>
```

```
 ( <edge_first_al_pattern>
[<edge_first_al_pattern>…,n] )
<al_pattern_quantifier>
 }
 | ( {<node_first_al_pattern>
[<node_first_al_pattern> …,n] )
<al_pattern_quantifier>
 LAST_NODE( <node_alias> ) | <node_alias>
 }
<edge_first_al_pattern> ::=
 { (
 { -( <edge_alias> )-> }
 | { <-( <edge_alias> )- }
 <node_alias>
 )
 }
<node_first_al_pattern> ::=
 { (
<node_alias>
 { <-( <edge_alias> )- }
 | { -( <edge_alias> )-> }
 )
 }
<al_pattern_quantifier> ::=
 {
 +
 | { 1 , n }
 }
n - positive integer only.
```

Аргументи

graph_search_pattern — задає шаблон для пошуку чи шляху для проходу в графі. Для проходу по графу використовує синтаксис ASCII-арта. Шаблон вказує перехід від одного вузла до другого у напрямку, на який вказує стрілка. Імена чи псевдоніми ребер зазначаються у дужках. Імена вузлів чи псевдоніми відображаються з двох кінців стрілки. Стрілка може вказувати будь-який напрям у шаблоні.

node_alias — ім’я чи псевдонім таблиці вузла, яке вказане в FROM.

edge_alias — ім’я чи псевдонім таблиці ребра, яке вказане в FROM.

SHORTEST_PATH — ця функція використовується для пошуку найкоротшого шляху між двома заданими вузлами в графі чи між певним вузлом та всіма іншими вузлами в графі. Вхідними даними буде шаблон довільної довжини пошук якого повторно виконується в графі.

arbitrary_length_match_pattern — вказує на вузли та ребра, які потрібно повторно обходити, поки не буде знайдено потрібний вузол чи досягнуть максимальне число ітерацій, вказаних у шаблоні.

al_pattern_quantifier — шаблон довільної довжини, який містить квантифікатор шаблона за стилем подібні до регулярного виразу, щоб вказувати на кількість повторів у шаблоні пошуку. Підтримуються наступні квантифікатори шаблону пошуку:

- + : 1 чи більше повторів шаблону. Дія припиняється при находженні найкоротшого шляху;
- {1,n} : від 1 до "n" повторів шаблону. Дія припиняється при находженні найкоротшого шляху.

Основний шаблон для навігації по вузлах графа має вид:

```
(<вузол, з якого виходить ребро>) - (<ребро>) ->
 (<вузол, в який входить ребро> )
```

Шаблон може визначати один чи кілька взаємозв'язків. Для кожного зв'язку потрібно ідентифікувати початковий і кінцевий вузли, а також ребро, яке зв'язує два вузли разом і вказувати напрямок зв'язку між вузлами. Якщо необхідно визначати один зв'язок, який починається з вузла 1 і закінчується вузлом 2, потрібно використовувати такий синтаксис:

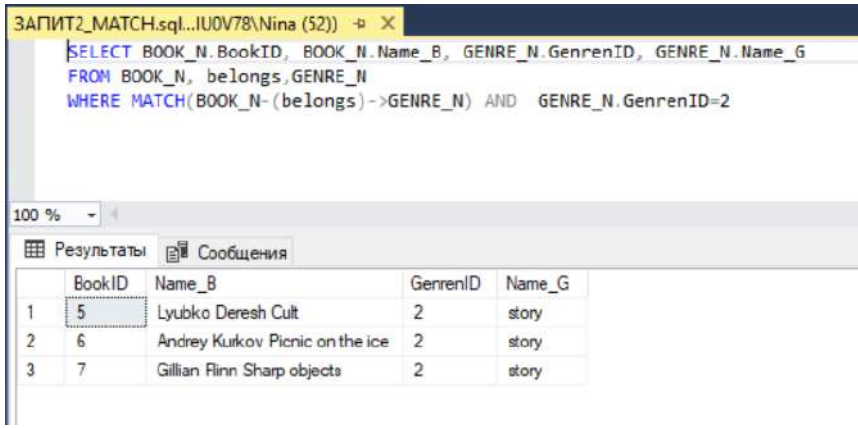
```
WHERE MATCH(node1-(edge)->node2)
```

За потреби при іншій орієнтації ребра можна змінити порядок, вказавши node1 справа від шаблона пошуку і node2 зліва, а стрілка буде вказувати в протилежний напрямок:

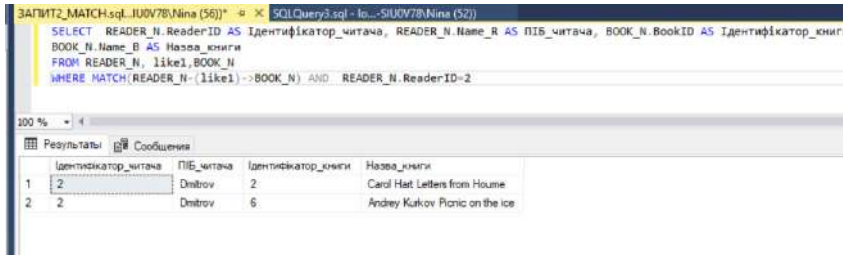
```
WHERE MATCH(node2<-(edge)-node1)
```

У функції MATCH можуть повторюватися імена вузлів, тобто в одному запиті через певний вузол можна проходити кілька раз. Імена ребер не можуть повторюватися в одному запиті з використанням функція MATCH. Ребра повинні мати чітку орієнтацію. Функція MATCH може в шаблоні пошуку використовувати AND, але не підтримує OR і NOT.

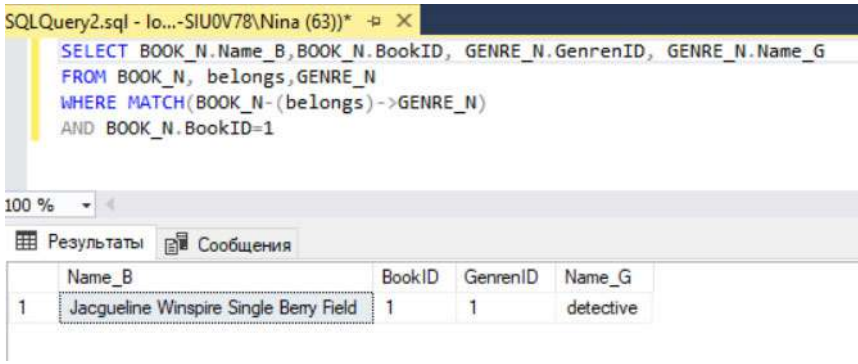
Приклад створення запита вибірки , який виконує вибірку story (романів) з таблиці вузла BOOK_N:



Нижче наведено запит вибірки книг, які подобаються читачу за його кодом (ReaderID=2):



Далі наведено запит вибірки книги за її кодом (BookID=1):



Нижче наведено запит вибірки книг за кодом жанру.

7.4. Завдання для опрацювання в лабораторній роботі

1. Як предметну область, для якої доцільно створювати графову базу, можна використати предметну область з лабораторної роботи, яка виконувалась ву середовищі Neo4J.
2. Запроектуйте графову модель базу даних для вибраної предметної області, а саме: створіть таблиці вузли та таблиці ребра і завантажте їх контрольним прикладом.
3. Відпрацюйте внесення змін до таблиць вузлів і таблиць ребер.
4. Реалізуйте запити до запроектованової графової бази даних.

7.5. Питання для самоконтролю

1. Дайте визначення графової моделі БД і характеризуйте її основні компоненти.
2. Як створюється графова база даних у середовищі СКБД Microsoft SQL Server 2017?
3. Які таблиці використовуються для моделювання графів у SQL Server 2017?
4. Чи обов'язкові первинні ключі в таблицях ребер і вузлів?
5. Які особливості виконання операції оновлення (UPDATE) у графовій базі даних?
6. Які оператори мови T-SQL можна використовувати для реалізації запитів до графової бази даних?

7.6. Література до теми

1. Створення бази даних іта виконання деяких шаблонних запитів з використанням T-SQL [Електронний ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/relational-databases/graphs/sql-graph-sample?view=sql-server-2017.

Тема 8.

КОЛОНКО-ОРІЄНТОВАНА ТА МОДЕЛЬ ТИПУ «КЛЮЧ-ЗНАЧЕННЯ»

8.1. Перелік знань і вмінь

При вивченні даної теми студенти знайомляться з такими моделями баз даних NoSQL як колонко-орієнтована та «ключ-значення». Отримають базові знання з побудови моделей, а саме, вивчать типи і формати полів та основні їх структурні елементи. Будуть знати цільове призначення моделей і вивчать їх перевагами і недоліки.

*Вивчивши матеріал теми, ви будете **знати**:*

- сутність моделі NoSQL «ключ-значення»;
- структуру та вибір ключів у моделі «ключ-значення»;
- обмеження для практичне використання моделі «ключ-значення»;
- сутність колонко-орієнтована бази даних NoSQL;
- характеристику основних структурних елементів колонко-орієнтованої моделі БД;
- призначення та практичне використання колонко-орієнтованих БД.

*Вивчивши матеріал теми, ви будете **вміти**:*

- інсталювати на робочу станцію нереляційної СКБД Redis, яка підтримує модель «ключ-значення»;
- виконувати первинне налаштування операційної системи Windows для роботи із СКБД Redis;
- виконувати базові команди для роботи із СКБД Redis.

8.2. Термінологічний словник

Колонко-орієнтована БД — це агрегатно-орієнтована база даних NoSQL, в якій агрегат містить дані стовпчика таблиці. Тобто в цих моделях одиницею зберігання даних на відміну від реляційних є не рядок, а стовпчик однотипних даних. Зберігання даних у вигляді стовпчиків використовуються для тих задач, в яких потрібно зчитувати по кілька стовпчиків одночасно. Одиницею зберігання в базі даних може бути група (сімейство) стовпчиків. Це зручно при обробленні архівних даних іта каталогів, в яких пошук і вибірки виконуються по стовпчиках, а не по рядках. Тобто колонко-орієнтовані бази даних можна представляти як

дворівневу агрегатну структуру, в якій є ключ і рядок-агрегат, що представляє собою асоціативний масив значень, представлених одним типом і форматом. Бази даних цього типу організують стовпчики в сімейства (асоціативні масиви). Кожен стовпчик повинен бути частиною певного сімейства і одиницею доступу, при цьому дані конкретного сімейства можуть бути одночасно доступними.

Кластеризований індекс — це індекс, який міняє фізичний порядок зберігання записів у таблиці. Дані зберігаються відсортованими по значенню ключа індексації. Кластеризований індекс автоматично формується при створенні первинного ключа (PRIMARY KEY). Таблиця, що не має кластеризованого індексу, називається «кучею».

Некластеризований індекс — це індекс, в якому логічний порядок індексу не відповідає фізичному зберігання записів у таблиці на диску. Такий індекс містить значення ключа індексації і посилання на рядок даних, що містять відповідні дані. Таблиця може мати кілька некластеризованих індексів. Некластеризовані індекси можна створювати для таблиць с кластеризованим індексом так і без нього. Саме ці індекси значно підвищують продуктивність БД при реалізації запитів.

***Модель типу «ключ-значення» (key-value, KV*)** — це база даних NoSQL, основними структурними елементами якої є ключ і відповідне йому значення. Значення це дані, що асоціюються з ключем, тобто розкривають його сутність**.** Значення, що може бути представлено даними довільної структури, значення підтримує практично необмежену семантику. Аналог моделі «ключ-значення» в реляційній БД є таблиця, що складається з двох атрибутів: перший це первинний ключ, а другий атрибут — його значення. Відмінність полягає в тому, що значення в моделі «ключ-значення» це агрегат, тоді як атрибути в реляційній базі даних мають бути атомарними. У моделі «ключ-значення» значення виступає як великий чорний ящик. Значення є непрозорими (opaque), тому БД NoSQL «ключ-значення» виступає у ролі великого сховища, метою якого є лише зберігання даних. Тобто запити з пошуку значень певних агрегатів у переважній більшості є не допустимими. БД типу «ключ-значення» ефективні в плані доступності (Availability) стійкість до розподілу (Partition tolerance), але явно програють в узгодженості даних (Consistency).

***Простір ключів (Keyspace)* —** об’єднує споріднені певним чином сімейства стовпчиків. Це поняття властиве деяким колонко-орієнтованим СКБД.

In-memory СКБД — це системи, що використовують оперативну пам'ять для зберігання даних. In-memory системи характеризуються значно більшою продуктивністю, так як процесор значно швидше працює з ОЗП ніж з дисковою пам’яттю. Появі БД, що зберігаються в оперативній пам’яті, сприяла тенденція, що намітилась у середині 2000-х років до зниження вартості ОЗУ та поява багатоядерних процесорів. Недостатність оперативної пам’яті одного комп’ютера компенсуються об'єднанням їх у кластери. Є два типи таких систем. СКБД, що зберігають всі дані лише в ОЗУ, тобто при виконанні запитів вони працюють з оперативною пам’яттю і не звертаються до дисків. Прикладом такої системи є Memcached. Другий тип In-memory СКБД — це системи, в яких дані зберігають в оперативній пам’яті, але кожна операція, що змінює дані, записується в журнал транзакцій. Прикладом такої системи є Redis.

Стратегії зберігання даних у Redis:

- зберігання даних лише в пам’яті. Персистентна база даних перетворюється в кешуючий сервер (keshuyuchyy server);
- періодичне зберігання даних на диск (за замовчуванням). Періодичне створення копій (snapshot) один раз в 1–15 хвилин залежно від часу створення попередньої копії і кількості змінених ключів;
- лог транзакцій. Синхронний запис кожної зміни в спеціальний append-only лог-файл;
- реплікація. Кожному серверу можна вказати майстер-сервер, після чого всі зміни на майстері будуть відображатися на слейвах.

Конкретний спосіб зберігання даних визначається при конфігуруванні сервера Redis.

Резидентна база даних (*in-memory database, IMDB*) — база даних, що розміщена в оперативній пам’яті.

Резидентна СКБД — система керування резидентними базами даних, яка підтримує резидентні обчисленні (*in-memory computing*). Велика кількість баз даних типу «ключ-значення», підтримуються резидентними СКБД, які забезпечують високу продуктивність в умовах горизонтального масштабування. Крім NoSQL резидентних СКБД з середини 2010-х років механізми роботи з in-memory базами даних мають переважна більшість комерційних реляційних баз даних. Починаючи з MS SQL Server 2014 Microsoft надає можливість використання технології In-Memory таблиць. У 2016 версії ця технологія удосконалена. Всі операції з In-Memory таблицями є транзакційними й відповіда-

ють ACID-умовам: атомарність (atomic), узгодженість (consistent), ізольованість (isolated), надійність (durable).

СКБД Redis (Remote Dictionary Server) — це розподілена система з відкритим кодом, що підтримує модель БД типу «ключ-значення», яка зберігає дані в оперативній пам'яті (in memory). Однією з переваг Redis, перед іншими базами типу «ключ-значення», є те, що вона дуже швидко виконує операції читання та запису, але її розмір обмежений розміром оперативної пам'яті. Хоча офіційно Redis не підтримується Windows проте починаючи з Windows 10 можлива інсталяція в цій операційній системі.

Поняття «температура даних» використовується для класифікації даних залежно від частоти звертань до них. Згідно цієї ознаки дані поділяються на «гарячі» і «холодні». Гарячі дані — це критично важливі дані, до яких часто звертаються. Холодні дані це не особливо термінові дані, до яких не так часто звертаються і вони частіше зберігаються для цілей аудиту та подальшого архівування. Таким чином, гарячі дані повинні зберігатися способами, що гарантують швидкий доступ і модифікацію, а холодні дані можуть зберігатися більш економними способами.

Хеш-функція — це функція, що перетворює ключ key в індекс і рівний h(key), де h(key) — хеш-код. Хеш-функція на вході приймає рядок і повертає числове значення. Весь процесс формування індексів хеш-таблиці має назву — хешування.

Хеш-таблиця (hash table) — це спеціальна структура даних для зберігання пар ключ і значення. Хеш-таблиця дозволяє за мінімальний час виконувати операції пошуку, вставки і вилучення. Тобто хешування і зберігання даних у хеш-таблицях забезпечують швидше оброблення, ніж опрацювання даних у звичайних таблицях. Однією з структур даних зберігання в СКБД Redis є хеш-таблиці.

8.3. Інструктивні та методичні вказівки для виконання лабораторної роботи №9 «Інсталяція та перше знайомство з СКБД Redis»

Мета роботи: знайомство із СКБД Redis, інсталяція нереляційної БД на робочу станцію, отримання навиків первинного налаштування для роботи з СКБД Redis.

Завдання роботи

1. Вивчити основи побудови бази даних NoSQL типу «ключ-значення» та характеристики основних її компонент.

2. Вивчити питання інсталяції СКБД Redis.
3. Познайомитися з основними типами представлення даних у СКБД Redis.
4. Виконати базові команд для роботи із СКБД Redis.
5. Відповісти на питання, поставлені в роботі.
6. Оформити звіт з лабораторної роботи.

Основна інформація про СКБД Redis

Redis — розподілене сховище пар ключ-значення, які зберігаються в оперативній пам'яті, з можливістю забезпечувати довговічність зберігання за бажанням користувача. Це програмне забезпечення з відкритим кодом написане на ANSI C. Redis надає схожі на Memcached функції для зберігання даних у форматі ключ/значення, розширені підтримкою структурованих даних, таких як списки, хеші і множини. На відміну від Memcached, Redis забезпечує постійне зберігання даних на диску і гарантує збереження БД у разі аварійного завершення роботи. Клієнтські бібліотеки доступні для більшості популярних мов, включаючи Perl, Python, PHP, Java, Ruby і Tcl.

Ключі в Redis є бінарно-безпечними (binary safe) рядками.

Занадто довгі ключі — погана практика, не тільки через кількість займаної пам'яті, але так само і в зв'язку зі збільшенням часу пошуку певного ключа. Варто дотримуватися схеми при побудові ключів: «object-type: id: field».

Типи даних Redis

Рядки (strings). Базовий тип даних Redis. Рядки в Redis бінарно-безпечні, можуть використовуватися так само як числа, обмежені розміром 512 Мб.

Списки (lists). Класичні списки рядків, впорядковані в порядку вставки, яка можлива як з боку голови, так і з кінця списку. Максимальна кількість елементів — 2^{32}.

Множини (sets). Множини рядків у математичному розумінні: не впорядковані, підтримують операції вставки, перевірки входження елемента, перетину і різниці множин. Максимальна кількість елементів — 2^{32}.

Хеш-таблиці (hashes). Класичні хеш-таблиці або асоціативні масиви. Максимальна кількість пар «ключ-значення» — 2^{32}.

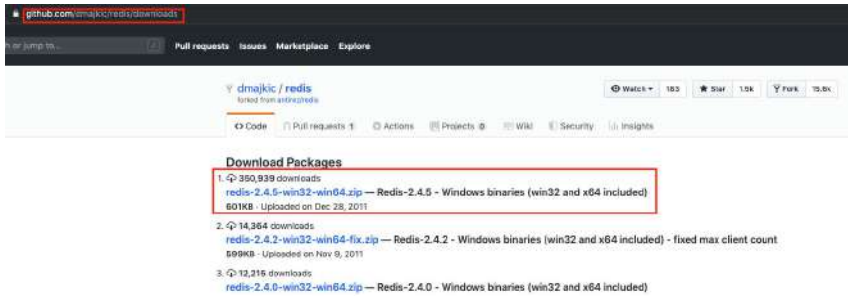
Впорядковані множини (sorted sets). Впорядкована множина відрізняється від звичайної тим, що її елементи впорядковані по особливому параметру «score».

За виникнення будь-яких запитань можна звернутись до документації на офіційному сайті https://redis.io/documentation .

Інсталяція СКБД Redis

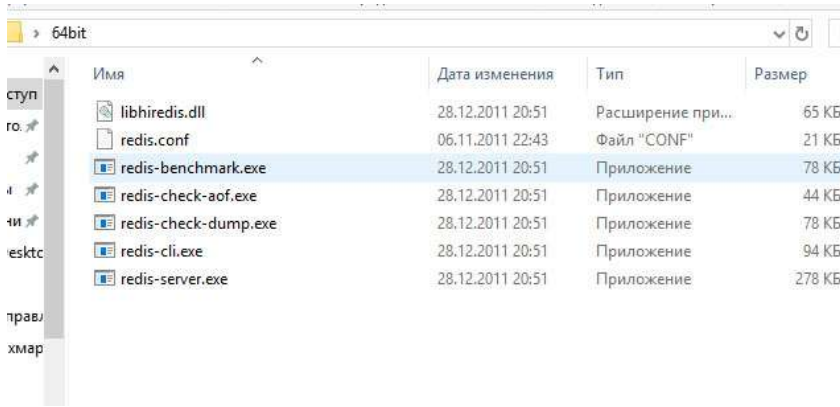
Для практики на ОС Windows можемо завантажити файли серверу із даного репозиторію: https://github.com/dmajkic/redis/downloads .

Остання стабільна версія — 2.4.5.



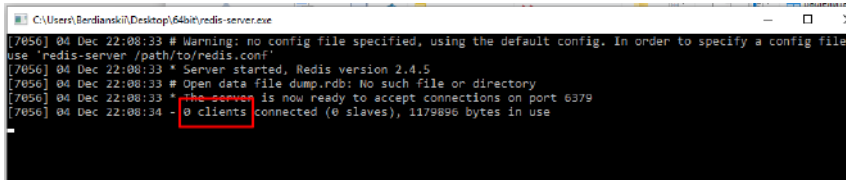
Завантажуємо архів, у якому одразу знаходяться версії як для 32-х бітної системи, так і для 64-х бітної.

Розпаковуємо папку для 64-х бітної системи та бачимо такі файли:



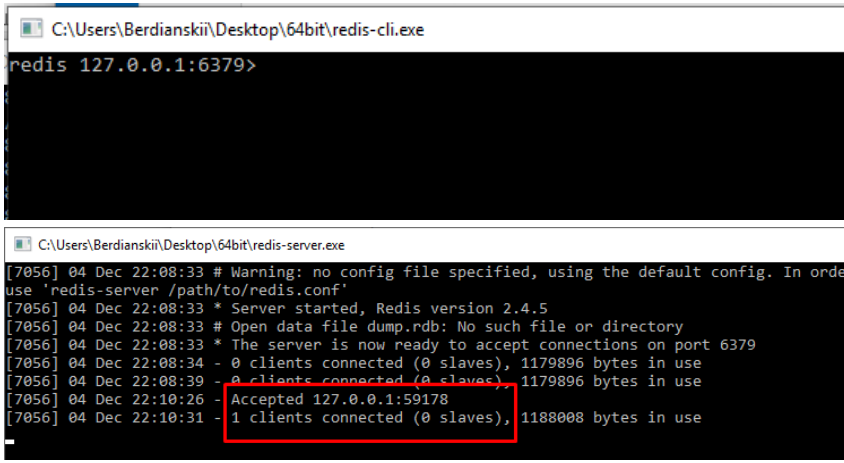
Для нас важливими є 2 основні файли — «redis-server» і «redis-cli». Перший файл потрібен для запуску сервера на локальній машині, а другий — для роботи з сервером і виконання команд.

Запустимо сервер:



Бачимо, що сервер успішно запущено, але до нього поки не приєднався жоден клієнт. У нашому випадку клієнтом виступає «redis-cli».

Запустимо «redis-cli»:

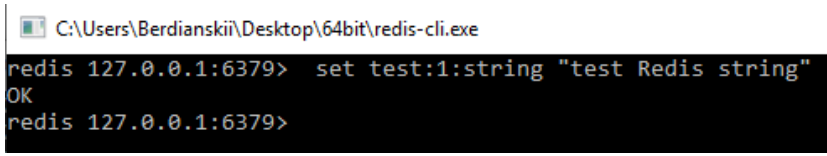


Бачимо, що клієнт успішно запущено, це відображається у консолі сервера.

Виконання базових команд для роботи із Redis сервером

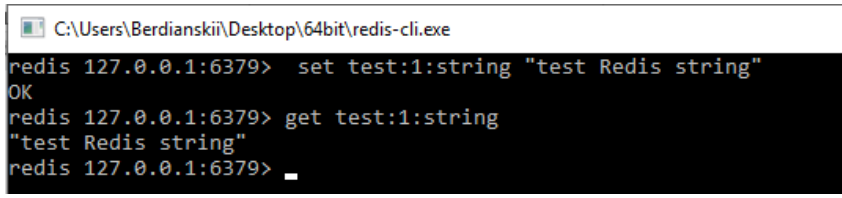
Напишемо у базу перше текстове значення із ключем «test» за допомогою команди:

```
set test:1:string
```



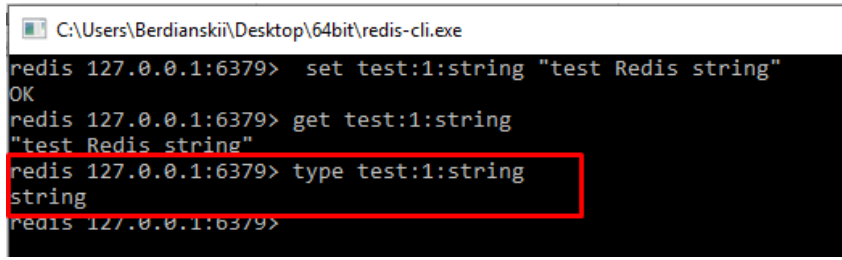
Дістанемо із бази раніше створений запис за допомогою команди:

```
get test:1:string
```

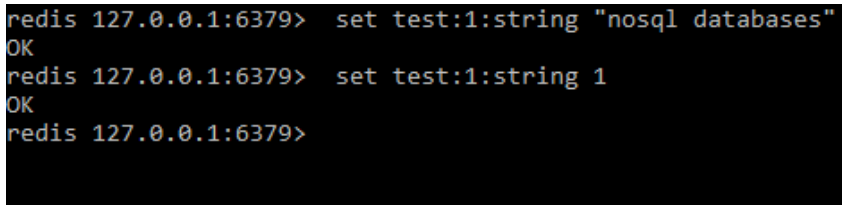


Визначимо тип даних раніше створенної змінної за допомогою команди:

```
type test:1:string
```

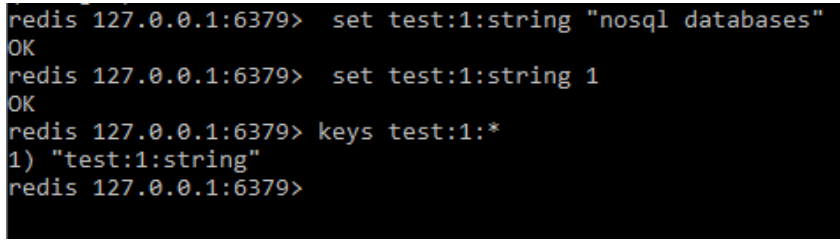


Додамо ще два записи до бази:



Переглянемо усі записи, збережені у базі за допомогою команди:

```
keys test:1:*
```



Основні команди Redis

SET — команда установки ключа та його значення, з додатковим необов'язковими параметрами, які задають термін дії ключа:

• EX *секунди* — встановлює вказаний час закінчення в секундах;
• PX *мілісекунди* — встановлює вказаний час закінчення, у мілісекундах;
• NX — встановлює ключ, лише якщо він ще не існує;
• XX — встановлює ключ, лише якщо він уже існує;
• KEEPTTL — зберігає час для дії, пов'язаний з ключем;
• GET — повертає старе значення, збережене в ключі, або нуль, коли ключа не існувало.

Приклад установки ключа house зі значенням "heat indicators", параметр EX вказує термін існування в секундах:

```
127.0.0.1:6379> SET house "heat indicators"
OK
127.0.0.1:6379> SET house "heat indicators" ex
5
OK
```

GET — команда для отримання значення ключа. У разі, коли запис значення ключа перевищить термін його існування, то видається значення nil.

Приклад:

```
127.0.0.1:6379> GET house
"heat indicators"
# якщо закінчиться термін дії ключа
127.0.0.1:6379> GET house
(nil)
```

EXISTS — команда перевіряє існування вказаного ключа. Вона видає 1 при існуванні та 0 при відсутності.

Приклад:

```
127.0.0.1:6379> EXISTS house
(integer) 1
```

FLUSHALL — команда повністю вилучає всі дані поточного сеансу.

GETSET — команда повертає поточне значення і встановлює нове. Використовується для атомарного управління даними.

Приклад:

```
127.0.0.1:6379> SET house " indicators"
OK
127.0.0.1:6379> GETSET house " heat "
" indicators "
127.0.0.1:6379> GET house
" heat "
```

DEL — команда вилучає ключ і відповідне значення.

Приклад:

```
127.0.0.1:6379> DEL house
(integer) 1
```

APPEND — команда додає у відповідний ключ додаткове значення та видає кількість символів нового значення.

Приклад:

```
127.0.0.1:6379> SET house "heat"
OK
127.0.0.1:6379> APPEND house "indicators"
(integer) 14
127.0.0.1:6379> GET house
"heat indicators"
```

TTL — якщо ключ встановлено з певним терміном дії (наприклад SET house EX 10), то командо можна переглянути час, що залишився.

Приклад:

```
127.0.0.1:6379> SET house 10 EX 10
OK
127.0.0.1:6379> TTL house
(integer) 6
127.0.0.1:6379> TTL house
(integer) 5
127.0.0.1:6379> TTL house
(integer) 2
127.0.0.1:6379> TTL house
(integer) -2
```

PERSIST — цією командою можна вилучити термін дії ключа.

Приклад:

```
127.0.0.1:6379> PERSIST house
(integer) 1
127.0.0.1:6379> TTL house
(integer) -1
127.0.0.1:6379> GET house
"bar"
```

RENAME — команда перейменовує ключі.

Приклад:

```
127.0.0.1:6379> RENAME house house 2
OK
127.0.0.1:6379> GET house
(nil)
127.0.0.1:6379> GET house 2
"bar"
```

8.4. Завдання для опрацювання в лабораторній роботі

1. Виберіть предметну область для якої доцільно створювати базу типу «ключ-значення».
2. Завантажте базу даних тестовим прикладом.
3. Відпрацюйте основні команди роботи з базою даних.

8.5. Питання для самоконтролю

1. Дайте визначення моделі типу «ключ-значення» та характеризуйте її основні компоненти.
2. Як визначаються ключі в моделі типу «ключ-значення»?
3. Що представляє собою значення в моделі типу «ключ-значення»?
4. Які типи даних підтримує СКБД Redis?
5. Характеризуйте основні напрями практичного застосування СКБД Redis .Характеризуйте основні переваги та недоліки моделі типу типу «ключ-значення»?

8.6. Література до теми

1. What is Redis? / Офіційний портал компанії Redis Inc. [Електронний ресурс]. URL: https://www.redis.io
2. Redis: дані Вікіпедії [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Redis
3. Он-лайн керівництво із Redis [Електронний ресурс]. URL: https://metanit.com/nosql/Redis/

Тема 9.
ІНТЕГРАЦІЙНІ ПРОЦЕСИ В РОЗВИТКУ СКБД

9.1. Перелік знань і вмінь

При вивченні даної теми студенти знайомляться з причинами інтеграційних процесів у розвитку СКБД. Вивчать характеристику інтеграції на прикладі СКБД Microsoft SQL Server.

*Вивчивши матеріал теми, ви будете **знати**:*

- відмінності представлення атомарних і агрегованих даних у JSON форматі;
- основи створення елементів документно-орієнтованих моделей у середовищі реляційної СКБД;
- оператори мови T-SQL для створення даних у JSON - форматі;
- вбудовані функції SQL Server SQL Server для роботи з даними в JSON-форматі.

*Вивчивши матеріал теми, ви будете **вміти**:*

- створювати гібридні таблиці мультимодельної бази даних, що містять атомарні дані та їх агрегати в JSON-форматі;
- створювати запити завантаження JSON-даних у реляційну таблицю;
- використовувати вбудовані функції SQL Server для роботи з даними в JSON форматі.

9.2. Термінологічний словник

Багатоваріантне зберігання («polyglot persistence») — це використання різних технологій зберігання, різних моделей представлення даних і різних СКБД у рамках одної інформаційної системи.

Гібридна таблиця — це таблиця мультимодельної бази даних, що містить атомарні атрибути, згідно вимог реляційних моделей та агрегати NoSQL, наприклад, документи у JSON-форматі.

Локальна змінна Transact-SQL — це об'єкт, що містить значення певного типу. Для ініціалізації локальної змінної використовується оператор DECLARE, який завжди починається з символа @ і представляється таким синтаксисом:

```
DECLARE @назва змінної тип даних
```

JSON дані в SQL Server зберігаються у форматі NVARCHAR, наприклад:

```
DECLARE @Json NVARCHAR(max)
```

Установка значення локальної змінної виконується оператором SET:

```
SET @ назва змінної = значення
```

Значення — містить конкретні значення, які присвоюється змінній.

Мультимодельна база даних (multi-model databases) — це система, що може зберігати та підтримувати роботу з даними, що описуються різними моделями. Довгий час СКБД підтримували лише одну модель даних, наприклад реляційну, документно-орієнтовану чи іншу. На сьодення є кілька різновидів мультимодельних систем:

- СКБД, які позиціонують себе як мультимодельні, не маючи успадкованої основної моделі. Наприклад, ArangoDB, OrientDB, CosmosDB (сервіс у складі хмарної платформи Microsoft Azure);
- мультимодельні системи, що розширили свій функціонал на базі основної моделі. Наприклад, реляційна СКБД SQL Server підтримує документарні, графові та колонко-орієнтовані NoSQL моделі даних.

ISJSON — вбудована функція перевіряє правильність синтаксису перед записом в БД JSON-даних, які імпортуються в SQL Server з інших систем.

JSON_VALUE — вбудована функція вибору скалярних значень з JSON-даних.

JSON_QUERY — вбудована функція вибору об'єкта чи масиву з JSON-даних.

JSON_MODIFY — вбудована функція для внесення змін у рядку даних у форматі JSON.

OPENJSON — вбудована функція, яка перетворює JSON-дані в набір рядків таблиці. Над цим набором можна виконувати SQL-запити.

FOR JSON — вбудована функція, яка перетворює табличне представлення даних у JSON-формат.

9.3. Інструктивні та методичні вказівки для виконання лабораторної роботи №10 «Робота з даними у форматі JSON в СКБД SQL Server»

Мета роботи: опрацювати та набути практичних знань і навичок роботи з мультимодельними базами даних. Зокрема підтримки та роботи з елементами документарної моделі даних у середовищі СКБД SQL Server.

Завдання роботи

1. Ознайомитись із можливостями використання формату JSON у реляційних базах даних.
2. Отримати навички створення таблиць з полями у JSON-форматі та вміти їх завантажувати конкретними даними.
3. Вміти перетворювати дані, що представлені в JSON-форматі, за допомогою функції OPENJSON, у табличне представлення.
4. Відпрацювати створення запитів з перенесення JSON-даних у реляційну таблицю.
5. Відповісти на питання, поставлені в роботі.
6. Оформити звіт з лабораторної роботи.

Інструкція з виконання лабораторної роботи

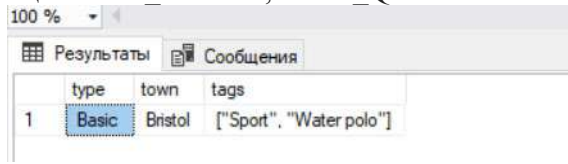
Починаючи з Microsoft SQL Server 2016 і пізніших версій надається можливість підтримки даних у JSON-форматі. JSON-дані в SQL Server дозволяють об'єднувати елементи документно-орієнтованої NoSQL і реляційні моделі в одній базі даних.

Для перевірки тексту, представленого в JSON-форматі, на валідність використовуються вбудовані функції JSON_VALUE, JSON_QUERY.

Запит перевірки на валідність наведено нижче.

```
DECLARE @json NVARCHAR(4000)
SET @json =
N'{
 "info":{
 "type":1,

 "address":{
 "town":"Bristol",
 "county":"Avon",
 "country":"England"
 },
 "tags":["Sport", "Water polo"]
 },
 "type":"Basic"
}'

SELECT
 JSON_VALUE(@json, '$.type') as type,
 JSON_VALUE(@json, '$.info.address.town') as town,
 JSON_QUERY(@json, '$.info.tags') as tags
```

Результат виконання запиту перевірки на валідність з використанням функцій JSON_VALUE, JSON_QUERY:



В таблиці бази даних можуть одночасно зберігатися як атомарні дані, так і агреговані дані у форматі JSON. Розглянемо приклад створення таблиці Viddil_1 з полем Person, яке представлено у форматі JSON. Сценарій створення даної таблиці має вид:

```
USE [BD_NOSQL]
GO

/****** Object: Table [dbo].[Viddil_1] Script
Date: 14.06.2019 18:15:46 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Viddil 1](
    [Kod V] [tinyint] NOT NULL,
    [Name V] [nchar](40) NULL,
    [Person] [varchar](max) NOT NULL,
 CONSTRAINT [PK_Viddil_1] PRIMARY KEY
CLUSTERED
(
    [Kod V] ASC
)WITH (PAD INDEX = OFF, STATISTICS NORECOMPUTE
= OFF, IGNORE DUP KEY = OFF, ALLOW ROW LOCKS =
ON, ALLOW PAGE LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Для завантаження даних до таблиці визначимо локальну змінну, що буде містити значення, що будуть зберігатися в полі Person у JSON -форматі.

```
DECLARE @Json NVARCHAR(max);
SET @Json='[
 { "id" : 2,"firstName": "Jvan", "lastName": "Karpov",
 "age": 25, "dateOfBirth": "2007-03-25T12:00:00" },
```

```
 { "id" : 5,"firstName": "Nika", "lastName": "Kirpa",
 "age": 35, "dateOfBirth": "2005-11-04T12:00:00" },
 { "id" : 7,"firstName": "Petr", "lastName": "Sowa",
 "age": 15, "dateOfBirth": "1983-10-28T12:00:00" },
 { "id" : 8,"firstName": "Elena", "lastName": "Druz",
 "age": 12, "dateOfBirth": "1995-07-05T12:00:00" },
 { "id" : 9,"firstName": "Anton", "lastName":
"Savchuk",
 "age": 37, "dateOfBirth": "2015-03-25T12:00:00" }
]
';
```

У поле Person, що задано в JSON-форматі, будуть записані особисті дані співробітників відділів, а саме: firstName (ім'я), lastName (прізвище), age (вік) і dateOfBirth (дата народження).

Сценарій запису даних про всіх співробітників відділу 'Soft' до таблиці Viddil_1 має такий вид:

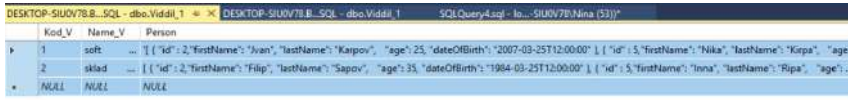
```
USE [BD_NOSQL]
GO
DECLARE @Json NVARCHAR(max);
SET @Json='[
 { "id" : 2,"firstName": "Jvan", "lastName": "Karpov",
   "age": 25, "dateOfBirth": "2007-03-25T12:00:00" },
 { "id" : 5,"firstName": "Nika", "lastName": "Kirpa",
   "age": 35, "dateOfBirth": "2005-11-04T12:00:00" },
 { "id" : 7,"firstName": "Petr", "lastName": "Sowa",
   "age": 15, "dateOfBirth": "1983-10-28T12:00:00" },
 { "id" : 8,"firstName": "Elena", "lastName": "Druz",
   "age": 12, "dateOfBirth": "1995-07-05T12:00:00" },
 { "id" : 9,"firstName": "Anton", "lastName": "Savchuk",
   "age": 37, "dateOfBirth": "2015-03-25T12:00:00" }
]
';
INSERT INTO [dbo].[Viddil_1]
           ([Kod_V]
           ,[Name_V]
           ,[Person])
     VALUES
           ( 1, 'Soft' , @Json)
GO
```

```
%
Сообщения

(затронута одна строка)
```

Аналогічно завантажується другий рядок таблиці, що містять дані про співробітників підрозділу з назвою - 'sklad':

```
USE [BD_NOSQL]
GO
DECLARE @Json NVARCHAR(max);
SET @Json='[
 { "id" : 2,"firstName": "Filip", "lastName":
"Sapov",
 "age": 35, "dateOfBirth": "1984-03-25T12:00:00"
},
 { "id" : 5,"firstName": "Inna", "lastName":
"Ripa",
 "age": 19,dateOfBirth": "2000-11-04T12:00:00" },
 { "id" : 7,"firstName": "Petr", "lastName":
"Stafov",
 "age": 36,ateOfBirth": "1983-10-28T12:00:00" },
 { "id" : 8,"firstName": "Liza", "lastName":
"Franko",
 "age": 24, "dateOfBirth": "1995-07-05T12:00:00"
},
 { "id" : 9,"firstName": "Karpo", "lastName":
"Spivak",
 "age": 25, "dateOfBirth": "1994-03-25T12:00:00" }
]
';
INSERT INTO [dbo].[Viddil_1]
 ([Kod V]
 ,[Name V]
 ,[Person])
 VALUES
 (2,'sklad',@Json)
GO
```

Таблиця Viddil_1 з завантаженими даними має вигляд:

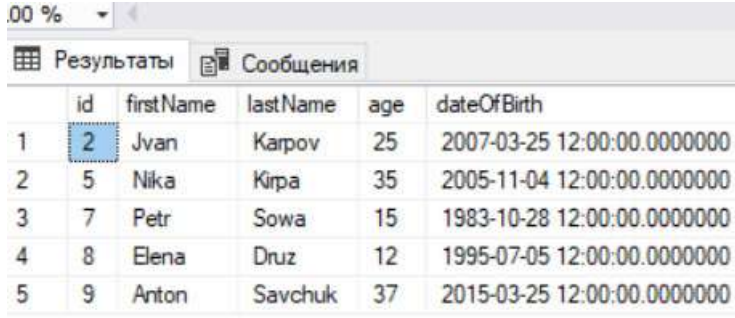


	Kod_V	Name_V	Person
▸	1	soft	... [{ "id" : 2,"firstName": "Ivan", "lastName": "Karpov", "age": 25, "dateOfBirth": "2007-03-25T12:00:00" }, { "id" : 5,"firstName": "Nika", "lastName": "Kirpa", "age...
	2	sklad	... [{ "id" : 2,"firstName": "Filip", "lastName": "Sapov", "age": 35, "dateOfBirth": "1984-03-25T12:00:00" }, { "id" : 5,"firstName": "Inna", "lastName": "Ripa", "age": ...
*	NULL	NULL	NULL

Дані, що представлені в JSON-форматі, за допомогою функції OPENJSON для перегляду можна представити у вигляді таблиці. Нижче наведено сценарій виконання такого запиту з використанням локальної змінної @Json.

```
DECLARE @Json NVARCHAR(max);
SET @Json='[
 { "id" : 2,"firstName": "Jvan", "lastName":
"Karpov",
 "age": 25, "dateOfBirth": "2007-03-25T12:00:00" },
 { "id" : 5,"firstName": "Nika", "lastName":
"Kirpa",
 "age": 35, "dateOfBirth": "2005-11-04T12:00:00" },
 { "id" : 7,"firstName": "Petr", "lastName":
"Sowa",
 "age": 15, "dateOfBirth": "1983-10-28T12:00:00" },
 { "id" : 8,"firstName": "Elena", "lastName":
"Druz",
 "age": 12, "dateOfBirth": "1995-07-05T12:00:00" },
 { "id" : 9,"firstName": "Anton", "lastName":
"Savchuk",
 "age": 37, "dateOfBirth": "2015-03-25T12:00:00" }
]
';
SELECT *
FROM OPENJSON(@json)
 WITH (id int, firstName nvarchar(50), lastName
nvarchar(50),
 age int, dateOfBirth datetime2)
GO
```

Результат виконання описаного запиту з функцією OPENJSON буде таким:

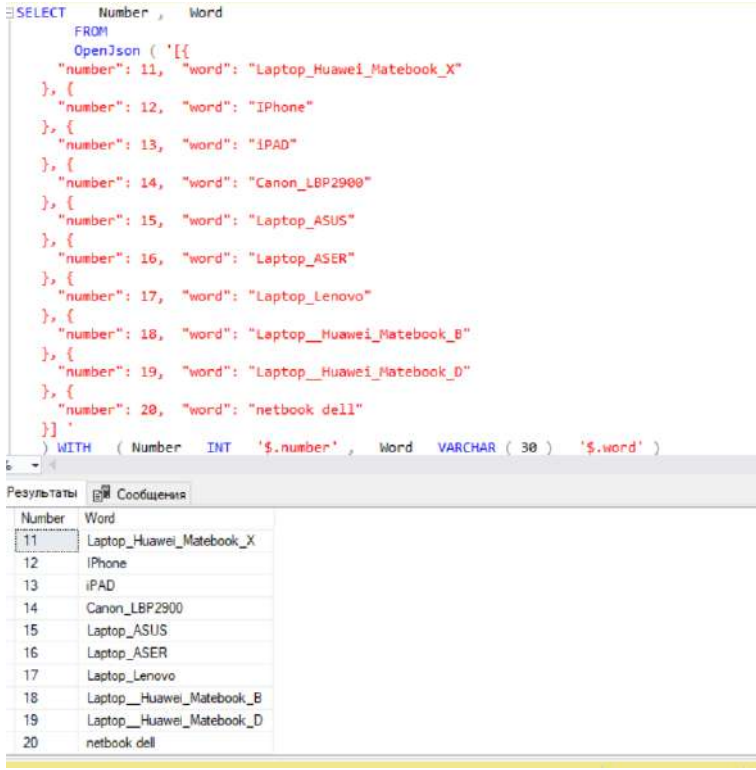


00 %

Результаты | Сообщения

	id	firstName	lastName	age	dateOfBirth
1	2	Jvan	Karpov	25	2007-03-25 12:00:00.0000000
2	5	Nika	Kirpa	35	2005-11-04 12:00:00.0000000
3	7	Petr	Sowa	15	1983-10-28 12:00:00.0000000
4	8	Elena	Druz	12	1995-07-05 12:00:00.0000000
5	9	Anton	Savchuk	37	2015-03-25 12:00:00.0000000

Інший сценарій використання функції OPENJSON можна для перегляду представити у вигляді таблиці без локальної змінної.



Для того, щоб зберегти JSON -дані в базі даних у вигляді реляційної таблиці необхідно створити нову таблицю. Створюємо таблицю Person:

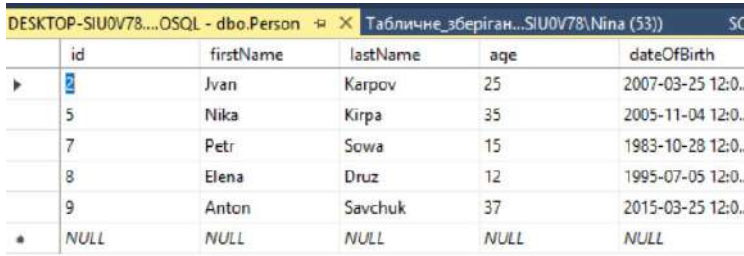
```
USE [BD_NOSQL]
GO

/****** Object: Table [dbo].[Person] Script
Date: 14.06.2019 15:15:25 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Person](
    [id] [int] NULL,
    [firstName] [nvarchar](50) NULL,
    [lastName] [nvarchar](10) NULL,
```

```
    [age] [int] NULL,
    [dateOfBirth] [datetime2](7) NULL
) ON [PRIMARY]
GO
```

На наступному кроці створюємо запит перенесення даних у таблицю:

```
DECLARE @Json NVARCHAR(max);
SET @Json='[
 { "id" : 2,"firstName": "Jvan", "lastName":
"Karpov",
 "age": 25, "dateOfBirth": "2007-03-
25T12:00:00" },
 { "id" : 5,"firstName": "Nika", "lastName":
"Kirpa",
 "age": 35, "dateOfBirth": "2005-11-
04T12:00:00" },
 { "id" : 7,"firstName": "Petr", "lastName":
"Sowa",
 "age": 15, "dateOfBirth": "1983-10-
28T12:00:00" },
 { "id" : 8,"firstName": "Elena", "lastName":
"Druz",
 "age": 12, "dateOfBirth": "1995-07-
05T12:00:00" },
 { "id" : 9,"firstName": "Anton", "lastName":
"Savchuk",
 "age": 37, "dateOfBirth": "2015-03-
25T12:00:00" }
]
';

INSERT INTO Person (id, firstNAme, lastName,
age, dateOfBirth)
 SELECT id, firstNAme, lastName, age,
dateOfBirth
 FROM OPENJSON(@json)
 WITH (id int,
 firstName nvarchar(50), lastName nvarchar(50),
 age int, dateOfBirth datetime2)
```

Після виконання запиту таблиця буде вміщувати такі дані:


DESKTOP-SIU0V78....OSQL - dbo.Person | Табличне_зберіган...SIU0V78\Nina (53)) | SQL

id	firstName	lastName	age	dateOfBirth
2	Jvan	Karpov	25	2007-03-25 12:0...
5	Nika	Kirpa	35	2005-11-04 12:0...
7	Petr	Sowa	15	1983-10-28 12:0...
8	Elena	Druz	12	1995-07-05 12:0...
9	Anton	Savchuk	37	2015-03-25 12:0...
NULL	NULL	NULL	NULL	NULL

9.4. Завдання для опрацювання в лабораторній роботі

1. У середовищі SQL Server 2017 створіть одну реляційну таблицю, яка окрім атомарних полів, буде містити поля у JSON-форматі.
2. Реалізуйте запит завантаження даними створену таблицю.
3. Відпрацюйте запити з використанням функцій OPENJSON і FORJSON.

9.5. Питання для самоконтролю

1. Характеризуйте необхідність використання представлення даних у JSON-форматі в реляційних таблицях.
2. Характеризуйте відмінності представлення реляційних даних і даних у JSON-форматі.
3. Які вбудовані функції SQL Server виконують перевірку на валідність даних у JSON-форматі?
4. За допомогою якої функції можна перетворювати в табличне представлення даних у JSON-форматі?
5. Характеризуйте особливості створення реляційних таблиць, що містять поля у JSON форматі.
6. Як виконується завантаження даних у JSON-форматі в реляційну таблицю?

9.6. Література до теми

1. Дані JSON в SQL Server / Офіційний портал компанії Microsoft. [Електронний ресурс]. URL: https://docs.microsoft.com/ru-ru/sql/relational-databases/json/json-data-sql-server?view=sql-server-2017
2. Робота з JSON в Microsoft SQL Server / Офіційний портал компанії Microsoft. [Електронний ресурс]. URL: https://info-comp.ru/programmirovanie/544-working-with-json-in-microsoft-sql-server.html

Додаток 1

ПЕРЕЛІК РЕКОМЕНДОВАНИХ ТЕМ З ВИКОНАННЯ ІНДИВІДУАЛЬНИХ ЛАБОРАТОРНИХ РОБІТ

Завдання. Вивчити предметну область, яку характеризує задача. Визначити, які дані по задачі можна представити у документо-орієнтованій базі та запроектувати її в середовищі СКБД MongoDB.

Наведена тематика є орієнтовною. Здобувач може запропонувати свою тему, узгодивши її з викладачем.

1. Проектування БД рішення задачі з обліку наявності та руху особового складу організації.
2. Проектування БД рішення задачі з обліку основних засобів.
3. Проектування СД з аналізу обсягів продаж продукції торговельною фірмою.
4. Проектування БД з аналізу доходу від збуту готової продукції.
5. Проектування БД з аналізу страхового портфеля страхової фірми.
6. Проектування БД з аналізу клієнтської бази комерційного банку.
7. Проектування БД з аналізу кредитного портфелю комерційного банку.
8. Проектування БД рішення задачі з обліку акціонерів комерційного банку.
9. Проектування БД рішення задачі з обліку депозитів комерційного банку.
10. Проектування БД рішення задачі з обліку емісії пдастикових карток.
11. Проектування БД з обліку страхових договорів (полісів) за окремими видами страхових продуктів.
12. Проектування БД з обліку роботи мобільного оператора.
13. Проектування БД з контролю виконання навантаження викладачами ВУЗу.
14. Проектування БД з аналізу і контролю успішності здобувачів ВУЗу.
15. Проектування БД з надання послуг дистанційної освіти.
16. Проектування БД з управління електронним магазином з продажу побутової техніки.
17. Проектування БД з обліку та аналізу роботи з продажу обчислювальної техніки.

18. Проектування БД туристичного агенства.
19. Проектування БД автотранспортного підприємства.
20. Проектування БД бібліотеки.
21. Проектування БД з обліку роботи поліклініки.
22. Проектування БД з обліку роботи видавництва.
23. Проектування БД з обліку роботи кінотеатру.
24. Проектування БД з обліку роботи готелю.
25. Проектування БД з обліку роботи торговельно-закупівельного підприємства.
26. Проектування БД з обліку та аналізу роботи мережі магазинів парфюмерії.
27. Проектування БД з обліку та аналізу роботи мережі аптек.
28. Проектування БД з обліку та аналізу роботи мережі пунктів атосервісу.
29. Проектування БД з обліку та аналізу роботи мережі автосалонів.
30. Проектування БД з обліку та аналізу роботи мережі піццерій.
31. Проектування БД з обліку та аналізу роботи центра зайнятості.
32. Проектування БД з обліку та аналізу роботи мережі поштових відділень.
33. Проектування БД з обліку та аналізу роботи рієлторської компанії.
34. Проектування БД з обліку та аналізу роботи тепличного господарства.
35. Проектування БД з обліку та аналізу роботи пункту прокату велосипедів.
36. Проектування БД з обліку та аналізу роботи ветеринарної клініки.
37. Проектування БД з обліку та аналізу роботи клінінгової компанії.
38. Проектування БД з обліку та аналізу врожайності агрокультур.
39. Проектування БД студії звукозапису.
40. Проектування БД з служби таксі.
41. Проектування БД мережі медичних діагностичних лабораторій.

Навчальне видання

СИТНИК Ніна Василівна,
ЗІНОВ'ЄВА Ірина Сергіївна

ОРГАНІЗАЦІЯ БАЗ ДАНИХ NoSQL

ПРАКТИКУМ

Редактор *І. Савлук*
Коректор *Н. Підлужна*
Верстка *Т. Мальчевської*

Підп. до друку 30.06.2022. Формат 60×84/16. Папір офсет. № 1.
Гарнітура Тип Таймс. Друк офсет. Ум.-друк. арк. 9,76.
Обл.-вид. арк. 11,11. Зам. 21-5707

Державний вищий навчальний заклад
«Київський національний економічний університет імені Вадима Гетьмана»
03680, м. Київ, проспект Перемоги, 54/1

Свідоцтво про внесення до Державного реєстру
суб'єктів видавничої справи (серія ДК, № 235 від 07.11.2000)

E-mail: litera_kneu@ukr.net