ДРОГОБИЦЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА

Ірина Шаклеіна, Надія Ших

НЕРЕЛЯЦІЙНІ БАЗИ ДАНИХ

Навчальний посібник для студентів
спеціальності «122 Комп'ютерні науки»

Дрогобич
2020

УДК.004(07)
Ш17

Рекомендовано до друку вченою радою Дрогобицького державного педагогічного університету імені Івана Франка як методичні рекомендації до виконання лабораторних робіт (протокол № 9 від 30.06.2020 р.)

Рецензенти:

Столярчук Ігор Дмитрович, доктор фізико-математичних наук, професор кафедри фізики навчально-дослідного ІФМЕІТ Дрогобицького державного педагогічного університету імені Івана Франка.

Гадзаман Іван Васильович, кандидат фіз.-мат. наук, доцент кафедри фізики навчально-дослідного ІФМЕІТ Дрогобицького державного педагогічного університету імені Івана Франка.

Шаклеіна І., Ших Н.

Нереляційні бази даних: навчальний посібник / Ірина Шаклеіна, Надія Ших – Дрогобич: Редакційно-видавничий відділ Дрогобицького державного педагогічного університету імені Івана Франка, 2020. – 75 с.

Посібник укладено відповідно до робочої програми навчальної дисципліни «Організація баз даних та знань» для підготовки фахівців освітнього ступеня «Бакалавр» галузі знань: «12 Інформаційні технології» спеціальності «122 Комп'ютерні науки», затвердженої науково-методичною радою Дрогобицького державного педагогічного університету імені Івана Франка.

УДК.004(07)

ЗМІСТ

ПЕРЕДМОВА

Практично будь-яка сучасна галузь пов’язана з накопиченням та обробкою даних. З огляду на це одним з важливих завдань для фахівця з інформаційних технологій є знання технологій роботи з базами даних.

Ядром бази даних (БД) є її прикладна частина, яка складається із сервера баз даних, джерел даних та мережевого програмного забезпечення для підключення клієнта до мережі.

На сучасному етапі розвитку інформаційних технологій спостерігається тенденція збільшення ваги нереляційних баз даних, які відрізняються від SQL-орієнтованих реляційних баз даних структурою, способом організації та типами даних, підходом до збереження інформації тощо. Це зумовлено потребами зберігання та швидкого опрацювання великих обсягів неструктурованої чи слабоструктурованої інформації.

Однією з головних відмінностей нереляційних баз даних (NoSQL) є велика продуктивність та наявність горизонтального масштабування. У реляційних базах даних зазвичай підтримка масштабування потребує більш суттєвих витрат ресурсів.

Посібник містить загальні відомості про неререляційні бази даних та особливості роботи з даними в документо-орієнтованих і графових базах даних. Зокрема розглянуто особливості опрацювання даних, що зберігаються в БД, розроблених засобами MongoDB, FireBase, Neo4j та OrientDB.

До кожної теми підібрані контрольні запитання та завдання для самостійного виконання. Наприкінці посібника наведено перелік літературних джерел, що можуть бути корисними студентам при вивченні навчального матеріалу.

НЕРЕЛЯЦІЙНІ БАЗИ ДАНИХ: ОСНОВНІ ТИПИ ТА ОСОБЛИВОСТІ

Значна кількість СКБД (SQLServer, MySQL, Oracle, Postgresql, SQLite) є реляційними СКБД, що збрерігать дані у таблицях, між якими втановлені зв'язки певного типу. Кожна таблиця має унікальну назву та перелік полів (атрибутів, колонок), а всі таблиці і зв'язки між ними утворюють схему реляційної бази даних. Реляційні бази даних залишаються дотепер єдиним варіантом збереження даних для багатьох розробників, оскільки структура таблиці їм зрозуміла і знайома. Однак протягом останього часу популярності набувають бази даних, які використовують інакші підходи до організації даних та їх оброблення. Це зумовлено необхідністю обробки величезних об'ємів даних даних, які часто не мають дуже чіткої структури.

Наприклад, дані, що потребують опрацювання, можуть бути різнотипними, мати різні атрибути. З огляду на це у реляційній БД потрібно зазвичай зберігати дані у різних таблицях, об'єднуючи їх між собою складними зв'язками. Проте, такий підхід приводить до необхідности використання складних запитів на вибірку даних та до суттєвого збільшення навантаження на сервер бази даних.

Саме з огляду на наявність даних такого типу (слабоструктурованих, різнотипних, великого об'єму) набирають ваги бази даних, які не мають схеми у вигляді таблиць і звязків – нереляційні БД (або NoSQL БД)

Нереляційні (NoSQL) БД – це категорія баз даних, які часто використовуються для опису систем керування даними, що не належать до SQL, або підходу до організації та обробки даних, який передбачає використання не тільки SQL. Існує низка технологій NoSQL, включаючи бази даних документів, сховище пар "ключ-значення", сховища класів стовпців, а також графові бази даних, які часто використовуються в іграх, додатках для роботи з соціальними мережами.

Можна виділити чотири основні типи нереляційних баз даних, які мають різні моделі даних, інший підхід до розподіленості даних та реплікації.

Бази даних “ключ-значення”

Бази даних “ключ-значення” – це найбільш простий вид нереляційних баз даних. Модель даних є хеш-таблицею, яка складається з пар ключ-значення, тобто кожному значенню (об'єкту) ставиться у відповідність унікальний ключ Для додавання даних потрібно вказати ключ та об'єкт даних, що має бути збережений. Щоб зробити запит, потрібно відправити ключ і отримати назад пов'язаний з ним об'єкт.

Оскільки немає чіткої схеми зв'язків між об'єктами бази даних, у таких БД часто зберігають не пов'язані між собою дані різних типів.

Наприклад, сховище даних Redis зберігає у собі пари ключ-значення. Ключем повинен бути строковий літерал, але як значення можна використовувати один з шести доступних типів даних: string (як рядки можна використовувати довільні дані, однак набір операцій для роботи з цим типом даних обмежений), list (реалізований як пов'язаний список), hash (містить набір пар ключ-значення, причому у ролі ключів можуть використовуватися тільки рядкові значення), множина (невпорядковані набори рядків), впорядкована множина (симбіоз звичайних множин і списків; містять тільки унікальні значення, але кожному значенню відповідає число), hyper log logs (використовується для швидкого підрахунку унікальних елементів).

Використання такої схеми роботи з даними дає змогу суттєво збільшити масштабованість, оскільки немає зв’язків між значеннями, кількість записів обмежується лише технічними потужностями. Недоліком СКБД такого типу є низька швидкість виконання операцій запису даних при збільшенні об’єму БД

Також пошук даних, за їхніми значеннями, а не за ключами, є набагато триваліший порівняно з реляційними базами даних. Тому дуже складно аналізувати наявну в БД інформацію та отримати статистику про дані, що зберігаються.

З огляду на це бази даних такого типу використовуються у випадках, коли зміст інформації, яка міститься у певній комірці, не має значення для оператора бази даних, тобто зв’язки між окремими осередками БД повністю відсутні.

Документо-орієнтовані бази даних

Документо-орієнтована БД добре підходить для зберігання та опрацювання ієрархічних структур даних у форматах JSON (BSON) або XML.

Основним об'єктом документо-орієнтованих баз даних є документ. Для зручності документи часто групуються в колекції за схожою структурою чи вмістом для більш ефективної роботи з даними. Часто всередині документа може бути розташований інший документ' а колекції можуть містити інші колекції – підколекції.

Документи всередині документо-орієнтованої бази даних можуть мати різну структуру. Наприклад, один з документів колекції може мати таку структуру:

```
{
    name: "Alex",
    age: 18,
    languages_p: [ "php", "c++", "pithon"]
}
```

Іншим документом колекції може бути такий:

```
{
    name: "John",
    age: 25,
    project: [
              {name_pr:"DataBank", date_d:"15.10.11"}
```

```
            {name_pr:"WorkSearch", status: true}
            ]
}
```

Наведені документи мають деякі спільні поля, однак кожен має і власні унікальні елементи. Особливістю документно-орієнтованих баз даних є відсутність порожніх полів у документах, що дає змогу спростити роботу зі слабоструктурованими даними.

Кожен документ БД має унікальний ключ, який використовується для пошуку та отримання інформації. З метою пришвидшення пошуку документів зазначені ключові поля індексуються.

Отже, документо-орієнтовані БД можливлюють отримання даних без повного завантаження документів у оперативну пам'ять та є ефективним рішенням для роботи з даними у тих випадках, якщо об'єкти, які зберігаються в БД, мають різні атрибути та між ними немає чітких зв'язків або ці зв'язки не дуже суттєві.

Графові бази даних

Графові моделі баз даних відрізняються наявністю сильних зв’язків між об'єктами, що зберігаються. Зазвичай графові бази даних представляють дані у вигляді окремих вузлів – вершин графа, що можуть мати деяку кількість параметрів та між якими встановлені зв'язки (які часто також мають певні параметри). Графові бази даних є зручними при опрацюванні даних, для яких важливі зв’язки, тому стають дуже популярними для великих онлайнових систем із глибокими зв'язками між об'єктами. У таких випадках вони суттєво випереджають інші типи БД за швидкістю виконання запитів, простотою внесення змін і наочністю подання інформації. Деякі операції над даними набагато простіше виконати з використанням цього типу моделей.

Графові бази знайшли застосування при розробці соціальних мереж, допомагають виявляти складні схеми шахрайства. Аналіз взаємозв'язків у

графових базах даних дає можливість обробляти фінансові операції і операції, пов'язані з покупками, практично в режимі реального часу. За допомогою швидких запитів до графа можна, наприклад, визначити, що потенційний покупець використовує ту саму адресу електронної пошти та кредитну карту, які вже «засвітилися» у відомому випадку шахрайства.

Графові бази даних також дають змогу виявляти певні шаблони взаємозв'язків, наприклад, коли кілька людей пов'язані одні з одними адресою електронної пошти чи коли кілька людей використовують один IP-адрес, але проживають за різними фізичними адресами.

На відміну від інших моделей нереляційних баз даних, графові СКБД підтримують ACID-транзакції, оскільки означена модель не є агрегатною.

На рис. 1.1 наведено приклад графової структури даних. Фундаментальна модель даних графових баз є такою: вузли, з'єднані ребрами (дугами).

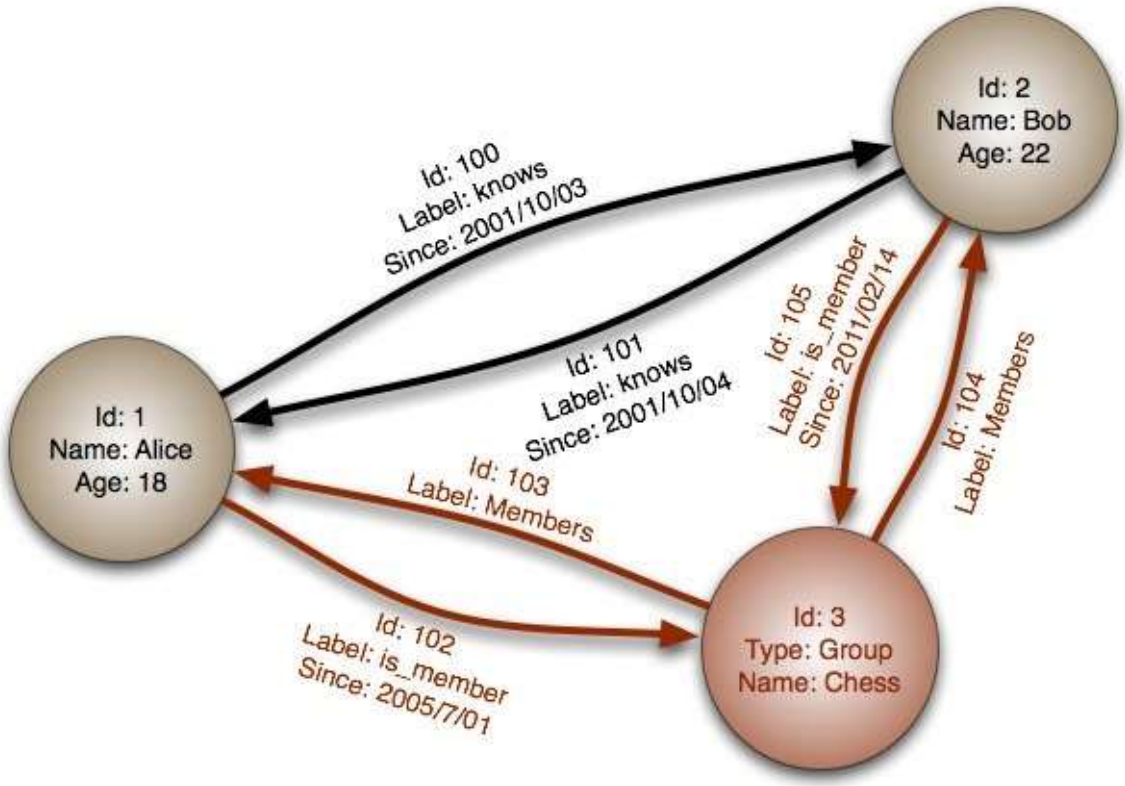


Рис. 1.1. Фрагмент графа, що відображає дані та зв'язки між ними

Однак у різних моделях даних можливі відмінності, зокрема, в тому, які механізми використовуються для зберігання вузлів і ребер у базі даних. Наприклад, база FlockDB – це проста сукупність вузлів і ребер без будь-якого механізму для додаткових атрибутів. Neo4J уможливлює приєднання Java-об'єктів як властивостей вузлів і ребер у неструктурованому вигляді; а Infinite Graph зберігає Java-об'єкти, які є екземплярами підкласів таких вбудованих типів, як вузли і ребра.

У графових базах даних обхід вузлів відбувається зазвичай дуже швидко. В основному це пояснюється тим, що графові бази даних переносять велику частину роботи, пов'язаної з навігацією по зв'язках, з моменту запиту на момент вставки. Це виправдовує себе в ситуаціях, коли продуктивність запиту є більш важливою за швидкість додавання даних.

Деякі графові бази даних використовують спеціалізовані сховища графів, призначені і оптимізовані для зберігання й обробки саме графів. Але таку технологію зберігання використовують не всі графові бази даних. Деякі серіалізують графи і розміщують їх у реляційній, об'єктно-орієнтованій чи іншій базі даних.

Графові бази даних дають змогу будувати моделі будь-якої складності, що дуже добре відображають предметну область. Отримані моделі простіше та більш наочні, ніж ті, що створюються за допомогою традиційних реляційних баз даних або інших NoSQL-сховищ.

Однією з вагомих причин вибору графової бази даних є великий приріст продуктивності при роботі із взаємопов'язаними даними. На великих наборах даних продуктивність графових баз даних залишається незмінною зі збільшенням обсягу збережених даних. Це пов'язано з тим, що запити локалізуються у певній частині графа. У результаті час виконання кожного запиту залежить від розміру частини графа, яку потрібно обійти для виконання запиту, а не від загального розміру графа.

До недоліків баз даних такого типу можна віднести те, що графова БД займає більше місця на диску, ніж реляційні СКБД, та робота з ними потребує спеціальних навиків.

Бази даних "Column Family"

Інформація у базах баних такого типу зберігається у вигляді розрідженої матриці, стовпці та рядки якої використовуються як ключі. Тобто, дані впорядковані по стовпцях і рядках.

Організаційно такі сховища стовпців не дуже суттєво відрізняються від реляційної бази даних. Суттєва перевага сховища даних у вигляді стовпців полягає у можливості структурувати розріджені дані без нормалізації, що пов'язано зі стовпцево-орієнтованим методом зберігання даних.

Бази даних типу "Column Family" можна представити як набір "табличних даних", в яких стовпці поділяються на певні групи або класи стовпців. Кожен клас стовпців включає у себе набір логічно пов'язаних стовпців, які розглядаються як єдине ціле. Дані, які використовуються в інших процесах, зберігаються окремо в інших класах стовпців. Є можливість динамічно додавати нові стовпці, а рядки не обов'язково повинні мати значення для кожного стовпця.

Зазавичай бази даних такого типу використовуються для обробки дуже великих об'ємів даних та веб-індексування. Найбільш відомими СКБД такого типу можна вважати Google BigTable, Apache Cassandra та SimpleDB Amazon.

Кожна з цих категорій або моделей має свої особливості, переваги та обмеження. Не можна сказати, що якась з розглянутих моделей є кращою за інші, однак певні типи баз даних краще розв'язують конкретні проблеми.

Дослідження показують, що нереляційні БД зазначених типів мають суттєві переваги у швидкодії, коли мова йде про опрацювання величезних масивів даних.

Суттєвою перевагою NoSQL баз даних є горизонтальне мастабування – запити обробляються паралельно різними серверами. Орієнтовані на роботу в розподілених системах, СКБД такого типу часто проєктуються так, що процедури шардингу та реплікації даних виконуються самою NoSQL базою.

Варто відзначити, що часто нереляційні бази даних використовуються спільно з реляційними – основна інформація зберігається у реляційній БД, а за кеш відповідає нереляційна.

Завдання для самостійного виконання

1. Розглянути особливості роботи з даними у БД типу ключ-значення на прикладі сховища Redis. Завантажити останню версію Redis можна за посиланням github.com/MSOpenTech/redis/releases, після чого видобути файли у відповідний каталог та запустити сервер (файл redis-server.exe) й клієнт (файл redis-cli.exe). Основні команди для роботи з даними наведені в розділі Commands на офіційному сайті https://redis.io/.

2. Визначити, які з нереляційних СКБД є найбільш популярними у розробників (мають найбільшу кількість користувачів).

Контрольні запитання

1. Що означає термін "NoSQL бази даних"?
2. Чому виникла потреба часткової відмови від реляційних баз даних? Коли доцільно використовувати нереляційцні бази даних?
3. Які основні типи нереляційних баз даних можна виділити?

4. У яких випадках доцільно використовувати бази даних типу "ключ-значення"?
5. Яку структуру мають документно-орієнтовані бази даних?
6. Назвіть основні переваги документно-орієнтованих баз даних.
7. Яку структуру мають графові бази даних?
8. Нереляційні бази даних якого з розглянутих типів підтримують ACID-транзакції?
9. Що таке реплікація та шардинг?
10. До якого типу нереляційних баз даних належать Google Cloud Datastore?

ОПРАЦЮВАННЯ БАЗ ДАНИХ MONGODB. ОБРОБКА ТА ПОШУК ДАНИХ

MongoDB – крос-платформенна, потужна, документо-орієнтована система керування базами даних з відкритим вихідним кодом. На відміну від реляційних баз даних, MongoDB не використовує табличну структуру з чітко заданою кількістю стовпців і типів даних. Усі дані в базах даних, розроблених засобами MongoDB, зберігаються у форматі бінарного JSON – BSON. Вона також має вбудовану підтримку MapReduce-style Aggregation та Geospatial індексів.

До соболивостей MongoDB можна віднести: підтримку індексів, MapReduce, легке масштабування, періодичний вихід оновлень та профілювання запитів.

Усередині MongoDB може бути нуль або більше баз даних, кожна з яких складається з колекцій. Колекція має унікальне ім'я – довільний ідентифікатор, що складається з не більше ніж 128 різних алфавітно-цифрових символів і знака підкреслення. Колекції можуть бути проіндексовані, що поліпшує продуктивність вибірки і сортування даних.

Колекції складаються з нуля або більше «документів». Документ можна уявити як об'єкт, який зберігає деяку інформацію. Будь-який документ всередині колекції може мати власний унікальний набір полів, подібно до рядків у реляційних СКБД, однак може містити набагато більше інформації.

Документ – це набір пар ключ-значення. Значення можуть відрізнятися за типом даних. Є такі типи значень:

- **String** – рядковий тип даних (для рядків використовується кодування UTF-8);

- **Array –** тип даних для зберігання масивів елементів;
- **Binary data** – тип для зберігання даних у бінарному форматі;
- **Boolean** – зберігає логічні значення TRUE або FALSE;
- **Date** – зберігає дату в форматі часу Unix;
- **Double** – числовий тип даних для зберігання чисел з плаваючою крапкою;
- **Integer** – використовується для зберігання цілочисельних значень;
- **JavaScript** – тип даних для зберігання коду JavaScript;
- **Null** – тип даних для зберігання значення Null;
- **Object** – рядковий тип даних;
- **ObjectID** –тип даних для зберігання id документа;
- **Regular expression** – застосовується для зберігання регулярних виразів;

Об'єктна мова запитів у стилі JSON дає змогу проводити великий набір операцій над даними (умовний пошук, складна вставка/оновлення і таке інше) та має підтримку різних типів даних (підтримка масивів зокрема).

«Індекси» в MongoDB майже ідентичні індексам у реляційних базах даних. Результатом запиту до MongoDB є повернення курсору, з якими можна працювати, не завантажуючи при цьому самі дані.

Робота з даними в MongoDB

Для роботи з даними потрібно спочатку запустити сервер (файл mongod.exe, розташований в каталозі mongodb\bin). Для цього треба скачати відповідні файли з офіційного сайту https://www.mongodb.com/. Слід розпакувати папку з файлами та створити в ній папку з назвою «db», оскільки при запуску сервер шукає директорії data/db або db в тій же директорії.

Далі слід запустити командний рядок у режимі адміністратора, перейти в ньому розпаковану папку та виконати команду

```
mongod.exe  - - dbpath =D:\mongodb\db\
```

Тут D:\mongodb\db\ – шлях до створеного катологу «db», що розташований в скачаній з сайту та розпакованій папці mongodb. Після розгортання серверу можна працювати з БД в консолі, запустивши файл mongo.exe з папки mongodb\bin.

Робота з БД MongoDB *в консолі*

Перед початком роботи з даними потрібно встановити потрібну нам БД як поточну, щоб потім її використовувати, командою Use <назва_БД> (рис. 2.1). Якщо такої БД не існує, то MongoDB автоматично створить її при додаванні до неї даних.

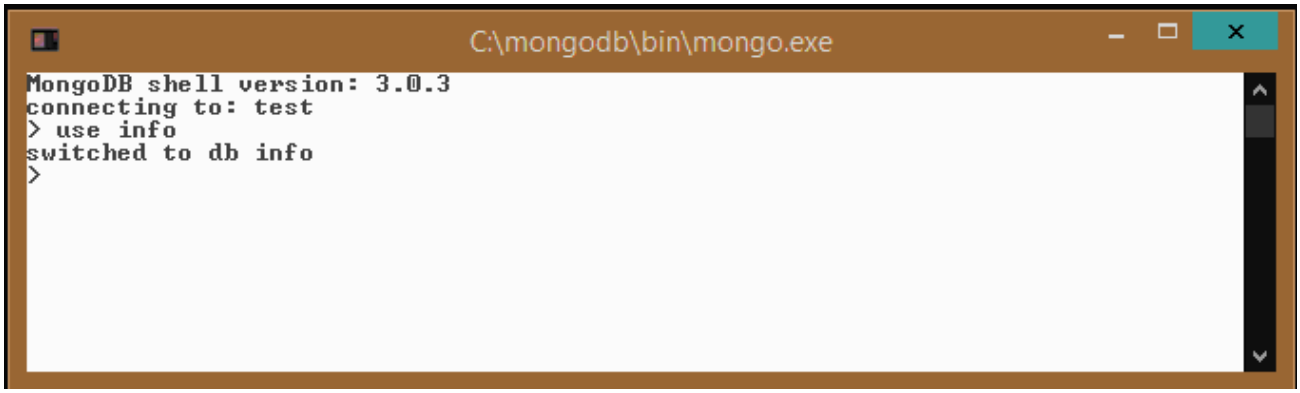


Рис. 2.1. Використання команди Use

Для того, щоб переглянути всі наявні БД в консолі, використовують команду `show dbs`. Список всіх колекцій в поточній БД можна подивитися за допомогою команди `show collections`. Переглянути всі доступні команди для роботи з БД в MongoDB можна командою `db.help()`. Так, наприклад, щоб переглянути команди для роботи з колекціями в БД info потрібно скористатися командою `db.info.help()`.

Команда `db.stats()` дає змогу **переглянути статистику** щодо поточної БД. Щоб переглянути статистику за окремою колекцією, потрібно ввести команду `db.<назва_колекції>.stats()`.

Для **створення колекції** застосовують метод `db.createCollection (name, options)`, де name – назва колекції, а options – необов'язковий об'єкт з додатковими налаштуваннями ініціалізації. Наприклад:

```
db.createCollection ("student").
```

Ім'я колекції не повинно починатися з префікса system, позаяк він зарезервований для внутрішніх колекцій (наприклад, колекція system.users містить всіх користувачів бази даних). Також ім'я колекції не повинно містити знака долар – $.

Щоб **перейменувати** наявну колекцію, слід використовувати функцію renameCollection:

```
db.student.renameCollection("нова назва")
```

Для **додавання** до колекції даних використовується функція insert():

```
db.student.insert ({"name": "Alex", "age": 18,
languages_p: [ "php", "c++", "pithon"]})
```

У цьому випадку в колекцію student додається простий об'єкт. В результаті вдалого додавання консоль виводить вираз:

```
WriteResult ({ "nInserted": 1})
```

Є другий спосіб додавання в БД документа, який містить два етапи: визначення документа (document = ({...})) і власне його додавання:

```
document = ({ "name": " Alex", "age": 18, languages_p:
[ "php", "c++", "piton"]})

db.student.insert(document)
```

Щоб **переглянути** додані дані, використовують функцію `find()` (рис. 2.2) або `find().pretty()` – для форматного виведення даних.

Рис. 2.2. Запит на виведення даних

Зазначимо, що для кожного документа у MongoDB автоматично створюється поле "_id" типу **ObjectID** для зберігання унікального id документа.

Для **видалення** з БД документів за певним значенням поля використовується команда

```
db.db_name.remove({"field":"value"}).
```

Щоб **оновити** документи згідно з певним критерієм використовується функція `update()`, яка приймає три параметри для оновлення документа:

- документ, який буде змінений;
- документ, на який буде змінений;
- параметри оновлення документів (upsert і multi). Якщо параметр upsert має значення true, то буде оновлено документ, якщо він знайдений, і створений новий, якщо документа немає; якщо параметр upsert має значення false, то не буде створено новий документ, якщо знайденого документа немає. Парамет multi визначає, чи всі документи вибірки потрібно оновлювати (якщо приймає значення true), чи повинен оновитися лише перший елемент у вибірці (використовується за умовчанням).

Наприклад, запит

`db.users.update({name:"Olga", lastname: "Petrenko"}, {$set:{lastname:"Ivanenko"}}, {multi:true, upsert:false})` змінить всі документи з вибірки.

Працювати з БД MongoDB можна за допомогою графічних оболонок Robo 3T або Compas.

Робота з даними засобами графічної оболонки Robo 3T

Розглянемо опрацювання БД засобами оболонки Robo 3T. Після її встановлення та запуску потрібно встановити з'єднання з сервером MongoDB.

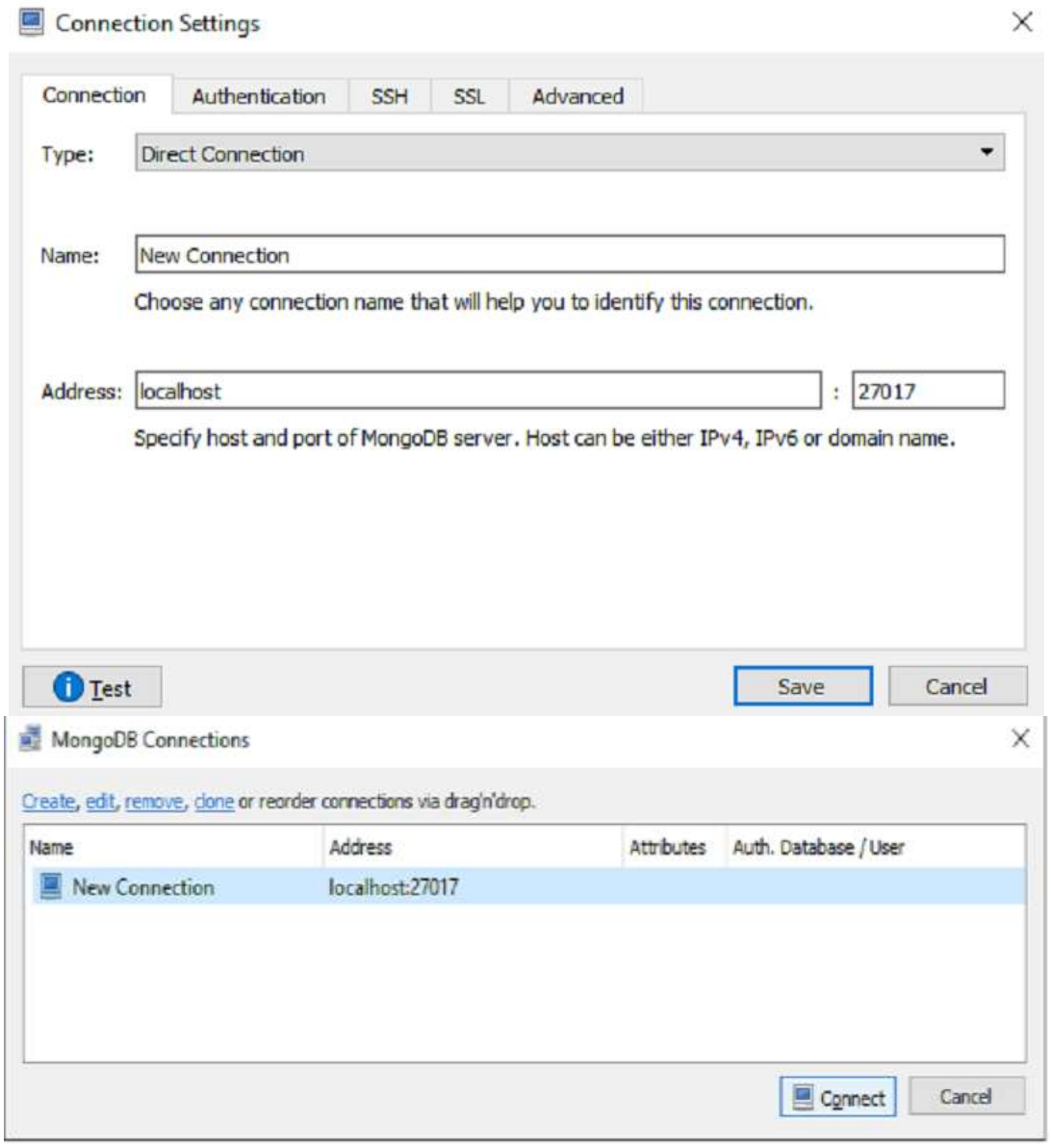


Рис. 2.3. Підключення до сервера MongoDB

Після цього у верхній лівій частині користувач побачить всі БД, що є на сервері (рис. 2.4)

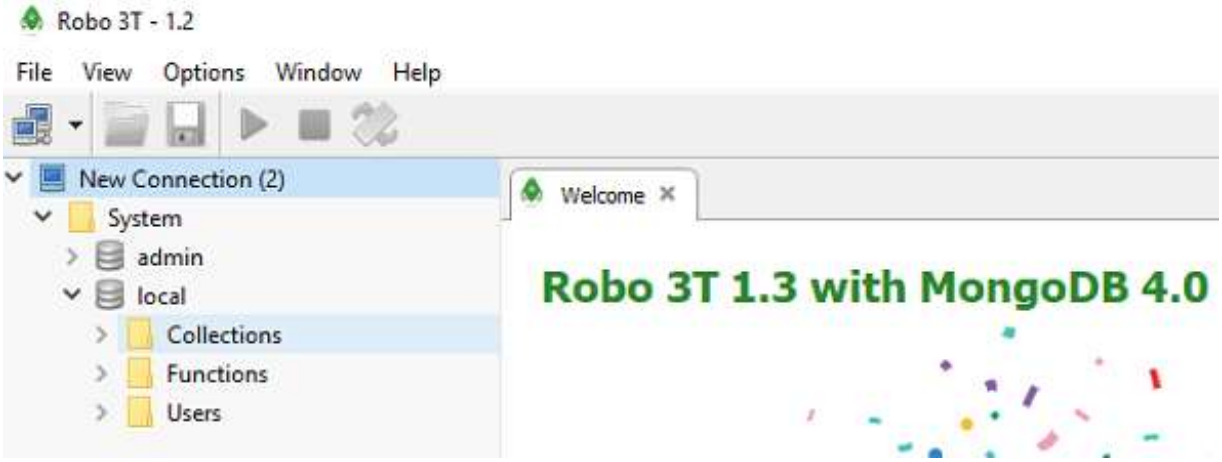


Рис. 2.4. Підключення до сервера MongoDB

Для створення нової БД засобами Robo 3T, потрібно викликати контекстне меню пункту New Conection та обрати пункт Create Database (рис. 2.5). У вікні, що з'явилося, слід ввести ім'я ДБ.

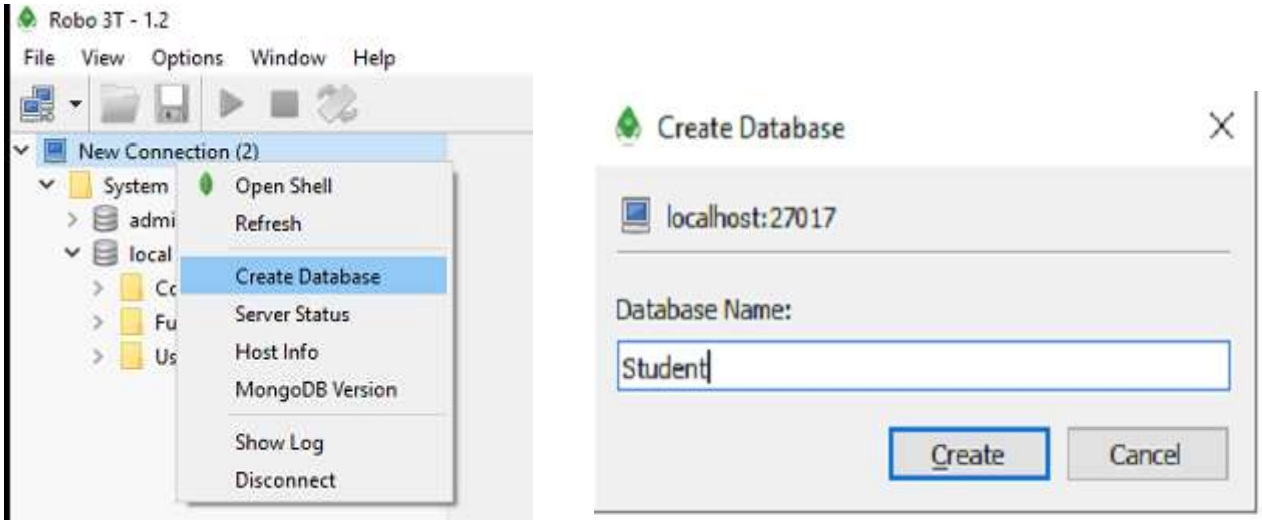


Рис. 2.5. Створення нової БД MongoDB у Robo 3T

Далі слід створити колекції, використовуючи контекстне меню пункту Collection, створеної БД. Усі наявні в базі даних колекції, можна переглянути у пункті Collection.

Для вставки документів до колекції потрібно обрати відповідну

колекцію та обрати команду Insert Collection. У вікні, що відкриється, потрібно вказати відповідні поля та їх значення у форматі JSON (рис. 2.6). Документи, на відміну від записів у реляційних базах даних, можуть мати різну структуру, тобто мати різні поля та містити різні типи даних.

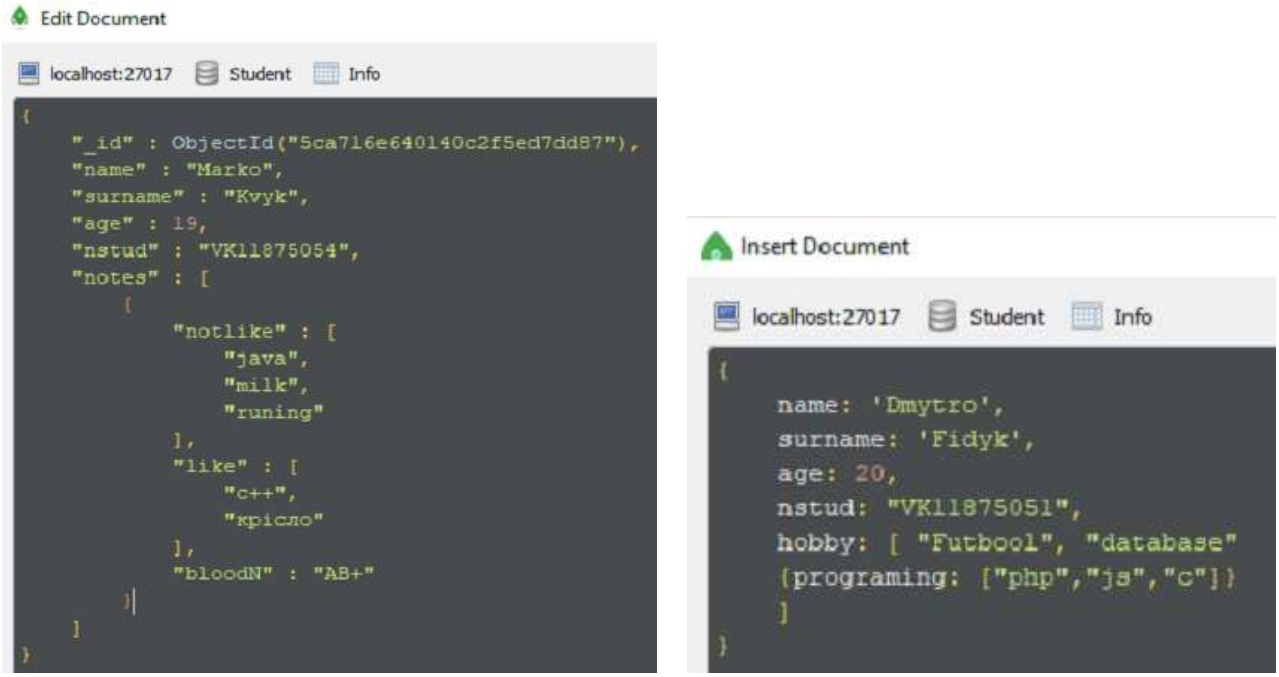


Рис. 2.6. Додавання документів до колекції

У Robo 3T всі запити можна писати у рядку виконання, що розташований вгорі кожної вкладки. Переглянути всі документи колекції та їх вміст у Robo 3T можна запитом `db.getCollection(назва_колекції).find()` (автоматично формується, якщо скористатись командою View Document контекстного меню обраної колекції) або `db.назва_колекції.find()`. Якщо потрібно вивести дані згідно з певним критерієм, можна також скористатись функцією `find()`. Як аргумент функції потрібно вказати відповідні поля та їхні значення. Так, наприклад, команда `db.Info.find({name: "Marko"})` виведе всі документи колекції Info, в яких значення поля name = Marko (рис. 2.7).

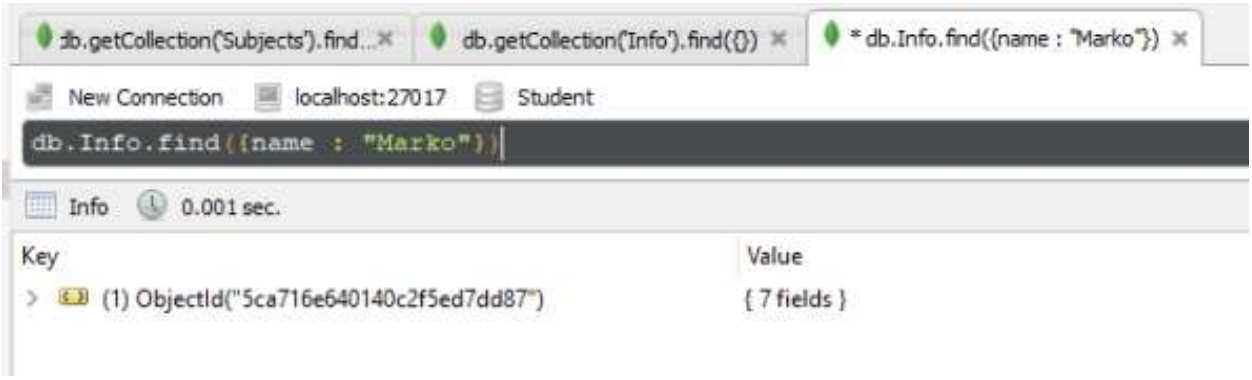


Рис. 2.7. Виведення інформації згідно з деяким критерієм

Розглянемо основні команди, що можуть бути потрібні при формуванні запитів на вибірку даних.

Умовні оператори:

- $eq – дорівнює(=);
- $gt – більше (>);
- $lt – менше (<);
- $gte – більше або рівне (>=);
- $lte – менше або рівне (<=);
- $ne – не дорівнює (<>);
- $in – визначає масив значень, одне з яких має мати поле документа (рис. 2.8);
- $nin – визначає масив значень, одне з яких не має мати поле документа.

Запит `db.Info.find({"age": {$gt: 18, $lte: 21}})` поверне всі документи колекції Info, в яких значення поля age перебувають у діапазоні (18; 21].

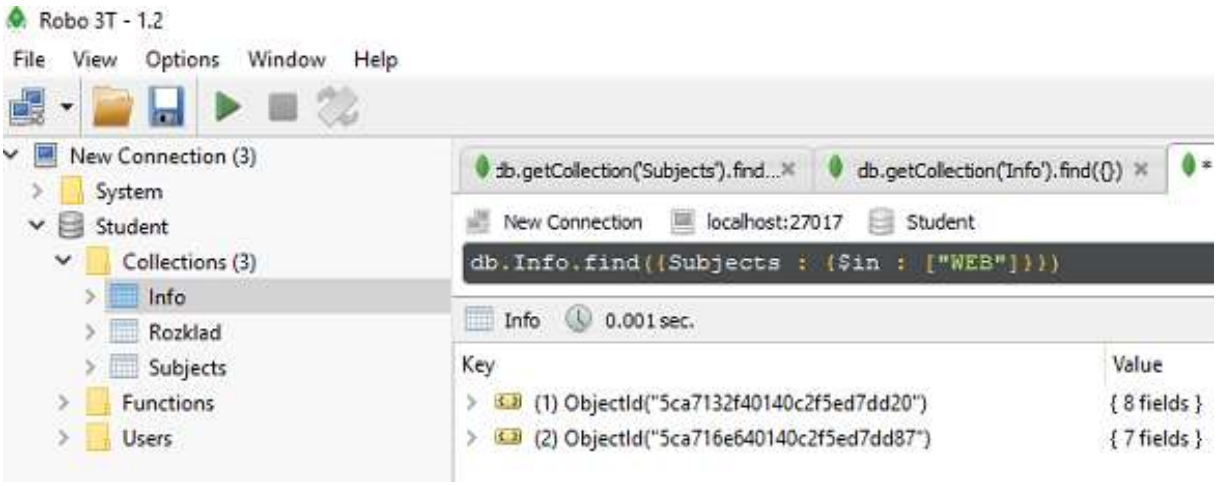


Рис. 2.8. Виведення інформації по всіх студентах, що вивчають дисципліну "Web"

Запит `db.Info.find({"age": {$in: [18, 21]}})` поверне всі документи колекції Info, в яких значення поля age дорівнює 18 або 21.

Запит `db.Info.find({"name": {$in: ["Marko", "Dmytro"]}})` поверне всі документи колекції Info, в яких значення поля name дорівнює Marko або Dmytro.

Логічні оператори:

- $ Or – з'єднує дві умови, документ повинен відповідати одній з цих умов;
- $ And – з'єднує дві умови, і документ повинен відповідати обом умовам;
- $ Not – документ повинен НЕ відповідати умові;
- $ Nor – з'єднує дві умови, і документ повинен НЕ відповідати жодній умові.

Наприклад, запит

```
db.Info.find({$and: [{suname:"Petrenko"}, {age :{$gt :32}}]})
```

поверне всі документи, в яких значення поля suname є "Petrenko" та значення поля age більше за 18.

Для отримання кількості документів в колекції використовується

функція `count()`. Як аргумент функції можна вказати критерій для підрахунку документів.

Для вибірки даних із сортуванням за деяким полем можна скористатись функцією `sort()`. Аргументами функції є назва поля, за яким проводиться сортування та значення 1 для сортування за зростанням і (-1) для сортування за спаданням відповідно. Наприклад, `db.Info.find().sort({name: 1})`.

Для роботи з даними можна використовувати функції JavaScript. Створена функція може бути параметром функції `find()`. Наприклад, наступний запит поверне всі документи, в яких поле name має значення "Ivan":

```
func = function() { return this.name == "Ivan"; }
db.Info.find(func)
```

На рис. 2.9 наведено результат запиту на повернення всіх документів, що не містять поля " notes".

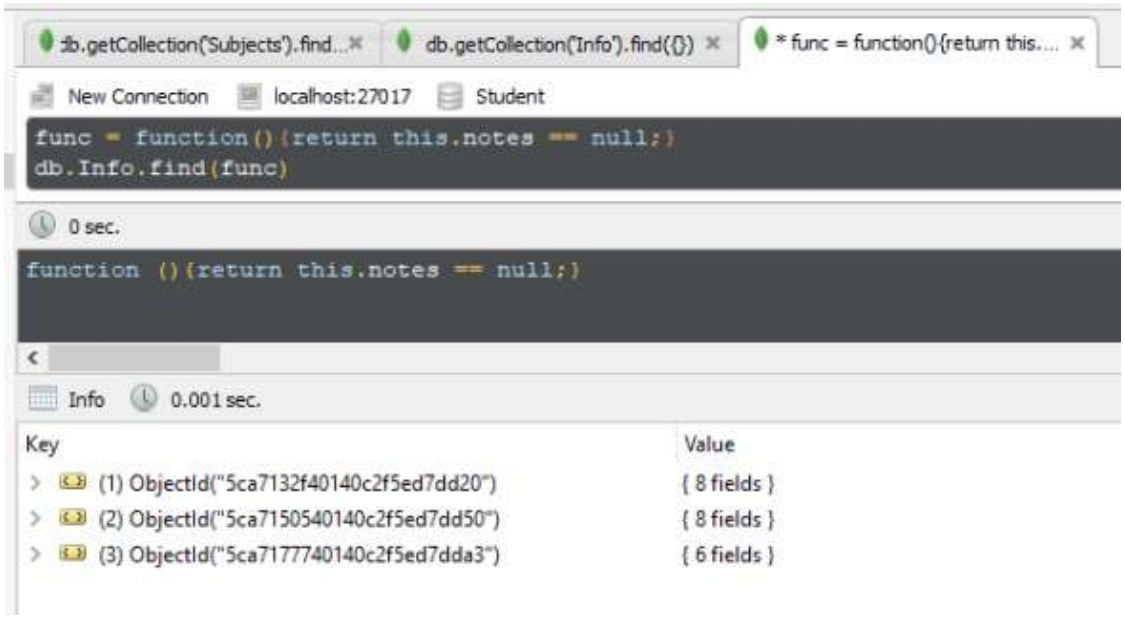


Рис. 2.8. Використання функцій при формуванні запитів

Для створення індексу за полем використовують функцію `createIndex()`. Якщо параметр "unique" має значення true створиться унікальний індекс:

```
db.Info.createIndex({"name" : 1}, {"unique" : true}).
```

Щоб переглянути всі індекси з колекції, використовують запит `db.назва_колекції.getIndexes()` (рис. 2.9), для видалення індексу – `db.назва_колекції.dropIndex("назва_індексу")`.

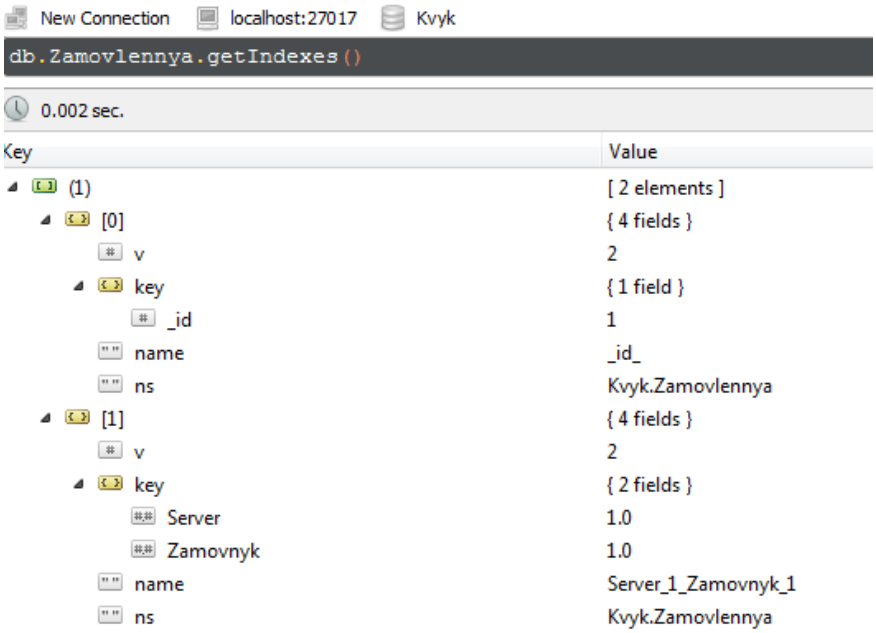


Рис. 2.9. Перегляд індексів в колекції

Робота з даними засобами графічної оболонки Compas

Для роботи з даними в Compas потрібно скачати відповідний файл з офіційного сайту та встановити його. Після встановлення та запуску потрібно налагодити з'єднання з сервером MongoDB.

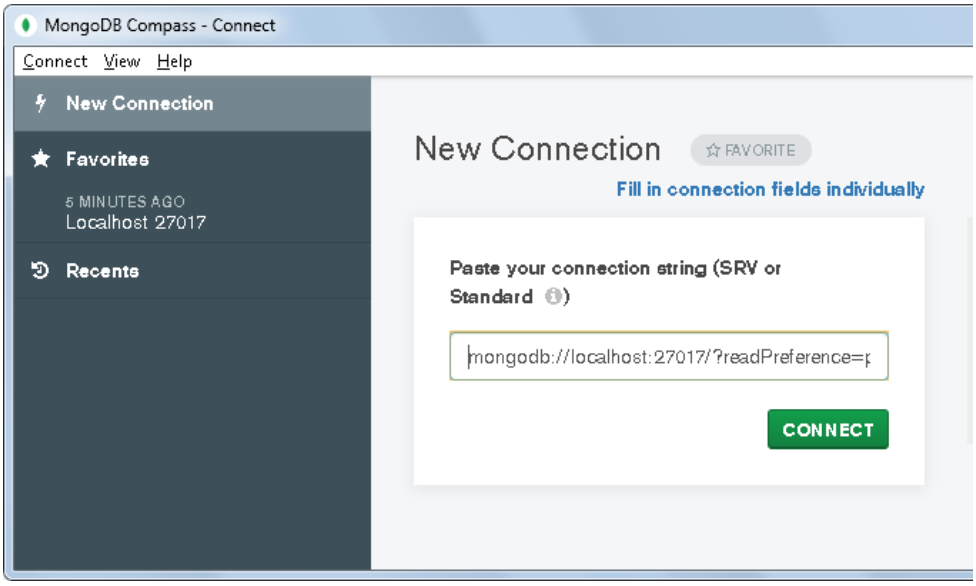


Рис. 2.10. Підключення до сервера MongoDB

Після підключення користувач побачить всі бази даних, що зберігаються на сервері (рис. 2.11)

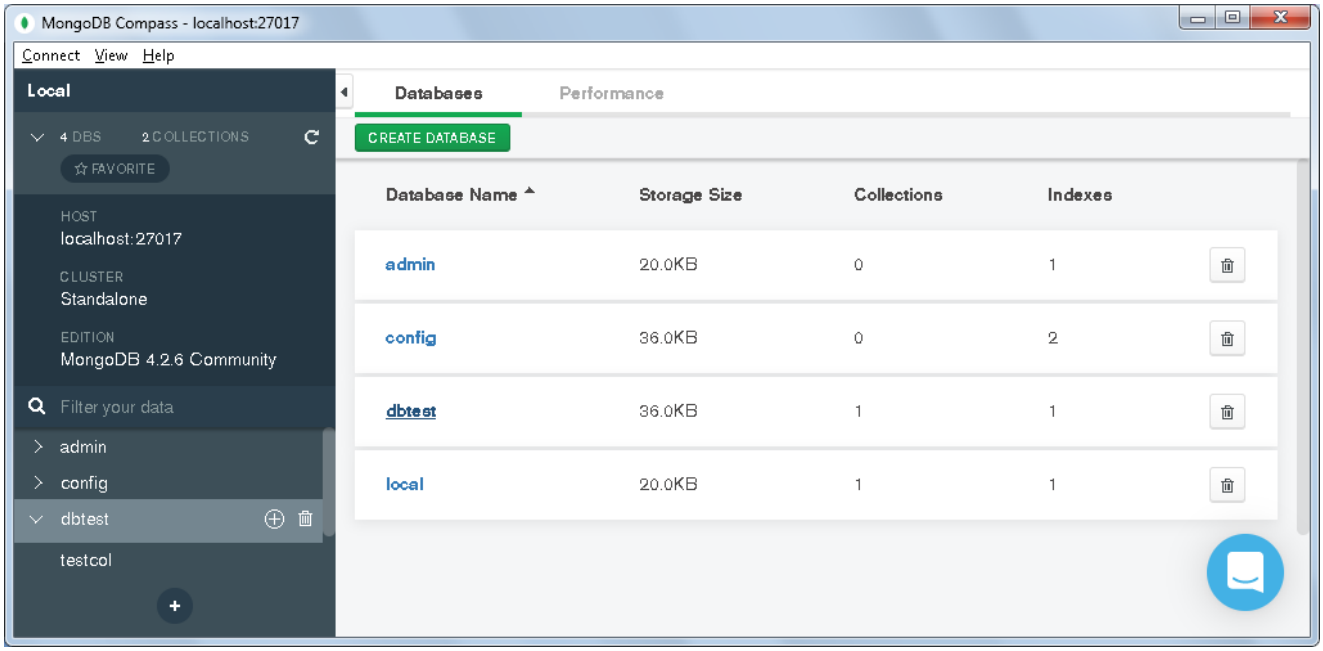


Рис. 2.11. Відображення існуючих баз даних

Для перегляду вмісту БД потрібно відкрити її. Compas пропонує три способи відображення вмісту та структури документів (рис. 2.12)

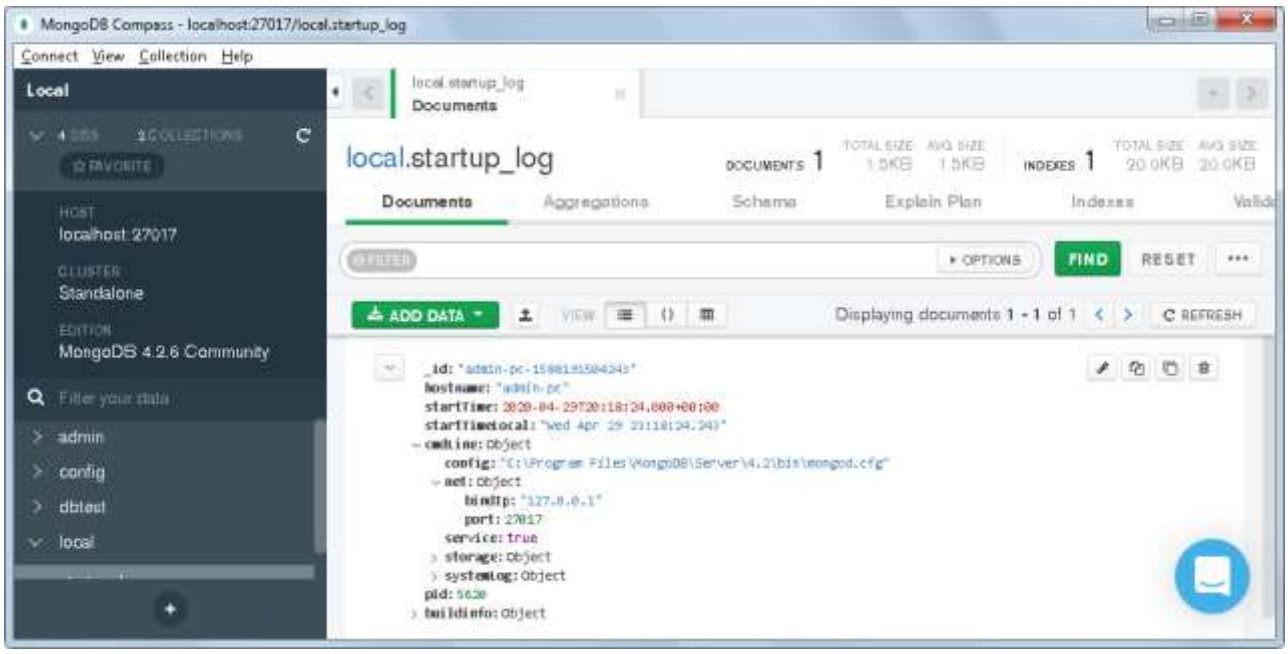


Рис. 2.12. Вміст системної БД local

Для створення нової БД засобами Compas потрібно обрати пункт Create Database. У вікні, що з'явилось, слід ввести ім'я ДБ та їм'я

колекції (рис. 2.13).

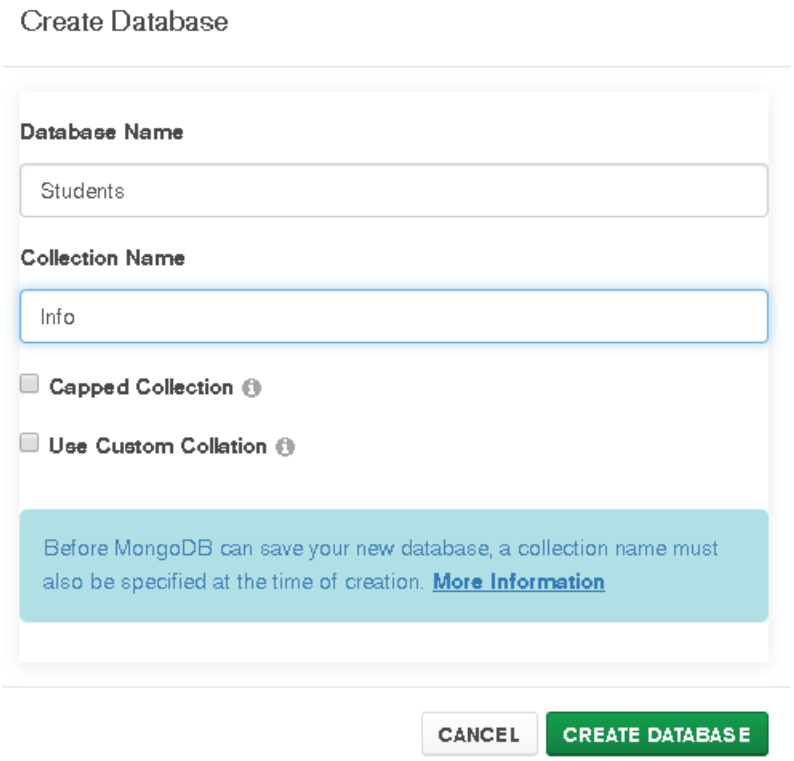


Рис. 2.13. Створення нової БД MongoDB у Compas

Після створення БД можна додати інші колекції, обравши пункт Create Collection. Для вставки документів до колекції потрібно обрати відповідну колекцію та обрати команду Add Data – Inset Document для додавання даних з клавіатури. У вікні, що відкриється, потрібно вказати відповідні поля, обрати тип поля з розкривного списку праворуч поля та задати значення поля у форматі JSON (рис.2.14).

Для імпорту даних із зовнішнього джерела у форматі JSON або CSV слід обрати пункт Add Data - Import Fille.

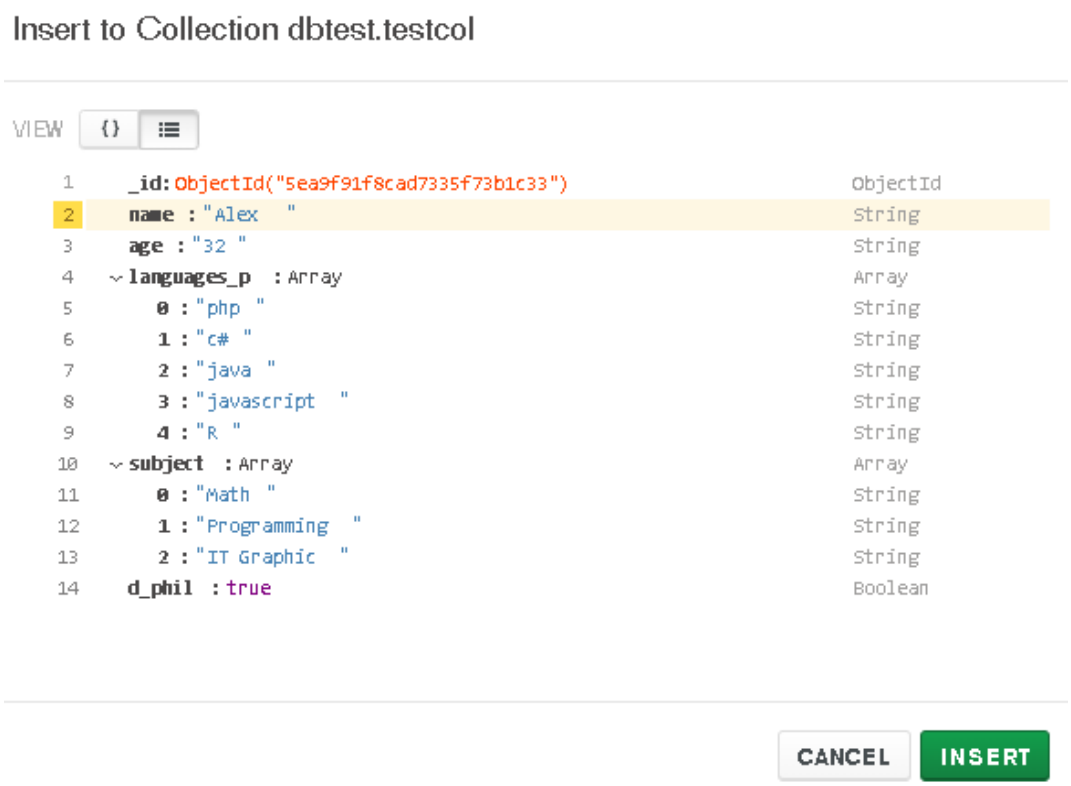


Рис. 2.14. Додавання документів до колекції

Запити у Compas можна писати у рядку виконання, що розташований вгорі першої вкладки. Особливістю цієї графічної оболонки є те, що не потрібно писати запит цілком, як при роботі з даними у командному рядку чи у Robo 3T. Ця графічна оболонка пропонує у рядку запитів такі пункти: Filter, Project, Sort, Collation, які розкриваються при натисканні на стрілку кнопки Option (рис. 2.15). Так, запити написані у рядку Filter, мають містити лише умову пошуку, не включаючи команди find(). Аналогічно, у рядку Sort достатньо вказати назву поля, за яким проводитиметься сортування та умову сортування (1 – за зростанням, -1 – за спаданням).

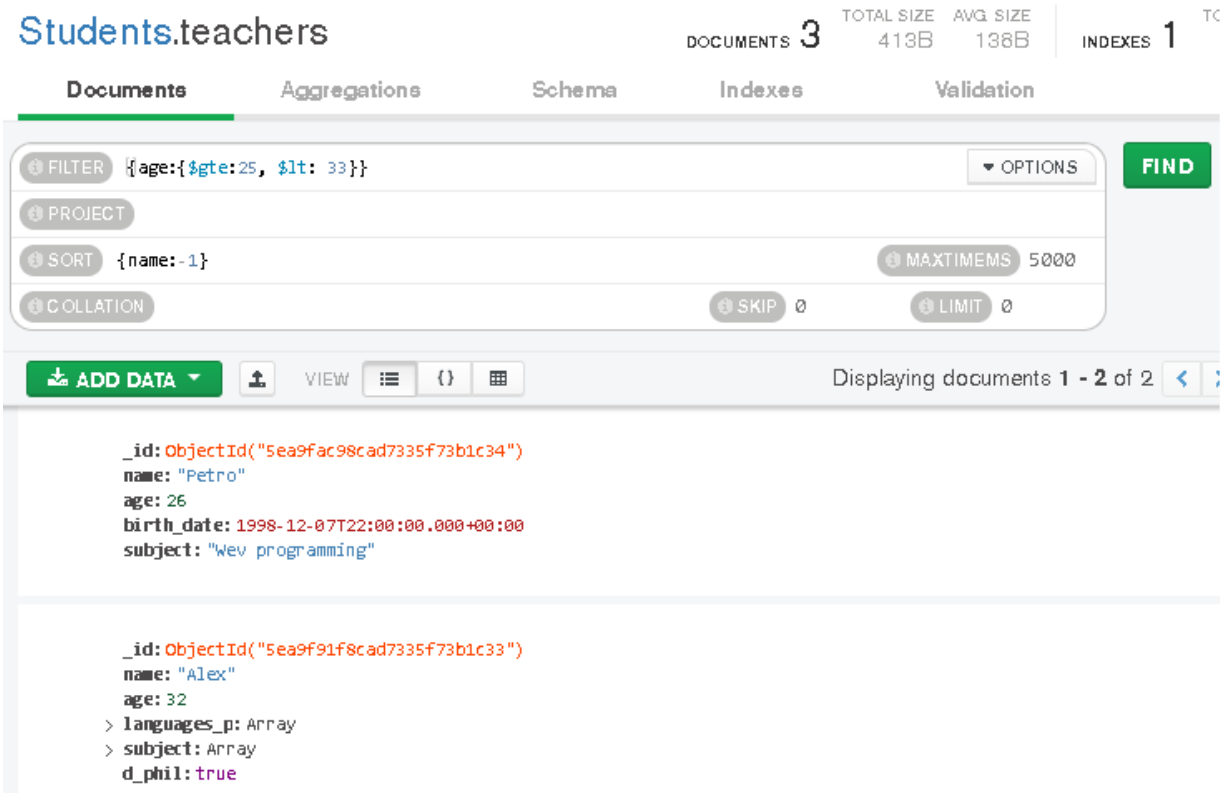


Рис. 2.15. Пошук певного документу в Compas

Слід відзначити, що розробники цієї графічної оболонки також передбачили можливість обрати при написанні запиту потрібну функцію з випадаючого списку (рис. 2.16).

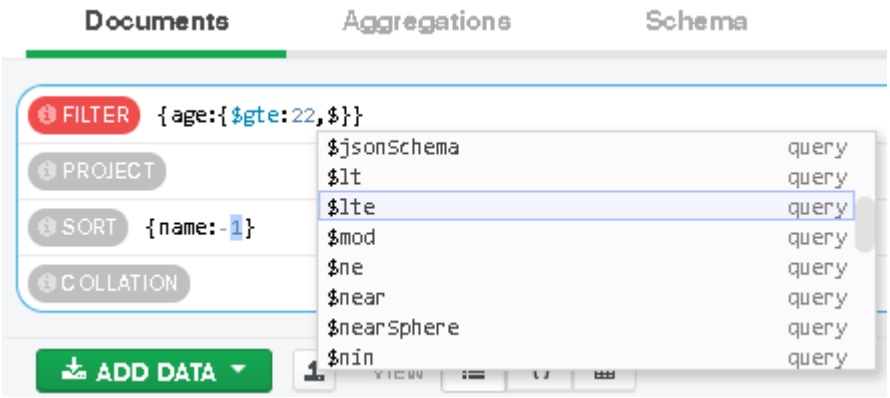


Рис. 2.16. Вибір відповідної команди з переліку

Усі індекси для означеної колекції можна переглянути на вкладці Indexes. Для створення нового індексу за полем слід скористатися командою Create Index вкладки Indexes (рис. 2.17). У вікні, що відкриється, користувач має вказати назву індексу, поле чи декілька полів (пункт Add another field) та обрати відповідні параметри даного

індексу.

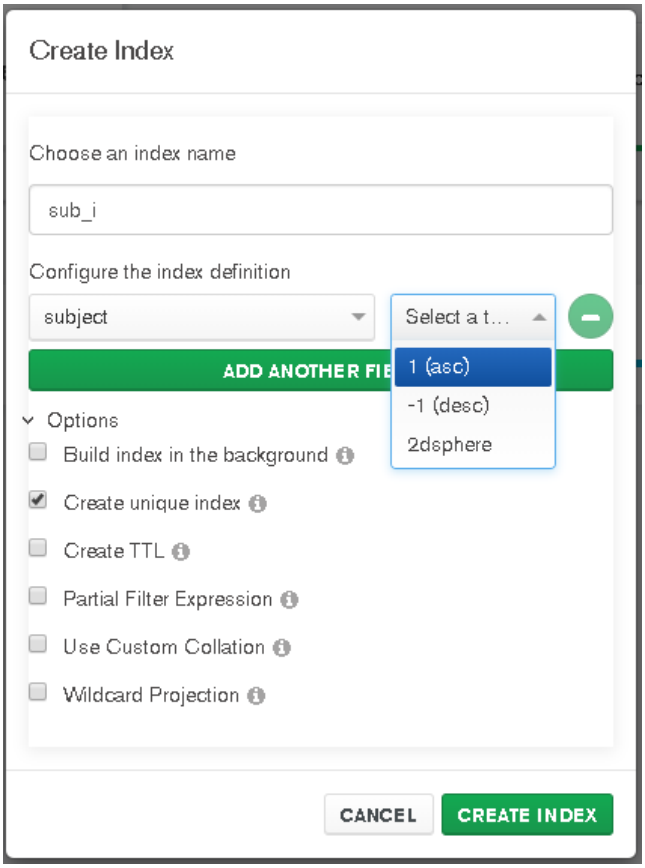


Рис. 2.17. Створення індексу у оболонці Compas

Завдання для самостійного виконання

1. Розглянути наведені вище особливості роботи з даними засобами MongoDB.
2. Використовуючи наведені вище вказівки, розгорнути сервер MongoDB (файл mongod.exe).
3. Створити нову БД згідно з вашим варіантом за допомогою графічної оболонки Compas або Robo 3T, що складається з 3–4 колекцій.
4. Розглянути основні команди для роботи з БД, колекціями та документами, що містяться в БД. Додати до колекцій відповідні документи (по 3–4 документи). Використати різні типи даних (String, Array, Boolean, Date, Integer та інші). Проаналізувати структуру бази даних.

5. Оновити дані згідно з деяким критерієм, використовуючи параметри upsert і multi функції update().
6. Сформувати декілька BSON-запитів на вибірку даних. Обов’язково використати пошук даних, що належать деякому діапазону; провести вибірку даних, що передбачає перевірку на належність множині; провести сортування даних. Проаналізувати запити, їх призначення та результат виконання.
7. Використовуючи js-функцію, вивести документи що містять/не містять вказані поля. Розробити та використати інші js-функції для роботи з даними.
8. Розглянути роботу з даними типу Object та функцією Timestamp().
9. Створити унікальний індекс та індекс за декількома полями. Переглянути всі індекси в базі даних та в деякій колекції.
10. Підключитись до розробленої БД в консолі. Переглянути структуру БД. Вивести дані згідно з деяким критерієм.

Контрольні запитання

1. До якого типу баз даних належать БД, рзроблені засобами MongoDB? Яку структуру вони мають?
2. Які обмеження накладаються на ім'я колекції в MongoDB?
3. Яку команду використовують для того, щоб переглянути всі наявні БД в MongoDB?
4. Як видалити колекцію з БД в MongoDB?
5. Які типи даних підтримуються в MongoDB?
6. Як можна додати нові дані до наявної колеції?
7. За що відповідають параметри upsert і multi функції update()?
8. За домогою яких команд можна реалізувати видалення даних з колекції?
9. Які особливості синтаксису запитів для роботи з даними у графічній оболонці Compas?
10. Імпорт яких форматів даних підтримується у Compas?

РОБОТА 3 БАЗАМИ ДАНИХ НА ПЛАТФОРМІ FIREBASE

Firebase – це платформа розробки мобільних та веб-застосунків. Містить багато сервісів і служб, доступ до повних функцій яких завичай є платним.

Firebase дозволяє, зокрема, працювати з документно-орієнтованими базами даних, що складаються з колекцій та документів, у режимі реального часу, що надає змогу розробникам зберігати й синхронізувати дані. Firebase надає програмістам API, який дає можливість синхронізувати дані додатків між клієнтами та зберігати їх у хмарі.

На сьогодні Firebase надає змогу працювати з двома типами баз даних – Realtime Database та Cloud Firestore.

На відміну від Realtime Database, де дані зберігаються у реальному часі в дереві JSON, у Cloud Firestore дані зберігаються в документах (як набори пар ключ-значення) і колекціях (набір документів). До основних переваг цього сервісу можна віднести: поліпшену базову файлову структуру; автономну підтримку веб-клієнта; атомарність операцій запису і транзакцій; підвищення надійності і продуктивності; більш безпечне й автоматичне масштабування.

Для роботи из сервісами потрібно зайти на офіційний сайт Firebase https://firebase.google.com і авторизуватись за допомогою облікового запису Google. Для початку роботи треба обрати пункт Get Started (рис. 3.1).

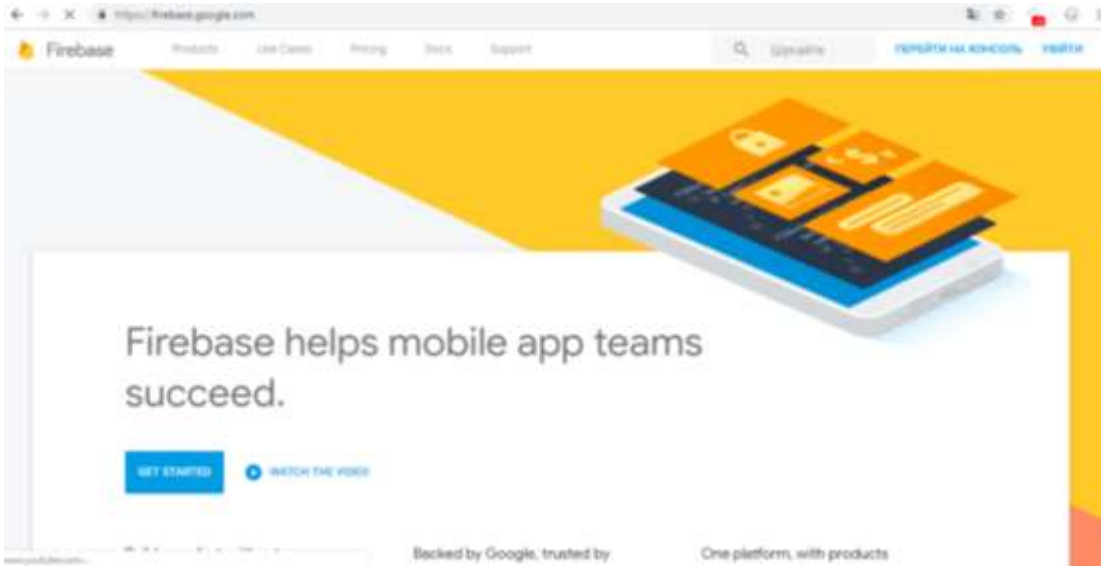


Рис. 3.1. Початок роботи з даними на платформі Firebase

Якщо потрібно, слід створити новий проєкт і задати йому назву (рис. 3.2).

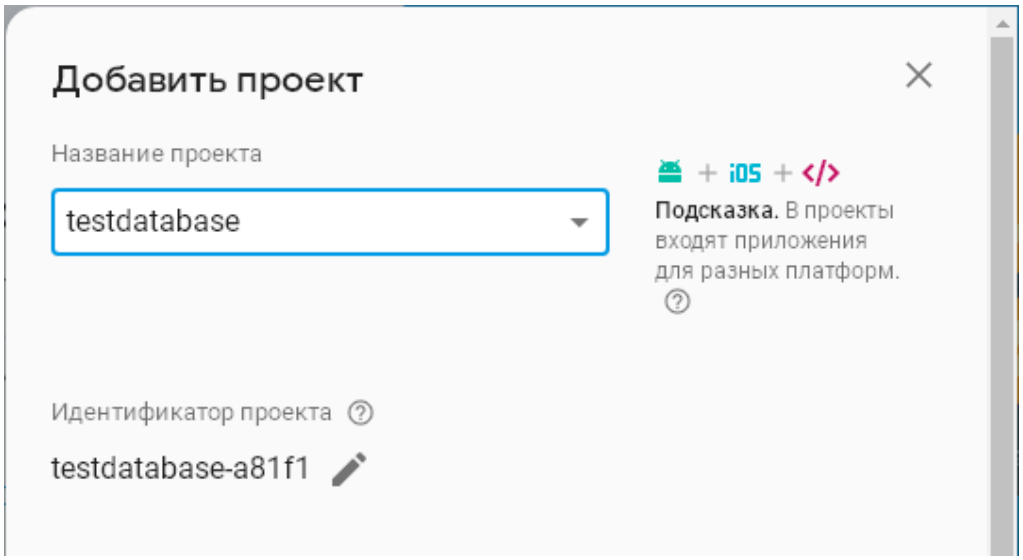


Рис. 3.2. Створення нового проєкту на платформі Firebase

Надалі усі створені й активні проєкти будуть відображатися на цій вкладці.

Для роботи з базами даних потрібно перейти у пункт Database вкладки Develop (рис. 3.3) і створити базу даних. Firebase пропонує за замовчуванням режим Cloud Firestore. Щоб працювати у режимі Realtime Database, потрібно обрати відповідний пункт трохи нижче у тому самому вікні.

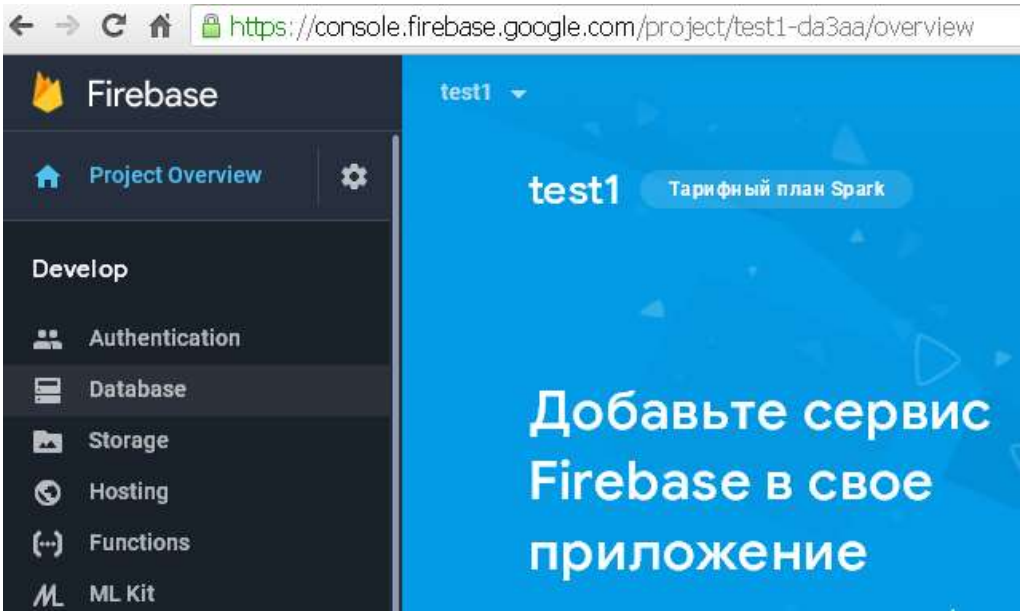


Рис. 3.3. Створення нового проєкту на платформі Firebase

Firebase надає змогу працювати з даними у тестовому та захищеному режимах (рис. 3.4). Тестовий режим дозволяє використання вашої бази даних всім користувачам, і всі вони мають права адміністартора. У ***захищеному режимі*** цього робити не можна.

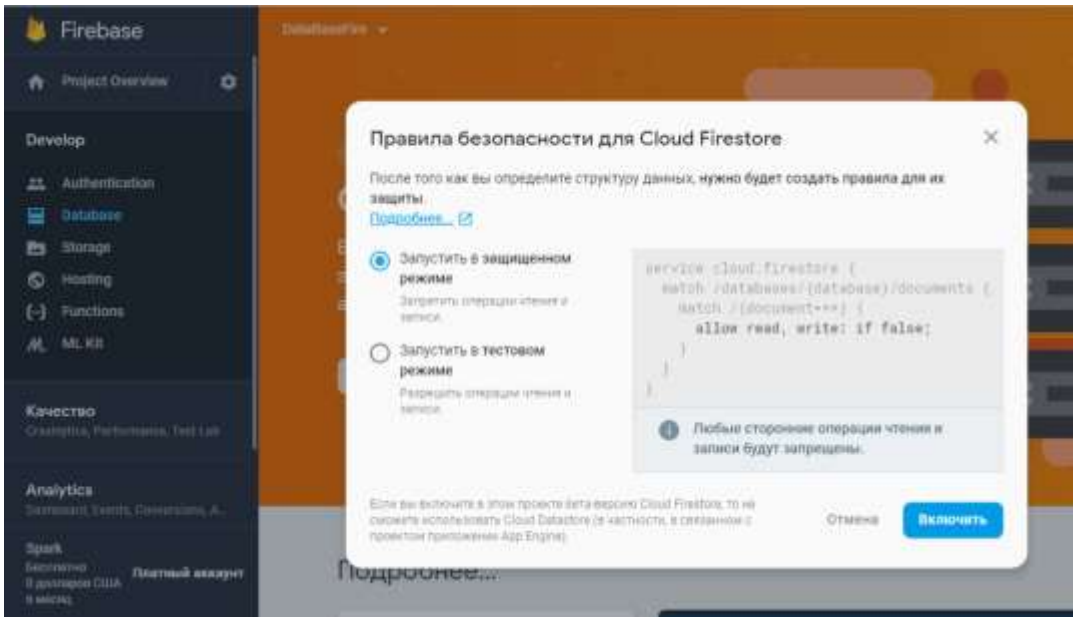


Рис. 3.4. Вибір режиму роботи з базою даних

Користувач згодом може змінити обраний режим у пункті Правила вкладки Database. Для цього достатньо змінити false на true у вікні, наведеному на рис. 3.5.

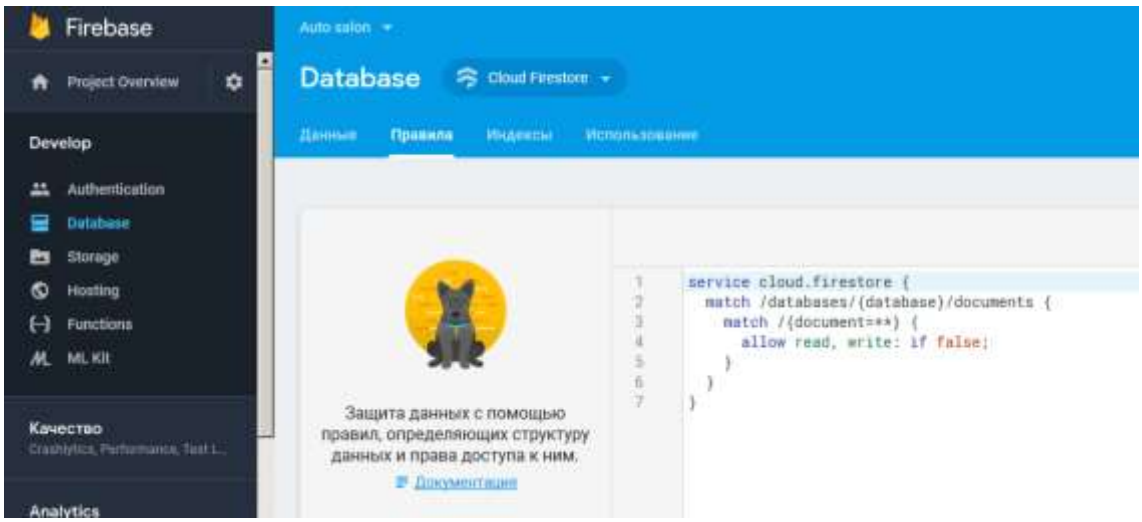


Рис. 3.5. Вікно зміни режимів роботи з базою даних

Робота з даними у Cloud Firestore

Розглянемо особливості роботи з базою даних у Cloud Firestore. На початку роботи потрібно створити нову колекцію і наповнити її документами. У вікні додавання нової колекції потрібно задати ідентифікатор колекціїї.

Кожен документ у колекції складається з пар ключ-значення. Для кожного поля потрібно явно вказати його тип. Можливі типи даних відображені у випадаючому списку у вкладці «Тип» (рис. 3.6).

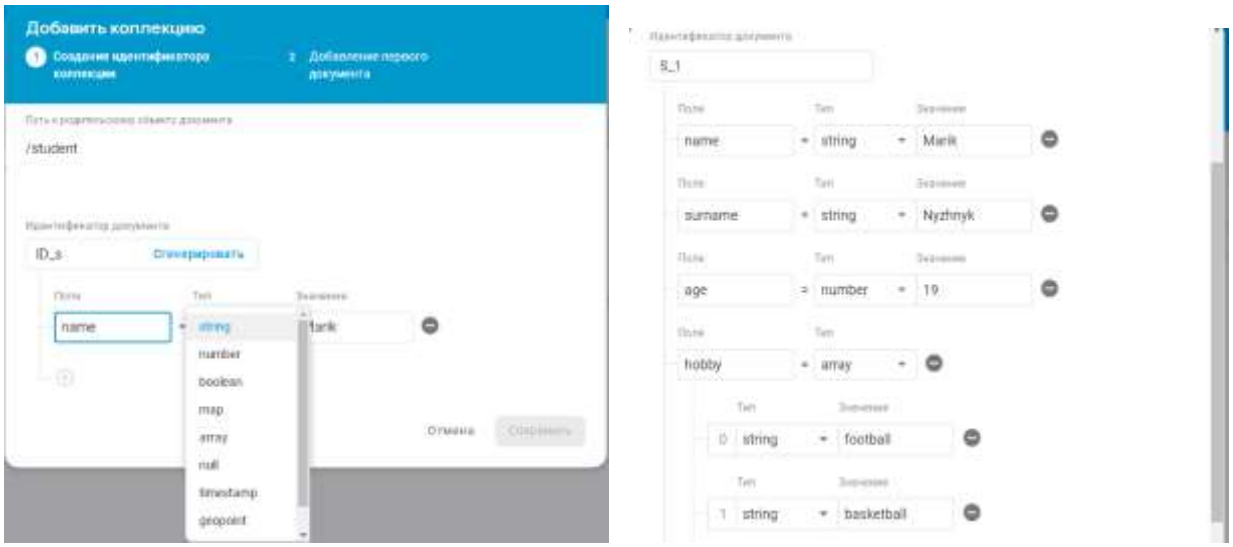


Рис. 3.6. Створення нового документа у колекції бази даних

Розглянемо коротко основні типи даних:

- **String –** примітивний тип, який представляє рядкове значення**;**
- **Number –** значення типу Integer або типу Float;
- **Boolean –** примітивний тип, який представляє логічне значення (**true** або **false);**
- **Map –** використовується для простих відображень ключ-значення;
- **Array –** примітивний тип, який представляє масив значень;
- **Null –** тип, який представляє порожнє значення (тобто нічого);
- **Timestamp –** часова мітка, є собою моментом часу, незалежним від будь-якого часового поясу або календаря, представлений у вигляді секунд та долей секунд з наносекундним дозволом під час UTC Epoch;
- **Geopoint –** використовує надані значення широти і довготи;
- **Reference –** тип, який має можливість надати шлях до іншого документа.

Після додавання та збереження даних до документа користувач має змогу наглядно побачити додані дані та їх структуру (колекції, документи та поля документів) (рис. 3.7).

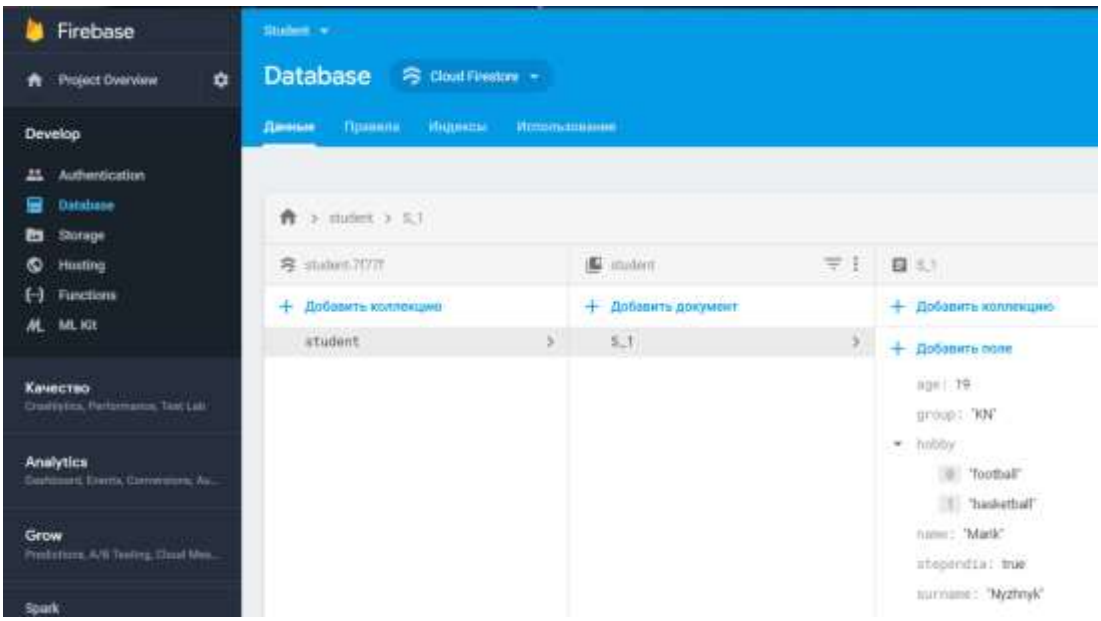


Рис. 3.7. Перегляд даних

У базах даних, створених засобами Firebase Cloud Firestore, колекції та документи можуть містити вкладені підколекції.

Для зручності можна згрупувати дані або задати умову, згідно з якою відображаються дані, скориставшись відповідними командами контексного меню документа (рис. 3.8).

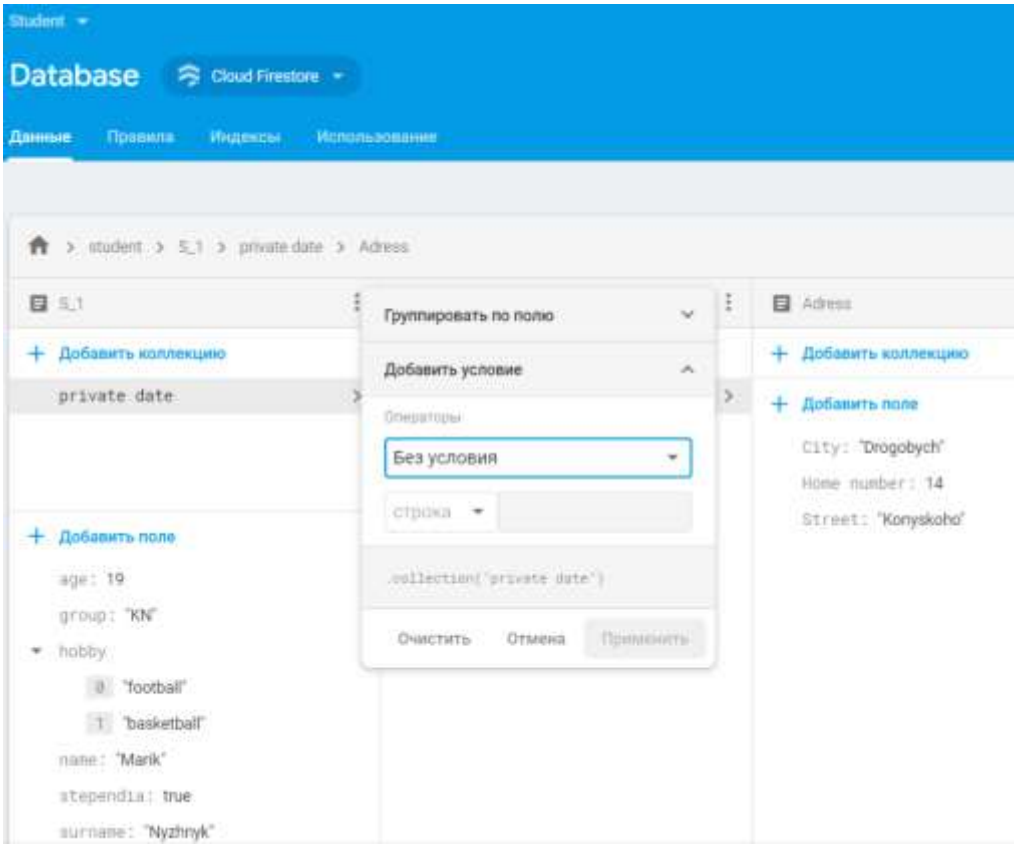


Рис. 3.8. Формування умови для відображення даних

Для видалення документа чи колекції можна скористатись відповідно командами **Видалити колекцію** чи **Видалити документ.**

Для оновлення даних у документі потрібно клікнути на відповідне поле та натиснути на значок редагування (рис. 3.9).

Варто пам’ятати що бази даних створені засобами Realtime Database та Cloud Firestore – це дві різні бази даних, а не їхні дублікати.

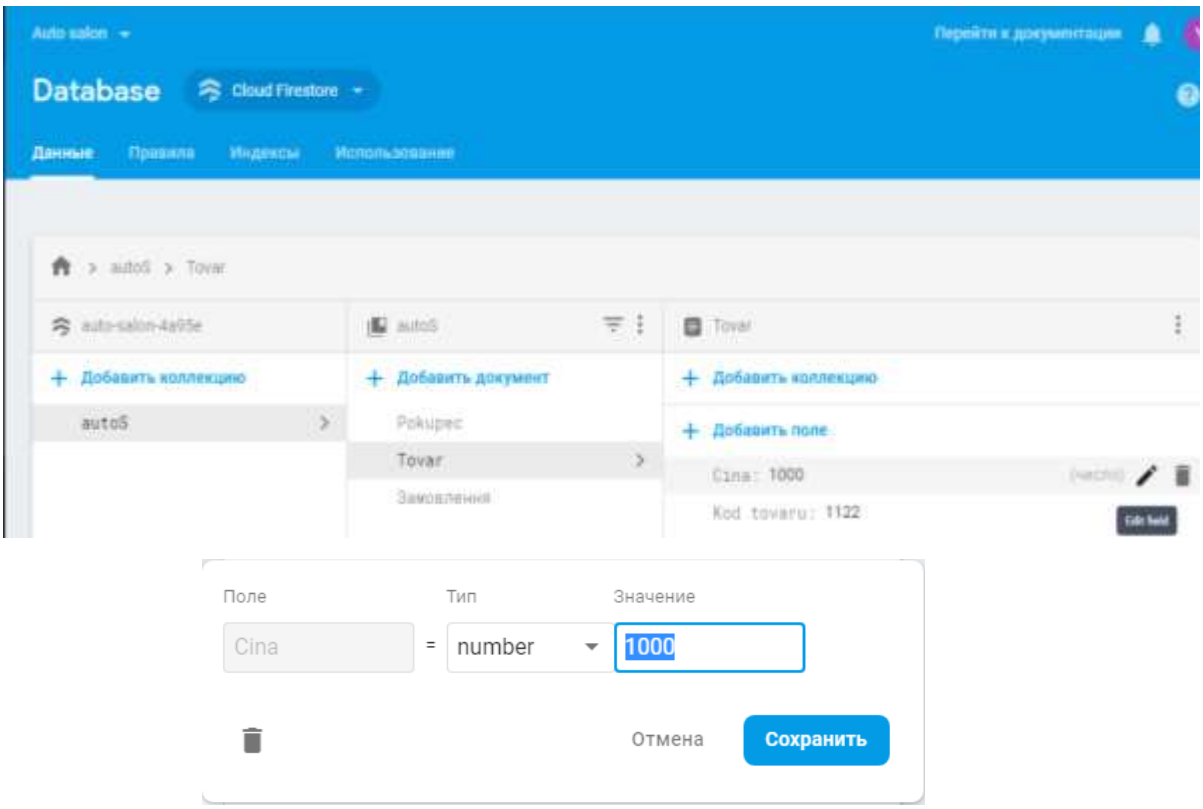


Рис. 3.9. Оновлення даних в документі

Завдання для самостійного виконання

1. Розглянути наведені вище особливості роботи з даними засобами Cloud Firestore.
2. Використовуючи наведені вище вказівки, авторизуватися на платформі Firebase та створити новий проєкт.
3. Створити у тестовому режимі нову БД Cloud Firestore згідно з вашим варіантом, що складається з 3–4 колекцій. Використати різні типи даних (Number, Map, Timestamp, Reference, String, Array, Boolean та інші).
4. Змінити тестовий режим на захищений.
5. Розглянути основні команди для роботи з даними (створення колекцій та документів, групування і вибірка даних, оновлення та видалення тощо).
6. Додати до деякого документа підколекцію та наповнити її даними.

Контрольні запитання

1. До якого типу NoSQL баз даних можна віднести бази даних, розроблені засобами Cloud Firestore на платформі Firebase?
2. Які особливості має тестовий режим роботи з даними? Як змінити тестовий режим на захищений?
3. Яку структуру мають бази даних, розроблені засобами Cloud Firestore?
4. Які обмеження накладаються на ім'я колекції?
5. Яку команду використовують для того, щоб переглянути дані, які відповідають деякому критерію?
6. Чи потрібно явно вказувати тип даних при додаванні даних у БД в Cloud Firestore?
7. Як видалити колекцію з БД в Cloud Firestore?
8. Які типи даних підтримуються у Cloud Firestore?
9. Які особливості мають типи даних Map та Array?
10. Чи можна експортувати БД з в Cloud Firestore?

РОБОТА З ГРАФОВИМИ БАЗАМИ ДАНИХ В NEO4J

Neo4j – одна з найбільш популярних графових баз даних. На відміну реляційних БД, дані тут зберігаються не у вигляді таблиці, а у вигляді графа, що дозволяє гнучко оперувати зв'язками між нодами (вершинами графа). Neo4j використовує модель орієнтованого графа та надає для додатків серверну частину з підтримкою транзакцій, яка відповідає вимогам ACID.

Робота з даними в Neo4j

Для роботи з даними потрібно спочатку скачати відповідні файли з офіційного сайту https://neo4j.com/ (для закачування даних обираємо в розділі Download Neo4j пункт Download Community Server або Download Neo4j Desktop для роботи з сервером в графічній оболонці, а не через браузер)(рис. 4.1)

Рис. 4.1. Завантажування даних з офіційного сайту

Після встановлення та запуску сервера можна працювати з даним в браузері – потрібно прописати в адресному рядку адресу сервера, вказану в рядку підключення (рис. 4.2) та під'єднатись до потрібної бази даних.

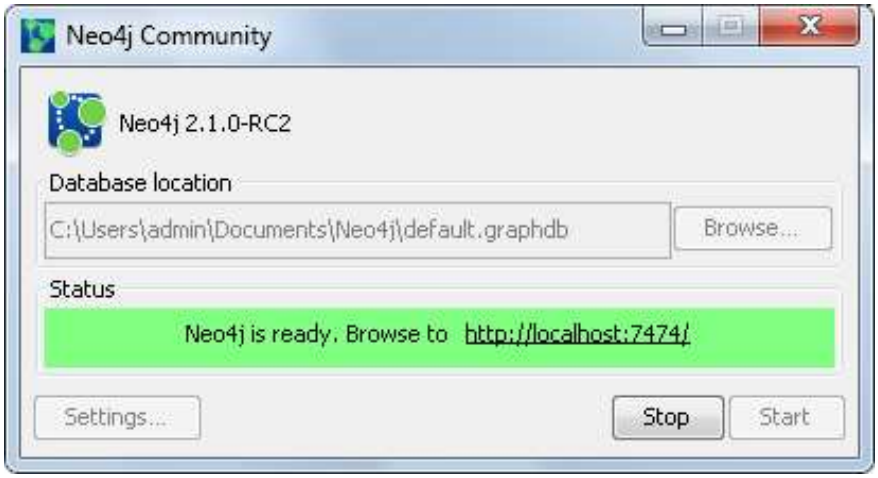


Рис. 4.2. Сервер Neo4j запущено, адреса сервера localhost:7474

На рис. 4.3. Наведено вікно роботи з даними у Neo4j Browser після встановлення з'єднання з сервером.

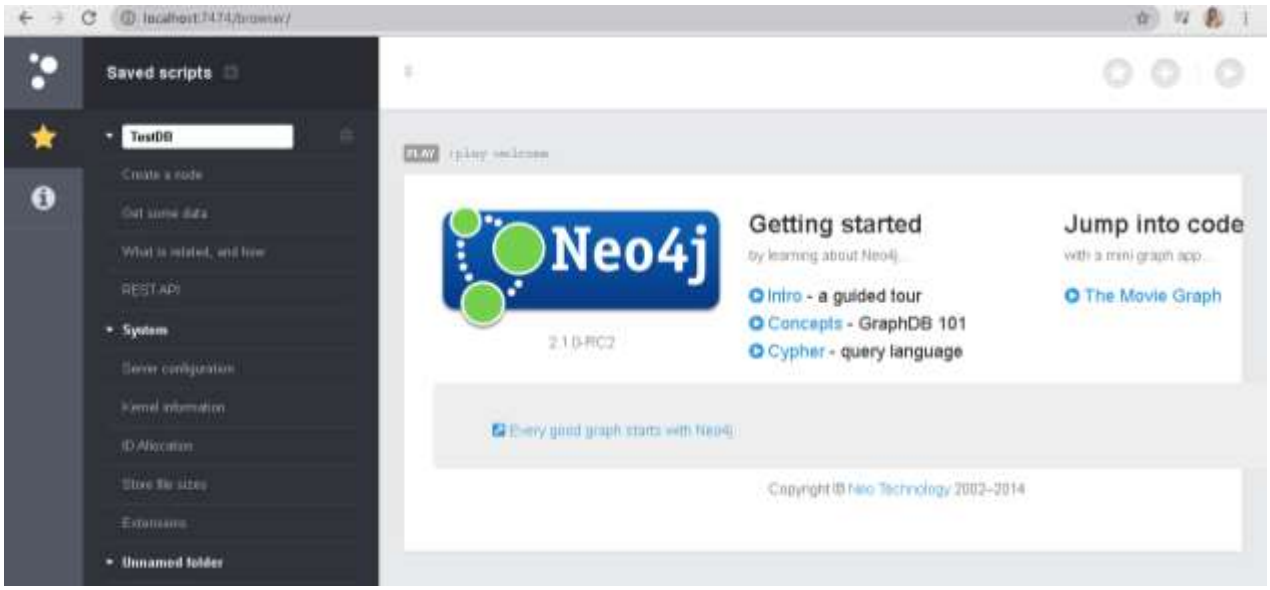


Рис. 4.3. Робота з даними у Neo4j Browser

Оболонка Neo4j Desktop потребує встановлення, натомість має більший функціонал та дозволяє працювати з даними за умови відсутності з'єднання з мережею Інтернет (рис. 4.4). Для створення нової

бази даних потрібно скористатись командою Add DataBase пункту My Project. Користувач має змогу створити нову локальну базу даних (Create a local Graph) або підключитись до чинної БД (Connect to Remote Graph). При створенні БД потрібно вказати назву БД, пароль для доступу та обрати версію сервера.

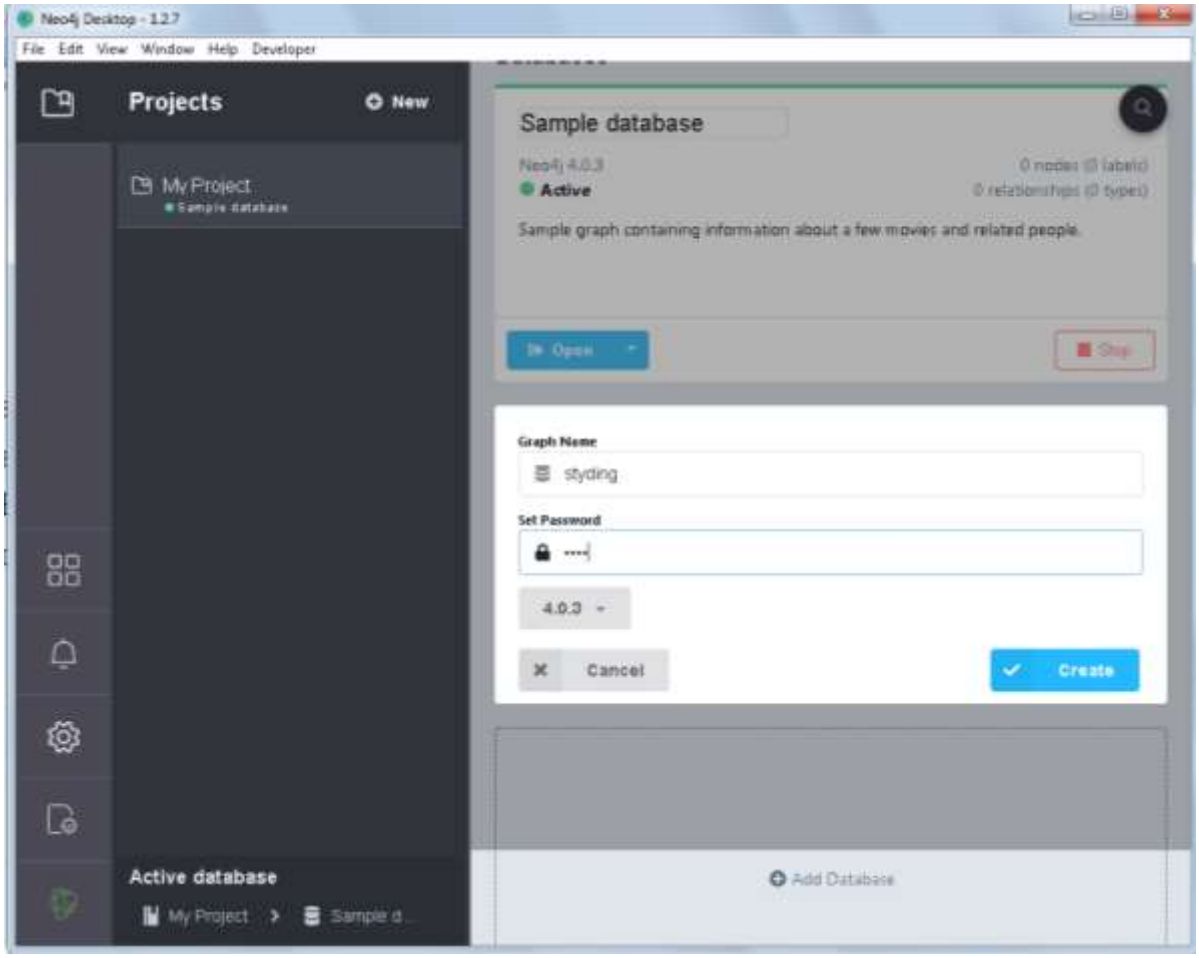


Рис. 4.4. Створення БД у Neo4j Desktop

За потреби, користувач також може працювати з даними безпосередньо з браузера, як описано вище (рис. 4.5). У вікні з'єднання з сервером потрібно ввести ім'я користувача та пароль до бази даних, до якої потрібно підключитись.

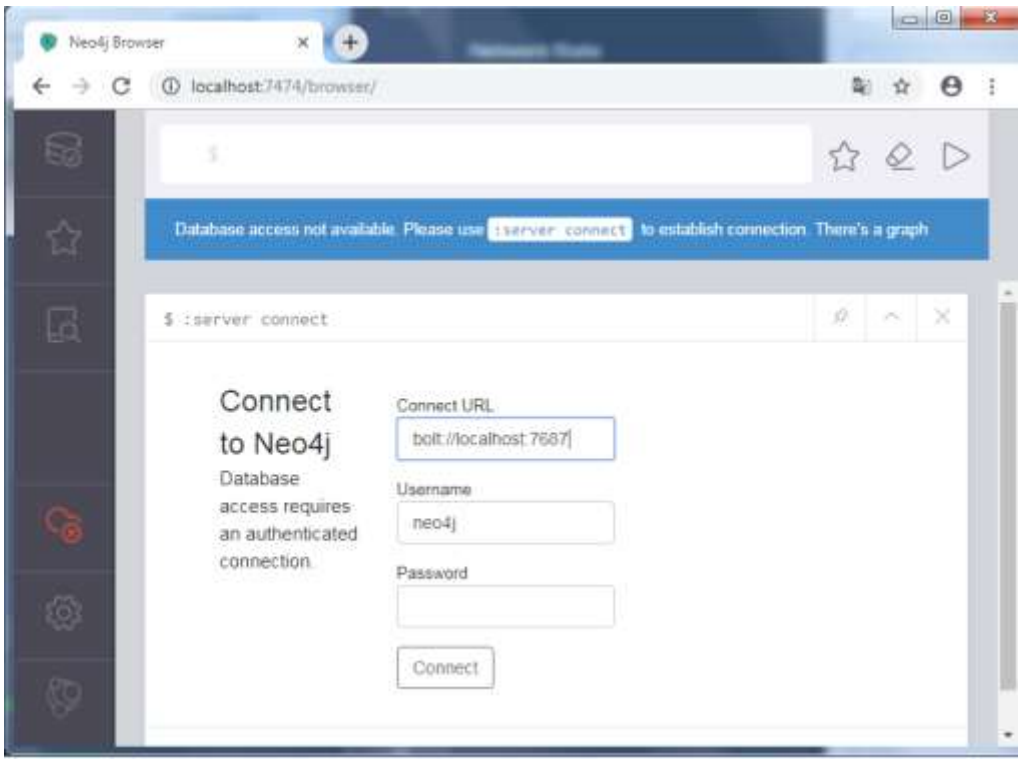


Рис. 4.5. Підключення до бази даних у браузері

Створення нод і робота з ними

У Neo4j для роботи з даними використовується мова запитів Cypher, що базується на потужності SQL, проте оптимізована спеціально для графів. Синтаксис мови є простим та наочним, що дозволяє користувачам легко записувати всі звичайні операції CRUD простим і доступним способом. Основні команди та правила побудови запитів можна переглянути у пункті “Help”.

Основними елементами БД є вузли (ноди) та зв'язки між ними. Щоб створити нову ноду, потрібно прописати команду:

```
CREATE ([назва_змінної]:[назва_мітки] {Поле1: значення_поля1, Поле2: значення_поля2, … })
```

Змінна створюється лише на момент запиту, і потрібна для звертання до ноди під час запиту, тому її можна не вказувати у звичайному запиті на створення вершини. Назву мітки потрібно вказати обов’язково, щоб мати змогу в подальшому групувати ноди за певною

ознакою. Наприклад, створимо ноду, що містить інформацію про навчальну дисципліну:

```
CREATE (:Subjects {name:"Data Base", hours:[130,24,36],
term:3, exam:true})
```

Команду CREATE можна також використовувати для створення кількох нод одразу. Для цього кожну створювану ноду потрібно брати в круглі дужки і розділяти комами:

```
CREATE (:Students {name:"Ivan", surname:"Vovk", age:
19, date_finish_st:"2020-6-21", subject: ["Web", "Data
Base","OS"], stipend: true}),
(:Students {name:"Petro", surname:"Franko", age: 21,
subject: ["Web","Data Base"],stipend: false}),
(:Students {name: "Maria", surname: "Petrenko",
subject: ["Web","Data Base", "Desigh"], stipend:
false})
```

До основних типів даних, що підтримуються в Neo4j, можна віднести цілі та дробові числа; рядки; булеві значення (true, false), масиви та дані типу "дата/час". Тип даних, як і в MongoDb, явно не вказується користувачем.

Neo4j, як і реляційні СКБД, дає змогу встановлювати унікальні значення для певних полів. Встановити унікальність значень поля name можна таким чином:

```
CREATE constraint on (s:Subjects)
ASSERT s.name is unique
```

При спробі додати ще одну ноду з таким самим значенням поля, яке визначене як унікальне, буде виведене повідомлення про наявне обмеження (рис. 4.6).

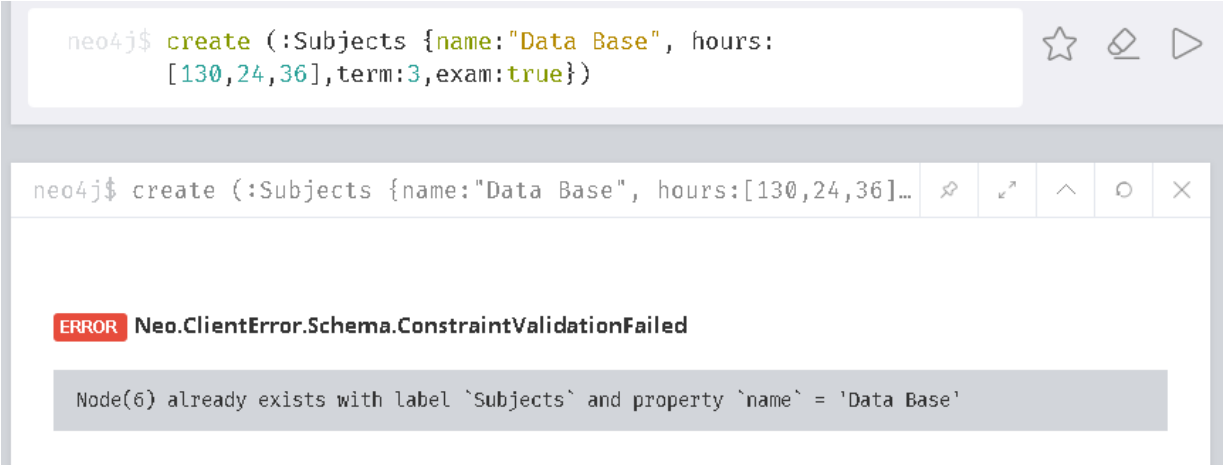


Рис. 4.6. Вивід повідомлення про обмеження унікальності значень

Щоб отримати ноди згідно з певним критерієм, використовують оператор MATCH. Для виведення даних можна виконати такий запит:

```
MATCH ([Змінна]:[Тип])[WHERE[умова]] RETURN[Змінна].
```

Команда `MATCH (назва_змінної) RETURN назва_змінної` поверне всі ноди (рис. 4.7). Також можна вказати максимальну кількість нод для відображення, вказавши її після команди LIMIT. Так, запит

```
MATCH (n)
RETURN n LIMIT 5;
```

поверне перші пять нод БД. При роботі з даними у Neo4j Desktop користувач може обрати один з чотирьох способів відображення результатів вибірки: граф (рис. 4.7), таблиця, текст та код. При роботі з базою даних у браузері пропонується лише відображення даних у вигляді таблиці чи графа.

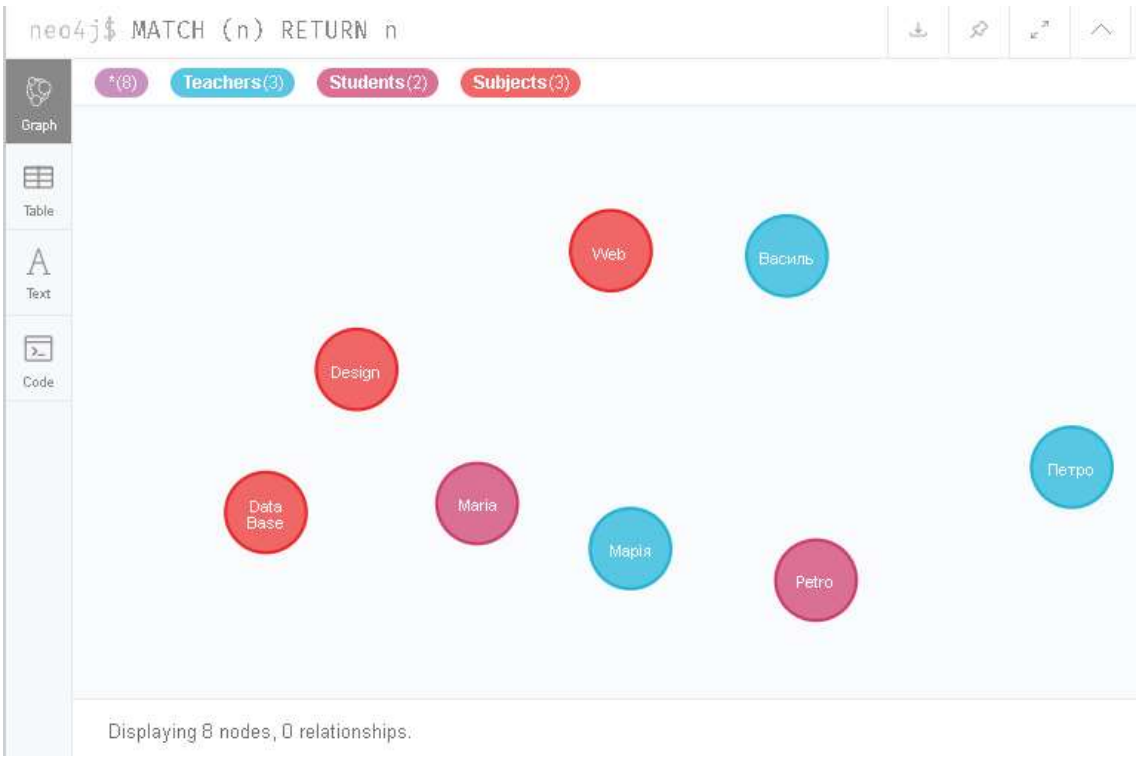


Рис. 4.7. Відображення всіх нод бази даних

Запит

```
MATCH (i:Students)
RETURN i
```

поверне всі ноди класу Students (з назвою Students) (рис. 4.8). Для того, що переглянути дані кожної ноди під час роботи з даними за допомого браузера, потрібно обрати її – у спливаючому вікні буде виведена інформація про дані, що містяться в обраному вузлі. У другій вкладці вікна можна обрати колір для відображення нод вказаного класу та поле, значення якого буде виведене на ноді при її графічному відображенні.

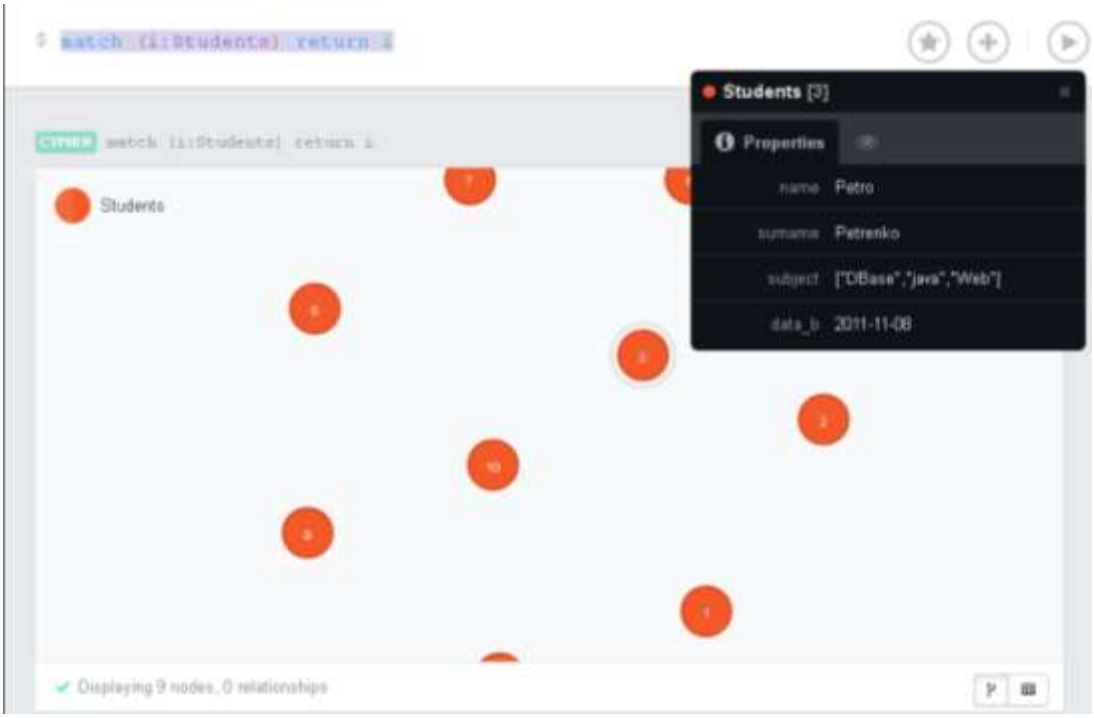


Рис. 4.8. Виведення всі нод класу Students

Запит

```
MATCH (m: Students) Where m.name = 'Petro'
RETURN m;
```

поверне лише одну ноду класу Students з атрибутом name = 'Petro'. Зауважимо, що цей запит рівнозначний запиту

```
MATCH (n:Students {name:"Petro"})
RETURN n.
```

При визначенні умови у секції WHERE, можна використовувати оператори =, <, >, >=, <=, та логічні оператори AND, OR, XOR, NOT.

Запит

```
MATCH (n:Students)where n.age>=18
RETURN n
```

поверне всі ноди класу Students з атрибутом age>=18.

Запит

```
MATCH (n:Students)
WHERE n.stipend=true and n.age>18
RETURN n
```

поверне всі ноди класу Students з атрибутом age>18 та stipend=true.

Функції для перевірки входження елемента до множини (вибірки):

ALL – true, якщо умова виконується для всіх елементів;

ANY – true, якщо умова виконується хоча б для одного;

EXISTS – true, якщо відповідність шаблону існує в графі, або якщо вказана властивість існує у вузлі, відношенні або карті;

NONE – true, якщо умова не виконується для всіх;

SINGLE – true, якщо умова виконується тільки для одного;

На рис. 4.9. наведено запит з перевіркою на входження елементу до вибірки та результат його виконання у вигляді таблиці.

```
MATCH (n:Teachers)
WHERE any (x in n.subject_s WHERE x= "Data Base")
RETURN n
```

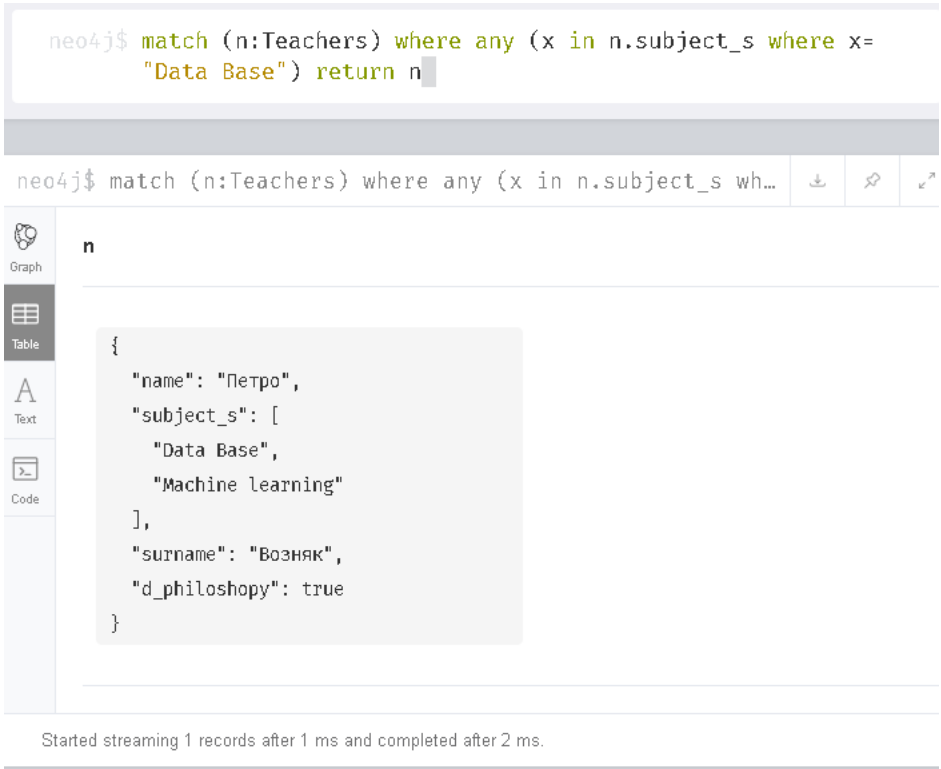


Рис. 4.9. Використання команди ANY

Тут x – це змінна, яка буде використовуватись у перевірці, а n.subject_s – це масив.

Для ***видалення ноди***, потрібно її вибрати командою MATCH і видалити командою DELETE. Наприклад, видалимо ноди з інформацією про студентів з атрибутом name="Ivan":

```
MATCH (n:Students {name:"Ivan"})
DELETE n
```

Щоб видалити ноду, обовязково потрібно, щоб у неї не було зв'язків з іншими нодами.

Зв'язки між нодами

Neo4j дозволяє створювати лише напрямлені зв'язки між нодами, тобто обов'язково вказувати напрям зв'язку. Для встановлення зв'язків між нодами також використовується команда CREATE, яка має такий синтаксис:

```
CREATE (n)-[:Назва_зв'язку{Поле1: значення_поля1, Поле2:значення_поля2, … }]-> (m)
```

Тут *n* та *m* – змінні, що визначають деякі ноди. Також, можна встановлювати зв'язок у зворотному напрямку:

```
CREATE (m)<-[:Назва_зв'язку{Поле1: значення_поля1, Поле2:значення_поля2, … }]- (n)
```

Встановити зв'язок між нодами класу Students з атрибутом name="Petro" та name="Maria" можна за допомогою наступного запиту (рис. 4.10):

```
MATCH (n:Students {name:"Ivan"}) match (m:Students{name:"Maria"})
CREATE (n)-[:bestfriend {type:"piory"]->(m)
```

або так:

```
MATCH (n: Students),(m: Students)
WHERE n.name = "Petro" AND m.name ="Maria"
CREATE (n)-[: bestfriend {type:"piory"]->(m)
```

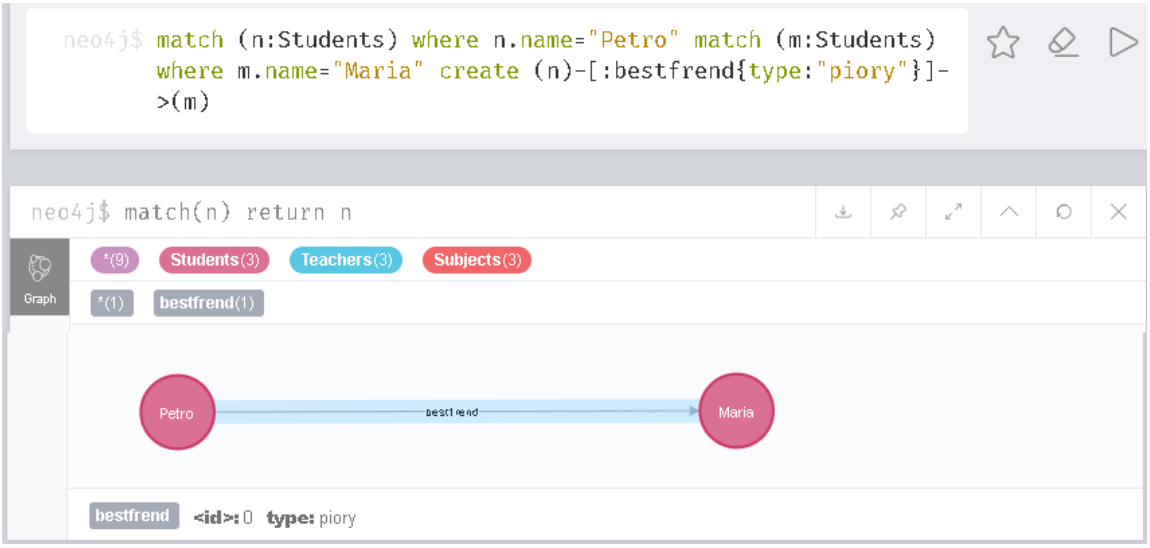
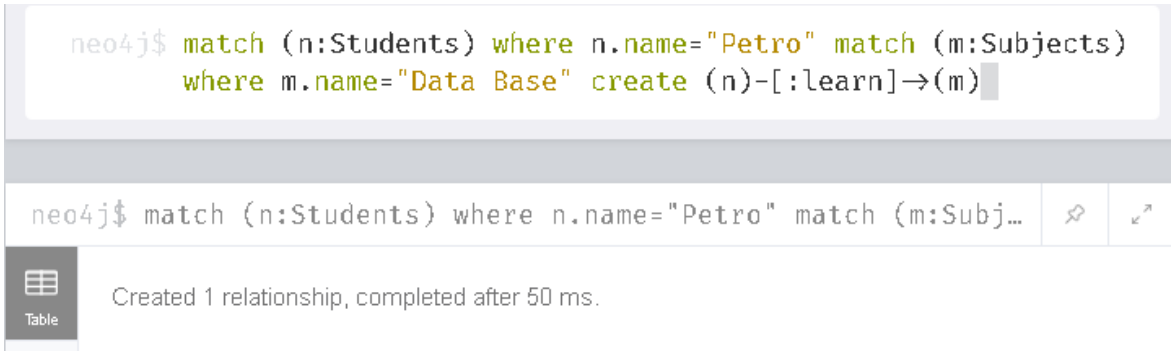


Рис. 4.10. Відображення створеного зв'язку bestfriend з параметром type="piory" між двома нодами

Наступний запит встановлює зв’язок між нодою класу Student з атрибутом name="Petro"та нодою класу Subjects з атрибутом name="Data Base":



У даному запиті встановлено лише один зв'язок. Наступний запит призводить до встановлення зв’язків між усіма нодами класу Student та нодами класу Subject:

```
MATCH (p:Student), (S:Subjects)
CREATE (p)-[:Learn]->(s)
```

Щоб переглянути всі ноди та всі зв’язки, можна скористатися командою `MATCH (назва_змінної) RETURN назва_змінної` (рис. 4.11).

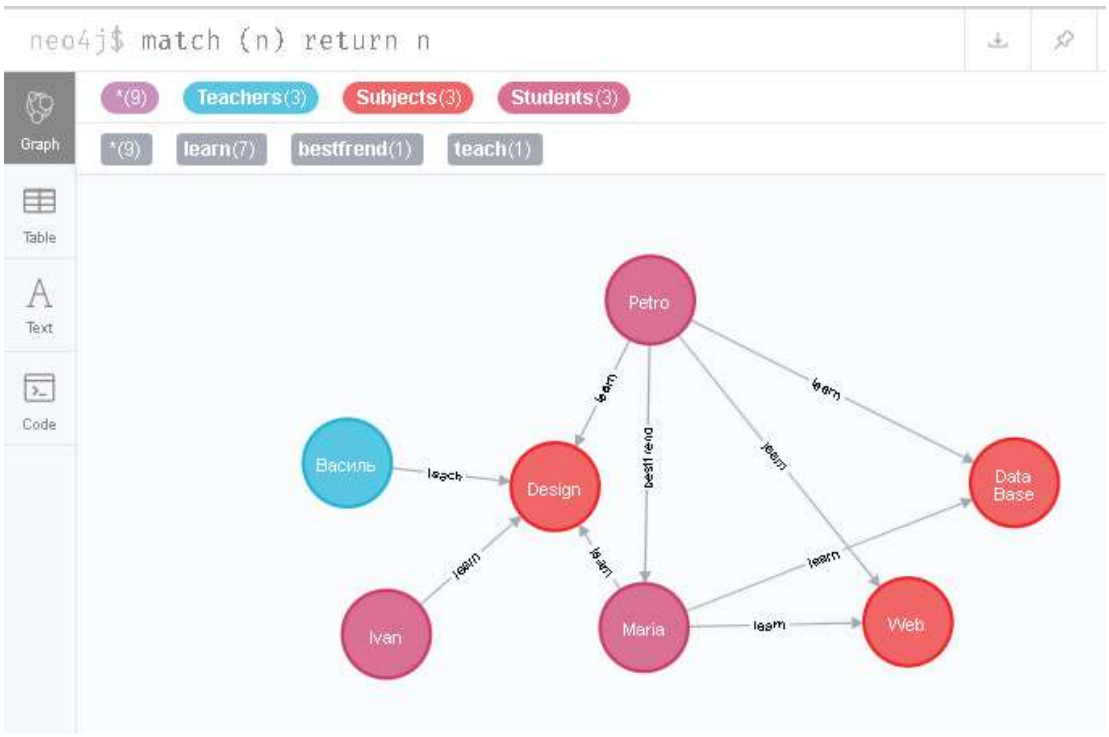


Рис. 4.11. Відображення всіх нод та зв'язків

Для перегляду певних елементів та зв’язків можна скористатись таким запитом:

```
MATCH (n)-[:назва_зв’язку]-(m)
RETURN m,n
```

Можна вибрати всі ноди, що пов'язані певним зв'язком. Наприклад, отримати перелік всіх дисциплін, що вивчає певний студент:

```
MATCH(n:Student)-[: Learn]->(m: Subjects)
WHERE n.name="Ivan"
RETURN m
```

Щоб отримати всі зв’язки для заданої ноди:

```
MATCH (n: Student {name:"Ivan" })-[r]->(m: Student )
RETURN r
```

На рис. 4.12 наведено запит для виведення даних щодо дисципліни «Дизайн»: викладач дисципліни та студенти, що її вивчають та результат його виконання.

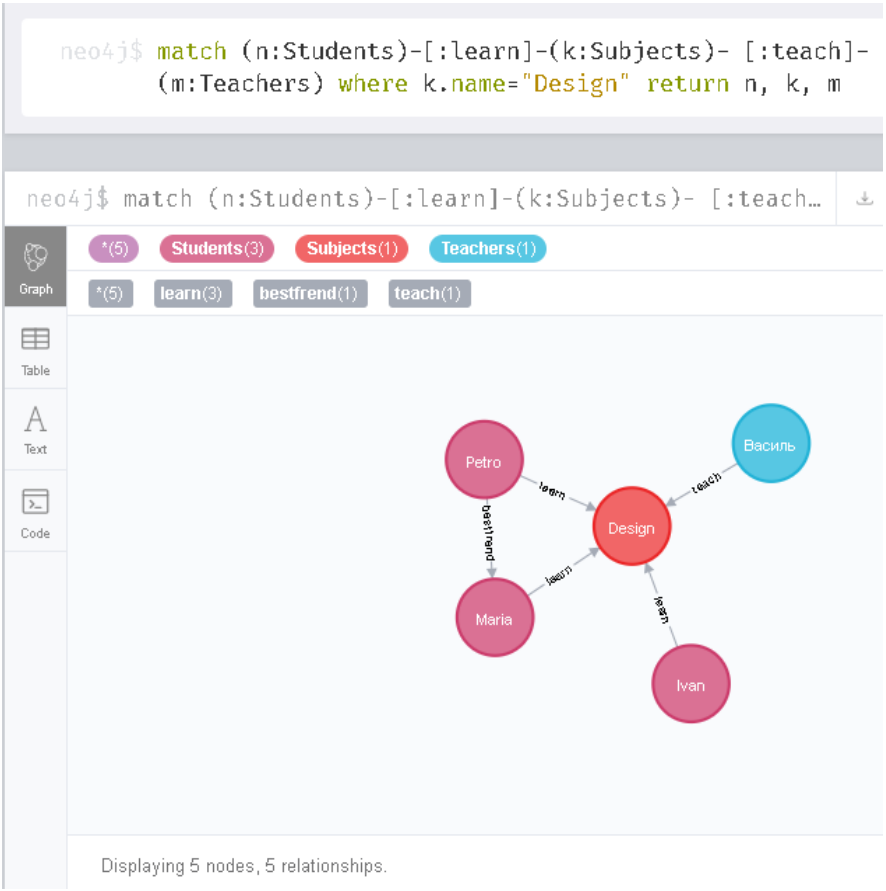


Рис. 4.12. Виведення нод згідно з певними зв’язками

Для видалення зв’язку, потрібно спочатку його вибрати командою MATCH, після чого видалити командою DELETE. Видалимо зв’язок між нодою з атрибутом name="Petro" класу Students та нодою з атрибутом name="Data Base" класу Subjects:

```
MATCH (n:Students {name:"Petro"}) -[m:learn]->(k:Subjects {name:"Data Base"})
DELETE m
```

Для зміни значень атрибутів об'єкта, потрібно його вибрати, і змінити за допомогою команди SET. Наступний запит реалізує вибірку всіх студентів, що вивчають дисципліну "Desigh", та встановлює значення false поля stipend:

```
MATCH(n:Student)-[: Learn]->(m: Subjects)
WHERE m.name="Desigh "
SET n.stipend:false
RETURN n
```

Для того, щоб здійснити вибірку даних із сортуванням за деяким критерієм, потрібно використати команду ORDERBY [поле], яку вказуємо після RETURN (рис. 4.13).

Рис. 4.13. Використання команди ORDERBY в Neo4j

Імпорт даних

Neo4j дозволяє імпорт даних у форматах CSV та JSON. Для імпорту даних з CSV-файла використовують команду load csv from “адреса файла”. Наприклад, для імпорту даних з файла testDB.csv, що заздалегідь розташований у паці Import і має наступну структуру:

```
Name,Age,Height,Adress
John,22,185,"Lviv, Shevchenka str,5"
Petro,31,178,"Kyiv, Franko str, 23"
Sam,28,175,"Kyiv, Central str, 173"
```

можна використати запит:

```
load csv with headers from "file:/// testDB.csv" as row
with row where row.Name is not null
create (:Person {Name: row.Name,
Age: to Integer(row.Age),
Height: to Integer(row.Height),
Adress: row.Adress})
```

Розглянемо цей запит детальніше. Параметр with headers вказує, що перший рядок у файлі містить заголовки стовпців. В другому рядку запиту вказано, що поле Name не може містити нульові значення і виступає в ролі ключового параметра.

У наступному рядку запиту визначаємо назву класу (в даному випадку створюється клас Person) та параметри нод, що створюються. За замовчуванням типи даних будуть визначені як string, тому для створення полів інших типів потрібно використати спеціальні вбудовані функції (наприклад, to Integer).

Завдання для самостійного виконання

1. Використовуючи наведені вище вказівки, запустити Neo4j та створити нову БД, згідно з вашою предметною областю, що складається з 3–4 класів нод. Використати різні типи даних (String, Array, Boolean, Date, Integer та інші).
2. Вивести створені ноди. Розглянути різні випадки – всі створені ноди, ноди, що належать деякому класу, дані згідно з деяким критерієм тощо.

3. Оновити дані згідно з деяким критерієм, використовуючи команду SET.
4. Встановити унікальні значення для деяких полів.
5. Встановити відповідні зв'язки між нодами. Встановити певні параметри для зв'язків. Проаналізувати структуру бази даних.
6. Сформувати декілька запитів на вибірку даних. Використати пошук даних, що належать деякому діапазону; провести вибірку даних за зв'язками між нодами; провести вибірку даних, що передбачає перевірку на належність множині; провести сортування даних.
7. Провести групування даних, використовуючи функцію count(). Розглянути використання інших агрегатних функцій: sum(), collect(), avg(), min(), max().
8. Розглянути роботу з індексами у Neo4j. Розглянути особливості імпорту даних до Neo4j.

Контрольні запитання

1. Які типи даних підтримуються у Neo4j?
2. Яка мова запитів використовується у Neo4j для роботи з даними?
3. Яке призначення команд MATCH та RETURN?
4. Для чого призначена функція count()?
5. Як у Neo4j встановити зв’язок між двома нодами? Чи обов’язково вказувати напрям зв’язку?
6. Яка команда використовується для видалення ноди? Які обмеження накладаються на цю операцію?
7. Як можна встановити унікальні значення для деяких полів?
8. Який запит виведе всі зв’язки для деякої ноди?
9. Для чого використовується команда DISTINCT?
10. Як здійснити вибірку даних із сортуванням за деяким параметром за спаданням?

ОПРАЦЮВАННЯ БАЗ ДАНИХ ORIENT DATABASE

OrientDB – це нереляційна графова система керування базами даних, яка підтримує документно-орієнтовані, об'єктні моделі та моделі "ключ-значення". Однак під час роботи з документо-орієнтованими даними взаємодія між документами реалізована прямими посиланнями між записами як у графових базах даних.

OrientDB опрацьовує кожен запис (документ) як об'єкт, реалізує зв'язок між ними через прямий вказівник на об’єкт, а не через посилання між ними. Це сприяє збільшенню швидкості опрацювання запитів в порівнянні з реляційними СКБД.

При роботі із запитами ключ-значення для організації сховища даних використовується алгоритм розподіленої хеш-таблиці. Також в OrientDB можлива підтримка інтерфейсу об'єктно-орієнтованої БД, який працює поверх документо-орієнтованого шару.

Особливості:

- підтримка транзакції ACID, які гарантують надійну обробку всіх транзакцій у базі даних, і в разі аварії всі відкладені документи відновлюються та виконуються.
- підтримка SQL-запитів із розширеннями для обробки зв'язків, керування деревами та графами пов’язаних документів.
- наявність унікального механізму індексації, який потребує значно менше пам’яті, зберігаючи при цьому швидкість Red-Black Tree за рахунок балансу операцій додавання та оновлення даних, та забезпечує високу швидкість оборобки запитів.
- підтримка протоколів HTTP, RESTful та наявність додаткових бібліотек або компонентів JSON.
- підтримка мультимастерської реплікації, включаючи географічно

розподілені кластери.

- наявність локального режиму для використання бази даних в обхід сервера.
- використання вбудованих засобів шифрування даних на диску, що запобігає доступу неуповноважених користувачів до вмісту бази даних або навіть до обходу безпеки OrientDB.
- можливість імпорту реляційних бази даних в OrientDB
- наявність функцій безпеки, що забезпечують розширювану структуру для додавання зовнішніх аутентифікаторів, перевірки паролів, розширених можливостей аудиту та підтримки системного журналу.
- OrientDB може бути розгорнутою у хмарі і підтримує таких провайдерів: Amazon Web Services, Microsoft Azure, CenturyLink Cloud, Jelastic, DigitalOcean.

Для початку роботи з базою даних потрібно завантажити відповідні файли з офіційного сайту https://orientdb.com. Після розпакування архіву потрібно запустити сервер (файл server.bat, розташований в каталозі orientdb\bin). У вікні, що відкриється (рис. 5.1), потрібно вказати пароль, що буде використаний для доступу до даних (якщо пароль не вказати, він буде згенерований автоматично). Ім’я користувача за замовчуванням “root”.

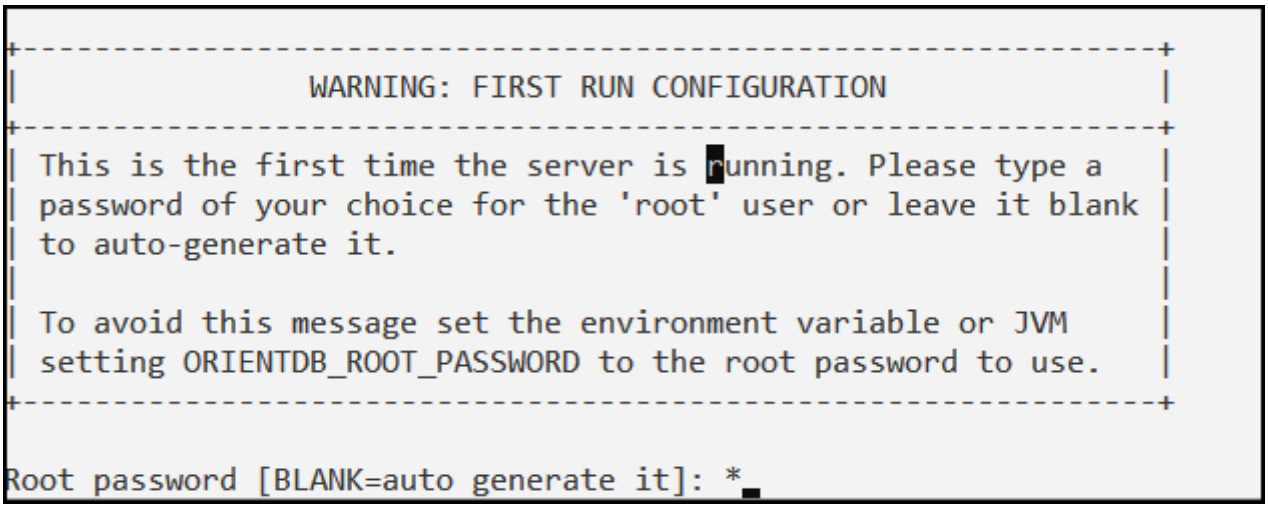


Рис. 5.1. Запуск сервера OrientDB

Після запуску OrientDB потрібно ввести URL-адресу http://localhost: 2480 у вікні браузера, після чого відкриється вікно, наведене на рис. 5.2.

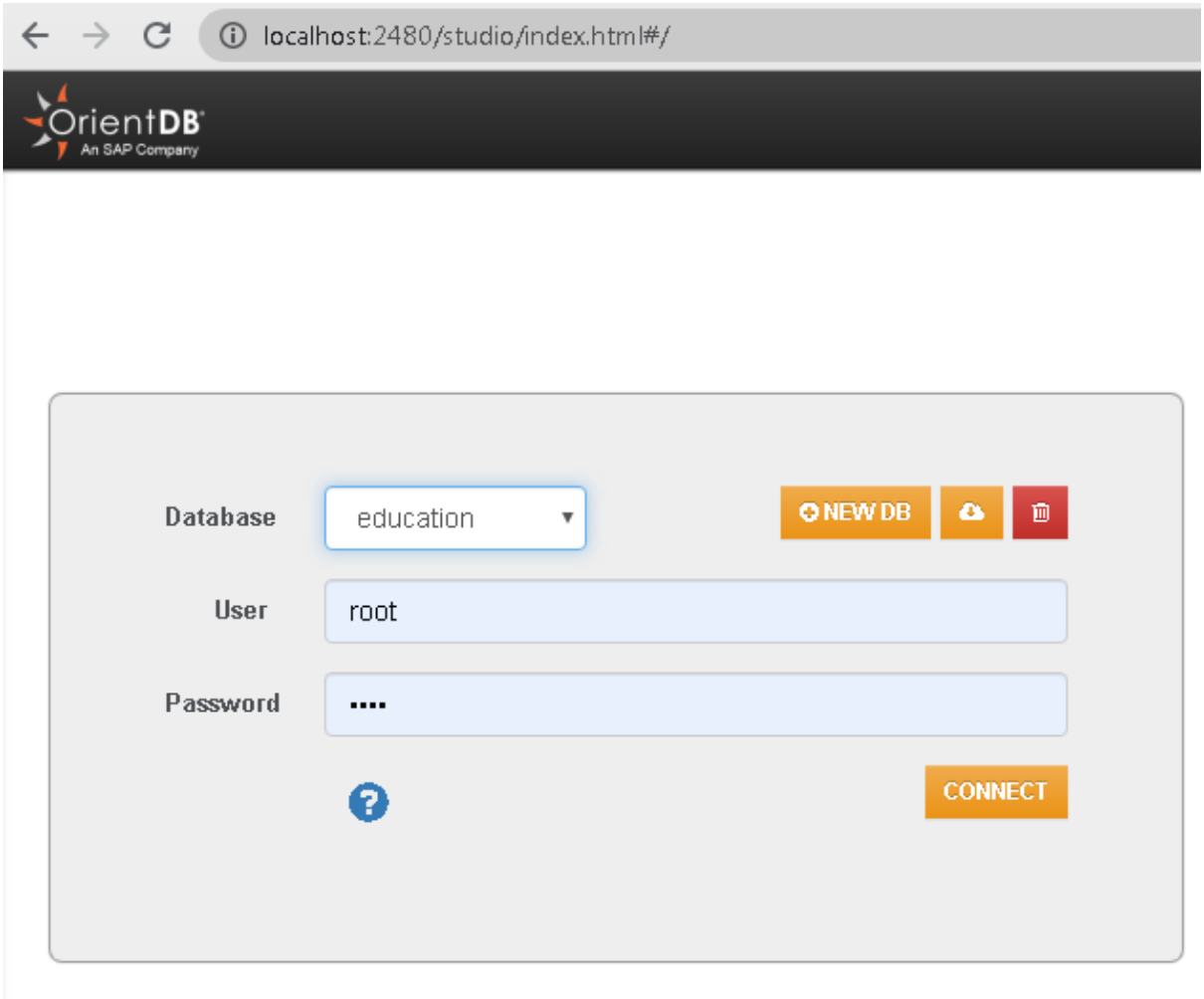


Рис. 5.2. Запуск OrientDB у браузері

Щоб підключитись до вже створеної бази даних, необхідно обрати її у списку, ввести вказані при створенні ім'я користувача та пароль і натиснути кнопку Connect. Для створення нової бази даних необхідно обрати пункт «NEW DB»; ввести у вікні, що відкрилось, назву бази даних, ім’я користувача та пароль і натиснути кнопку Create Database (рис. 5.3). За потреби можна імпортувати потрібну базу даних до OrientDB чи видалити наявну.

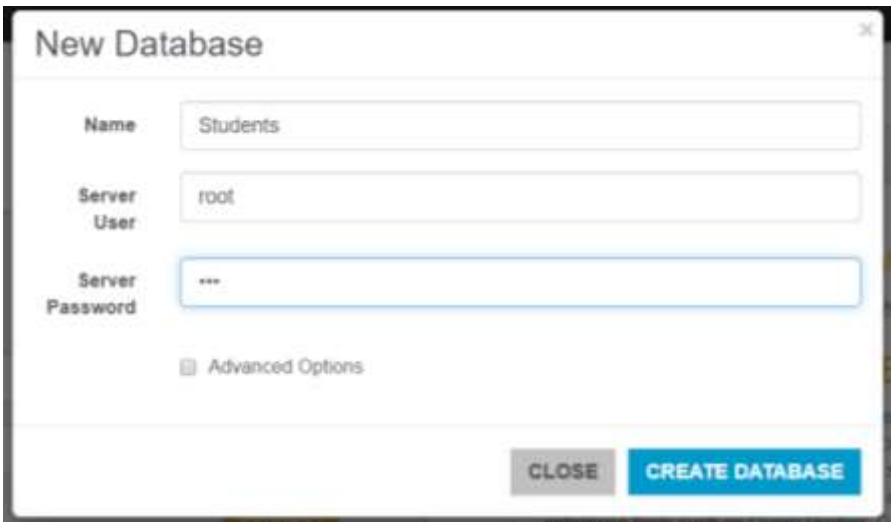


Рис. 5.3. Створення нової бази даних

Після підключення до створеної БД буде здійснений перехід у студію розробки (рис. 5.4). У пункті меню Browse користувачі OrientDB мають змогу писати і виконувати запити та переглянути історію запитів за потреби. При виборі пункту Schema відображається структура БД, у детальній формі.

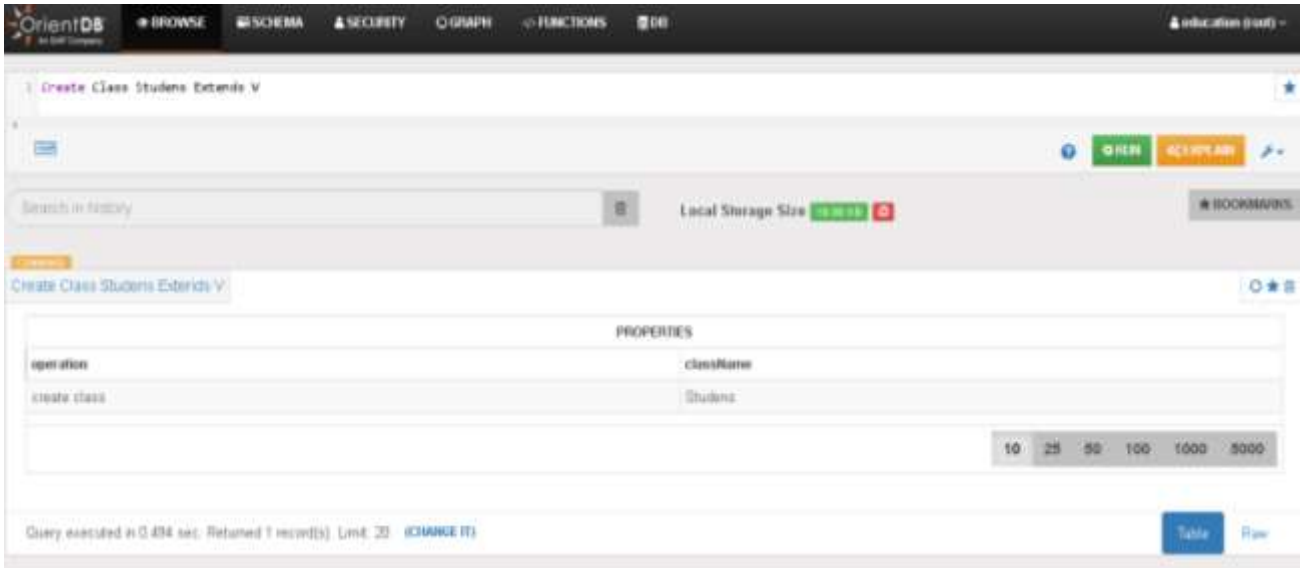


Рис. 5.4. Головне вікно OrientDB у браузері

У пункті меню Security користувачі мають змогу здійснювати адміністрування БД, тобто додавати чи видаляти користувачів, встановлювати їх ролі (рис. 5.5).

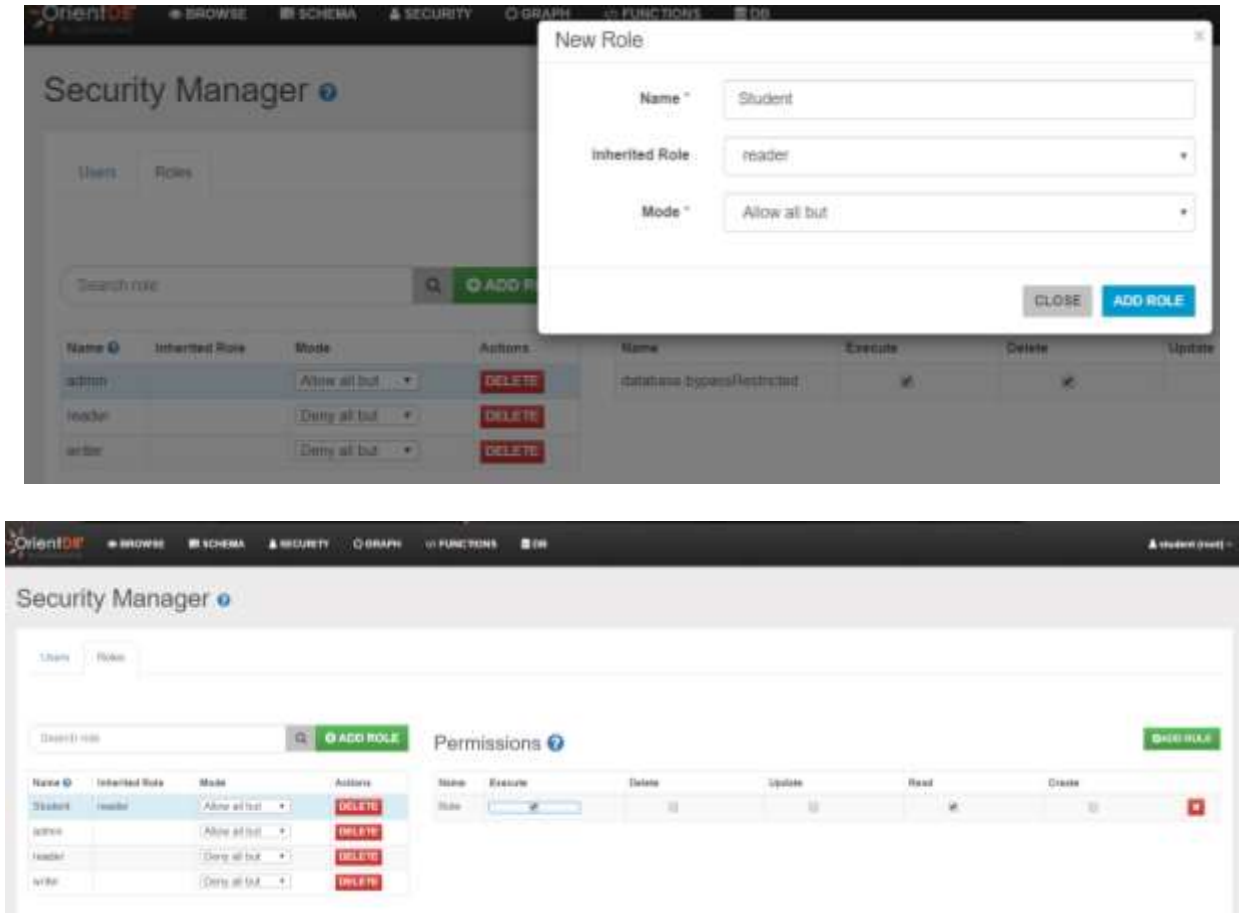


Рис. 5.5. Створення ролі Student та встановлення правил

Пункт меню Graph дає змогу працювати з даними, представленими у вигляді графа, змінювати параметри відображення вузлів та зв'язків між ними тощо. При виборі пункту Functions відкривається вікно Functions Management, в якому розробник БД має змогу створювати власні SQL- чи JavaScript-функції, та запускати їх на виконання (рис. 5.6). Пункт меню DB містить основну інформацію щодо поточної бази даних.

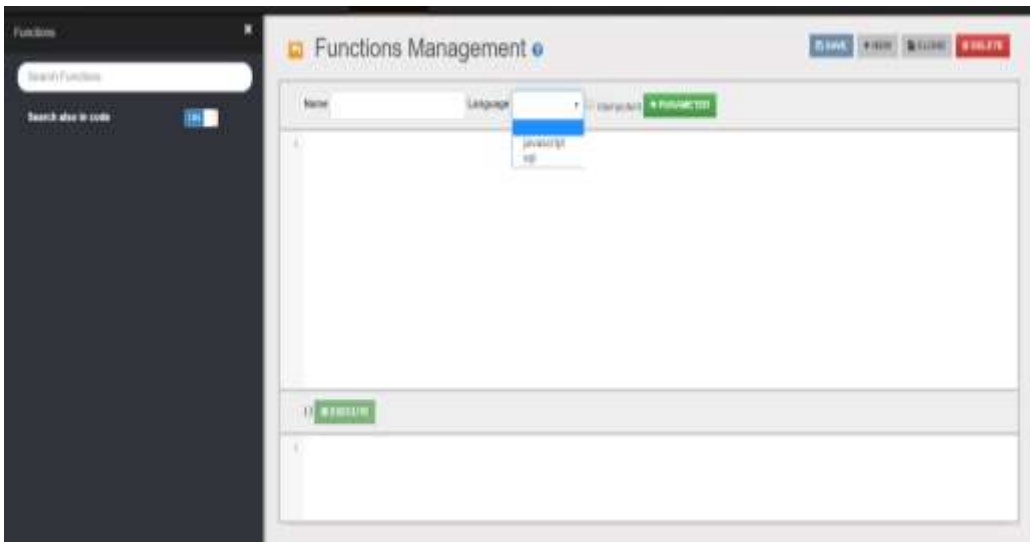


Рис. 5.6. Вікно Functions Management

Робота з графами

У графових базах даних система баз даних відображає дані в мережеві структури, що складаються з вершин і ребер. У моделі OrientDB Graph вершина визначається як об'єкт, пов'язаний з іншими вершинами, ребро як об'єкт, що зв'язує дві вершини.

Усі об’єкти БД належать до певного класу. Клас становить концепцію, взяту з парадигми об'єктно-орієнтованого програмування і дозволяє визначати певні правила для записів, які належать до нього. У документно-орієнтованій моделі бази даних він може бути порівняний з колекцією, а в реляційній моделі – з таблицею.

OrientDB має узагальнений клас вершин, що називається V (Vertix), та клас ребер E (Edge) та дозволяє розширювати стандартні класи V і E відповідно до конкретних потреб користувача. Перевагами такого підходу є можливість використовувати додаткові обмеження на рівні класу, що забезпечує об'єктно-орієнтоване успадкування серед елементів графа.

Для створення інших класів, вершин та зв’язків можна використовувати такі команди:

`CREATE CLASS` – створити новий клас;

`CREATE VERTEX` – створити нову вершину;

`CREATE EDGE` – створити ребро графа (зв'язок між вершинами);

Для оновлення та видалення зазначених об'єктів БД використовуються відповідні SQL-команди: DELETE та UPDATE.

Розглянемо роботу з об'єктами на прикладі бази даних про вивчення дисциплін на кафедрі. Створимо спочатку класи нод (рис. 5.7):

```
Create Class Students Extends V

Create Class Subject Extends V

Create Class Teacher Extends V
```

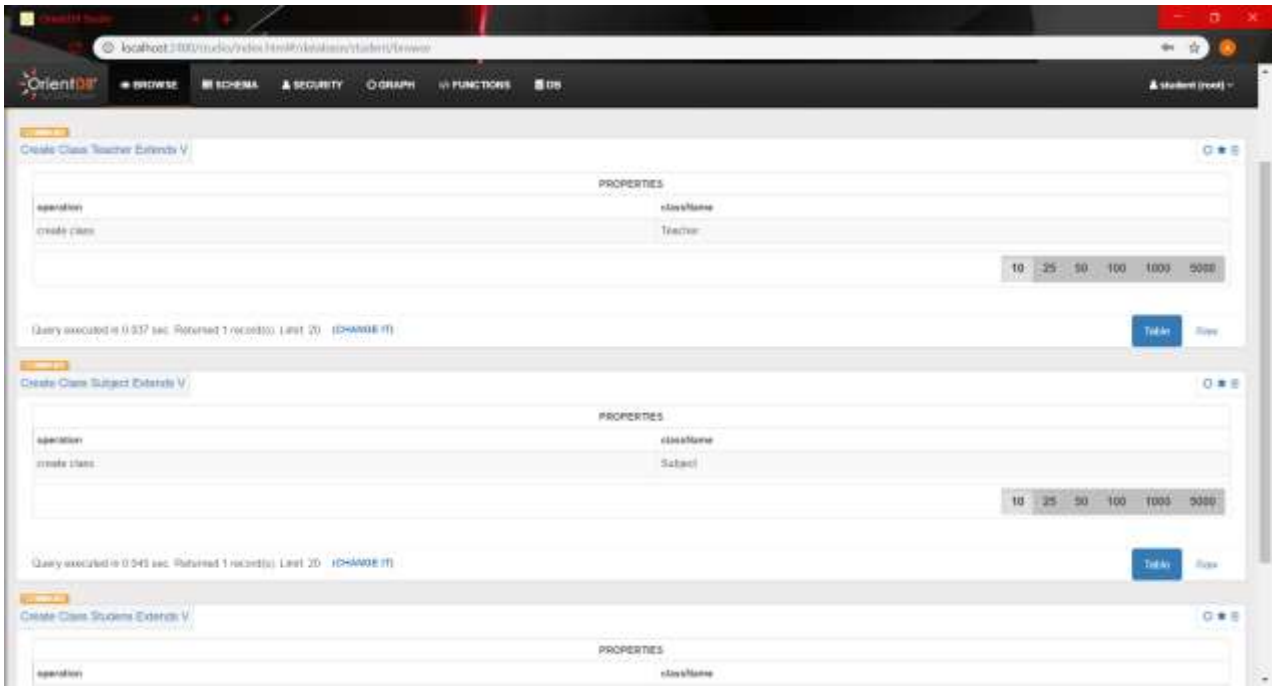


Рис. 5.7. Створення вершин графа

Таким чином, створено схему бази даних. Тепер потрібно заповнити граф даними. Основні типи даних, що підтримуються у OrientDB:

- **Boolean** – логічний тип даних, приймає значення True або False;
- **Integer** – 32-розрядні цілі числа;
- **Short** – невеликі 16-бітні цілі числа;

- **Long** – великі 64-розрядні цілі числа;
- **Double – д**есяткові числа з високою точністю;
- **Float** – десяткові числа;
- **Byte** – невеликі 8-бітні цілі числа;
- **Datetime** – будь-яка дата з точністю до мілісекунд;
- **Date** – будь-яка дата у форматі рік–місяць–день;
- **Decimal –** десяткові числа без округлення;
- **String** – будь-який рядок як алфавітно-числова послідовність символів;
- **Binary** – може містити будь-яке значення як масив байтів;
- **Link** – посилання на інший запис.
- **Link list** – посилання на декілька інших записів, причому зберігаються лише ідентифікатори записів; аналог відношення один-до-багатьох,
- **Link set** – посилання на інші записи.
- **Link map** – посилання на інші записи як значення записів, а ключі можуть бути тільки рядками. Зберігаються лише ідентифікатори записів;
- **Any** – невизначений тип даних, що використовується для оголошення колекцій змішаного типу і null.

Явно вказувати тип даних, що містяться в БД, користувач може, якщо додає дані, скориставшись пунктами New Vertex New Edge у вікні Uses Classes пункту Schema. При додаванні даних запитами явно вказувати тип даних не потрібно.

Додамо запитами по декілька вузлів до кожного створеного класу користувача.

Клас Students:

```
Create Vertex Students set Name = 'Christina', Age = 19,
Lengh = ['C++','Java']
```

```
Create Vertex Students set Name = 'Dima', Age = 20 ,
Likes = 'DataBase', Stipend = true
Create Vertex Students set Name = 'Marko', Age = 19
```

Клас Subject:

```
Create Vertex Subject Set Name = 'OOP'
Create Vertex Subject Set Name = 'DataBase', Semestr =
4, Hours = 100
Create Vertex Subject Set Name = 'web', Semestr = 4,
Status = True
```

Клас Teacher:

```
Create Vertex Teacher Set Name ='Ivanenko',Status
='Docent',ID_T = 1
Create Vertex Teacher Set Name ='Petrenko',Status
='Teacher', ID_T = 2
Create Vertex Teacher Set Name
='Shakleina',ATitle='PH_Doctor', ID_T=5
```

Для встановлення параметрів об’єкта використовується оператор SET. Елементи масиву вказуються у квадратних дужках.

На рис. 5.8 наведено відображення створених об’єктів БД у пункті Schema.

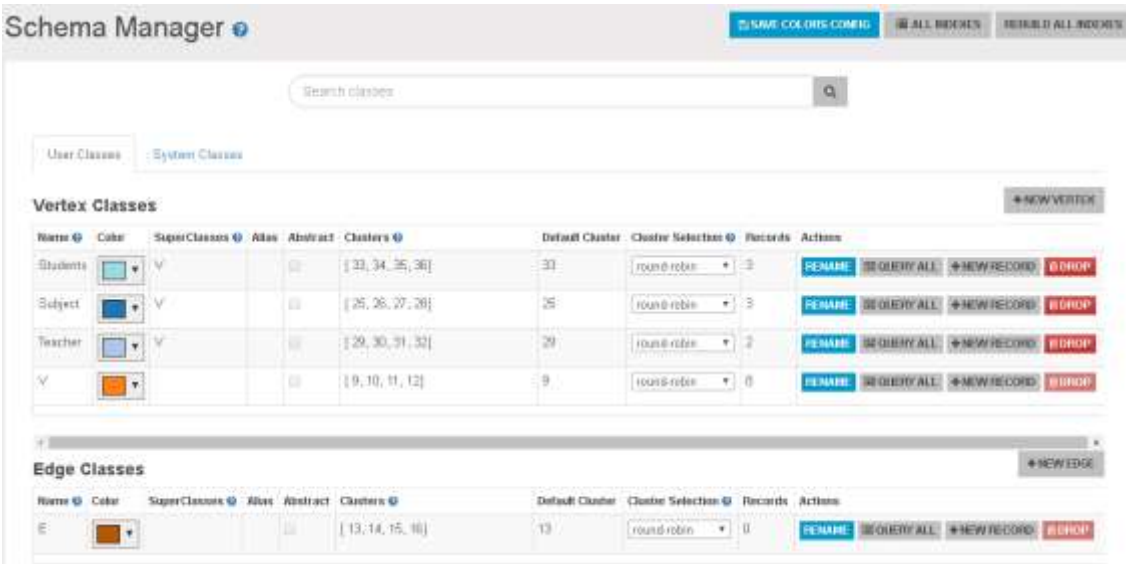


Рис. 5.8. Перегляд об’єктів БД у пункті Schema

За замовчуванням при перегляді структури БД у вигляді графа біля кожної вершини зазначено унікальний ID даної вершини, що створюється автоматично. Однак можна вибирати параметр для відображення у полі Display пункту Setings (рис. 5.9).

Рис. 5.9. Відображення всіх створених вершин графа

Для виведення всіх вершин БД, можна скористатись відповідним SQL запитом `select * from V` або `select * from <назва_класу>`, щоб вивести всі вершини деякого класу.

Для видалення вершини (чи класу вершин) використовують команду Delete Vertex:

```
Delete Vertex From Studens  where Name = 'Orest'
```

або за унікальним ідентифікатором вершини:

```
Delete vertex from Teacher where @rid="#29:0"
```

Видалити можна лише той вузол, що не має зв'язків з іншими вузлами.

Встановлення та опрацювання зв’язків

На даний момент створені вершини, незалежні одна від одної, пов'язані між собою лише класами, до яких вони належать. Тобто вони ще не з'єднані ребрами. Перш ніж встановити зв'язок між вершинами, потрібно створити відповідні класи ребер графа (Edge) (буде працювати і без класів ребер, але тоді не зможе бути використана графова модель безпосередньо).

```
CREATE CLASS Study EXTENDS E
CREATE CLASS Teach EXTENDS E
```

Тепер можна встановити зв'язок між потрібними вершинами (між вузлами графа). Створимо зв'язок, який відображає, що всі студенти другого курсу вивчають дисципліну “ООР”. Для цього скористаємось таким SQI-запитом.

```
Create EDGE Study FROM (Select from Students) to (select
from Subject where Name='OOP')
```

Встановимо зв’язок між дисципліною та викладачем кафедри, що її викладає (рис. 5.10):

```
Create EDGE Teach FROM (select from Teacher where
Name='Shakleina') to (select from Subject where
Name='DataBase')
```

При створенні зв’язків між вершинами потрібно звертати увагу на орієнтацію ребер графа, оскільки вона надає значення відношенню. Наприклад, створення зв’язку в протилежному напрямку (від дисципліни до викладача), потребує створення окремого класу; наприклад, «є в індивідуальному плані» або «читається».

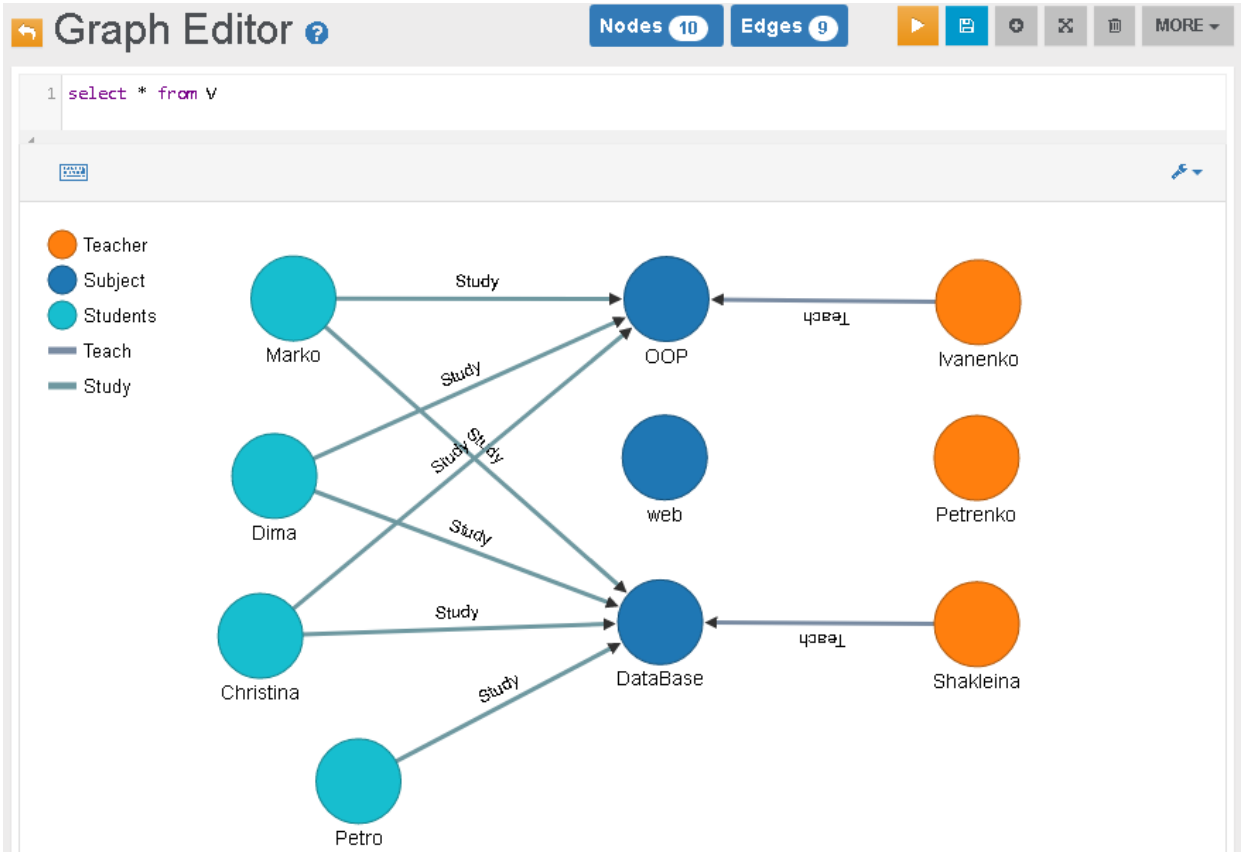


Рис. 5.10. Відображення створених зв’язків між класами Students,Subject та Teacher.

Зв'язок можна встановлювати використовуючи унікальний ідентифікатор вершин, що створюється OrientDB автоматично при додаванні об’єктів. Відмітимо, що деякі 2 студенти є друзями. Створимо для цього клас friend та додамо відповідні об’єкти (рис. 5.11):

```
CREATE CLASS friend EXTENDS E
create EDGE friend from #23:0 to #21:0
create EDGE friend from #21:0 to #23:0
```

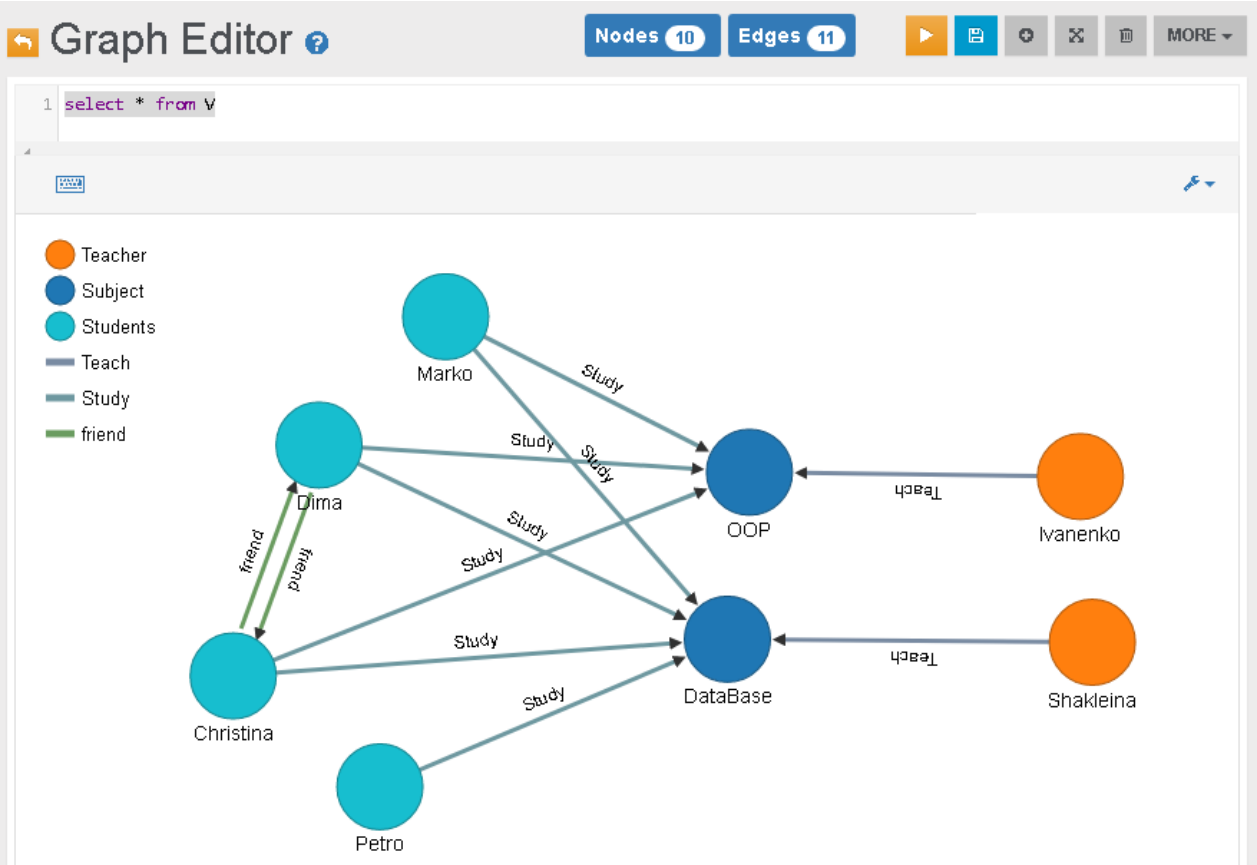


Рис. 5.11. Відображення створених зв’язків класу friend

Використання функцій для роботи з даними

Для роботи з даними можна використовувати відповідні запити чи функції (як стандартні, так і функції користувача). Так, наприклад, запит

```
select from Teacher where Name='Shakleina'
```

поверне вузол графа, в якого атрибут Name має значення 'Shakleina', а запит

```
select from V
```

виведе всі вузли БД.

Щоб працювати з даними, можна використовувати також спеціальні функції, які розширюють можливості стандартних SQL-запитів. Розглянемо деякі з них:

•OUT () – дає змогу отримати сусідні вихідні вершини. Наприклад, запит `select out(Study) from Students` повертає індекси всіх

вершин класу Studens, що пов’язані з іншими відношенням Study (рис. 5.12).

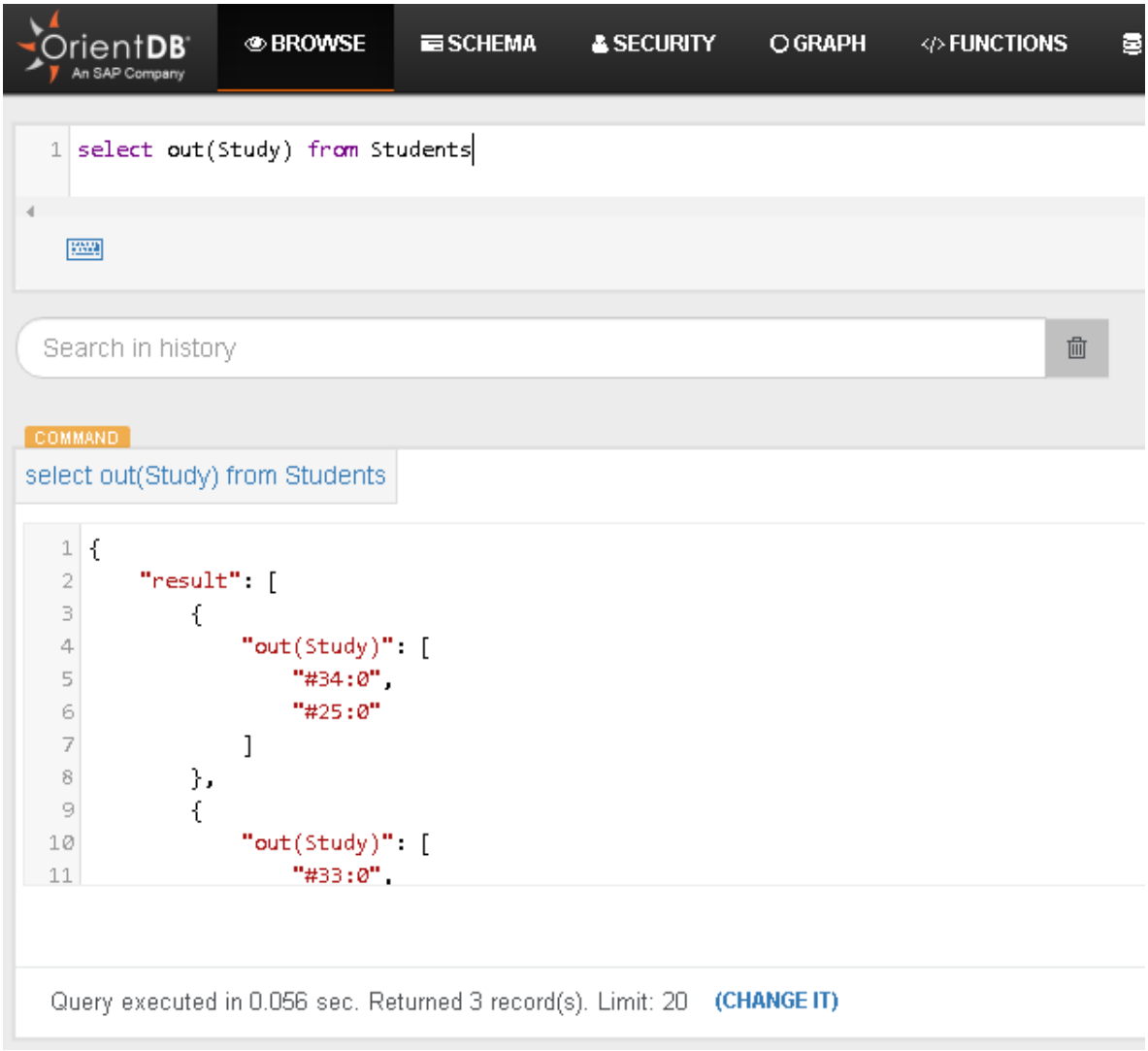


Рис. 5.12. Відображення вихідних вершин вузла графу

•IN () – дає змогу отримати суміжні вхідні вершини. Наприклад, запит `SELECT in('friend') FROM Students` повертає індекси всіх вхідних вершин класу Students, що пов’язані з іншими відношенням friend

•BOTH () – дає змогу отримати сусідні вхідні і вихідні вершини. Наприклад, запит `SELECT both() FROM #13:33` повертає індекси всіх вершин, що пов’язані з вершиною #13:33. Зручно використовувати, коли напрямок зв'язку між вершинами не важливий. Наприклад, потрібно отримати всі вершини, що пов'язані з вершиною, в якої параметр name= Christina, відношенням 'friend':

```
SELECT EXPAND(BOTH('friend')) FROM Students WHERE name = 'Christina'
```

Функція EXPAND () використовується для перетворення колекції вершин у набір результатів, розширюючи їх.

Для створення посилань на інших об'єкти можна використати запит:

```
CREATE LINK <link-name> FROM <source-class>.<source-property> TO <target-class>.<target-property>
```

Наприклад:

```
CREATE LINK comments FROM Students.Name TO Subject.Status INVERSE
```

Завдання для самостійного виконання

1. Розглянути наведені вище особливості роботи з даними засобами OrientDB.
2. Використовуючи наведені вище вказівки, створити нову БД згідно з Вашим варіантом, що складається з 3-4 класів.
3. Додати до кожного класу об’єкти. Використати різні типи даних (Number, Datetime, Date, Boolean, String, Array, Boolean та інші). Встановити звязки між створеними вершинами.
4. Створити посилання на інший запис (Link) та на декілька інших записів (Link list або Link set)
5. Розглянути основні команди для роботи з даними (створення вузлів та звязків, групування та вибірка даних, оновлення та видалення тощо). Сформувати та виконати 10 запитів на вибірку даних, використовуючи основні команди OrientDB. Проаналізувати запити та результати їхнього виконання.
6. Реалізувати вибірку даних, використовуючи змінну $depth, що показує глибину зв'язку.

7. Організувати роботу з даними, використовуючи стандартні функції OrientDB (BOTH(), IN() та інші).

8. Реалізувати експорт даних у формат JSON. Занотувати в звіт стуктуру БД та відповідний JSON-файл.

9. Імпортувати до OrientDB створену заздалегідь реляційну БД.

Контрольні запитання

1. Яку структуру мають бази даних, розроблені засобами OrientDB?
2. Яка мова запитів використовується для роботи з даними у OrientDB?
3. Яким чином додавати чи видаляти користувачів та встановлювати їх ролі у OrientDB?
4. Як створити новий клас вершин? Клас ребер графа?
5. Який оператор використовується для встановлення параметрів об’єкта?
6. Чи потрібно явно вказувати тип даних, що додаються до БД?
7. За домогою яких команд можна реалізувати видалення вершин графа з БД? Які обмеження накладаються на цю операцію?
8. Який запит виведе всі зв’язки для деякої вершини?
9. Чи потрібно звертати увагу на орієнтацію ребер при створенні зв’язків між вершинами? Чому?
10. Для чого використовується змінна $depth?

ВАРІАНТИ ЗАВДАНЬ ДЛЯ РОЗРОБКИ ВЛАСНОЇ БАЗИ ДАНИХ

	Предметна область	Орієнтовний вміст БД
1.	Облік студентів на факультеті ЗВО	Особисті дані, рік народження, адреса, дата зарахування, номер наказу, факультет, група, куратор групи,форма навчання, успішність, наявність стипендії
2.	Облік пацієнтів у поліклініці	Номер, прізвище, ім'я, по батькові, дата народження пацієнта; прізвище, ім'я, по батькові лікаря, посада та спеціалізація лікаря; діагноз, поставлений даним лікарем даному пацієнту, чи необхідно амбулаторне лікування, строк втрати працездатності,чи є на диспансерному обліку, примітка
3.	Облік товарів на складі	Найменування товару, тип товару, кількість на складі, дата надходження, виробник, постачальник, опис товару, вартість.
4.	Облік відвідувачів спортивної школи	Ім'я та прізвище вихованя, дата народження, адреса, телефон, секція, команда, особисті дані тренерів, розклад занять, облік змагань та досягнень за роками, примітка
5.	Облік товарів та продаж у магазині спортоварів	Наявні на складі товари, товари під замовлення, назва товару, вартість товару, дата продажу, відповідальний менеджер, опис товару, тип товару.
6.	Облік автомобілів в автосалоні	Модель авто, вартість, колір кузова, наявність або відсутність автоматичної коробки передач, кількість одиниць, клієнти та менеджери салону.

7.	Облік товарів і продаж у продуктовому магазині	Наявні на складі товари, товари, що є на полицях, назва товару, тип товару, вартість товару, час та дата продажу, відповідальний менеджер, упаковка
8.	Книжкове видавництво	Автори, назва, розділ (технічна, суспільно-політична і т.п.), рік видання, тираж, відаповідальний за друк, кількість сторінок, кількість томів; номер тома, ціна
9.	Облік рейсів в аеропорту	Назва аеропорту, марка літака, розклад рейсів, номер рейсу, пункт призначення, дата рейсу, час вильоту, час у дорозі, чи є маршрут міжнародним, пілоти, стюардеси, кількість місць, вартість квитків, кількість проданих квитків
10.	Облік книжок і читачів у бібліотеці	Прізвища читачів, номер читацького квитка, рік народження, дата запису, вид читача (студент, аспірант, викладач тощо), назви взятих книг, ціна, чи є новим виданням, коротка анотація, дати їх видачі, дати повернення, працівники бібліотеки
11.	Облік робіт у мережі автомайстерень	Особисті дані водія, номер прав водія, адреса та телефон власника автомобіля; номер, марка, потужність і колір автомобіля особисті дані механіка, кваліфікація механіка, назва, адреса та телефон ремонтної майстерні
12.	Облік товарів, що виробляються на підприємстві	Код виробу, назва виробу, адреса та телефон підприємства, кількість співробітників, обсяг продукції, що випускається, тип продукції, рік випуску та обсяг випуску певного виробу, замовлення, клієнти

13.	Облік науково-викладацьких кадрів на кафедрі та їх навантаження	Прізвище та ім'я викладача, рік народження, адреса, телефон, кафедра, посада, науковий ступінь, стажування, навчальне навантаження, дисципліни, закріплені за викладачем, наукові праці
14.	Облік місць у поїздах у касах продажу	Тип потягу,сполучення, кількість місць, вартість квитка, кількість проданих квитків,заброньовані квитки
15.	Облік тварин і рослин, занесених в Червону книгу України	Нава тварини, вид тварини, рід, сімейство, дата занесення в книгу, чисельність популяції, живе чи ні в Україні, необхідні для порятунку заходи, назва рослини, вид, ареал поширення, дата занесення в книгу
16.	Облік рослин та робіт у лісовому господарстві	Найменування зеленого масиву, площа, основна порода, чи є заповідником, дата останньої перевірки; прізвище обслуговуючого лісника, примітка
17.	Облік авто та замовлень у автотранспортному підприємстві	Номерний знак автомобіля, марка автомобіля, технічний стан авто, вантажопідйомність, середня швидкість, витрата палива; табельний номер водія, особисті дані водія, стаж роботи, оклад; дата виїзду, дата прибуття, місце призначення, відстань, витрата пального, маса вантажу
18.	Облік замовлень та турів в турфірмі	Наявні тури, час поїздки, вартість туру, тривалість туру, кількість турів, дані про клієнтів, замовлені тури, менеджери турфірми

19	Облік замовлень на роботи в будівельній фірмі	Прізвище, ім'я, по батькові клієнта, номер рахунку, адреса, телефон клієнта; номер замовлення, дата виконання, вартість замовлення, витрачені будівельні матеріали, їх ціна та кількість, працівники, що виконували замовлення, примітка
20.	Облік товарів в аптеці	Наявні на складі товари, назва товару, місце збереження товару (холодильник, сейф, склад), тип товару (ліки, косметичні засоби, супутні товари), лікарська форма, вартість товару, термін придатності, час та дата надходження, відповідальний менеджер, упаковка, примітки

ЛІТЕРАТУРА

1. Садаладж П.Дж. NoSQL: Новая методология разработки нереляционных баз даннях / П. Дж.Садаладж, М. Фаулер– М. : Вильямс, 2013. – 172 с.

2. Демиденко М.А. Введення в сучасні бази даних : навч. посіб. / М.А. Демиденко; НТУ «Дніпровська політехніка». – Д. : 2020. – 38 с.

3. Dan McCreary, Ann Kelly. Making Sense of NoSQL: A guide for managers and the rest of us. – Manning Publications, 2013. – 312 p.

4. MongoDB. Documentations. [Електронний ресурс] – Режим доступу: https://docs.mongodb.com/manual/

5. Кайл Бэнкер. MongoDB в дії / переклад з англ. / Кайл Бэнкер – М. : ДМК «Пресс», 2012. – 395с.

6. Robinson, I. and Webber, J. and Eifrem, E. Graph Databases. – O'Reilly Media, 2013. – 178 p.

7. Neo4j. Developer Guides. [Електронний ресурс] – Режим доступу: https://neo4j.com/developer/graph-database/

8. OrientDB. Documentation [Електронний ресурс] – Режим доступу: https://orientdb.org/docs/3.0.x/

9. Эванс Э. Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем. – М.: Вильямс,

2011. – 448 с.

10. Shashank Tiwari. Professional NoSQL. – John Wiley & Sons Inc, 2011. – 384 pGirish Kumar. Exploring the different types of nosql databases [Електронний ресурс] – Режим доступу: https://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases

11. Тереник Д., Кучук Г. Порівняння SQL і NoSQL баз даних на прикладі проектування аффілейт репорт систем // Радіоелектронні і комп'ютерні системи. Спеціалізовані системи обробки інформації. 2020, – №1(93). – С. 83–89

12. Швець М., Заруба Д., Хохлов Ю. Порівняння SQL і NoSQL баз даних // Вчені записки ТНУ імені В.І. Вернандського. Серія: Технічні науки, 2018, – Т. 29 (68), № 6 – С. 21–25