Міністерство освіти і науки України

Ніжинський державний університет імені Миколи Гоголя

Навчально-науковий інститут природничо-математичних, медико-біологічних наук та інформаційних технологій

Кафедра інформаційних технологій, фізико-математичних та

економічних наук

Освітня програма: Комп’ютерні науки

Спеціальність:122 Компʼютерні науки

**КВАЛІФІКАЦІЙНА РОБОТА**

на здобуття освітнього ступеня *магістр*

**Аналіз NoSQL-рішень для управління великими даними**

студента **Крикуна Віталія Анатолійовича**

**Науковий керівник:**

Лисенко Ірина Миколаївна,

кандидат фізико-математичних наук, доцент

**Рецензент:**

Компан Сергій Володимирович

канд. фіз-мат. наук, доцент, Державний університет інформаційно-комунікаційних технологій

Чернишова Еліна Олександрівна

к. т. н., доцент

**Допущено до захисту:** \_\_\_\_ \_\_\_\_\_\_ 2024 р.

Завідувач кафедри

проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Казачков І.В.

Ніжин − 2024

ЗМІСТ

[ВСТУП 3](#_Toc182296918)

[РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ТА КЛАСИФІКАЦІЯ NoSQL-БАЗ ДАНИХ 6](#_Toc182296919)

[1.1 Визначення та основні характеристики NoSQL баз даних 6](#_Toc182296920)

[1.2 Класифікація NoSQL-рішень 12](#_Toc182296921)

[1.3 Архітектура NoSQL баз даних 18](#_Toc182296922)

[Висновки до розділу 1 25](#_Toc182296923)

[РОЗДІЛ 2. АНАЛІЗ NOSQL-РІШЕНЬ ДЛЯ РОБОТИ З ВЕЛИКИМИ ДАНИМИ 27](#_Toc182296924)

[2.1 Особливості великих даних та організації їх сховищ 27](#_Toc182296925)

[2.2 Використання NoSQL баз даних для обробки великих даних 33](#_Toc182296926)

[2.3 Аналіз кейсів використання NoSQL для Big Data 35](#_Toc182296927)

[Висновки до розділу 2 39](#_Toc182296928)

[РОЗДІЛ 3. РОЗРОБКА ІНТЕРФЕЙСУ ДЛЯ ОБРОБКИ ДАНИХ НА ОСНОВІ PySpark ТА NoSQL 40](#_Toc182296929)

[3.1 Огляд архітектури та вибір технологій для розробки 40](#_Toc182296930)

[3.2 Розробка бази даних NOSQL для MongoDB 42](#_Toc182296931)

[3.3 Тестування та аналіз продуктивності розробленого рішення 53](#_Toc182296932)

[Висновки до розділу 3 57](#_Toc182296933)

[ВИСНОВКИ 58](#_Toc182296934)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 60](#_Toc182296935)

[ДОДАТКИ 63](#_Toc182296936)

**ВСТУП**

Поява комп’ютерних систем і швидкі зміни у виробництві підвищили вимоги до якості та продуктивності товарів і послуг. Для вирішення цієї задачі стало необхідним автоматизувати процеси реального світу та впроваджувати комп’ютерні системи.

Довгий час системи реляційних баз даних були основою інформаційних систем. Однак нездатність таких систем обробляти сучасні робочі навантаження призвела до проблем масштабованості, продуктивності та доступності через її жорстку структуру схеми. Компанії в усьому світі почали запроваджувати нові способи зберігання та масштабування великих обсягів даних.

Актуальність теми дослідження обумовлена стрімким зростанням обсягів даних, які організації у різних галузях щоденно генерують та використовують для прийняття рішень, прогнозування і аналітики. Традиційні реляційні бази даних, хоча й є надійними та перевіреними засобами зберігання, часто не здатні ефективно обробляти великі обсяги неструктурованих та напівструктурованих даних. Це породило потребу в нових підходах і технологіях, зокрема в NoSQL-рішеннях, які забезпечують високу продуктивність, масштабованість і гнучкість у роботі з великими даними. Аналіз таких рішень є надзвичайно важливим для розуміння їх можливостей та обмежень у контексті обробки й управління великими даними.

В сучасних дослідженнях особливу увагу приділяють аналізу організації взаємодії NoSQL з великими даними. Завдяки цьому були сучасні вимоги до обробки та зберігання інформації. Оскільки обсяги даних постійно зростають, особливо у таких галузях, як соціальні мережі, електронна комерція, фінанси, медіа та Інтернет речей (IoT), традиційні реляційні бази даних не завжди здатні ефективно справлятися з обробкою цих даних. Саме тут NoSQL-системи стали критично важливими, оскільки вони пропонують гнучкі та продуктивні моделі зберігання, що дозволяють обробляти різноманітні типи даних із високою швидкістю.

NoSQL бази даних забезпечують високу масштабованість за рахунок горизонтального розподілу даних на декількох серверах, що є вирішальним у контексті великих даних, де важливо не лише зберігати інформацію, але й забезпечувати доступ до неї з мінімальними затримками. Завдяки різним моделям узгодженості та гнучкості в структурі, NoSQL дозволяє компаніям і організаціям обробляти як структуровані, так і неструктуровані дані, такі як текст, відео, зображення, логи і дані сенсорів.

Важливість аналізу NoSQL-систем полягає також у тому, що різні NoSQL-технології мають специфічні можливості та обмеження, які впливають на ефективність системи в певних завданнях. Оцінка продуктивності, узгодженості даних, вимог до швидкості та надійності допомагає приймати обґрунтовані рішення щодо вибору NoSQL-рішень для конкретних завдань.

Загалом, аналіз використання NoSQL для великих даних є важливим етапом при створенні масштабованих і продуктивних систем зберігання та обробки інформації, оскільки допомагає підвищити ефективність бізнес-процесів і прийняття рішень. Це особливо актуально для компаній, що працюють з великими масивами даних і прагнуть оптимізувати свої ресурси, забезпечуючи при цьому швидкий доступ до інформації та надійне зберігання.

**Мета дослідження** полягає у визначенні й порівнянні основних характеристик різних NoSQL-технологій для управління великими даними, їх відповідності сучасним вимогам до зберігання, доступності, швидкості обробки і аналізу великих обсягів інформації.

**Завдання дослідження** включають:

1. Проведення огляду сучасних NoSQL-рішень, які використовуються для управління великими даними.
2. Визначення основних типів NoSQL-баз даних та їх особливостей.
3. Оцінку характеристик NoSQL-технологій з точки зору масштабованості, доступності, продуктивності та гнучкості.
4. Виявлення переваг і недоліків використання NoSQL-баз даних для різних сценаріїв роботи з великими даними.
5. Розробку бази даних NoSQL-рішень та перевірка її продуктивності.

**Об’єктом дослідження** є системи управління базами даних, що застосовуються для обробки та зберігання великих обсягів інформації.

**Предметом дослідження** є особливості архітектури, функціональні можливості та характеристики NoSQL-рішень для управління великими даними.

**Методи дослідження.** У роботі теоретичною основою досліджень є:

* Методи аналізу літератури та наукових джерел: огляд публікацій пов’язаних з архітектурою, особливостями використання NoSQL рішень, застосуванням баз даних для зберігання та обробки великих даних;
* Методи роботи з базами даних: для розробки бази даних NoSQL;
* Методи алгоритмічного програмування для створення функціоналу рекомендаційних систем;
* Методи аналізу для вибору відповідних технологій, платформи для розробки бази даних.

**Практичне значення.** Результати аналізу, проведеного в даному дослідженні, та їх узагальнення можуть бути покладені в основу рішень щодо обґрунтованого вибору інструментів та платформ для розробки NoSQL рішень.

**Апробація дослідження.** Результати дослідження були **апробовані** на XVІ Міжнародній науково-практичній конференціії молодих учених, аспірантів та студентів «Основні напрями розвитку наукових досліджень молодих учених» й опубліковані в збірнику «Наукові горизонти: збірник наук. праць (Секція: Інформаційно-комунікаційні технології)». Вип. XVI-XVII.

**Структура та обсяг випускної кваліфікаційної роботи.** Випускна кваліфікаційна робота складається із вступу, трьох розділів, висновків, списку використаних джерел із 22 найменування, додатків і містить 64 сторінки основного тексту, 23 рисунки.

**РОЗДІЛ 1**

**ТЕОРЕТИЧНІ ОСНОВИ ТА КЛАСИФІКАЦІЯ NoSQL-БАЗ ДАНИХ**

**1.1 Визначення та основні характеристики NoSQL баз даних**

NoSQL (Not Only SQL) бази відрізняються від традиційних реляційних баз даних тим, що вони не використовують стандартну SQL-мову запитів і часто оптимізовані для роботи з великими обсягами розподілених даних. Такі системи характеризуються своєю здатністю обробляти великі обсяги даних з високою продуктивністю та масштабованістю, а також надають розробникам гнучкість в управлінні структурою даних [16].

Основна відмінність NoSQL баз даних від реляційних полягає в їх структурі та методах зберігання даних. Реляційні бази даних будуються на основі таблиць, стовпців і рядків та використовують жорстко задані схеми даних, які вимагають визначення перед початком роботи. На відміну від них, NoSQL бази даних можуть використовувати різні підходи, включаючи документо-орієнтований, графовий, ключ-значення та колонкові моделі зберігання, які дозволяють зберігати дані у більш гнучкій та оптимізованій формі для конкретних типів запитів та операцій [4].

До того ж, NoSQL бази даних часто надають можливість горизонтального масштабування, що означає додавання додаткових серверів для обробки обсягу даних, що збільшується, без необхідності їх централізації. Це контрастує з вертикальним масштабуванням у реляційних базах даних, яке зазвичай обмежується збільшенням потужності сервера.

Розробка NoSQL баз даних була стимульована кількома ключовими факторами [3].

1. Зростання обсягів даних. Сучасні програми генерують величезні обсяги даних, що створює проблеми для традиційних реляційних систем з погляду продуктивності та масштабованості.
2. Різноманітність типів даних. Необхідність в обробці не тільки структурованих, а й напівструктурованих та неструктурованих даних стимулювала розвиток гнучких схем даних, які можуть легко адаптуватися до вимог, що змінюються.
3. Розподілені обчислення. Зі зростанням інтернет-технологій та хмарних обчислень виникла потреба в базах даних, які можуть ефективно працювати у розподілених середовищах.
4. Високі вимоги до доступності та стійкості до відмов. Додатки, що потребують цілодобової доступності та високої стійкості до збоїв, знайшли рішення в NoSQL, оскільки ці системи забезпечують більш просунуті можливості реплікації та відновлення даних.

Перелічені фактори призвели до широкого прийняття та розвитку NoSQL технологій, оскільки вони пропонують рішення, які можуть впоратися із сучасними викликами обробки та зберігання даних.

NoSQL (Not Only SQL) бази даних з’явилися як відповідь на зростання обсягів даних і складність роботи з традиційними реляційними базами. Вони забезпечують можливість зберігання великих обсягів структурованих, напівструктурованих і неструктурованих даних, а також забезпечують гнучкість у масштабуванні. На відміну від реляційних баз даних, які використовують фіксовану схему таблиць, NoSQL бази орієнтовані на зберігання даних в гнучкому форматі, що підходить для широкого спектра сучасних додатків.

Основні характеристики NoSQL баз даних [4, 22].

1. NoSQL бази даних не вимагають заздалегідь визначеної схеми. Це означає, що структуру даних можна легко змінювати без необхідності переробляти всю базу. Ця гнучкість особливо корисна для великих систем, де формат даних часто змінюється. Наприклад, у документо-орієнтованих базах даних, таких як MongoDB, документи можуть мати різну структуру, що полегшує адаптацію системи до нових вимог.
2. NoSQL бази даних побудовані для горизонтального масштабування, що дозволяє розподіляти дані між кількома серверами. Це важливо для обробки великих обсягів інформації та збільшення продуктивності системи. Завдяки цьому NoSQL бази з легкістю можуть обробляти мільйони запитів на секунду, підтримуючи масштабні вебзастосунки та аналітичні платформи.
3. Зазвичай NoSQL бази даних забезпечують високу продуктивність навіть при значних обсягах даних завдяки відсутності складних операцій з’єднання (JOIN), характерних для реляційних баз. Доступ до даних реалізується з мінімальними затримками, що особливо корисно для додатків у режимі реального часу, таких як системи електронної комерції, соціальні мережі чи аналіз потокових даних.
4. Підтримка різних моделей зберігання даних. NoSQL бази класифіковані за моделями зберігання, кожна з яких відповідає певним сценаріям використання [15]:
   1. Документо-орієнтовані бази даних (MongoDB, Couchbase) зберігають дані у форматі документів (JSON або BSON).
   2. Колонкові бази даних (Cassandra, HBase) оптимізовані для зберігання даних у стовпцях, що дозволяє швидко обробляти аналітичні запити.
   3. Ключ-значення бази даних (Redis, DynamoDB) працюють з простими парами ключ-значення, що робить їх ефективними для кешування і зберігання сеансів.
   4. Графові бази даних (Neo4j, ArangoDB) підходять для роботи зі зв’язками між об’єктами та широко використовуються в рекомендаційних системах та соціальних мережах.
5. Багато NoSQL баз даних не дотримуються повного набору ACID-властивостей (атомарність, узгодженість, ізольованість, довговічність), натомість надаючи пріоритет швидкості й доступності (модель BASE). Це означає, що в NoSQL системах можуть бути тимчасові невідповідності даних, які з часом узгоджуються. Однак для багатьох сучасних додатків тимчасова неконсистентність є прийнятною, якщо це підвищує продуктивність.
6. Різноманітність інструментів для обробки та аналізу даних означає, що NoSQL бази даних підтримують різні запити та інструменти для аналітики, що дозволяє ефективно проводити аналіз великих даних. Більшість NoSQL баз підтримують інтеграцію з інструментами для обробки даних, такими як Apache Spark, що дозволяє здійснювати масштабовану обробку та аналіз на великих кластерах даних.

Реляційні бази даних вимагають попередньо визначеної моделі даних і структурованих даних. Вони пропонують розширену функціональність для керування, оновлення та запитів даних за допомогою SQL. Бази даних SQL зберігають надійність і цілісність даних і транзакцій, дотримуючись властивостей ACID (атомність, послідовність, ізоляція, міцність) [12].

Однак забезпечити властивості ACID важко досягти у випадку великих зростаючих наборів даних. Тому в базах даних NoSQL покладаються на принципали BASE (в основному доступність, м’який стан, остаточна узгодженість). Вони пропонують гнучку архітектуру для обробки не тільки структурованих даних, але також неструктурованих і напівструктурованих даних.

Варто зазначити, що властивості ACID і BASE є наслідком з теореми CAP (узгодженість, доступність, стійкість до розподілення). Принципи BASE більш гнучкі, ніж принципи ACID, тоді як властивості ACID забезпечують більшу послідовність і надійність транзакцій. Однак ці дві якості досягаються за рахунок продуктивності та збільшення витрат.

Крім того, реляційним базам даних не вистачає ефективності при роботі з великими даними. Насправді продуктивність реляційних баз даних має тенденцію до зниження зі збільшенням обсягу даних, особливо при роботі з напівструктурованими даними у великих сховищах. Крім того, вони вимагають значних витрат, коли потрібно збільшити масштабованість (наприклад, додавання серверів для зберігання та обробки великих наборів даних вимагає придбання додаткових ліцензій). Також зростаюча потреба в аналізі в реальному часі великих змінних неоднорідних обсягів даних додає ще один рівень складності [12].

На відміну від RDMS, бази даних NoSQL були адаптовані та вдосконалені для забезпечення масштабованості, продуктивності та гнучкості, необхідних для випадків використання великих даних. Наприклад, до мільярда даних можна вводити щодня в Hypertable-сховище колонок Zvent, а Google може обробити 20 петабайт даних, які зберігаються в BigTable через MapReduce. Крім того, вони засновані на більш доступному обладнанні та технологіях порівняно з реляційними базами даних. NoSQL дозволяє уникнути непотрібної складності реляційних баз даних [3]. Наприклад, сайти соціальних медіа та великі веб-додатки не обов’язково потребують надійних транзакцій і властивостей ACID (наприклад, оновлення статусу Facebook або коментарів у Twitter). Нульова втрата даних, нульове переривання обслуговування не є вирішальними в таких випадках. Крім того, впровадження властивостей ACID RDMS може бути дорогим порівняно з корисністю соціальних медіа [14].

З усіх цих причин реляційні бази даних не підходять для зберігання великих даних. Бази даних NoSQL забезпечують кращу доступність, масштабованість, продуктивність і гнучкість. Вони можуть обробляти всі типи даних (структуровані, напівструктуровані та неструктуровані дані).

Крім того, в реляційних базах даних дуже важко керувати змінами. Користувачі повинні визначити схему бази даних перед введенням даних. Слід ретельно вивчати будь-які зміни в схемі бази даних або таблицях. В іншому випадку такі зміни можуть призвести до збою служби, зниження продуктивності або можуть вимагати обслуговування та додаткових інвестицій для адаптації модулів програми.

Навпаки, бази даних NoSQL дозволяють легко керувати змінами. Насправді немає необхідності вказувати заздалегідь жорстку схему бази даних. Це дає користувачам можливість зберігати дані без попередньо визначеної схеми. Крім того, можна змінити будь-яку модель даних через деякий час, не впливаючи на продуктивність системи чи програми.

Користувачі реляційних баз даних використовують для запитів мову Structured Query Language (SQL). Однак не існує мови запитів для баз даних NoSQL. Кожна база даних NoSQL має свій унікальний спосіб керування, вилучення та запиту даних. З іншого боку, бази даних SQL є потужним засобом для обробки складних запитів через стандартизований інтерфейс. В той час, як базам даних NoSQL не вистачає продуктивності при роботі зі складними запитами. Натомість NoSQL є більш адекватним для обробки паралельних обчислень і математичних розрахунків на розподілених великих наборах даних.

Для NoSQL використовується шардинг, тобто розподіл великих обсягів даних між серверами та віртуальними вузлами даних. Бази даних NoSQL використовують сегментування, щоб збалансувати навантаження і забезпечити паралельне зберігання та обробку. Вони пропонують можливість додавати або видаляти сервери з рівня даних без впливу на продуктивність програми [16].

Традиційні засоби резервування даних зосереджені на віддзеркаленні даних. Вони копіюють дані через цільові масиви в центрі обробки даних або через віддалений сайт. Даний метод займає багато місця для зберігання, особливо у випадку великих наборів даних, які перевищують петабайти. Насправді для організації зберігання великих потоків даних (дані в русі), а також великих архівів даних за допомогою традиційних засобів є накладними витратами та дорогими витратами. Щоб запобігти втраті даних, більшість баз даних NoSQl забезпечують автоматичну реплікацію даних для відмовостійкості. Вони тиражують дані між серверами кластерів і навіть центрами обробки даних. Таким чином, використовуючи технології Big Data і бази даних NoSQL, розробникам не потрібно хвилюватися ні про складність гетерогенного середовища зберігання, ні про механізми паралельної обробки. Прикладом реплікації даних у реальному часі є IBM InfoSphere Data. Реплікація даних у реальному часі дозволяє забезпечити постійну високу доступність даних як у гетерогенному, так і в однорідному середовищі. Реплікація даних у реальному часі має вирішальне значення для створення звітів, інтерактивного аналізу та забезпечення синхронізації транзакцій. Це допомагає отримати точну інформацію, підтримує швидке прийняття рішень і оптимізує ресурси. Ще одним прикладом забезпечення резервування даних є використання алгоритму виправлення помилок під назвою Erasure Coding, який поєднується з технікою об’єктного зберігання. Таке рішення є альтернативою реплікації даних у розподіленому середовищі. Наприклад, об’єкт даних (документ із його метаданими) розбивається на сегменти. Кожен сегмент кодується та розрізається на фрагменти, які зберігаються на різних серверах. Таким чином, якщо деякі фрагменти більше не доступні через збій диска, організація все ще може реконструювати вихідні дані. Це рішення знижує вартість, споживає менше пам’яті та гарантує відмовостійкі репозиторії.

**1.2 Класифікація NoSQL-рішень**

Бази даних NoSQL можна класифікувати за допомогою різних підходів і різних критеріїв. Деякі експерти класифікують їх відповідно до моделі даних, і більшість із них виділяють чотири основні типи баз даних NoSQL, зокрема: бази даних ключ-значення, бази даних документів, баз даних, орієнтованих на стовпці і графових баз даних.

У NoSQL базах даних «ключ-значення» кожен елемент зберігається як пара «ключ-значення», що робить їх простими і дуже швидкими. Ця модель реалізована за допомогою хеш-таблиці, де є унікальний ключ і вказівник на певний елемент даних, створюючи пару ключ-значення. Хеш-таблиця підходить для пошуку простих або складних значень у надзвичайно великих наборах даних. Бази даних ключ-значення [10] можуть обробляти дуже велику кількість записів. Вони можуть підтримувати велику кількість змін стану за секунду з мільйонами одночасних користувачів за допомогою розподіленої обробки та зберігання. Бази даних ключових значень покладаються на свою надмірність, щоб протистояти втраті вузлів зберігання та захищати програми [15].

Вони дуже корисні як для зберігання результатів аналітичних алгоритмів (наприклад, підрахунок фраз серед великої кількості документів), так і для отримання цих результатів у звітах. Однак бази даних «ключ-значення» не надають жодних традиційних можливостей бази даних. Таким чином, щоб забезпечити атомарність транзакцій або узгодженість кількох паралельних транзакцій, користувачі повинні покладатися на саму програму [1.

Іншим недоліком є те, що користувачі не можуть отримати доступ до даних за значенням. Дійсно, неможливо зробити запит до сховища даних типу ключ-значення, щоб отримати всі записи, які містять певний набір значень. Єдиний спосіб зробити запит до бази даних ключових значень – це вказати запит за ключем або діапазоном ключів.

Прикладами СУБД «ключ-значення» є Redis, Riak і Voldemort. Ці три рішення забезпечують масштабованість і відмовостійкість, а також певне підвищення продуктивності.

Riak розроблений для сильно розподілених середовищ, таких як хмара. Він забезпечує високу відмовостійкість, але з меншою продуктивністю, ніж Redis. Redis більше підходить для критичних за часом програм, оскільки він покладається на набір даних у пам’яті для швидкої відповіді. Redis підходить для обробки даних, що швидко змінюються, наприклад збору даних у реальному часі від датчиків і зв’язку в реальному часі. З іншого боку, Voldermort підходить для дуже великих наборів даних, таких як геологічні дані та метадані карт. Він може обробляти зберігання величезних обсягів без значного впливу на продуктивність [13].

Документо-орієнтовані NoSQL бази даних зберігають інформацію у вигляді документів, найчастіше у форматі JSON або BSON. Кожен документ є самостійною одиницею, яка містить структуровані дані з гнучкою схемою. Це дозволяє зберігати різнорідні дані у межах однієї колекції. Такі бази даних були розроблені для зберігання та керування великомасштабними документами. Даний тип бази даних присвоює ключове значення кожному документу. Документи містять кілька пар ключ-значення або пар ключ-масив або навіть вкладені документи. Документи кодуються в стандартному форматі обміну даними, наприклад XML, JSON (нотація параметрів Javascript) або BSON (двійковий JSON). Бази даних документів визнані потужним, гнучким і гнучким інструментом для зберігання великих даних [2].

Бази даних документів відрізняються від сховищ ключ-значення. Вони дозволяють користувачам шукати дані на основі вмісту документів. Вони можуть робити запити за ключами, значеннями або прикладами. Насправді закодовані документи містять об’єкти метаданих, тому можна запитувати дані за прикладом [11]. Це надає базам даних документів велику гнучкість, необхідну для деяких випадків використання. Для запуску запитів користувачі можуть покладатися на API програмування або мову запитів.

На відміну від простих сховищ ключ-значення, стовпець значення в базах даних документів містить напівструктуровані дані, зокрема пари ім’я/значення атрибута. Крім того, бази даних документів підтримують гнучку схему. Вони дозволяють зберігати сотні атрибутів в одній колонці схеми документа. Тому рядки можуть отримувати різну кількість і типи атрибутів.

Як приклад, CouchDB і MongoDB є відкритими документо-орієнтованими СУБД. Для обох дані зберігаються в документах із самодостатніми записами та без внутрішніх зв’язків. CouchDB зберігає дані на диску за допомогою файлів лише для додавання, тоді як MongoDB зберігає дані за допомогою Memory-Mapped Storage Engine. Вона використовує файли, відображені в пам’яті, для всього дискового вводу/виводу. Як формат обміну, CouchDB пропонує HTTP API як для доступу до даних, так і для адміністрування. Натомість MongoDB надає дротовий протокол на основі сокетів із BSON.

Для відмовостійкості CouchDB підтримує реплікацію як головний/головний, так і головний/підлеглий. Реплікацію можна точно налаштувати за допомогою фільтрів реплікації. Навпаки, Mongo керує реплікацією за допомогою форми асинхронної реплікації головного/підлеглого під назвою Набори реплік.

Обидва рішення мають хорошу підтримку спільноти. Однак CouchDB не пристосована до надзвичайно мінливих даних, а MongoDB підходить для динамічних запитів і забезпечує кращу продуктивність у великих базах даних.

Колонкові бази даних являють собою розширення моделі ключ-значення зі стовпцями. Вони були розроблені для обробки розподілених даних через пул інфраструктури. Колонкові бази даних засновані на гібридному підході, який спирається на декларативні характеристики реляційних баз даних і різні схеми сховищ ключ-значення.

Hbase, Cassandra та Accumulo є прикладами колонкових СУБД. HBase [17] – це система управління базами даних NoSQL, розроблена для роботи поверх HDFS. Це проект з відкритим вихідним кодом, який підходить для обробки різних великих наборів даних. Він заснований на моделі даних ключа/значення, орієнтованої на стовпці. HBase розроблено для підтримки високої швидкості оновлення таблиць і горизонтального масштабування в розподілених кластерах. HBase забезпечує гнучкий структурований хостинг для дуже великих таблиць у форматі, схожому на BigTable. HBase надає багато функцій, таких як запити в реальному часі, пошук природною мовою, послідовний доступ до джерел великих даних, лінійна та модульна масштабованість, автоматичне та настроюване шардинг таблиць [16]. Це популярна нереляційна СУБД, яка включена в багато рішень Big Data і вебсайтів, керованих даними.

Cassandra також є популярною СУБД NoSQL для дуже великих наборів даних. Це СУБД ключ-значення, яка використовує сховище, орієнтоване на стовпці, шардинг за діапазонами ключів і надлишкове сховище. З її допомогою забезпечується масштабованість, продуктивність читання/запису, а також стійкість до «гарячих» вузлів і збоїв вузлів. Cassandra дозволяє налаштовувати параметри для налаштування компромісів між узгодженістю та доступністю.

Apache Accumulo – це відсортоване розподілене сховище ключів/значення, яке забезпечує масштабованість і високу продуктивність. Apache Accumulo базується на дизайні Google BigTable і побудований на основі Hadoop, Zookeeper і Thrift. Це дозволяє контролювати доступ на рівні комірки в BigTable. Це також дозволяє змінювати пари ключ/значення на різних етапах процесу керування даними. Це забезпечується механізмом програмування на стороні сервера.

Як реляційні бази даних, так і бази даних NoSQL, такі як сховища ключів і значень, неефективні при роботі з даними. Їм не вистачає продуктивності та гнучкості, необхідних для обробки та запиту кількох зв’язків у великих наборах даних [17].

Незважаючи на те, що парадигма MapReduce у поєднанні з фреймворком Hadoop забезпечує масштабованість, відмовостійкість і інструменти, які легко програмувати для великих наборів даних, деякі бази даних NoSQL, такі як сховища ключ-значення, не завжди підходять для підключених даних і дуже великих графів [12].

Навпаки, графові бази даних придатні для зберігання не лише інформації про об’єкти, а й усіх зв’язків, що існують між ними. Вони покладаються на графову модель без схем, щоб легко моделювати та представляти зв’язані дані. Така модель включає вершини (наприклад, об’єкти або елементи, представлені вузлами) і ребра для представлення зв’язків між даними.

Графові бази даних корисні для зберігання, доступу та аналізу сили та характеру зв’язків між двома чи більше елементами (наприклад, наскільки близькі стосунки між двома людьми? Як далеко таксист знаходиться від іншого або від туристичного місця). Відповіді на такі питання дозволяють сформулювати цінні рекомендації в багатьох галузях промисловості.

Графові бази даних пропонують для багатьох випадків використання підвищену продуктивність (вони забезпечують меншу затримку порівняно з пакетною обробкою агрегатів), гнучку модель даних (простий спосіб вираження зв’язків і збагачення графа, коли дані та бізнес-вимоги стають точнішими) і гнучкість (можливість розвивати додатки контрольованим чином відповідно до практик розробки програмного забезпечення Agile і Test Driven) [8].

Neo4J, ArangoDB і OrientDB є прикладами графових СУБД. Neo4J – це база даних із відкритим вихідним кодом, яка зберігає високозв’язані дані у форматі графів, а не в таблицях (які більше підходять для агрегованих даних). Насправді дані зберігаються у вузлах, з’єднаних один з одним певним зв’язком. І вузли, і зв’язки мають свої властивості. База даних повністю написана на Java і підтримується технологією Neo. Це вбудований, дисковий, повністю транзакційний механізм збереження Java. Він забезпечує високу масштабованість, що дозволяє додавати до кількох мільярдів даних, високу доступність, навіть якщо дані розподіляються між багатьма машинами, швидкі запити та швидку ідентифікацію шляху через систему обходу. Крім того, аналітики даних можуть використовувати зрозумілу людині мову запитів, адаптовану для графових моделей. Neo4j пропонує також зручний і простий доступ (через інтерфейс Rest або об’єктно-орієнтований Java API). Він також забезпечує, як і реляційні бази даних, повні властивості ACID для надійних транзакцій [8].

ArangoDB – це розподілена та багатоцільова СУБД NoSQL з відкритим кодом. Вона підтримує кілька моделей даних, включаючи документи, графи та ключ-значення. Підходить для додатків, які потребують економії простору, високої продуктивності зі зручними інструментами запитів. Дійсно, це дозволяє використовувати SQL-подібну мову запитів, а також розширення JavaScript і Ruby. Мова запитів ArangoDB (AQL) розроблена для підтримки складних запитів, особливо щодо моделей даних ArangoDB. Зберігання та пошук даних базується на колекціях.

OrientDB – це система управління базами даних NoSQL з відкритим вихідним кодом, яка випущена за ліцензією Apache 2. СУБД написана на Java, тому може працювати в Linux, Windows і будь-якій системі, яка підтримує Java. OrientDB забезпечує гнучкість документів, а також хорошу продуктивність для роботи з розподіленими графами. OrientDB – це база даних на основі документів, яка складається з ODocuments (можливість динамічного додавання та видалення властивостей) [1]. Водночас це дозволяє керувати зв’язками, як у графових базах даних із прямими зв’язками між записами. Таким чином, OrientDB емулює властивість безіндексної суміжності документів. Він підтримує кілька режимів, у тому числі без схеми, повну схему та змішану схему. Крім того, OrientDB підтримує SQL як мову запитів із можливістю керування графіками пов’язаних документів. Завдяки OrientDB можливо обробляти зв’язки без об’єднань SQL.

**1.3 Архітектура NoSQL баз даних**

У NoSQL базах даних використовуються архітектурні патерни, що дозволяють забезпечити масштабованість, високу доступність і продуктивність. Через специфіку нереляційних баз даних та їхню схильність до обробки великих обсягів інформації, деякі з патернів, що активно використовуються в NoSQL, значно відрізняються від тих, що зустрічаються у реляційних системах. Розглянемо основні патерни архітектури NoSQL.

Шардинг – це горизонтальне розбиття даних між різними серверами або вузлами. Кожен вузол (шард) містить лише частину загальної бази даних, що дозволяє розподіляти навантаження (рис.1.1). Кожен шард відповідає за обробку тільки певної частини даних, що зменшує навантаження на окремі сервери. За необхідності, можна додати новий шард і перерозподілити дані для більш ефективного використання ресурсів. Шардинг дозволяє уникнути надмірного навантаження на один сервер, що підвищує надійність [12].

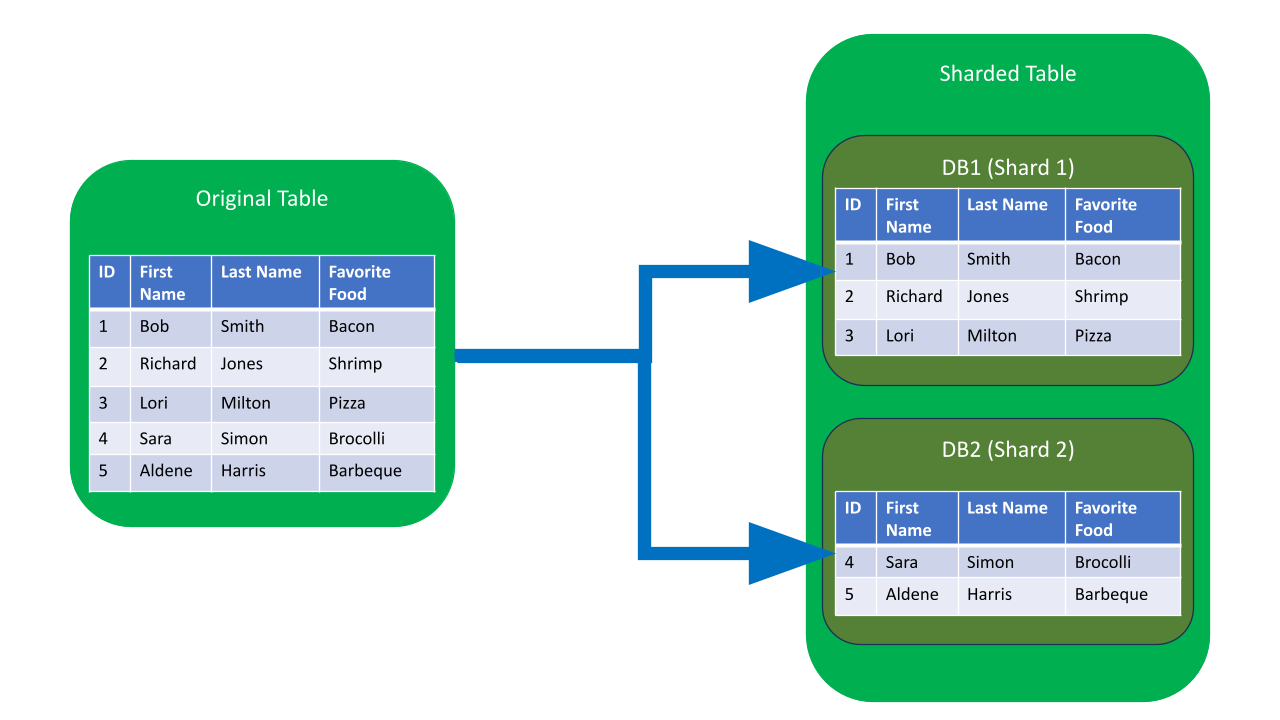


Рисунок 1.1 – Приклад шардингу

Реплікація в NoSQL базах даних – це процес створення копій даних на декількох серверах або вузлах, що дозволяє підвищити доступність і надійність системи, а також забезпечити відмовостійкість. Суть реплікації полягає в тому, що дані не зберігаються на одному вузлі, а розподіляються по різних серверах, кожен з яких має копію або частину даних. Це означає, що якщо один вузол вийде з ладу, система зможе продовжити роботу, оскільки інші вузли зберігатимуть потрібну інформацію [16].

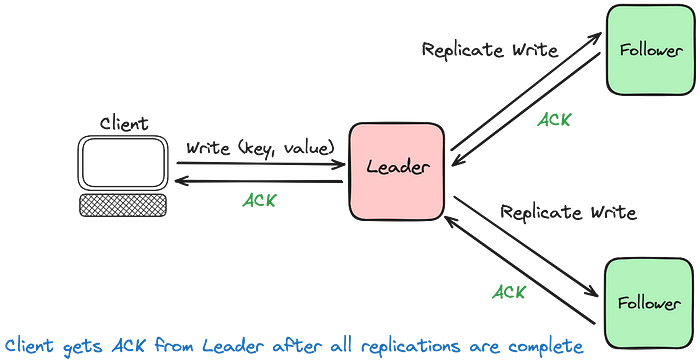


Рисунок 1.2 – Реплікація в БД

Існують два основних типи реплікації: синхронна та асинхронна. При синхронній реплікації зміни даних одразу розсилаються на всі вузли системи, і кожен з них підтверджує отримання оновлень. Це забезпечує високу узгодженість даних, проте уповільнює процес запису, оскільки запис вважається завершеним лише після оновлення всіх реплік. Такий підхід підходить для систем, де узгодженість даних важливіша за швидкодію. Асинхронна реплікація, навпаки, не вимагає негайного оновлення на всіх вузлах — головний сервер (primary) виконує запис і повертає підтвердження клієнту, а інші копії оновлюються з певною затримкою. Це збільшує швидкість роботи системи, але може призводити до тимчасової неузгодженості, коли не всі вузли мають однакові дані в один момент часу.

Реплікація дозволяє не лише забезпечити доступність даних, але й розподіляти навантаження на систему. Наприклад, запити на зчитування можна розподілити між кількома вузлами, зменшуючи навантаження на основний сервер. Багато популярних NoSQL СУБД, таких як MongoDB, Cassandra і Redis, підтримують реплікацію для підвищення надійності і доступності. У MongoDB використовується асинхронна реплікація через набір реплік, де один сервер виступає як основний, а інші — як резервні копії. У Cassandra також застосовується асинхронна реплікація, при якій копії даних можуть бути розміщені на різних вузлах, забезпечуючи масштабованість та надійність системи. Redis, у свою чергу, надає можливості як для синхронної, так і для асинхронної реплікації, що дозволяє налаштовувати систему відповідно до конкретних потреб.

Загалом, реплікація в NoSQL базах є необхідною для підтримки стабільності роботи великих розподілених систем. Вона дозволяє створювати резервні копії даних, швидко відновлювати доступ до них після збоїв та забезпечує гнучкість у налаштуванні рівня узгодженості і продуктивності відповідно до вимог бізнесу.

Модель узгодженості Eventual Consistency – це підхід до збереження даних у розподілених системах, при якому всі копії даних у системі, навіть якщо вони тимчасово не узгоджені, зрештою синхронізуються і стають однаковими. Ідея цієї моделі полягає в тому, що після завершення всіх операцій запису в систему та за відсутності нових запитів на оновлення, всі вузли, що зберігають копії даних, прийдуть до єдиного узгодженого стану через деякий час. Це дозволяє системі підтримувати доступність навіть при високих навантаженнях або збоях окремих вузлів, жертвуючи при цьому негайною узгодженістю даних (рис.1.3).

Eventual Consistency підходить для розподілених баз даних, де основне завдання – швидке обслуговування запитів на запис і читання, особливо в системах з великою кількістю користувачів. Наприклад, у соціальних мережах або у великих інтернет-магазинах модель eventual consistency дозволяє користувачам швидко зчитувати і записувати дані, навіть якщо інші копії цих даних будуть оновлені не миттєво [16].

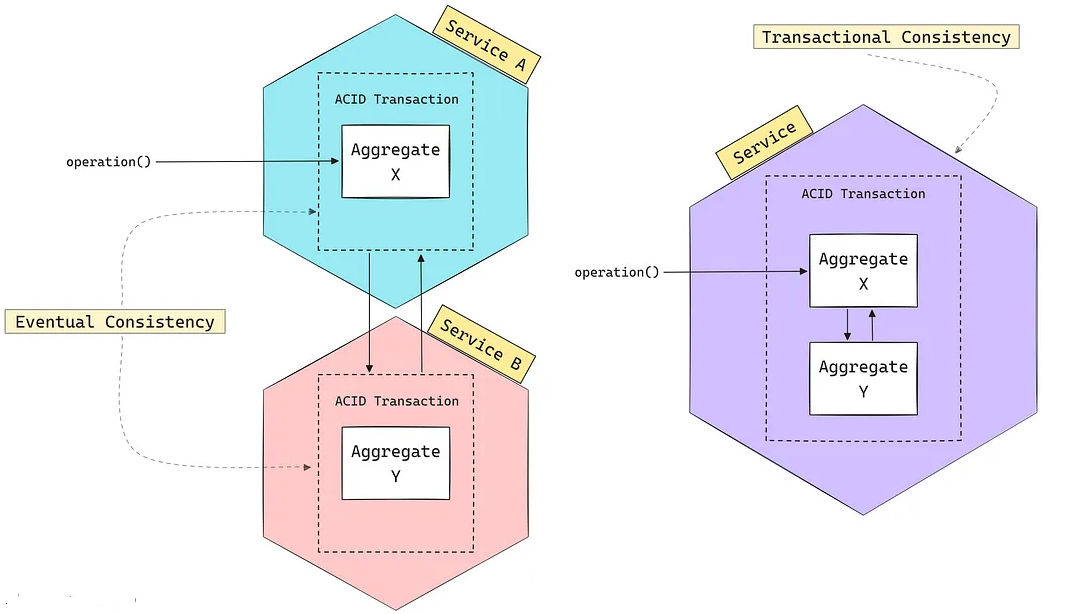


Рисунок 1.3 – Модель узгодженості Eventual Consistency

Важливою характеристикою цієї моделі є можливість тимчасової неузгодженості. Наприклад, якщо один користувач змінює дані, інші користувачі, які звертаються до іншої репліки, можуть бачити стару версію даних до тих пір, поки всі копії не синхронізуються. Водночас, система гарантує, що ця неузгодженість не є постійною: за достатньо тривалий час усі вузли матимуть однакові дані. Це дозволяє досягати високої продуктивності та відмовостійкості, але з можливим компромісом у вигляді тимчасових розбіжностей між копіями даних.

Таким чином, модель Eventual Consistency особливо підходить для систем, де важлива швидкодія і масштабованість, а миттєва узгодженість не є критичною.

Патерн CQRS (Command Query Responsibility Segregation) розділяє обробку команд (змін даних) і запитів (зчитування даних), забезпечуючи незалежні шляхи для роботи із системою залежно від типу операцій. Цей патерн базується на принципі, що команда (command) виконує операцію змін у системі, а запит (query) лише отримує дані, не змінюючи їх. Поділ команд і запитів підвищує ефективність роботи системи, забезпечує краще масштабування та спрощує роботу з різними вимогами до продуктивності і узгодженості [16].

Основна ідея CQRS полягає у використанні двох різних моделей для запису і читання даних. У системах, що використовують цей патерн, команди направляються до моделі, яка обробляє зміну даних, тоді як запити обробляються окремо – моделлю, яка виконує зчитування даних. Це дозволяє оптимізувати кожен шлях відповідно до конкретних потреб: операції запису можна налаштовувати для забезпечення надійності, а запити – для швидкодії. В результаті система стає більш гнучкою і масштабованою, особливо в умовах високих навантажень.

Завдяки CQRS можливо ізолювати логіку запису від логіки читання, що допомагає уникати конфліктів при великій кількості запитів, а також дозволяє розробникам оптимізувати окремі частини системи. Зокрема, для складних систем з великими обсягами даних CQRS часто використовується разом з Event Sourcing – підходом, коли кожна зміна стану системи зберігається як окрема подія. Це дає можливість відновлювати повний стан системи у будь-який момент часу.

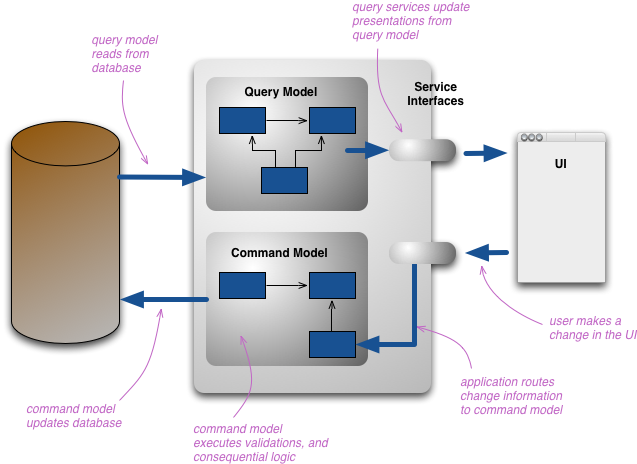


Рисунок 1.4 – Патерн CQRS (Command Query Responsibility Segregation)

Патерн CQRS також дозволяє використовувати різні бази даних для команд і запитів. Наприклад, у великих системах можна зберігати дані для запису у надійному сховищі, що забезпечує узгодженість, а дані для читання – у базі, оптимізованій для швидкого доступу. Це розподілення зменшує навантаження на основну базу даних і підвищує ефективність обробки даних у системі.

Загалом, CQRS допомагає будувати гнучкі, продуктивні та масштабовані системи, що особливо важливо для великих розподілених застосунків. Патерн дозволяє зменшити складність у роботі з великими обсягами даних і запитів, забезпечуючи чітке розмежування відповідальностей між командами і запитами.

Патерн MapReduce – це модель обробки даних, яка була розроблена для роботи з великими обсягами інформації в розподілених системах. Основна ідея цього патерну полягає у розподілі задачі на менші підзадачі, обробці цих підзадач паралельно на кількох вузлах системи, а потім об’єднанні результатів для отримання кінцевого результату. MapReduce складається з двох ключових етапів – Map та Reduce, кожен з яких виконує певну функцію [5].

На етапі Map вхідні дані розбиваються на менші частини і кожна з цих частин передається функції Map, яка обробляє її та перетворює на набір ключ-значення (key-value pair). Наприклад, якщо задача полягає у підрахунку слів у великому тексті, функція Map отримує кожен рядок тексту та створює пари ключів і значень, де ключ – це слово, а значення – кількість його появ у даному рядку (зазвичай це число 1 для кожного окремого слова). Цей етап відбувається паралельно на всіх вузлах, що дозволяє обробляти великі обсяги даних швидко та ефективно.

На етапі Reduce результати, отримані на етапі Map, групуються за ключами. Потім для кожного унікального ключа застосовується функція Reduce, яка об’єднує значення, пов’язані з цим ключем, для отримання кінцевого результату. У випадку підрахунку слів функція Reduce підсумовує всі значення для кожного слова, щоб отримати загальну кількість його появ у тексті. В результаті кожне слово матиме кінцевий підрахунок, який показує, скільки разів воно зустрічається в тексті (рис.1.5) [5, 6].

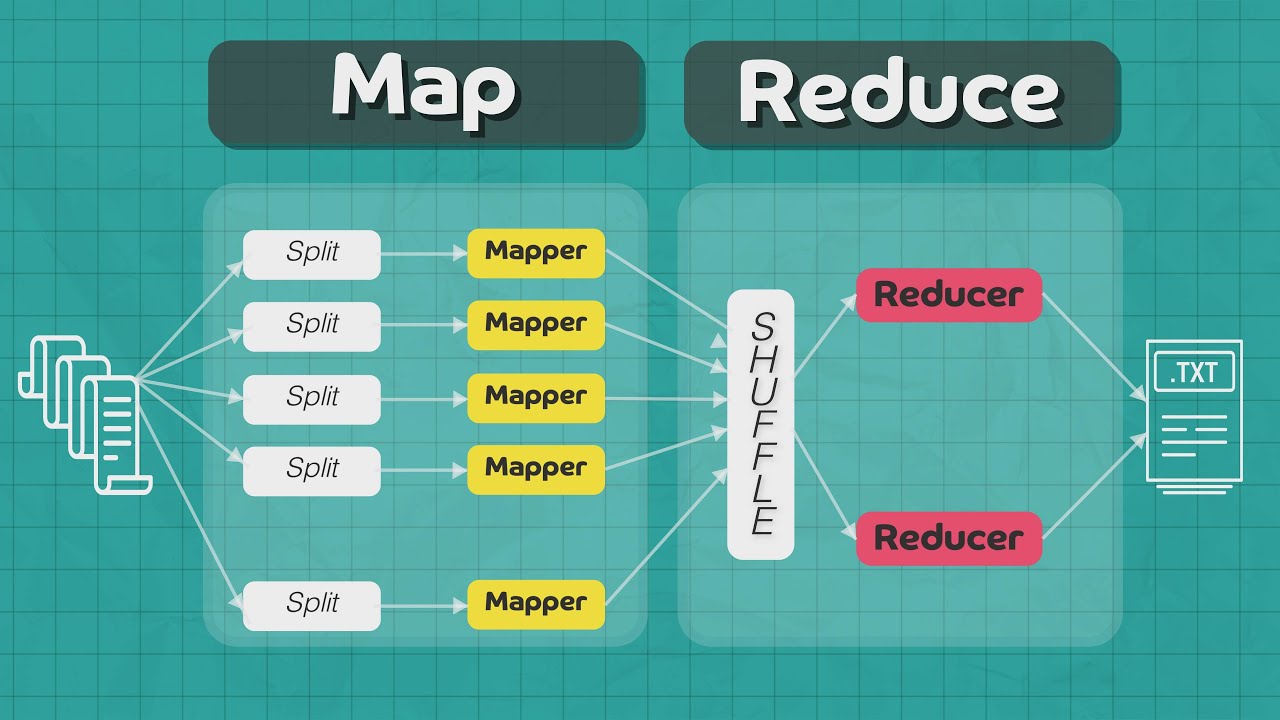


Рисунок 1.5 – Патерн MapReduce

Основною перевагою патерну MapReduce є його здатність масштабувати обробку даних на кластери з багатьма вузлами. Завдяки паралельній обробці, MapReduce дозволяє ефективно працювати з терабайтами і петабайтами даних, роблячи цей підхід популярним для великих аналітичних задач, обробки журналів подій, машинного навчання та інших завдань, пов’язаних з аналізом великих обсягів даних.

Патерн MapReduce є одним із ключових компонентів багатьох сучасних систем обробки даних, таких як Hadoop та Apache Spark, які використовують цей підхід для розподіленої обробки даних, забезпечуючи гнучкість і високу продуктивність при роботі з великими обсягами інформації.

Отже, архітектурні патерни є ключовими елементами у забезпеченні масштабованості, високої продуктивності та надійності NoSQL баз даних. Кожен патерн відповідає певним вимогам до обробки даних і забезпечує ефективність для специфічних типів додатків, таких як аналіз великих даних, управління транзакціями або забезпечення високої доступності в розподілених системах.

**Висновки до розділу 1**

У сучасних умовах швидкого зростання обсягів даних, що потребують ефективного та гнучкого зберігання і обробки, NoSQL-бази даних стали важливим інструментом, який пропонує альтернативу традиційним реляційним базам. Розглянуті теоретичні аспекти NoSQL-рішень показали, що такі системи є більш масштабованими та адаптованими до розподілених середовищ, дозволяючи обробляти великі потоки інформації та працювати з різними типами даних.

Класифікація NoSQL-баз за моделями даних — документо-орієнтовані, графові, колоночні та key-value сховища — відображає їхню здатність вирішувати різноманітні задачі в залежності від структури та особливостей даних, які необхідно зберігати. Кожен тип NoSQL баз даних має свої сильні сторони і підходить для конкретних випадків використання, таких як обробка текстів, графових структур, аналітики великих даних або зберігання великих обсягів неструктурованої інформації.

Загалом, дослідження теоретичних основ та класифікації NoSQL-баз даних дозволяє зробити висновок, що ці технології є необхідними для розв’язання задач сучасного світу даних, особливо коли йдеться про масштабованість, продуктивність, гнучкість та відмовостійкість.

**РОЗДІЛ 2**

**АНАЛІЗ NOSQL-РІШЕНЬ ДЛЯ РОБОТИ З ВЕЛИКИМИ ДАНИМИ**

**2.1 Особливості великих даних та організації їх сховищ**

Поява великих даних пов’язані з розширенням джерел інформації у світі.

Великі дані, за визначенням, запропонованим у 2011 р. Мервом Адріаном з компанії Gartner – це дані, збирання, керування та обробка яких неможливо здійснити за допомогою найчастіше використовуваних апаратних середовищ і програмних інструментів протягом допустимого для користувача часу. Словосполучення «великі дані» має на увазі не лише їхній обсяг. Згідно з Gartner Group слово «великі» – це не тільки збільшений обсяг, але і швидкість передачі, що зросла, і різноманітність джерел даних. Таким чином, доводиться мати справу не просто з великою кількістю даних, а з тим, що вони надходять дуже швидко, у складних формах та з різноманітних джерел.

Великим даним властиві такі особливості [2]:

* вони часто автоматично генеруються машиною без участі людини (так, вбудований у двигун датчик генерує дані, навіть якщо ніхто його про це не просить);
* великі дані зазвичай співвідносяться з новими джерелами даних;
* дані є структурованими, неструктурованими, напівструктурованими або навіть мультиструктурованими. Великі дані часто описуються як неструктуровані, а традиційні дані – як структуровані, тобто представлені в чітко визначеному, незмінному форматі, що полегшує роботу із ними.

Великі дані зазвичай описуються за допомогою моделі «5V», яка включає п’ять ключових характеристик:

1. Обсяг (Volume). Великі дані характеризуються значним обсягом, який постійно зростає. Це можуть бути терабайти або навіть петабайти інформації, що надходять з численних джерел, таких як соціальні мережі, сенсори, мобільні пристрої, інтернет та інші джерела.
2. Швидкість (Velocity). Великі дані генеруються з великою швидкістю, і в багатьох випадках необхідна їхня обробка майже в режимі реального часу. Наприклад, соціальні мережі постійно оновлюються великою кількістю публікацій та повідомлень, а датчики в інтернеті речей (IoT) передають інформацію безперервно.
3. Різноманіття (Variety). Дані мають різні формати: структуровані, неструктуровані та напівструктуровані. Сюди відносяться тексти, зображення, відео, логи, показники сенсорів та інші формати, кожен з яких має свої специфічні вимоги до обробки.
4. Достовірність (Veracity). Одна з проблем великих даних — це їх достовірність або точність. У великих масивах даних часто містяться похибки, неповна або невірна інформація, тому важливо забезпечити високу якість обробки і фільтрації даних.
5. Цінність (Value). Останньою, але надзвичайно важливою характеристикою великих даних є їхня цінність. Дані самі по собі не мають значення, якщо вони не можуть бути використані для отримання корисної інформації та прийняття обґрунтованих рішень.

Організація сховищ для Big Data – це процес побудови та управління інфраструктурою, яка дозволяє ефективно зберігати, обробляти і аналізувати великі обсяги різнорідної інформації. Сховища для великих даних повинні бути гнучкими, масштабованими, забезпечувати швидкий доступ до інформації, а також підтримувати високий рівень безпеки та надійності.

Сховища великих даних мають відповідати кільком ключовим вимогам [7]:

1. Масштабованість. Системи повинні легко розширюватись, щоб мати змогу зберігати зростаючі обсяги даних. Для цього використовуються розподілені системи зберігання, де дані розміщуються на багатьох серверах або вузлах, що дозволяє додавати нові сервери без зупинки роботи системи.
2. Продуктивність. У світі великих даних продуктивність є критичним аспектом, оскільки дані повинні оброблятися з високою швидкістю, особливо для обробки в реальному часі або при використанні для машинного навчання. Сховища повинні забезпечувати швидкий доступ до інформації для її аналізу та прийняття рішень.
3. Надійність та відмовостійкість. Великі обсяги даних потребують високої надійності. Розподілені системи зберігання, такі як Hadoop HDFS, використовують реплікацію даних — створення копій даних на різних вузлах, щоб забезпечити їх збереження навіть у разі відмови обладнання.
4. Різноманітність підтримуваних форматів даних. Великі дані є різнорідними за своїм складом (структуровані, напівструктуровані, неструктуровані), тому сховища мають підтримувати зберігання та обробку даних різних типів. Це дозволяє працювати з текстами, зображеннями, відео, таблицями та іншими форматами.

Для побудови сховищ великих даних використовуються кілька підходів, зокрема Data Lake, Data Warehouse та Data Mart, кожен з яких має свої специфічні переваги.

Data Lake або озеро даних – це сучасне сховище, яке дозволяє зберігати дані в їхньому первісному вигляді. Це рішення забезпечує організаціям можливість накопичувати великі обсяги різнорідної інформації, включаючи структуровані, неструктуровані та напівструктуровані дані, такі, як логи, аудіо, відео, текстові документи, дані з Інтернету речей (IoT). Основна перевага Data Lake полягає в тому, що дані можуть зберігатися у сирому вигляді без попереднього очищення або трансформації, що дозволяє легко повертатися до вихідної інформації для детального аналізу [2, 9].

Data Lake забезпечує зберігання сирих даних, даючи можливість зберігати будь-який тип даних у їхній оригінальній формі, що особливо цінно для роботи з неструктурованими даними. Це сховище є надзвичайно масштабованим: воно використовує хмарні платформи або розподілені обчислювальні системи, що дозволяє організаціям динамічно збільшувати обсяг збереження та обробки. Data Lake підтримує різні типи даних, що дозволяє зберігати таблиці баз даних, документи, графи, текстові файли, зображення, відео, аудіо, що забезпечує високу гнучкість для організацій, які мають доступ до різноманітних джерел інформації. Важливим аспектом є те, що Data Lake надає можливість аналітикам, інженерам даних, фахівцям з машинного навчання звертатися до сирих даних без їхньої попередньої обробки. Завдяки недорогому зберіганню, доступному на хмарних платформах, організації можуть зберігати великі обсяги даних без значних витрат [9].

Архітектура Data Lake складається з різних компонентів, зокрема джерел даних, що включають транзакційні бази, IoT-сенсори, журнали систем, соціальні мережі, які накопичують інформацію в розподіленому сховищі, здатному підтримувати різні формати, як-от Parquet, ORC, Avro, JSON. Управління метаданими допомагає відслідковувати інформацію про кожен файл, його тип, структуру та походження. Завдяки розвиненим засобам безпеки Data Lake надійно захищає інформацію від несанкціонованого доступу, використовуючи контроль доступу, шифрування та автентифікацію. Для обробки та аналізу Data Lake підтримує інструменти, такі як Apache Spark, Hadoop, Presto, що забезпечує трансформацію даних, аналіз і побудову моделей машинного навчання.

Серед переваг Data Lake – гнучкість у роботі з даними, що дозволяє зберігати інформацію у її оригінальному вигляді, особливо неструктуровані дані. Масштабованість розподілених хмарних сховищ забезпечує динамічне зростання обсягу збережених даних без великих витрат. Data Lake дозволяє проводити аналітику в реальному часі завдяки потужним інструментам обробки, таким як Apache Spark чи Hadoop, що є корисним для організацій, які потребують миттєвого аналізу великих обсягів інформації.

Data Warehouse або сховище даних – це централізована система, яка зберігає структуровані дані з різних джерел для підтримки аналітики та прийняття рішень у бізнесі. Сховище даних призначене для інтеграції інформації з різнорідних систем, таких як транзакційні бази даних, ERP-системи, CRM, і зберігає її у формі, оптимізованій для запитів і звітності. На відміну від оперативних баз даних, які підтримують поточні транзакції, Data Warehouse фокусується на історичних даних і побудові агрегованих звітів, забезпечуючи цілісність та послідовність інформації [9].

Однією з ключових особливостей Data Warehouse є процес ETL (Extract, Transform, Load), що включає витяг даних з джерел, їх трансформацію для забезпечення уніфікації та завантаження у сховище даних. Це гарантує, що інформація, яка надходить у сховище, є очищеною, перевіреною на відповідність та підготовленою для подальшого аналізу. Процес ETL забезпечує стандартизацію різних форматів даних, що полегшує їх використання для аналізу та звітності.

Архітектура Data Warehouse зазвичай складається з декількох рівнів. Джерела даних надають інформацію з різних операційних систем, яка обробляється в ETL-процесі і завантажується в центральне сховище. Інформація у Data Warehouse організована за темами (наприклад, продажі, фінанси, маркетинг), що дозволяє користувачам з різних відділів швидко отримувати доступ до релевантних даних. Дані в сховищі зазвичай представлені у вигляді таблиць з попередньо визначеними структурами, що робить їх легкими для використання у бізнес-аналізі.

Data Warehouse має багато переваг для організацій, особливо у бізнесі, де аналіз інформації є важливою складовою для прийняття стратегічних рішень. Сховище даних дозволяє інтегрувати дані з різних джерел і представляти їх у зручному для аналізу вигляді, надаючи єдине джерело правди для організації. Завдяки збереженню історичної інформації Data Warehouse підтримує трендовий аналіз, дозволяючи відстежувати динаміку та порівнювати показники у часі. Для забезпечення швидкого доступу до даних, сховища даних часто оптимізуються під запити за допомогою індексів, агрегацій і кешування.

Data Warehouse забезпечує високу точність і консистентність інформації. Доступ до сховища даних відбувається через інтерфейси звітності та бізнес-аналітики, як-от BI-інструменти (Power BI, Tableau), які дозволяють користувачам без технічної підготовки швидко отримувати звіти та проводити аналіз. Одним із важливих аспектів є безпека – Data Warehouse зазвичай має строгі механізми доступу та аудиту, що дозволяє контролювати, хто і до яких даних має доступ.

Data Mart (сегментоване сховище) – це підхід, коли дані зберігаються в сегментованих сховищах для конкретних підрозділів або потреб. Data Mart дозволяє організувати окремі секції даних, орієнтовані на специфічні завдання. Це часто знижує навантаження на центральне сховище, дозволяючи швидше отримувати потрібні дані для конкретних цілей [9].

Data Mart – це підмножина даних, створена для конкретного бізнес-підрозділу або функціональної сфери в межах організації, як-от продажі, фінанси, маркетинг. Це спрощене сховище даних, яке забезпечує швидкий доступ до інформації для виконання вузькоспеціалізованих завдань або аналізу. Data Mart зазвичай створюють на основі даних, які вже зберігаються у великому сховищі даних, або з окремих джерел даних, що забезпечує швидкий доступ до релевантної інформації для конкретного відділу чи групи користувачів.

Основна мета Data Mart – забезпечити ефективний доступ до важливої інформації для певного типу користувачів без необхідності звертатися до всього обсягу даних, що зберігається у Data Warehouse. Наприклад, відділ продажів може використовувати Data Mart для аналізу показників продажів, даних про клієнтів, регіональних звітів, тоді як відділ маркетингу може мати свій Data Mart з даними про кампанії, цільові аудиторії та поведінкову аналітику. Це не лише скорочує час доступу до потрібної інформації, але й зменшує навантаження на основне сховище даних.

Data Mart зазвичай має менш складну структуру і менші обсяги даних порівняно з Data Warehouse, що дає змогу виконувати запити швидше та забезпечувати більш гнучкий доступ до інформації. Архітектура Data Mart може бути незалежною, коли дані надходять безпосередньо з джерел, або залежною, коли вони витягуються з централізованого Data Warehouse. Існує також гібридний підхід, який поєднує обидва методи.

Data Mart надає багато переваг, серед яких спрощення доступу до даних, зменшення часу на виконання запитів і поліпшення продуктивності завдяки обробці невеликого обсягу даних. Такого типу сховища також забезпечують користувачам більш цілеспрямовану аналітику, адаптовану під їхні конкретні потреби. Data Mart допомагає зберігати дані, які стосуються виключно певного напряму діяльності компанії, що зменшує обсяг оброблюваних даних і підвищує швидкість аналітичних операцій.

Разом з тим, Data Mart має певні обмеження, зокрема ризик виникнення несумісності між різними Data Mart, якщо вони не інтегровані з основним сховищем даних. Крім того, дані у Data Mart можуть застарівати, якщо процеси оновлення не налагоджені, і виникає потреба у додатковому обслуговуванні, щоб забезпечити синхронізацію з іншими системами та актуальність інформації.

**2.2 Використання NoSQL баз даних для обробки великих даних**

Огляд особливостей великих даних показує, що для їх обробки необхідно застосовувати засоби, які забезпечують високу швидкість обробки, гнучкість і масштабованість. Саме тому NoSQL бази даних здобули велику популярність у сценаріях, пов’язаних з великими даними.

NoSQL бази даних забезпечують гнучкість і можливість обробки різноманітних структур даних, які зберігаються у вигляді документів, графів, ключ-значення або колонок. Відсутність фіксованої схеми дозволяє зберігати дані, що швидко змінюються, а також підтримувати складні, неоднорідні структури. Це дозволяє NoSQL базам ефективно обробляти великі обсяги даних, які не завжди підходять для традиційної реляційної моделі.

NoSQL бази часто забезпечують горизонтальне масштабування, що означає додавання нових серверів у кластер замість вертикального масштабування (збільшення потужності одного сервера). Такий підхід дає змогу обробляти великі обсяги даних, розподіляючи їх на декілька вузлів. Це важлива особливість для великих даних, де обробка має відбуватися швидко і безперервно.

NoSQL бази даних є ключовими інструментами для вирішення задач великих даних, серед яких [10]:

1. Зберігання та обробка неструктурованих та напівструктурованих даних. NoSQL бази легко зберігають дані у вигляді JSON документів, графів або колонок, що дозволяє зберігати інформацію з різних джерел – соціальних мереж, логів, даних IoT тощо.
2. Обробка потоків даних у реальному часі. NoSQL бази підтримують горизонтальне масштабування та реплікацію, що робить їх ідеальними для обробки даних у режимі реального часу, наприклад, у моніторингових системах або e-commerce аналітиці.
3. Машинне навчання і аналітика. NoSQL бази даних використовуються для зберігання великих обсягів даних для подальшої обробки у машинному навчанні та аналітичних алгоритмах. Документо-орієнтовані і колонкові NoSQL бази забезпечують ефективний доступ до даних для обчислення трендів, прогнозів та кластеризації.
4. Персоналізація і рекомендаційні системи. В рекомендаційних системах графові та документо-орієнтовані NoSQL бази дозволяють зберігати дані про взаємодію користувачів, створюючи на їх основі персоналізовані пропозиції, що підвищує залученість користувачів.

Використання NoSQL баз даних для великих даних надає численні переваги. По-перше, це здатність до горизонтального масштабування, що дозволяє обробляти значні обсяги даних без потреби в дорогих і потужних серверах. По-друге, NoSQL бази забезпечують високу гнучкість, яка дає змогу обробляти як структуровані, так і неструктуровані дані. Це особливо корисно для роботи з даними з різних джерел, які часто не мають єдиної структури.

Крім того, NoSQL бази дозволяють швидко реагувати на зміни в даних, що є критичним у випадках реальної аналітики та оперативної обробки інформації. Це стає можливим завдяки механізмам реплікації та збереження даних у кількох копіях, які забезпечують доступність даних і захист від втрати інформації.

Хоча NoSQL бази мають значні переваги для великих даних, існують і певні виклики. Наприклад, NoSQL бази часто не підтримують транзакції на тому ж рівні, що й реляційні системи, що може створювати труднощі у критично важливих фінансових додатках. Крім того, хоча NoSQL бази відмінно підходять для горизонтального масштабування, керування великими кластерами може бути складним завданням [6].

Забезпечення узгодженості даних також є викликом, особливо у розподілених системах, де використовується модель eventual consistency (узгодженість у кінцевому підсумку). Незважаючи на ці виклики, індустрія активно розробляє нові підходи та архітектури для покращення можливостей NoSQL баз у роботі з великими даними.

**2.3 Аналіз кейсів використання NoSQL для Big Data**

Бази даних NoSQL можна використовувати для різноманітних додатків, але є кілька поширених випадків їх використання для аналізу великих даних:

1. Програми електронної комерції

Бази даних NoSQL можуть допомогти компаніям електронної комерції керувати великими обсягами даних, включаючи каталоги продуктів, профілі клієнтів та історію транзакцій. Програми електронної комерції вимагають здатності керувати великим обсягом даних, включаючи каталоги продуктів, профілі клієнтів та історію транзакцій. Ці дані часто є неструктурованими або напівструктурованими, що ускладнює керування традиційними реляційними базами даних. Бази даних NoSQL розроблені для обробки цих типів даних і забезпечують необхідну масштабованість і продуктивність для підтримки програм електронної комерції з високим трафіком [3].

У програмі електронної комерції база даних NoSQL може зберігати дані про продукт, включаючи опис продукту, ціни, зображення та наявність. Бази даних NoSQL можуть легко обробляти великі каталоги продуктів, що полегшує клієнтам пошук і пошук потрібних їм продуктів.

Дані клієнтів є ще одним важливим компонентом програми електронної комерції. Бази даних NoSQL можуть зберігати профілі клієнтів, включаючи імена, адреси, історію покупок і вподобання. За допомогою баз даних NoSQL компанії електронної комерції можуть створювати персоналізований досвід для своїх клієнтів, надаючи цільові рекомендації щодо продуктів і персоналізовані пропозиції на основі їх історії покупок і вподобань.

Нарешті, бази даних NoSQL також можуть зберігати історію транзакцій, забезпечуючи повний запис усіх транзакцій, які відбулися в програмі електронної комерції. Ці дані можна використовувати для звітності.

2. Платформи соціальних мереж

Платформи соціальних медіа генерують величезну кількість неструктурованих даних у формі публікацій, коментарів, лайків, спільних публікацій і профілів користувачів.

Однією з головних переваг баз даних NoSQL для платформ соціальних медіа є їх здатність зберігати та обробляти неструктуровані дані у великому масштабі. Платформи соціальних мереж щодня генерують величезну кількість неструктурованих даних, і бази даних NoSQL можуть обробляти ці дані ефективно та результативно. Завдяки базам даних NoSQL платформи соціальних мереж можуть зберігати дописи, коментарі, оцінки «подобається» та публікації, а також швидко отримувати ці дані за потреби [3, 22].

3. Інтернет речей (IoT)

Інтернет речей (IoT) – це мережа підключених пристроїв, які генерують величезні обсяги даних із датчиків, камер та інших джерел. Ці дані часто є неструктурованими або напівструктурованими. Однією з головних переваг баз даних NoSQL для додатків IoT є їх здатність обробляти великі обсяги даних у режимі реального часу. Пристрої IoT безперервно генерують дані, а бази даних NoSQL можуть швидко й ефективно зберігати й обробляти ці дані. Завдяки базам даних NoSQL програми IoT можуть збирати й аналізувати дані з мільйонів пристроїв у режимі реального часу, надаючи цінну інформацію про поведінку користувачів, продуктивність і потреби в обслуговуванні [22].

Бази даних NoSQL також мають високу масштабованість, що важливо для додатків Інтернету речей, яким може знадобитися десятки тисяч або навіть мільйони пристроїв. Оскільки кількість пристроїв і обсяг даних зростає, бази даних NoSQL можуть масштабуватися горизонтально, щоб відповідати додатковому навантаженню. Ця масштабованість має вирішальне значення для додатків Інтернету речей, яким потрібно швидко й ефективно обробляти великі обсяги даних.

4. Мобільні додатки

Мобільні програми є частиною сучасного життя, мільярди користувачів у всьому світі генерують величезні обсяги даних. Бази даних NoSQL добре підходять для обробки даних, створених мобільними програмами, включаючи профілі користувачів, дані про місцезнаходження та статистику використання програми. Завдяки базам даних NoSQL мобільні програми можуть забезпечити швидкий і надійний доступ до даних у розподіленій мережі.

5. Ігри

Ігрові компанії створюють величезну кількість даних, від даних гравців до ігрових станів, рекордів тощо. Використовуючи бази даних NoSQL ігрові компанії можуть легко зберігати та керувати великими обсягами даних гравців. Крім того, бази даних NoSQL добре підходять для обробки великих обсягів трафіку, що робить їх чудовим вибором для ігрових додатків, які відчувають різкий попит [22].

6. Аналітика великих даних

Аналітика великих даних стала найважливішим інструментом для компаній, які прагнуть отримати уявлення про свою діяльність, клієнтів і ринки. Компанії створюють велику кількість напівструктурованих різнопланових даних, які потрібно обробляти. Використання баз даних NoSQL дають можливість обробляти такі масиви даних, забезпечуючи швидкий і ефективний доступ до них [3].

Бази даних NoSQL також пропонують гнучкі моделі даних, що важливо для аналітики великих даних. Ця гнучкість означає, що компанії можуть зберігати та аналізувати дані у спосіб, який найкраще відповідає їхнім потребам.

Розглянемо приклади застосування баз даних NoSQL в реальних рішеннях.

У системі WindyGrid [3], призначеній для моніторингу міста Чикаго, використовується нереляційна база даних MongoDB, яка зберігає великий обсяг неструктурованих даних. Платформа обробляє мільйони записів, що надходять щодня з 15 ключових департаментів Чикаго, включаючи поліцію, транспорт, пожежну службу, інформацію про дорожні роботи, затримки вивезення сміття, а також дані щодо дзвінків на 911. Крім того, збираються публічні твіти про діяльність міських служб, дані про місцезнаходження автобусів на маршрутах, кольори світлофорів у реальному часі та багато іншої інформації. MongoDB також використовується у хмарі Google Cloud як рішення для швидкого розгортання проєктів розробниками.

Компанія PayPal використовує базу даних NoSQL Couchbase для отримання інформації про користувачів, їх транзакції. Для цього обробляється мільйони оновлень на хвилину. В системі використовуються технології Kafka та Hadoop. Couchbase як надбудови над AWS (Amazon Web Services).

База даних Couchbase використовується іншими компаніями. eBay використовує її для створення списків товарів в реальному часів, а соціальна мережа для фахівців LinkedIn здійснює за допомогою даної бази даних кешування інформації.

База даних Redis використовується Github для маршрутизації інформації під час виконання HTTP запитів на отримання інформації із серверів. За допомогою цієї бази даних мережа Х здійснює кешування твітів. Таким же чином використовує цю базу даних StackOverflow.

еВay використовує графову базу даних Neo4j для надання покупцям рекомендацій. За допомогою даної бази даних фірма змогла вибудувати еволюційну систему, яка з кожним уточненням контексту намагається підібрати оптимальну рекомендацію.

Отже, загалом, бази даних NoSQL зараз вже досить розповсюджені в різних областях економіки, розширюючи їх можливості з обробки та зберігання даних.

**Висновки до розділу 2**

NoSQL бази даних стали важливим інструментом для обробки великих даних завдяки своїй гнучкості, можливості горизонтального масштабування та здатності працювати з різними типами даних. Різні типи NoSQL баз даних дозволяють вирішувати широкий спектр задач, пов’язаних з великими даними, від обробки неструктурованих даних до побудови складних аналітичних і рекомендаційних систем.

Хоча існують певні обмеження і виклики, NoSQL бази є перспективним рішенням для сучасних потреб великих даних, і їх використання у поєднанні з іншими технологіями продовжує зростати.

**РОЗДІЛ 3**

**РОЗРОБКА ІНТЕРФЕЙСУ ДЛЯ ОБРОБКИ ДАНИХ НА ОСНОВІ PySpark ТА NoSQL**

**3.1 Огляд архітектури та вибір технологій для розробки**

Розглянемо використання NoSQL баз даних для рішення економічних завдань. Розглянемо задачу побудови рекомендаційної системи на основі покупок та відгуків користувачів.

Для проєкту з аналізу даних електронної комерції та побудови рекомендаційної системи було обрано дві основні технології – MongoDB та PySpark. Кожна з них має особливості, які забезпечують гнучкість, масштабованість та високу продуктивність при роботі з великими обсягами даних, що є ключовими вимогами для цього проєкту.

Вибір – MongoDB для зберігання даних.

MongoDB – це популярна документно-орієнтована NoSQL СУБД, яка зберігає дані у вигляді документів JSON-подібної структури. Однією з основних причин вибору MongoDB є її здатність обробляти неструктуровані та напівструктуровані дані, які часто використовуються у сфері електронної комерції. Наприклад, такі дані містять інформацію про товари, користувачів, відгуки, замовлення, і вони мають різну структуру та обсяги, що можуть швидко змінюватися. MongoDB дозволяє зберігати ці дані гнучко, без потреби в попередньому визначенні чіткої схеми, надаючи можливість ефективного зберігання та доступу до цих даних [13].

MongoDB також пропонує розподілену архітектуру, яка забезпечує горизонтальне масштабування, реплікацію та надійність. Для проєктів, що працюють з великими обсягами даних, важливо мати можливість швидко збільшити потужність бази даних, додаючи нові вузли в кластері. Реплікація в MongoDB гарантує збереження даних навіть у випадку збоїв, що критично важливо для великих даних, зокрема для забезпечення стабільності аналітичних процесів.

PySpark – це інтерфейс для роботи з Apache Spark у середовищі Python, що дозволяє виконувати складні обчислення над великими обсягами даних. Spark є фреймворком для розподіленої обробки даних, який підтримує обробку на кластерах, тому його інтеграція з MongoDB ідеально підходить для аналітичних проєктів, де важлива швидкість і продуктивність обчислень. Spark підтримує моделі обробки, які базуються на концепціях RDD (Resilient Distributed Dataset) та DataFrame, що дозволяють ефективно працювати з великими обсягами даних, проводити їх обробку, агрегування, аналіз, а також здійснювати машинне навчання.

PySpark також має вбудовану бібліотеку MLlib для машинного навчання, що дозволяє реалізувати різні моделі для створення рекомендаційних систем. У даному проєкті PySpark використовується для побудови колаборативної фільтрації, що дозволяє надавати персоналізовані рекомендації для користувачів. Завдяки можливостям PySpark масштабувати обчислення на декілька вузлів, обробка даних може виконуватися швидше та ефективніше, що забезпечує продуктивність навіть для значних наборів даних [21].

У запропонованому проєкті архітектура передбачає, що MongoDB виступає основним джерелом даних, в якому зберігаються результати взаємодії користувачів з товарами, а саме, замовлення та відгуки. Вибір MongoDB обумовлений тим, що відгук користувача може містити доволі довгий неструктурований текст з різними позначками та символами. Кількість взаємодій з кожним днем зростає. Оскільки передбачається, що база даних буде зберігати історичні дані, очевидним є вибір інструментів, які забезпечать масштабованість та гнучкість.

Використання PySpark забезпечить зручний та надійний інструмент для глибокого аналізу великих даних. Його можливість взаємодіяти з даними різних форматів є також ключовою при аналізі різних інструментів обробленні даних.

Отже, загалом, обрана архітектура підтримує гнучку обробку та аналіз великих обсягів даних у розподіленому середовищі, де MongoDB відповідає за стабільне та ефективне зберігання даних, а PySpark – за розподілену обробку, аналіз і побудову моделі обробки.

Таким чином, комбінація MongoDB і PySpark дозволяє ефективно обробляти великі дані, забезпечуючи гнучкість, продуктивність та надійність, необхідні для досягнення цілей цього проєкту. Цей технологічний стек також відкриває можливості для майбутнього масштабування та легкої інтеграції нових аналітичних інструментів, що робить його оптимальним рішенням для роботи з великими даними в контексті аналізу даних електронної комерції.

**3.2 Розробка бази даних NOSQL для MongoDB**

MongoDB представляє найбільш популярну на даний момент документо-орієнтовану систему управління базами даних. MongoDB зберігає документи, на відміну від рядків, документи можуть зберігати складну за структурою інформацію. Ключ представляє просту мітку, з якою асоційовано певний шматок даних.

Однак при всіх відмінностях є одна особливість, яка зближує MongoDB і реляційні бази даних. У реляційних СУБД зустрічається таке поняття як первинний ключ. І якщо явно не вказати його значення, то MongoDB автоматично згенерує для нього значення.

Вся система MongoDB може представляти не тільки одну базу даних, яка розташовується на одному фізичному сервері. Функціональність MongoDB дозволяє розташувати декілька баз даних на кількох фізичних серверах, і ці бази даних можуть легко обмінюватися даними та зберігати цілісність.

Система зберігання даних у MongoDB представляє набір реплік. У цьому наборі є основний вузол, а також може бути набір вторинних вузлів. Якщо основний вузол стає недоступним, то один із вторинних вузлів стає головним через механізм автоматичного вибору нового головного вузла, забезпечуючи безперервність роботи системи.

Одним з популярних стандартів обміну даними та їх зберігання є JSON (JavaScript Object Notation). який називається BSON (бісон) або скорочення від binary JSON.

BSON дозволяє працювати з даними швидше: швидше виконується пошук та обробка. Хоча треба відзначити, що BSON на відміну від зберігання даних у форматі JSON має невеликий недолік: загалом дані у JSON-форматі займають менше місця, ніж у форматі BSON, з іншого боку, цей недолік з лишком окупається швидкістю [13].

Модель організації бази даних MongoDB можна представити таким чином (рис.3.1).

Рисунок 3.1 – Організація БД в MongoDB

В MongoDB база даних складається з колекцій. Кожна колекція має своє унікальне ім’я – довільний ідентифікатор, що складається з не більше ніж 128 різних алфавітно-цифрових символів та підкреслення.

На відміну від реляційних баз даних MongoDB не використовує табличну організацію із чітко заданою кількістю стовпців та типів даних. MongoDB є документоорієнтованою системою, в якій центральним поняттям є документ.

Документ можна подати як об’єкт, який зберігає певну інформацію. Він представляє набір пар ключ-значення. Наприклад, у виразі «name»: «Yolanda Owen» «name» представляє ключ, а Yolanda Owen» – значення (hbc/3/2).

Ключі є рядками. Значення можуть відрізнятися за типом даних.

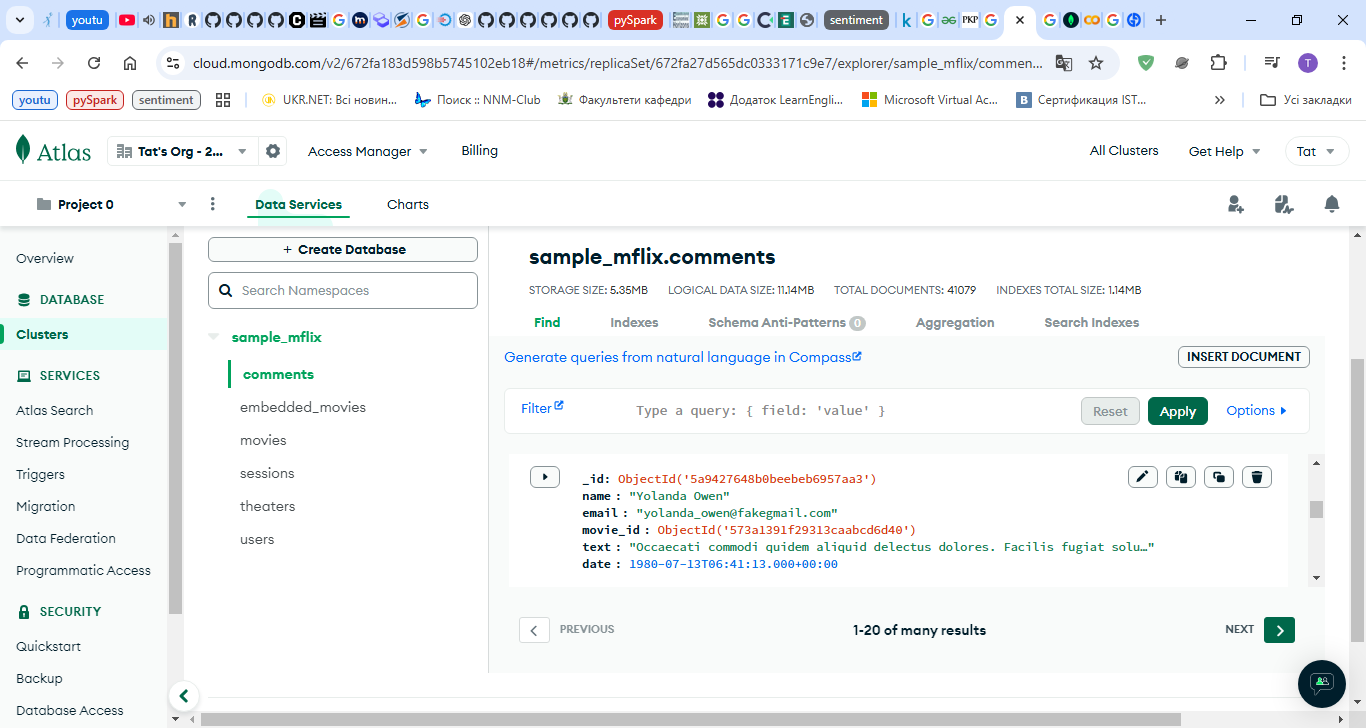


Рисунок 3.2 – Приклад фрагмента колекції

Для кожного документа MongoDB визначено унікальний ідентифікатор, який називається \_id. При додаванні документа до колекції цей ідентифікатор створюється автоматично. Однак розробник може сам явно задати ідентифікатор, а не покладатися на автоматично генеровані, вказавши відповідний ключ і його значення в документі.

Це поле має мати унікальне значення в межах колекції. І при спробі додати до колекції два документи з однаковим ідентифікатором буде додаватися лише один із них, а при додаванні другого користувач отримає помилку.

Якщо ідентифікатор не заданий явно, MongoDB створює спеціальне бінарне значення розміром 12 байт. Це значення складається з кількох сегментів: значення типу timestamp розміром 4 байти (що представляє кількість секунд з моменту початку епохи Unix), випадкове число з 5 байт і лічильник з 3 байт, який ініціалізований випадковим числом. Така модель побудови ідентифікатора гарантує з високою ймовірністю, що він матиме унікальне значення.

Перейдемо до представлення інформаційної моделі задачі. Інформаційна модель для рекомендаційної системи включає кілька ключових сутностей, що дозволяють організувати дані, необхідні для створення персоналізованих рекомендацій. Основними компонентами є користувачі, товари, замовлення та відгуки. Наведемо детальний опис кожної сутності:

1. Користувачі (Users). Сутність «Користувач» представляє осіб, які роблять замовлення і для яких формується система рекомендацій. Для зберігання інформації про користувачів використовуються такі атрибути:

user\_id: Унікальний ідентифікатор користувача.

name: Ім’я або нікнейм користувача.

age: Вік користувача.

email: Електронна адреса користувача.

location: місце проживання користувача.

2. Товари (Products). Сутність «Товар» описує всі продукти, доступні для купівлі та рекомендацій. Кожен товар має такі атрибути:

product\_id: Унікальний ідентифікатор товару.

name: Назва товару.

category: Категорія товару (наприклад, електроніка, одяг, книги).

price: Ціна товару.

stock: Кількість товару на складі.

3. Замовлення (Orders). Сутність «Замовлення» представляє транзакції, де відображається рол замлденні товари. Замовлення необхідні для збору даних про поведінку користувачів і створення рекомендацій. Атрибути:

order\_id: Унікальний ідентифікатор замовлення.

user\_id: Ідентифікатор користувача, який зробив замовлення (зв’язок із сутністю «Користувач»).

product\_id: Ідентифікатор товару, який був замовлений (зв’язок із сутністю «Товар»).

order\_date: Дата оформлення замовлення.

Amount: Вартість замовлення

4. Відгуки (Reviews). Сутність «Відгук» містить інформацію про оцінки та відгуки користувачів на товари. Ці дані використовуються для покращення рекомендаційної системи на основі оцінок товарів:

review\_id: Унікальний ідентифікатор відгуку.

user\_id: Ідентифікатор користувача, який залишив відгук (зв’язок із сутністю «Користувач»).

product\_id: Ідентифікатор товару, на який залишено відгук (зв’язок із сутністю «Товар»).

rating: Оцінка, що була надана товару (від 1 до 5).

comment: Текстовий відгук, залишений користувачем.

Опишемо зв’язки між сутностями.

Users - Orders: Зв’язок між користувачами і замовленнями вказує, що один користувач може зробити кілька замовлень.

Products - Orders: Зв’язок між товарами і замовленнями є один-до-багатьох, оскільки один товар може бути замовлений кількома користувачами.

Users - Reviews: Один користувач може залишити кілька відгуків на різні товари.

Products - Reviews: Один товар може отримати кілька відгуків від різних користувачів.

Перейдемо до створення бази даних, використовуючи графічний інтерфейс Compass.

Створимо базу даних Recommend, додаючи колекцію users (рис.3.3).

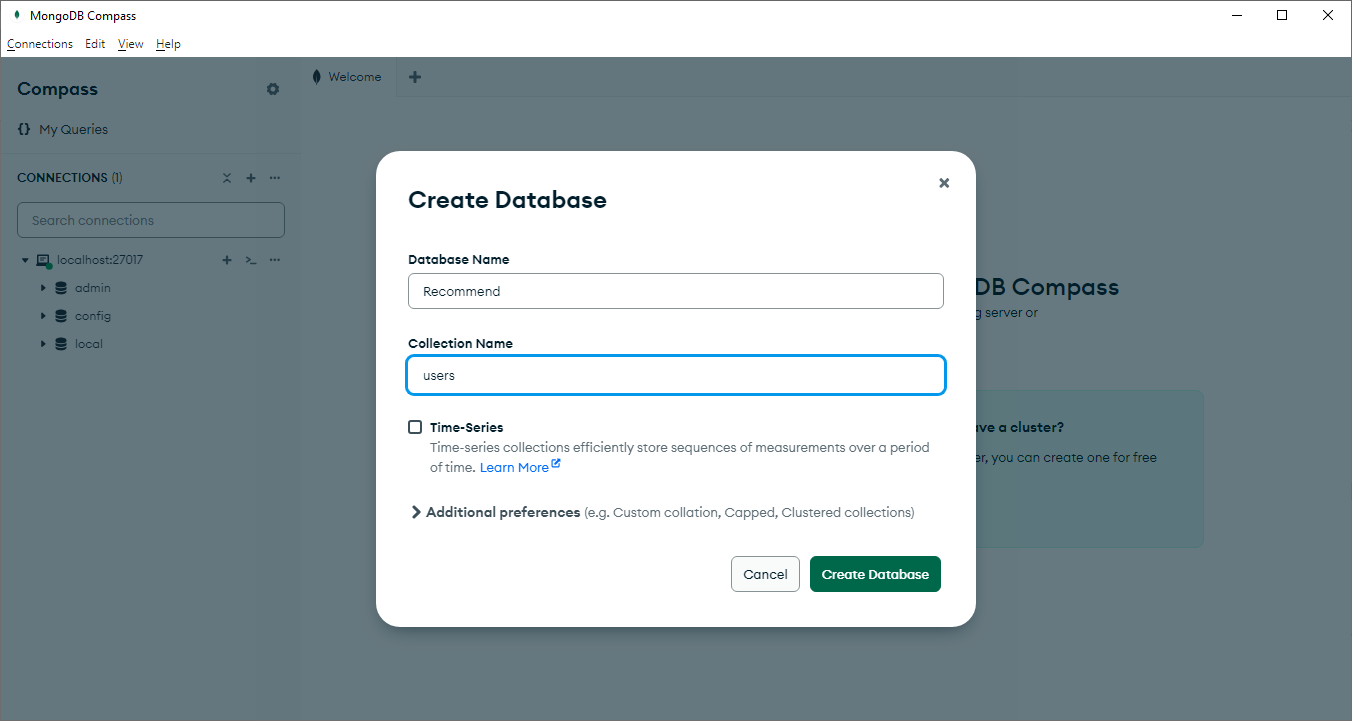


Рисунок 3.3 – Створення бази даних з колекцією users

Додамо до створеної колекції дані (рис.3.4).

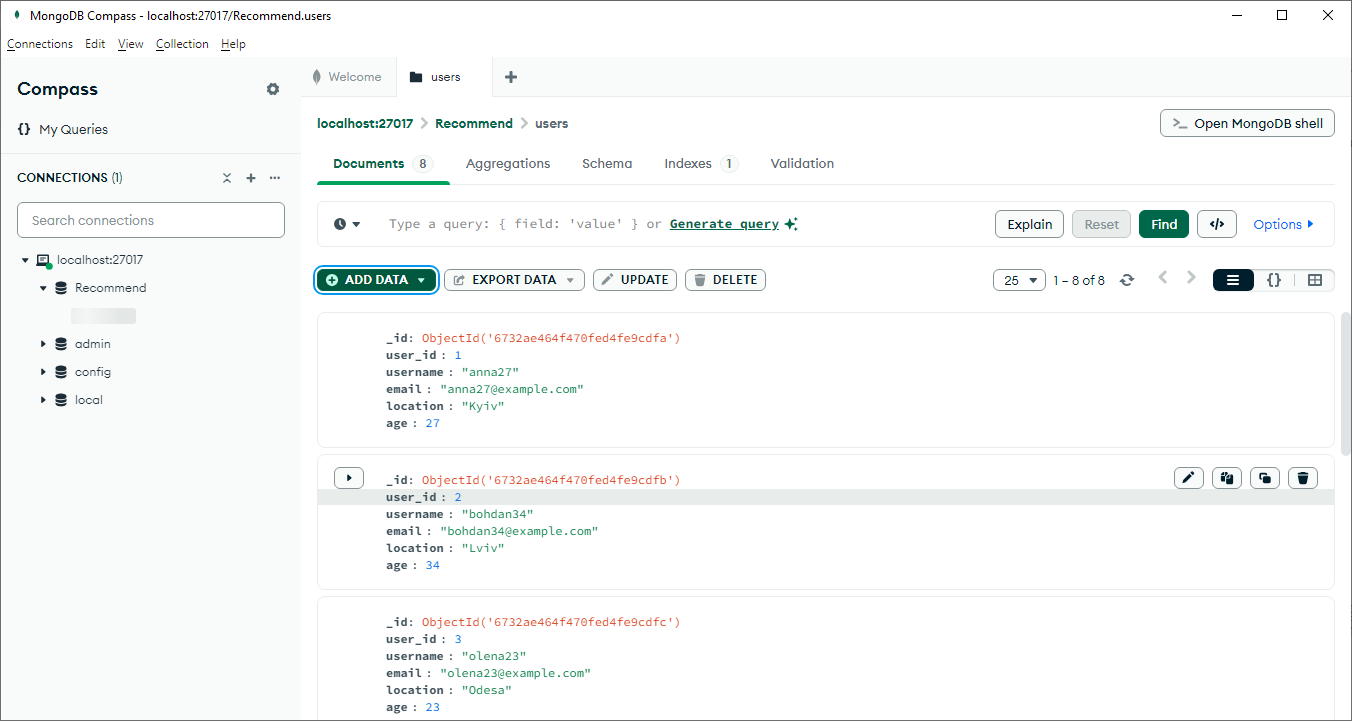


Рисунок 3.4 – Колекція users з документами

Аналогічно створимо 3 інші колекції.

Колекція products з вставленими документами представлена на рис.3.5.

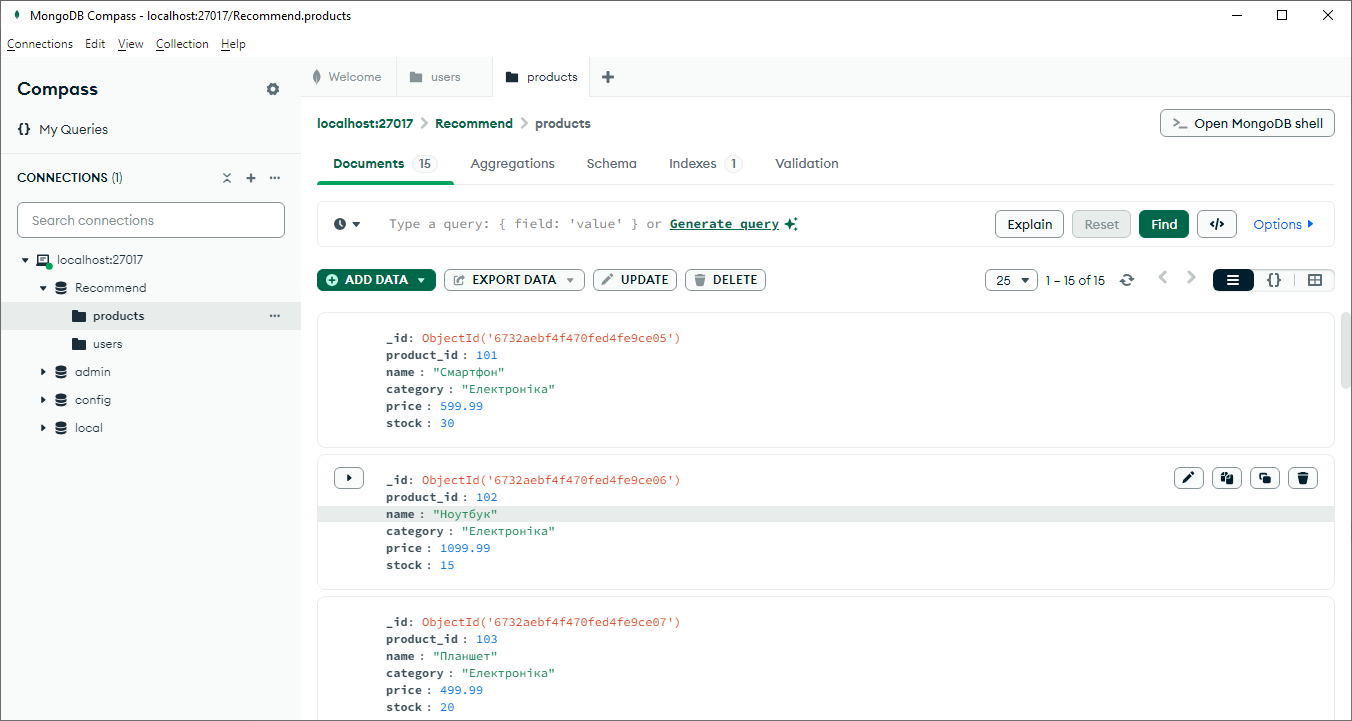


Рисунок 3.5 – Колекція products з документами

Колекція orders представлена на рис.3.6.

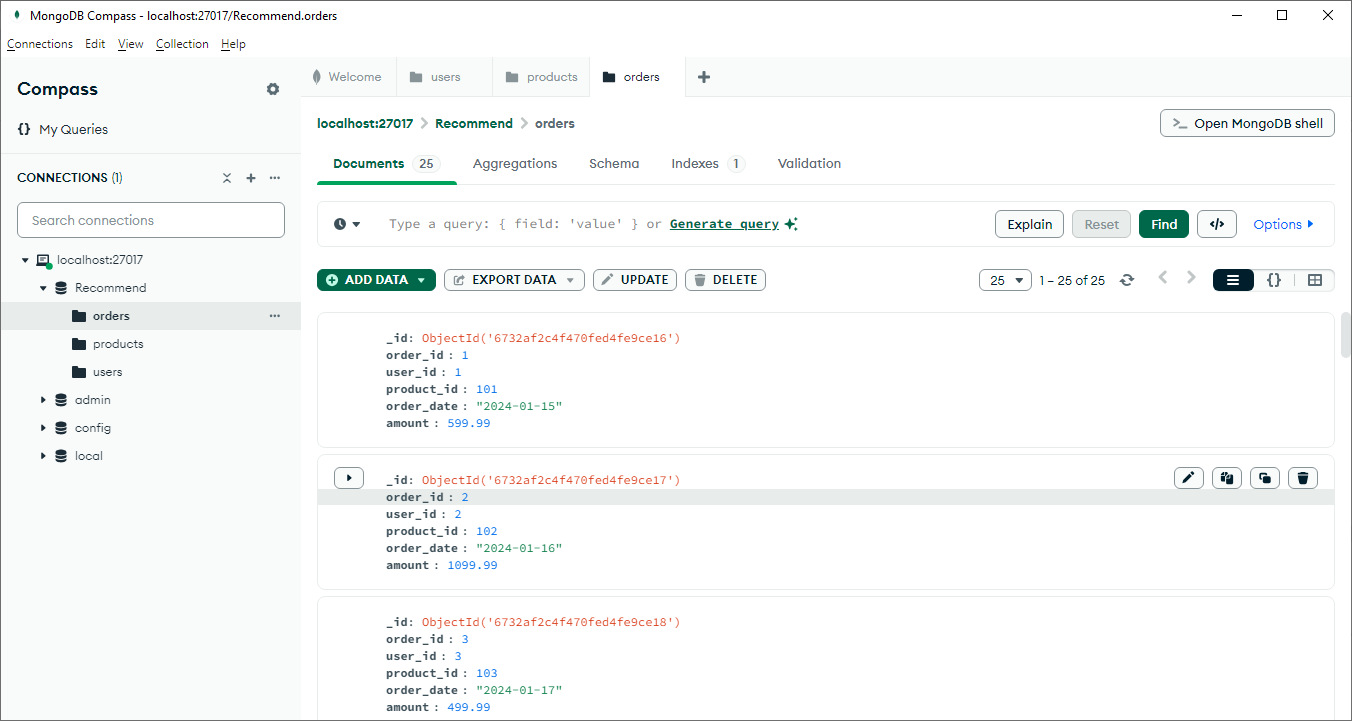


Рисунок 3.6 – Колекція orders з документами

Колекція reviews представлена на рис.3.7.

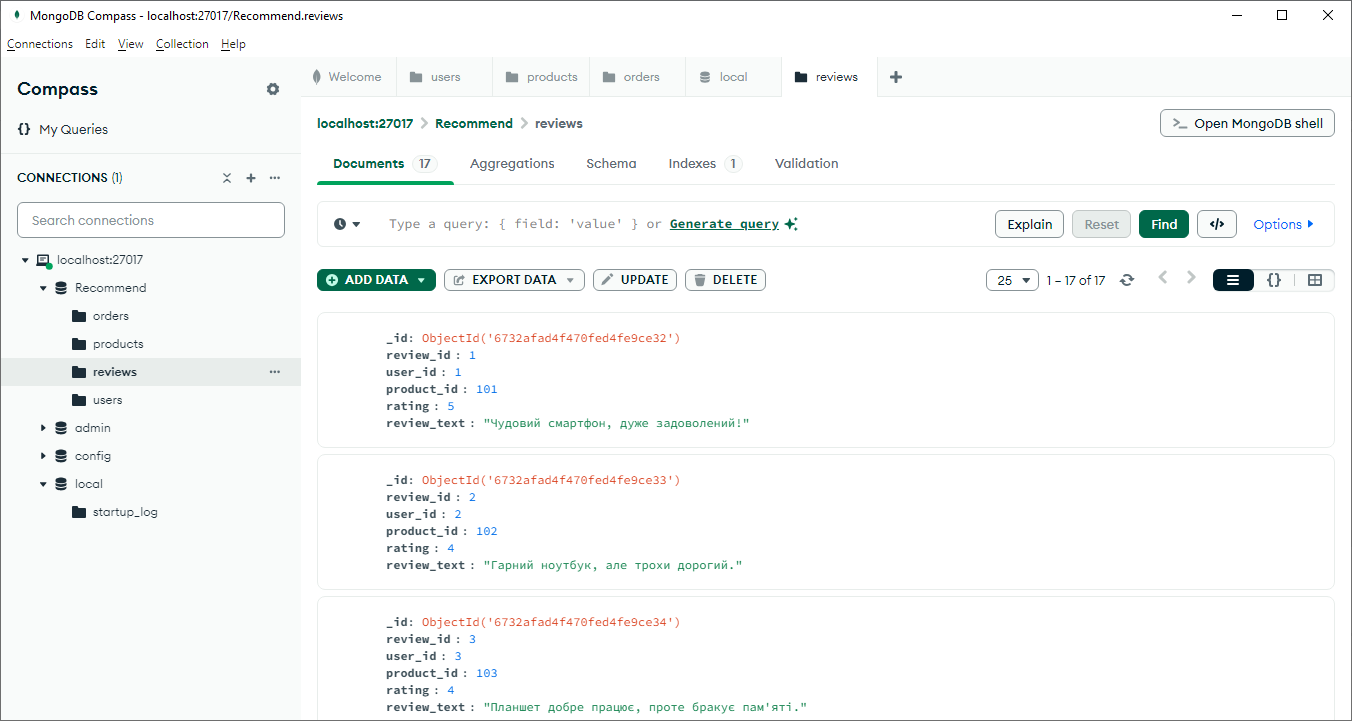


Рисунок 3.7 – Колекція reviews з даними

Переглянемо список користувачів, які проживають в Києві (рис.3.8)



Рисунок 3.8 – Пошук користувачів з Києва

Пошук замовлень, сума яких перевищує 500 у.о.

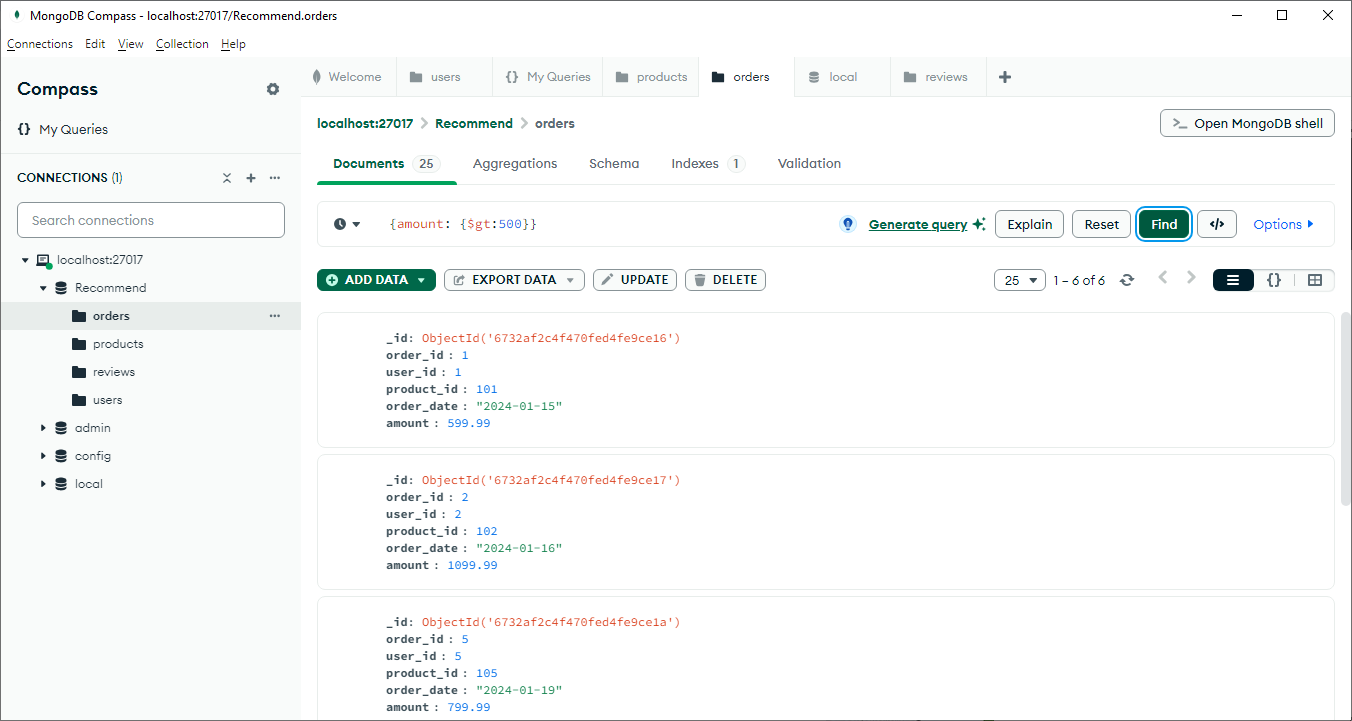


Рисунок 3.9 – Пошук замовлень, сума яких перевищує 500

Створимо конвеєр для розрахунку загальної суми всіх замовлень користувачів (рис.3.10).

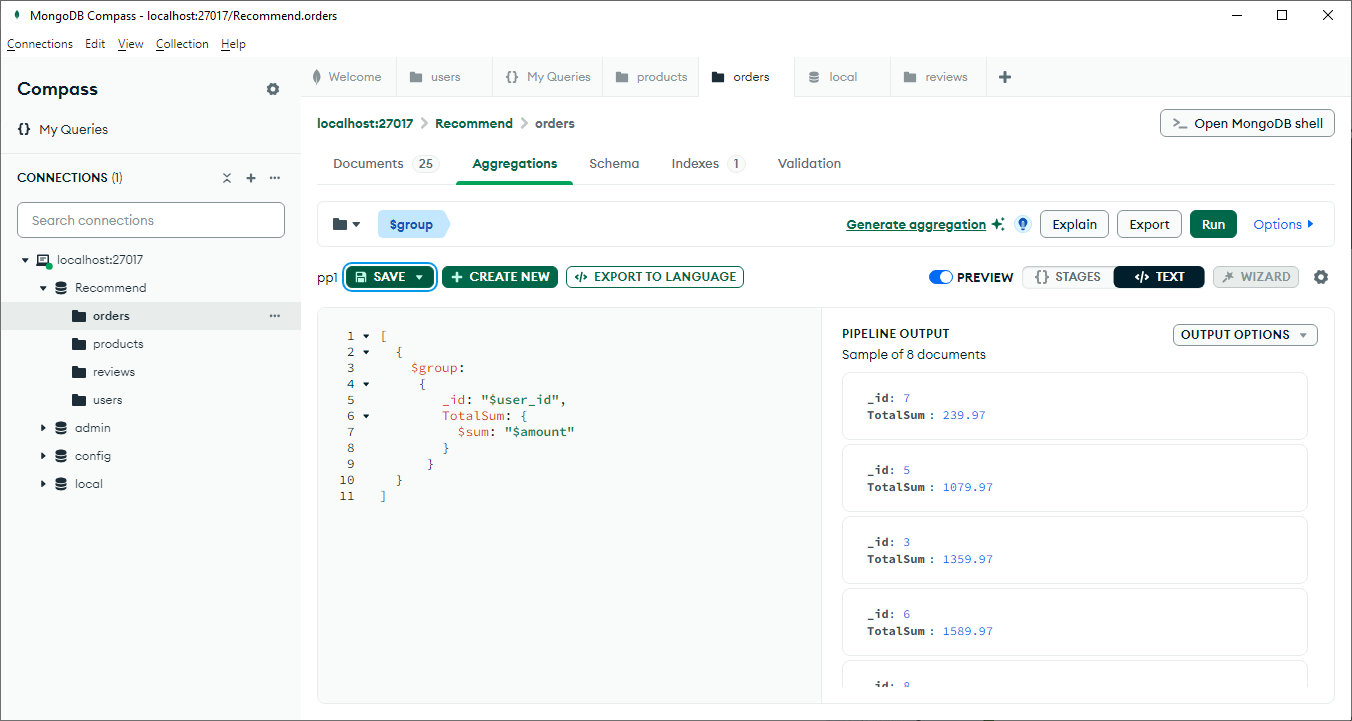


Рисунок 3.10 – Загальна сума замовлень всіх користувачів

Визначимо кількість товарів в кожній категорії (рис.3.11)

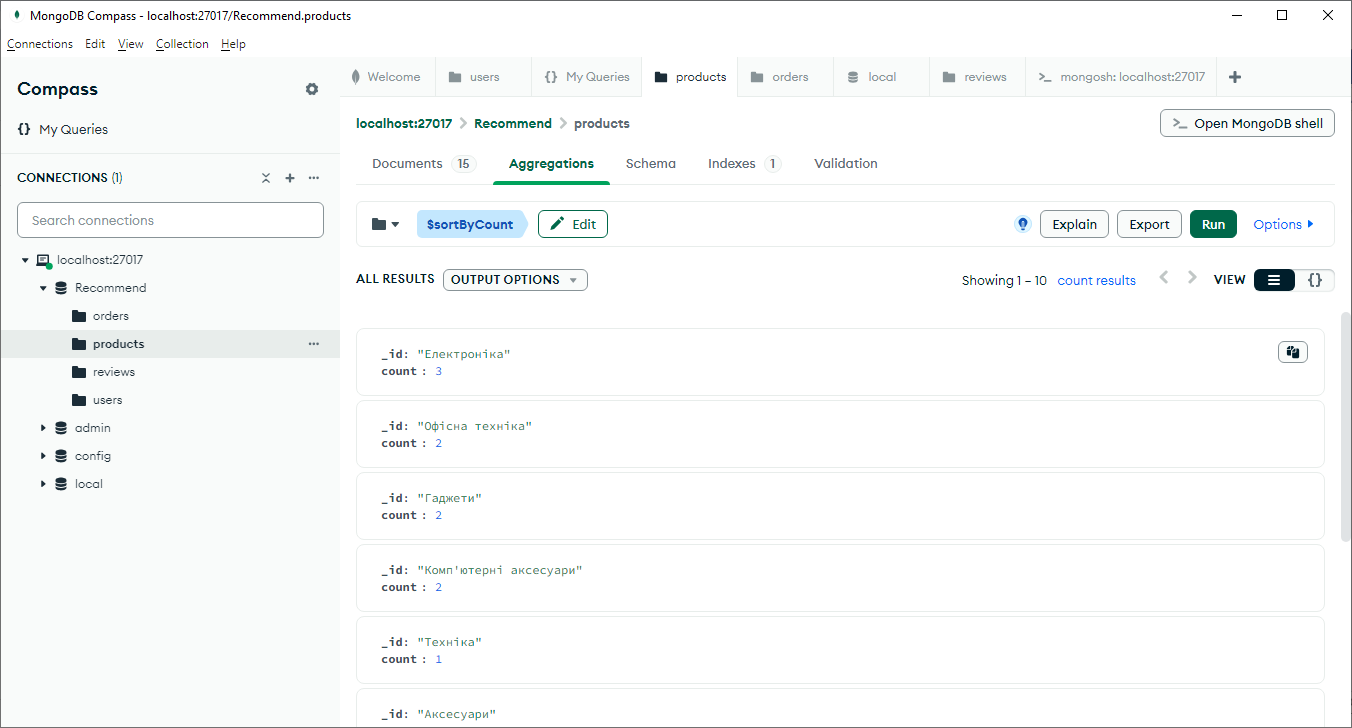


Рисунок 3.11 – Кількість товарів в кожній категорії

Визначимо кількість створених відгуків кожним користувачем (рис.3.12)

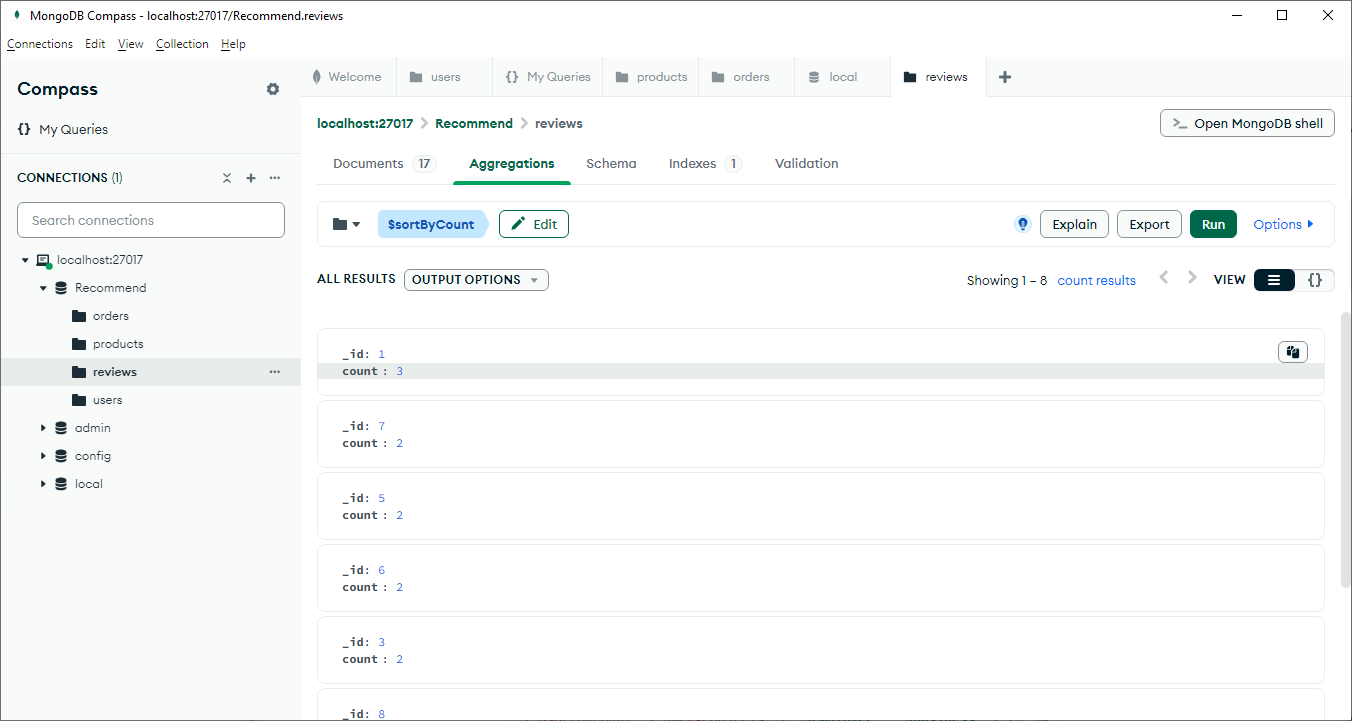


Рисунок 3.12 – Кількість відгуків для кожного користувача

З’єднання двох колекцій (рис.3.13).

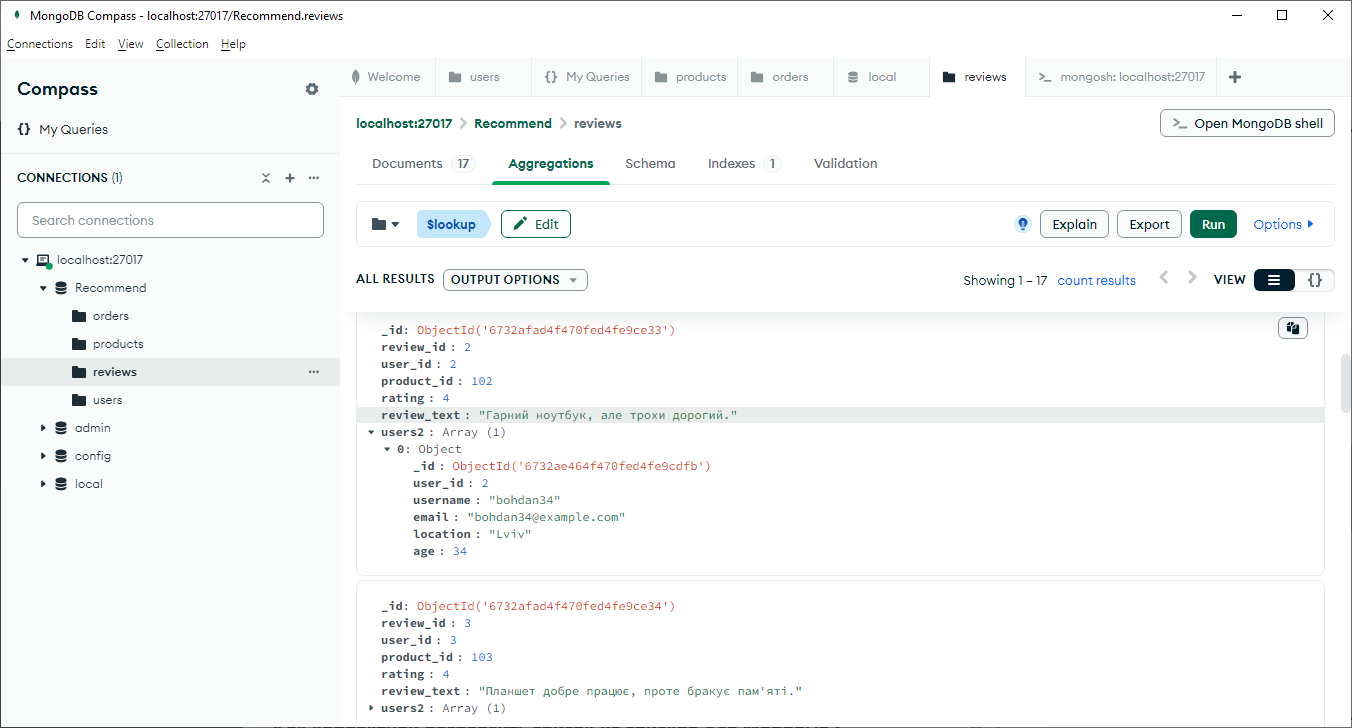


Рисунок 3.13 –З’єднання двох колекцій

Експорт даних та результатів запитів можна здійснювати або в форматі JSON, або в форматі CSV (рис.3.14), що дає змогу обробляти за допомогою інших інструментів.

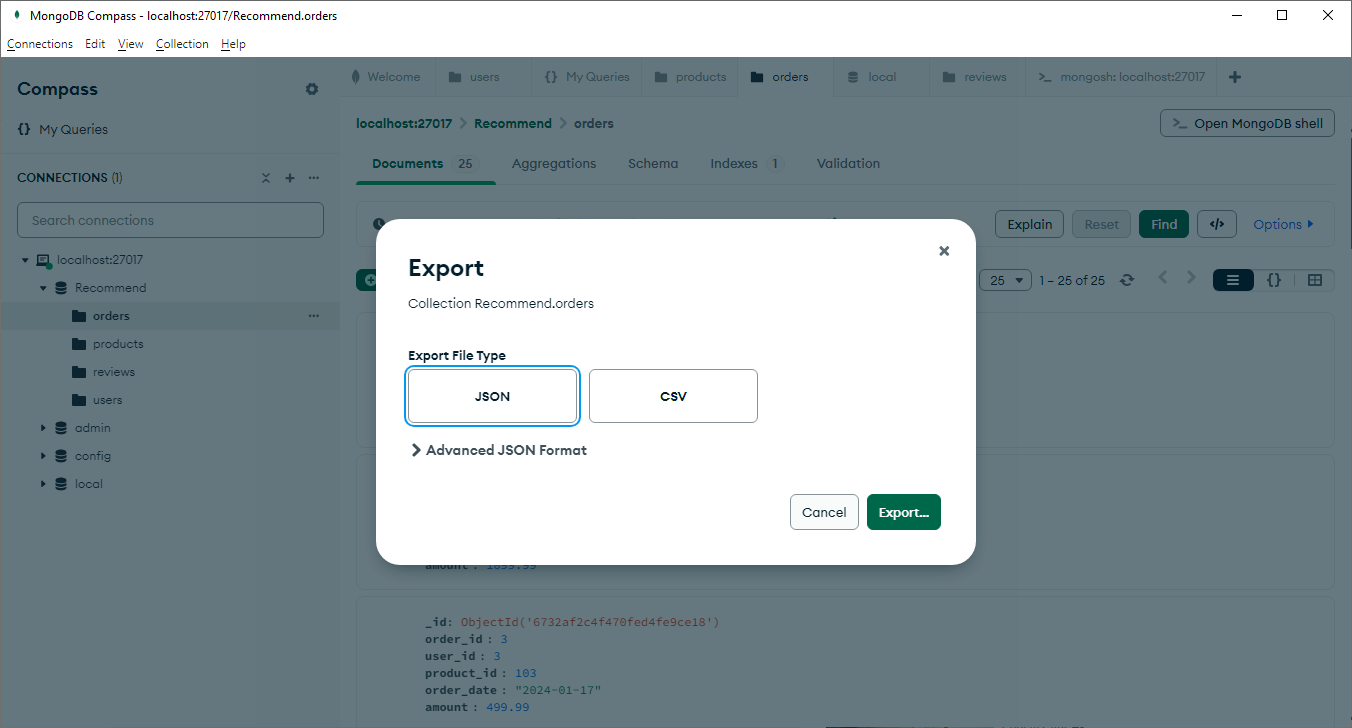


Рисунок 3.14 – Експорт даних із MongoDB

**3.3 Тестування та аналіз продуктивності розробленого рішення**

На основі даних, що можна експортувати з системи побудуємо рекомендаційну систему. Вибір в якості інструменту PySpark дає можливість використовувати потужні засоби для обробки даних.

Спочатку перевіримо правильність та продуктивність виконання запитів за допомогою PySpark та MongoDB.

**PySpark**

spark = SparkSession.builder \

.appName("PySpark Category Count") \

.getOrCreate()

start\_time = time.time()

df = spark.read.format("json").load("path\_to\_data.json")

result\_df = df.groupBy("category").count()

result\_df.show(4)

end\_time = time.time()

execution\_time = end\_time - start\_time

print(f"elapsed: {execution\_time:.4f} sec")

spark.stop()

**MongoDB**

spark = SparkSession.builder \

.appName("PySpark MongoDB Query") \

.config("spark.mongodb.input.uri", "mongodb://localhost:27017/mydb.mycollection") \

.getOrCreate()

start\_time = time.time()

df = spark.read.format("mongo").load()

result\_df = df.groupBy("category").count()

result\_df.show(4) # Показати лише перші 4 рядки

end\_time = time.time()

execution\_time = end\_time - start\_time

print(f"elapsed: {execution\_time:.4f} sec")

spark.stop()

Результати розрахунку кількості товарів в кожній категорії наведені на рис.3.15. Порівняння результатів виконання запиту показує, що обидва інструменти видають однаковий результат, в той же час, MongoDB витратила на виконання даного запиту менше часу.

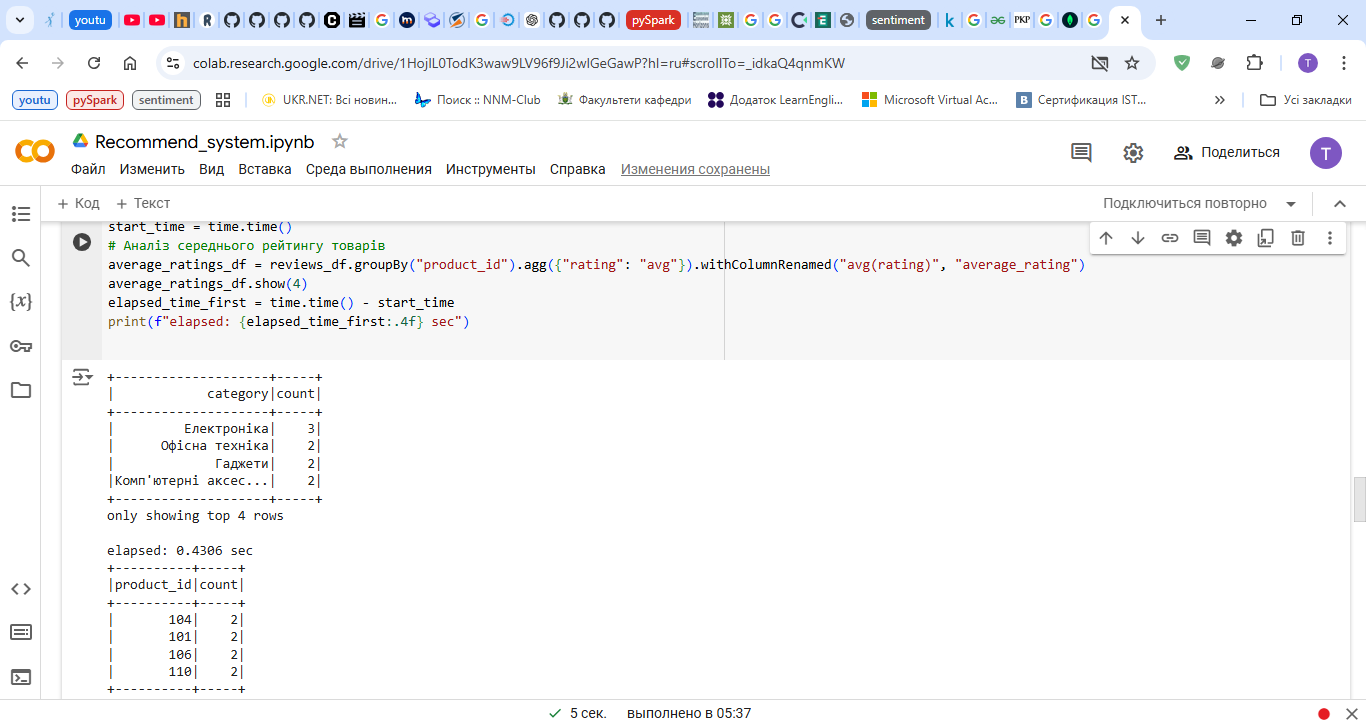


Рисунок 3.15 – Час виконання запиту з використанням PySpark

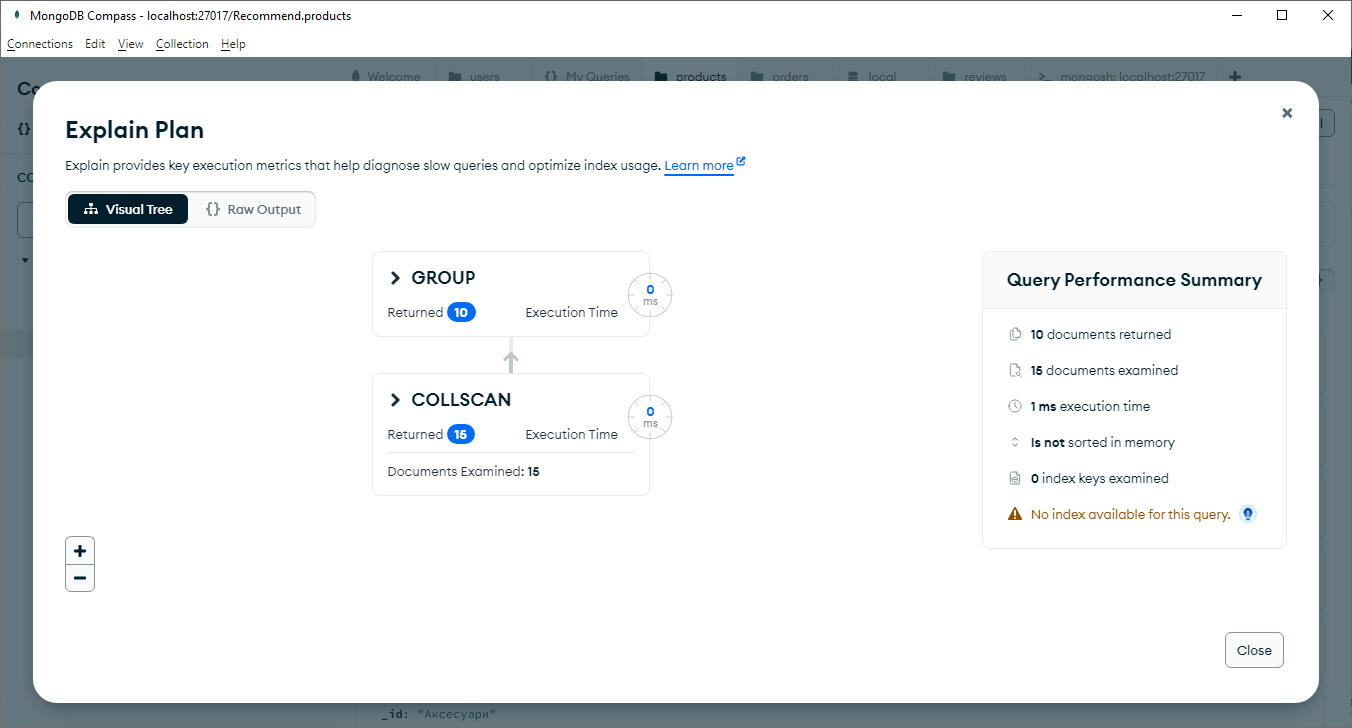


Рисунок 3.15 – Час виконання запиту на кількість товарів в кожній категорії

Аналогічно порівняємо результати запитів, щодо кількості проданих товарів (рис.3.16, 3.17).

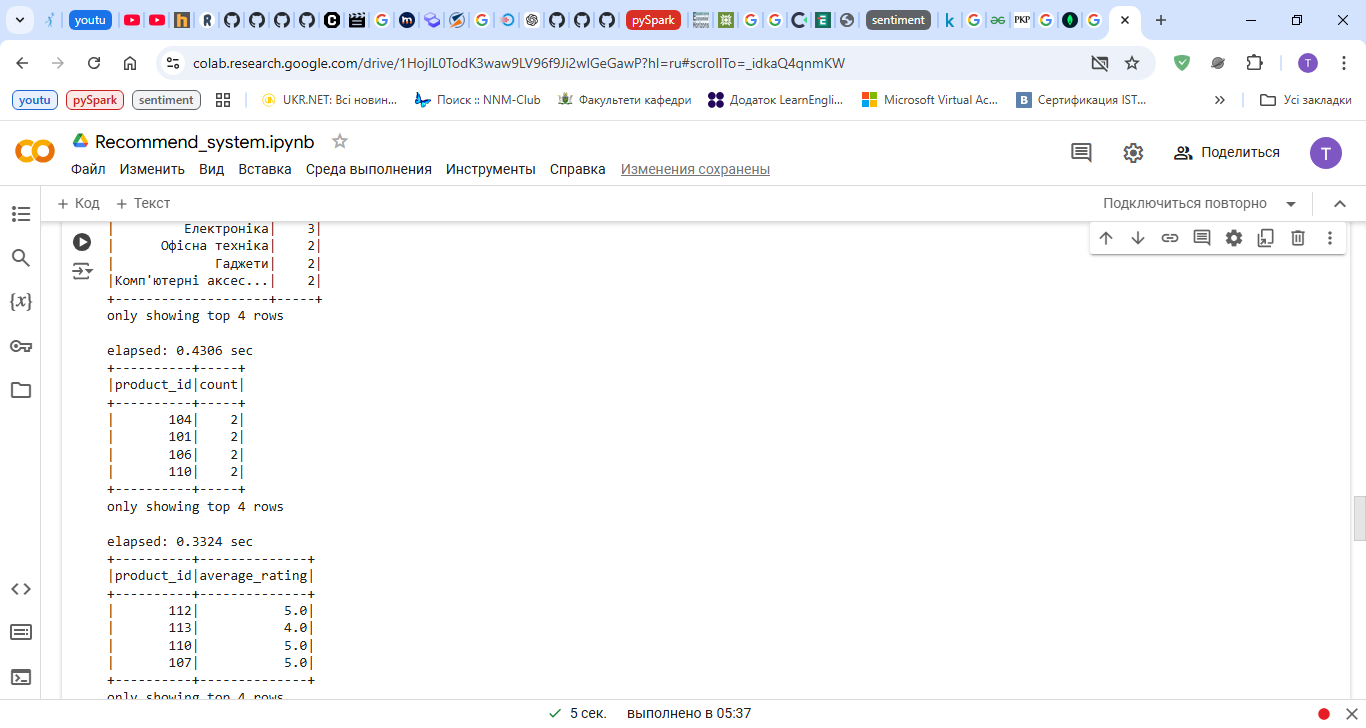


Рисунок 3.16 – Результат запиту на кількість проданого товару (PySpark)

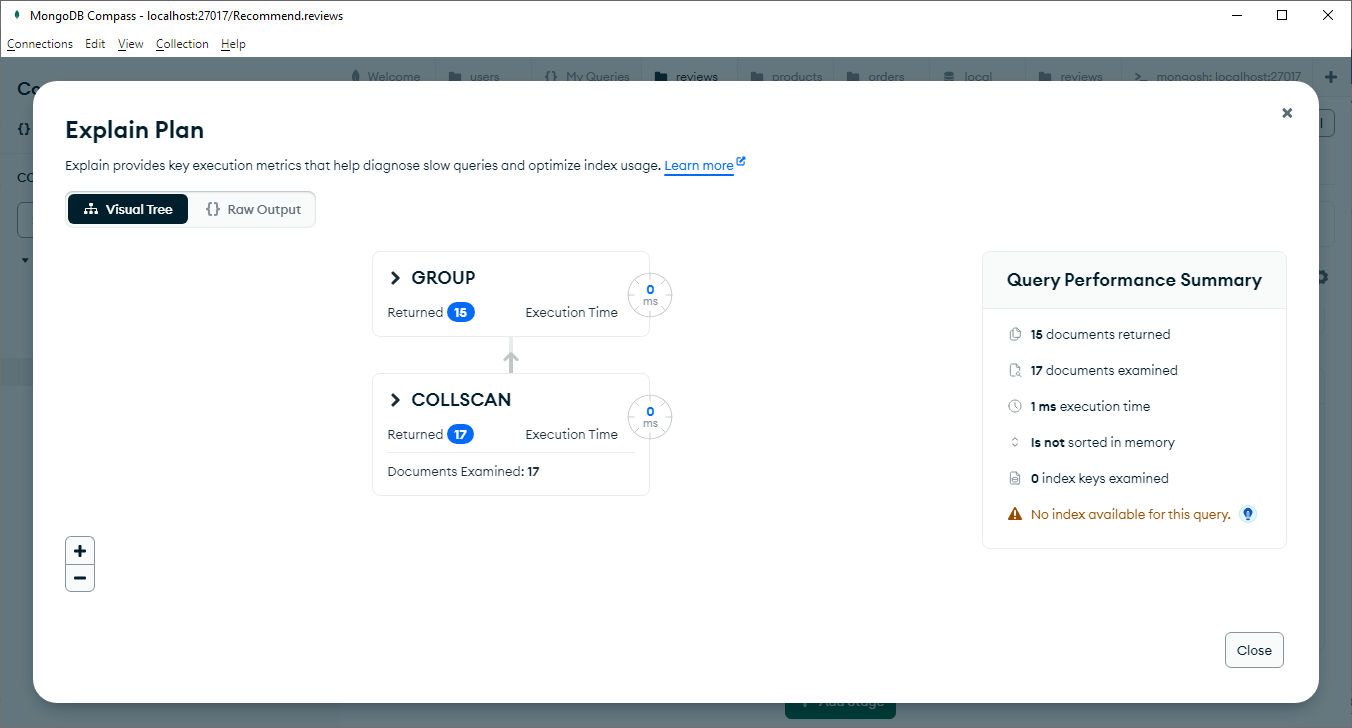


Рисунок 3.17 – Час, витрачений MongoDB на виконання запиту на кількість проданих товарів

Отримані результати свідчать про більшу продуктивність MongoDB порівняно з PySpark.

У той же час в PySpark є потужні інструменти для побудови моделей для аналізу даних, які відсутні в MongoDB. Прикладом такого інструменту є реалізація методу колаборативної фільтрації, що дає змогу побудувати рекомендаційну систему на основу відгуків та покупок користувачів.

Алгоритм ALS (Alternating Least Squares) є методом, який часто використовується для побудови рекомендаційних систем, зокрема на основі колаборативної фільтрації. Основна ідея ALS полягає у матричній факторизації, де велика матриця оцінок розбивається на дві менші матриці, що представляють користувачів і продукти. Цей підхід допомагає передбачити оцінки для комбінацій «користувач-продукт», які ще не були оцінені.

Приклад побудови моделі на платформі MongoDB

{ "user\_id": 1, "name": "Смартфон", "rating": 4.930703 },

{ "user\_id": 1, "name": "Планшет", "rating": 3.9666557 },

{ "user\_id": 1, "name": "Клавіатура", "rating": 3.928863 },

{ "user\_id": 2, "name": "Зовнішній диск", "rating": 4.934475 },

{ "user\_id": 2, "name": "Ноутбук", "rating": 3.9475799 },

{ "user\_id": 2, "name": "Смартфон", "rating": 2.2458904 },

{ "user\_id": 3, "name": "Смартфон", "rating": 4.334086 },

{ "user\_id": 3, "name": "Планшет", "rating": 3.9260662 },

{ "user\_id": 3, "name": "Клавіатура", "rating": 3.6038647 }

Результат побудови моделі наведений на рис.3.18.

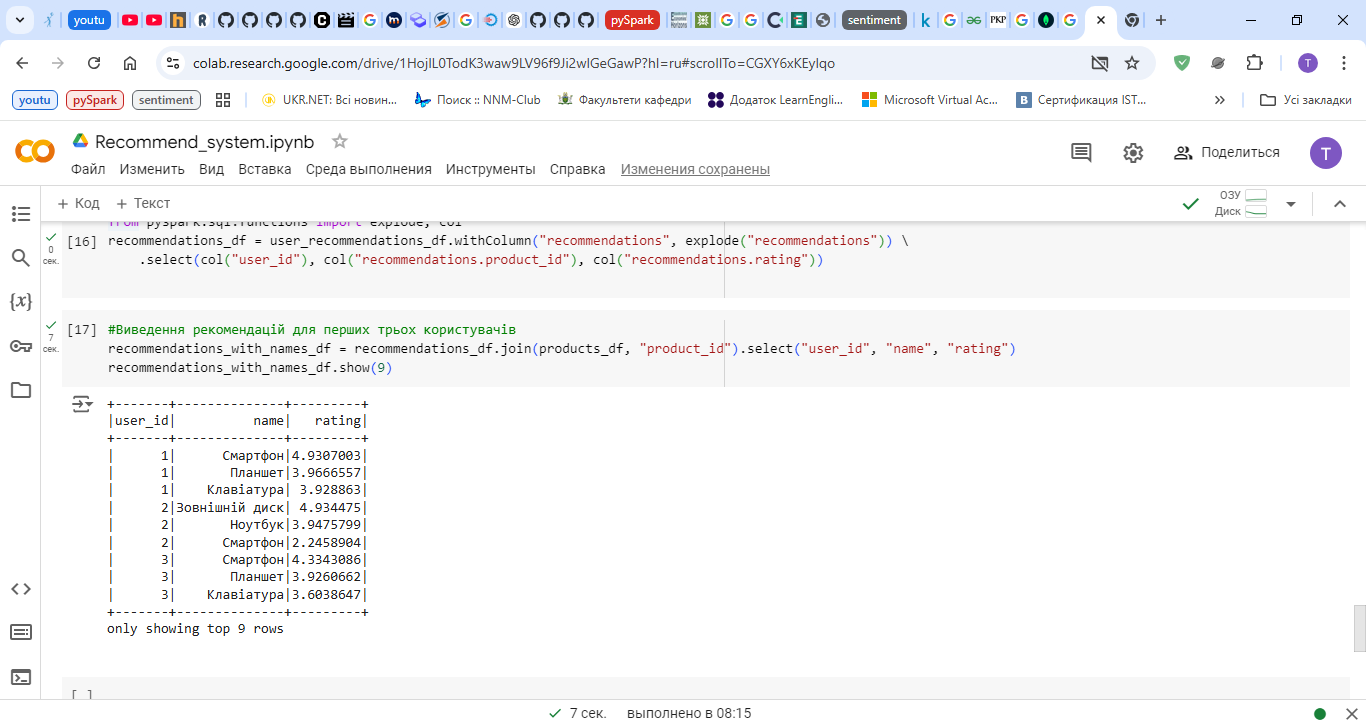


Рисунок 3.18 – Рекомендації для користувачів

Отже, для побудови рішень для великих даних на основі MongoDB та PySpark може бути достатньо ефективним з точки зору зберігання даних та складного аналізу. Для цього рекомендується використовувати MongoDB для зберігання даних та проведення первинної фільтрації та агрегування даних, а складний аналіз, пов’язаний з побудовою моделей машинного навчання, проводити в PySpark вже на відфільтрованих та підготовлених даних, експортованих з MongoDB в форматі JSON.

**Висновки до розділу 3**

В даному розділі було проведено обґрунтування використання для задач аналізу даних в електронній комерції MongoDB та PySpark.

Була розглянута задача побудови рекомендаційної системи на основі інформації про покупки та відгуки користувачів.

Аналіз інформаційної моделі задачі дав змогу розробити в MongoDB базу даних. Для цієї бази даних були розроблені декілька простих запитів з функцією агрегації. Результати цих запитів дозволити порівняти продуктивність MongoDB порівняно з PySpark.

Загалом для побудови ефективних систем обробки великих даних в електронній комерції пропонується використовувати MongoDB для зберігання та попередньої фільтрації даних, а складну обробку поводити з використанням потужних інструментів PySpark.

**ВИСНОВКИ**

У процесі дослідження було визначено, що NoSQL-технології є необхідним компонентом сучасних інформаційних систем, які працюють з великими обсягами даних. Завдяки своїй архітектурі, що орієнтована на високу масштабованість, гнучкість та продуктивність, NoSQL-рішення виявилися особливо корисними для обробки великих даних у режимах, де реляційні бази даних не можуть ефективно забезпечувати швидкість та надійність. Різні типи NoSQL-баз, такі як документоорієнтовані, ключ-значення, графові та колоночні бази даних, задовольняють потреби різних сценаріїв роботи з великими даними, кожен з яких має свої особливості та переваги.

Оцінка основних характеристик NoSQL-систем показала, що вони здатні ефективно забезпечувати масштабованість і доступність за рахунок горизонтального масштабування, що є критичним для великих обсягів неструктурованих або напівструктурованих даних. Використання NoSQL також дозволяє обробляти дані в режимі реального часу та адаптуватися до високого навантаження, яке виникає при роботі з великими потоками даних. При цьому, однак, важливо враховувати особливості узгодженості даних у NoSQL-базах: у багатьох системах реалізована модель eventual consistency, яка забезпечує високу продуктивність, але може бути неприйнятною для деяких критично важливих застосувань.

У ході дослідження також було виявлено, що кожна з NoSQL-технологій має свої сильні та слабкі сторони, залежно від типу даних, необхідних операцій, вимог до узгодженості та швидкості обробки. Так, документоорієнтовані бази є ідеальними для зберігання неструктурованих даних, у той час як графові бази краще підходять для аналізу взаємозв’язків між сутностями.

Результати дослідження надали можливість окреслити ключові аспекти вибору NoSQL-рішень для управління великими даними, зосереджуючись на аналізі особливостей даних, а також вимог до продуктивності та надійності систем. Наприклад, для систем, де важлива швидкість обробки неструктурованих даних із можливістю масштабування, рекомендовано використовувати документоорієнтовані бази даних на зразок MongoDB. А для складної обробки пропонується використовувати PySpark, що має високоефективну бібліотеку для реалізації моделей машинного навчання.

Отже, проведене дослідження підтвердило, що NoSQL-технології є перспективними та важливими для роботи з великими даними в сучасних інформаційних системах. Зважаючи на тенденцію до зростання обсягів даних і збільшення складності запитів, подальші дослідження та вдосконалення NoSQL-рішень мають велике значення для розвитку систем управління великими даними.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Крикун В.А. Технології NOSQL для великих даних: сучасні тенденції та виклики. Наукові горизонти: збірник наук. праць (Секція: Інформаційно-комунікаційні технології). Вип. XVI-XVII. Переяслав, 2024.
2. Ланде Д.В., Субач І.Ю., Гладун А.Я. Оброблення надвеликих масивів даних (Big Data) : навчальний посібник. Київ 2021. 168 с.
3. Провост Ф. Фоусетт Т. Data Science для бізнесу. Як збирати, аналізувати і використовувати дані. Київ, Наш формат, 2019. 400 с.
4. Ситник Н.В., Зінов’єва І.С. Організація баз даних NoSQL [Електронний ресурс] :практикум. Київ: КНЕУ,2022. 167 с.
5. Технології оброблення великих даних: конспект лекцій з дисципліни «Технології оброблення великих даних» [Електронний ресурс] : навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення» (освітня програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»)/ Л.М. Олещенко; КПІ ім. Ігоря Сікорського. Електронні текстові дані (1 файл: 5,55 Мбайт). Київ: КПІ ім. Ігоря Сікорського, 2021. 227 с.
6. Akerkar R. Models of Computation for Big Data Cham: Springer International Publishing, 2018.
7. Albert Y. Zomaya, Sherif Sakr (editors). Handbook of Big Data Technologies: Springer, 2017. 890 p.
8. An introduction to Graph Data Management. In Graph Data Management / R. Angles, C. Gutierrez. 2018. 32 p.
9. Balamurugan Balusamy, Nandhini Abirami R, Seifedine Kadry, Amir H. Gandomi Big Data: Concepts, Technology, and Architecture. Wiley; 1st edition. 2021, 368 p.
10. Big Data processing: methods, models and information technologies: Monograph / edited by Oleg I. Pursky. Shioda GmbH, Steyr, Austria, 2019. P.90-160.
11. Big Data Tutorial Library. Електронний ресурс: <https://dataflair.training/blogs/big-data-tutorials-home/> (дата звернення 1.10.2024)
12. Data Modeling Across the Evolution of Database Technology. A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years. 2018. Р.221–234.
13. Done P. Practical MongoDB Aggregation. 2021. URL: <https://www.practical-mongodb-aggregations.com/> (дата звернення 1.10.2024)
14. Elzein M., Hashem Y. Managing big RDF data in clouds: Challenges, opportunities, and solutions. *Sustainable Cities and Society*. 2018. №39. Р. 375–386.
15. Gandhi R., Gupta A., Povzner A., Belluomini W. and Kaldewey T. Mercury: Bringing efficiency to key-value stores. *Proceedings of the 6th International Systems and Storage Conference*, ser. SYSTOR ‘13. New York, NY, USA: ACM, 2013, pp. 6:1–6:6.
16. Ghavami P. Big Data Governance: Modern Data Management Principles for Hadoop, NoSQL & Big Data Analytics. CreateSpace Independent Publishing Platform, 2016. 204 p.
17. Lakshman and P. Malik, Cassandra: A structured storage system on a p2p network. *Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA ‘09. New York, NY, USA: ACM, 2009, pp. 47–47.
18. Liu Z., Natarajan S, He B., Hsiao H.-I., and Chen Y. Cods: Evolving data efficiently and scalably in column oriented databases. *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 1521–1524, Sep. 2010.
19. Measuring the Performance of Data Placement Structures forMapReduce-based Data Warehousing Systems. *International Journal of New Computer Architectures and Their Applications*. 2018. Р. 11–21.
20. Pedrycz W., Chen S.-M. (eds.), Information Granularity, Big Data, and Computational Intelligence, *Studies in Big Data* 8, DOI: 10.1007/978-3-319-08254-7, Springer International Publishing Switzerland 2015.
21. Rioux J. Data Analysis with Python and PySpark. Manning, 2022. 456 p.
22. Turner V. The digital universe of opportunities: Rich data and the increasing value of the internet of things. [retrieved: March, 2015]. URL: <http://www.emc.com/leadership/digitaluniverse/2014iview/digital-universe-of-opportunities-vernonturner.htm> (дата звернення 1.10.2024)

**ДОДАТКИ**

**Додаток А**

Лістинг файла з модулем рекомендаційної системи

!pip install pymongo

!pip install pyspark

from pyspark.sql import SparkSession

spark = SparkSession \

.builder \

.appName(«my spark app») \

.getOrCreate()

sc = spark.sparkContext

## Завантаження даних для аналізу

from pyspark.sql.functions import col

users\_df = spark.read.json(‘user.json’)

users\_df.printSchema()

users\_df.show(5)

products\_df = spark.read.json(‘product.json’)

products\_df.printSchema()

products\_df.show(5)

orders\_df = spark.read.json(‘order.json’)

orders\_df.printSchema()

orders\_df.show(5)

reviews\_df = spark.read.json(‘review.json’)

reviews\_df.printSchema()

reviews\_df.show(5)

### Тестування продуктивності

import time

start\_time = time.time()

# Аналіз даних: кількість товарів у кожній категорії

category\_count\_df = products\_df.groupBy(«category»).count().orderBy(«count», ascending=False)

category\_count\_df.show(4)

elapsed\_time\_first = time.time() - start\_time

print(f»elapsed: {elapsed\_time\_first:.4f} sec»)

start\_time = time.time()

# Аналіз найпопулярніших продуктів (за кількістю покупок)

popular\_products\_df = orders\_df.groupBy(«product\_id»).count().orderBy(«count», ascending=False)

popular\_products\_df.show(4)

elapsed\_time\_first = time.time() - start\_time

print(f»elapsed: {elapsed\_time\_first:.4f} sec»)

start\_time = time.time()

# Аналіз середнього рейтингу товарів

average\_ratings\_df = reviews\_df.groupBy(«product\_id»).agg({«rating»: «avg»}).withColumnRenamed(«avg(rating)», «average\_rating»)

average\_ratings\_df.show(4)

elapsed\_time\_first = time.time() - start\_time

print(f»elapsed: {elapsed\_time\_first:.4f} sec»)

## Розробка рекомендацій

from pyspark.ml.recommendation import ALS

# Створення ALS моделі для рекомендацій

# Формат: (user\_id, product\_id, rating)

orders\_df = orders\_df.withColumnRenamed(«user\_id», «order\_user\_id»)

# З’єднання та вибір необхідних колонок

ratings\_df = orders\_df.join(reviews\_df, «product\_id»).select(reviews\_df[«user\_id»], «product\_id», «rating»)

# Створення моделі ALS

als = ALS(maxIter=10, regParam=0.1, userCol=«user\_id», itemCol=«product\_id», ratingCol=«rating», coldStartStrategy=«drop»)

model = als.fit(ratings\_df)

# Генерація рекомендацій для кожного користувача

user\_recommendations\_df = model.recommendForAllUsers(3)

from pyspark.sql.functions import explode, col

recommendations\_df = user\_recommendations\_df.withColumn(«recommendations», explode(«recommendations»)) \

.select(col(«user\_id»), col(«recommendations.product\_id»), col(«recommendations.rating»))

#Виведення рекомендацій для перших трьох користувачів

recommendations\_with\_names\_df = recommendations\_df.join(products\_df, «product\_id»).select(«user\_id», «name», «rating»)

recommendations\_with\_names\_df.show(9)