Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)
Кафедра *Системотехніки*
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження процесів обробки запитів в різнорідних розподілених базах для зберігання великих даних
(тема)

Виконав:
Студент 2 курсу, групи *СПРм-18-1*
Стьопін В.І.
(прізвище, ініціали)

Спеціальність
122 – Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Системне проектування*

(повна назва освітньої програми)

Керівник *проф. Іванов В. Г.*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ *Гребеннік І. В.*
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)

Кафедра *Системотехніки*
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність *122 – Комп'ютерні науки*
(код і повна назва)

Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)

Освітня програма *Системне проектування*
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові *Стьопіну Владиславу Ігоровичу*
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Дослідження процесів обробки запитів в різнорідних розподілених базах для зберігання великих даних

затверджена наказом по університету від « » _____ 2019 р. № _____

2. Термін подання студентом роботи (проекту) _____

3. Вихідні дані до роботи (проекту) Функція: Встановлення та оптимізація різнорідних СУБД, розробка генератору синтетичних даних. Перелік використаних програмних засобів: OS Ubuntu 18.04, PHPStorm, MongoDB Compas, MongoDB Server 4.0 X64, MariaDB 10.3 X64, Studio 3T, Navicat for MySQL, MySQL Workbench, Mongo Shell, Microsoft Office 2010. Технічне забезпечення: комп'ютер, підключений до Інетрнету, зі встановленим локальним сервером для виконання запитів до бази даних.

4. Зміст пояснювальної записки (перелік питань, що потрібно розробити) Вступ, 1 Огляд сучасних підходів до зберігання та обробки даних, 2 Постановка задачі кваліфікаційної роботи, 3 Змістовий опис та аналіз різних типів СУБД і особливості їх застосування, 4 Особливості проектування різних типів баз даних і методи їх заповнення, 5 Налаштування СУБД та оптимізація запитів, 6 Порівняльний аналіз продуктивності різних СУБД, Висновки, Перелік посилань

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)

1.1 Статистика DB-Engines по використанню різнорідних СУБД, 1.2 Принцип різних типів зберігання даних у базі даних, 1.3 Статистика запитів у системі, яку наповнюють та складають люди, 1.4 Графік зміни продуктивності системи з ростом її складності, 2.1 Зміна популярності формату даних для передачі по мережі, 2.2 Методи створення розподіленої бази даних, 4.1 Логічна схема реляційної бази даних, 4.2 Логічна схема реляційної бази даних, 4.3 Функції генерації синтетичних даних, 4.4 Максимальний об'єм бази даних у експерименті, 5.1 Графік залежності часу виконання запиту від кількості запитів, 5.2 Приклади оптимізації запитів до бази даних, 5.3 Перегляд та зміна конфігурації СУБД, 6.1 Запити для тестування СУБД, 6.2 Виконання запиту додавання даних без оптимізації, 6.3 Виконання запиту читання даних без оптимізації, 6.4 Виконання запиту оновлення даних без оптимізації

6. Консультанти розділів роботи (проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапі виконання етапів роботи (проекту)	Примітка
1.	Отримання завдання на дипломне проектування	*01.09.19*	
2.	Аналіз завдання, літератури, накових статей та існуючих аналогів з теми дипломної роботи	*03.09 — 25.09.17*	
3.	Розробка власних алгоритмів створення та заповнення БД, тестування можливостей СУБД	*26.09— 18.10.19*	
4.	Тестування запитів на оптимізованих СУБД	*19.10 — 28.10.19*	
5.	Проектування структури СУБД для тестування складних запитів	*29.10 — 11.11.19*	
6.	Оформлення пояснювальної записки та графічних матеріалів	*12.11— 26.11.19*	
7.	Оформлення презентаційних матеріалів	*27.11 — 04.12.19*	
8.	Представлення на рецензування	*06.12.19*	
9.	Представлення дипломної роботи в дек	*10.12.19*	

Дата видачі завдання _____ _____20__ р.

Студент _____ Стьопін В.І.
(підпис)

Керівник роботи _____ проф. Іванов В.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до магістерської атестаційної роботи: 103 с., 3 табл., 23 рис., 2 додатки, 39 джерел інформації.

Об’єкт дослідження – різнородні ситеми управління базами даних великих обсягів, як складова частина розподіленої системи.

Предметом дослідження є параметри продуктивності СУБД і процесів обробки запитів різної складності у різнорідних типах баз даних.

Метою дослідження є виявлення найбільш продуктивної СУБД або комбінації різнорідних СУБД та їх можливостей для виконання запитів різної складності в умовах роботи з великими даними.

Метод дослідження – системний підхід, методи структурного аналізу і проектування.

У результаті виконання роботи розроблені практичні рекомендації щодо підвищення продуктивності роботи СУБД та виконання запитів до великих даних за умов розподілених баз даних в залежності від різних умов експлуатації системи. Подібні результати вперше отримано на останніх версіях СУБД за умов обробки великих даних. Порівняльний аналіз показав, що реляційна СУБД з кешуванням та детальними налаштуваннями серверу бази даних має переваги при виконанні запитів будь-якої складності. Тем не менш, документо-орієнтована база даних легше у використанні та має кращу масштаблованість, що дозволяє інженерам баз даних ефективно виконувати прості CRUD операції з великими даними. Для масштабних систем краще використовувати реляційні бази даних як сервіс (DBaaS), які розміщуються у хмарі, що дозволяє компенсувати недоліки масштабування та підвищи відказостійкість при потокових запитах з об’єднанням.

Галузь застосування – проектування будь-яких систем або сервісів, які використовуюсь базу даних, розподілені бази даних або хмарні рішення для зберігання і обробки даних. Результати роботи можуть допомогти інженерам баз даних або програмістам при формуванні запитів до бази даних або її оптимізації.

БАЗИ ДАНИХ, РОЗПОДІЛЕНІ БАЗИ ДАНИХ, NOSQL БАЗИ ДАНИХ , SQL ЗАПИТИ, MYSQL, MARIADB, MONGODB, ОПТИМІЗАЦІЯ, ВЕЛИКІ ДАНІ, ПРОЕКТУВАННЯ БАЗ ДАНИХ

ABSTRACT

Explanatory note to the master's certification project contains: 103 p., 3 tab., 23 pic., 2 applications, 39 sources.

Research object – different database management systems as a component of a distributed system.

Study subject is DBMS efficiency parameters and query processing of varying complexity to different database types.

The goal of the study is to identify the most productive DBMS or a combination of different DBMS and their possibilities to process queries of varying complexity in the context of Big data.

Research method – system approach, methods of structural analysis and design.

As a work result, practical recommendations for optimization and query processing to distributed databases were developed. Recommendations are based on the tasks set and different exploitation conditions of the system. Such results were obtained for the first time on the latest DBMS versions in the context of Big data processing. Comparative analysis showed that a relational database system with caching and detailed database server configuration has querying any complexity advantages. However, the document-oriented database is easier to use and it has better scalability, allowing engineers to efficiently perform simple CRUD operations with Big data. For large-scale systems, it is best to use relational databases as a service (DBaaS), which are hosted in the cloud. It can compensate for the scaling deficiencies and increase resiliency in streaming merged queries.

Application scope is designing any systems or services which are using databases, or distributed databases, or cloud solutions for storage and processing data. The results can help database engineers or programmers in querying or optimizing queries to database.

DATABASES, DISTRIBUTED DATABASES, NOSQL DATABASES, SQL QUERIES, MYSQL, MARIADB, MONGODB, OPTIMIZATION, BIG DATA, DATABASES DESIGN

ЗМІСТ

ВСТУП

Процес розробки програмного забезпечення зазнає позитивних змін і кожен рік вдосконалюється. Все більше процесів автоматизуються, використовується мікросервісна архітектура для побудови систем будь-якого призначення. Більшість даних зберігається і обробляється у хмарних сховищах. Переважна більшість СУБД пропонують автоматичну оптимізацію запитів, швидку роботу с даними та підтримку будь-якої мови програмування. Але з часом, коли об'єм даних стає великим, різниця між різними типами баз даних стає більш чіткою.

Кількість різноманітної інформації у сучасному світі тільки росте. Зі збільшенням кількості приладів, які підключені до Інтернету (Internet of Things), користувачами та інформації, що обробляється, збільшується і кількість даних, яку потрібно зберігати. Розповсюджена практика зберігати сирі дані (raw data) для подальшого аналізу з метою находження корисної інформації (data analysis). Це призводить до збільшення об'єму бази даних та виникнення проблеми обробки великих даних. Якщо питання підвищення продуктивності виконання запитів виникає у момент експлуатації системи – будь-яке рішення буде дорожче, ніж її передбачення на етапі проектування.

Мета дослідження – виявлення найбільш продуктивної СУБД або комбінації різнорідних СУБД та оптимізації виконання запитів різної складності в умовах роботи з великими даними. Мета досягається за рахунок проведення структурного і порівняльного аналізу існуючих СУБД, їх властивостей та характеристик, і виведення практичних рекомендацій та комбінацій їх використання для мінімізації часу виконання запитів різної складності.

Об'єктом дослідження є різнорідні СУБД великих обсягів, як складові частини єдиної розподіленої системи. Предметом дослідження є визначення показників продуктивності СУБД для виконання запитів різної складності до різнорідних баз даних. Дослідження проводиться завдяки методам системного підходу, структурного аналізу та системного проектування.

Результати роботи вперше отримані на сучасних версіях СУБД в умовах використання великих даних, що частково удосконалюють висновки попередніх робіт за цією темою. Проміжні висновки цієї роботи представлені у 7 публікаціях, одна з яких входить до науко метричної бази Scopus.

1 ОГЛЯД СУЧАСНИХ ПІДХОДІВ ДО ЗБЕРІГАННЯ ТА ОБРОБКИ ДАНИХ

Архітектура більшості сучасних систем полягає у роботі з даними. Завдяки технологічному прогресу, ми можемо збирати дані не тільки від людей, а й від машин, їх взаємодії між собою, навколишнього середовища, тощо. Кількість даних, які обробляються або підлягають обробці – швидко зростає. Розповсюджується практика збирання всієї аналітичної інформації, яка не несе практичної користі, але може бути використана аналітиками для отримання корисної інформації [1].

Завдяки появі сервісної архітектури, навіть невеликі системи використовують аналітику та збирають дані користувачів. Для збору і обробки даних навіть не завжди можна використовувати СУБД. Але зважаючи на об’єм даних, який необхідно обробляти кожного дня, бази даних є невід’ємною частиною кожної сучасної системи. До баз даних у сучасному світі ставлять більше задач, ніж лише зберігання даних. Від СУБД очікується, що вона стане інструментом аналітики, обробки даних та засобом спрощення взаємодії системи і сховища даних. Більше того, вимагається така можливість використання СУБД, яка дозволяє отримувати дані з однаковою швидкістю для кінцевого користувача прикладної програми, незалежно від того, якого об’єму дані зберігаються у базі.

Актуальність проблеми обробки інформації навколо системи підкреслюють самі товариства розробників, так як велика кількість СУБД є безкоштовними та підтримуються виключно за рахунок незалежних авторів. Також створюються легкі у використанні бази даних для аналітиків, які можуть без додаткових знань проводити аналіз даних, що зберігаються. Такі БД легко підключаються до сайту та зберігають вхідні дані автоматично. Завдяки таки інструментам стає можливо краще розуміти своїх користувачів, вдосконалювати інтерфейс або функціональність систем, збільшувати конверсії на сайтах, аналізувати дані у реальному часі та багато інших, досі недоступних операцій.

Перед створенням будь-якої завжди системи стоїть вибір типу бази даних, яка буде використана. Існують наступні моделі баз даних: реляційна, документно-орієнтована, зберігання у форматі «ключ-значення», колонкова, у вигляді графу та мульти-модельна, яка поєднує у собі декілька моделей, перерахованих вище. Кожна з моделей може бути ефективною за певних умов використання. Такі умови можуть бути: предметна область системи, кількість даних, які будуть

зберігатися, кількість запитів до бази даних, передбачуване навантаження, модель використання системи, ціль зберігання, аналізу та використання даних, та інші незначні критерії.

Найбільш популярними моделями баз даних є реляційні та не реляційні, у які входять всі інші моделі (документна, «ключ-значеня», тощо). У найпоширеніших СУБД є можливість використовувати декілька моделей одночасно, або у рамках декількох баз даних, які зв'язані між собою. Таким чином, розробникам дають можливість використовувати необхідні їм інструменти для роботи з даними, пов'язувати між собою різнорідну функціональність у рамках однієї системи, що прискорює процес розробки та полегшує обробку даних. На рисунку 1.1 зображено графік росту популярності теперішніх лідерів ринку СУБД.

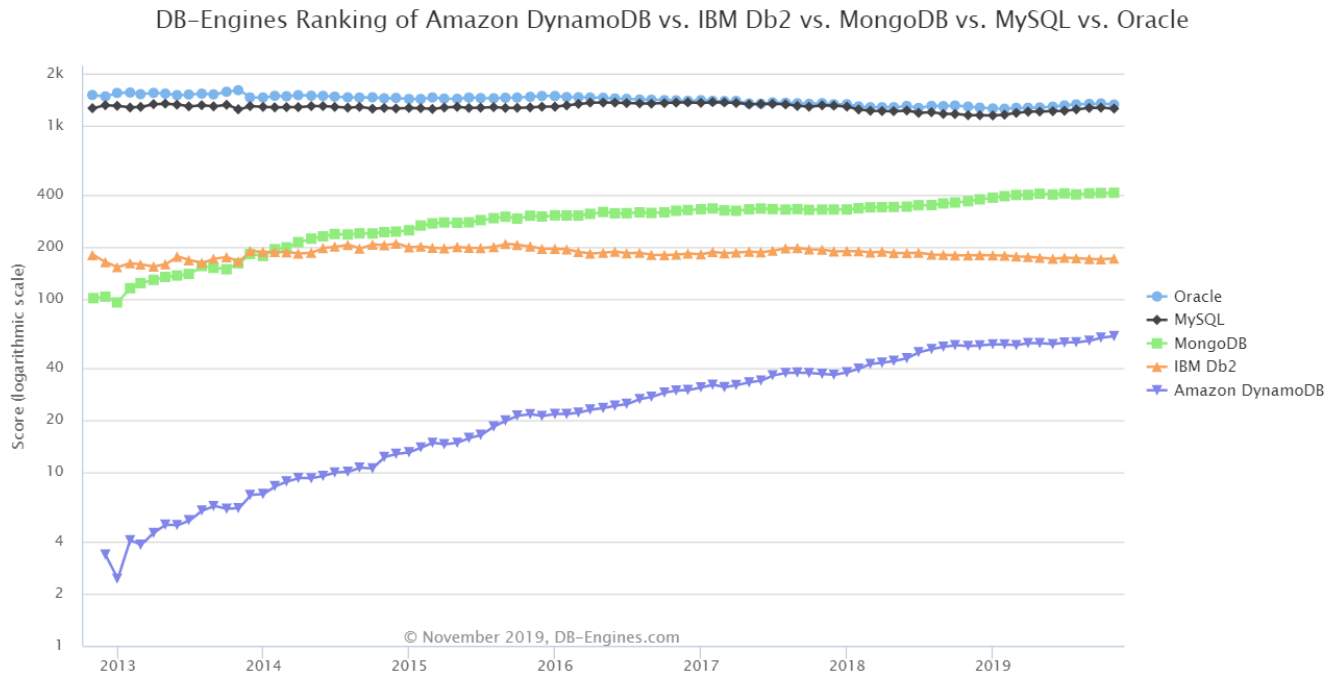

Рисунок 1.1 – Статистика DB-Engines по використанню різнорідних СУБД [13]

MySQL та Oracle мають найбільший рейтинг серед усіх інших за рахунок того, що вони виникли як перші СУБД з широкими можливостями для роботи з даними. Популярність та функціональність цих СУБД не перевищували жодні інші програмні рішення аж до 2012 року. Саме тоді почали з'являтися гарно розроблені аналоги, які могли краще виконувати деякі групи задач. З того ж

самого моменту системи починають ставати меншого розміру і не потребують великої бази даних, тому швидко набирає популярність документно-орієнтована модель баз даних, найкращим представником якої є MongoDB, що підтверджує графік на рисунку 1.1. Сучасні системи та найбільші в світі компанії працюють у Інтернеті. З покращенням технології мобільного зв'язку та мобільного Інтернету – все більше користувачів використовують свої телефони для роботи з даними на сайтах. Кількість запитів, які обробляють сайти великих компаній, сягають мільярдних значень. Саме тому популярність швидко набирають не лише хмарові сховища даних, а й хмарові сервіси по обробці та аналітиці даних [2]. Саме такі сервіси надає, наприклад, Amazon Web Services, але для їх використання краще зберігати свої дані у БД Amazon DynamoDB, так як тоді базу даних можна аналізувати без трати додаткового трафіку. Таким чином, найпопулярнішою безкоштовною СУБД на даний момент є MySQL, в більшості завдяки тому, що вона є базою даних за замовчуванням у більшості бібліотек для автоматизації створення програм та сайтів. Та вона має багато різних програмних вдосконалень, які перетворилися в самостійні СУБД: PostgreSQL, SQLite, MariaDB, тощо.

Більшість баз даних підтримують єдину мову запитів – SQL [3]. Завдяки цьому легко використовувати різні типи СУБД та моделі баз даних у будь-яких системах. Мова запитів дозволяє не тільки маніпулювати даними, а і фільтрувати, аналізувати та отримувати корисну інформацію на вимогу користувача.

У зв'язку з проблемою швидкісної обробки великих даних для різних користувачів системи виникли додаткові пошукові системи, які дозволяють ефективніше використовувати бази даних, наприклад Redis [4]. Завдяки таким системам програмні продукти мають змогу легко розширюватися та обробляти ще більше різнорідної інформації. Такі системи стали незамінними при роботі з великими даними, так як вони запам'ятовують запити користувачів та відповіді бази даних, таким чином, якщо поступає новий запит, який вже був раніше, то система видасть вже готову відповідь, економлячи час роботи самої БД та час очікування користувача. Система лише перевіряє, чи не змінилися дані у базі.

Високу швидкість обробки даних має нова модель бази даних – колонкова. Замість звичних рядків для зберігання даних використовуються колонки, що дозволяє сортувати та вибирати лише ті дані з бази, які задано у запиті. Взагалі, колонкові бази даних швидше обробляють інформацію та дозволяють зменшити трафік при роботі в мережі. Найбільшу конкурентну спроможність така модель баз даних набуває в аналітичних системах, де переважає операція «SELECT». У

такому разі перевага у швидкості навіть на серверах з нижчою потужністю досягає 100 разів. Такі над продуктивні показники виникають за рахунок використання меншого об’єму даних, ніж у стандартній РСУБД та використання компресії. Також колонкові бази даних займають менше простору для зберігання даних. Але, якщо запити до бази даних ускладнюються, тоді швидкість виконання запитів може дорівнювати базі даних зі строковим типом зберігання даних, а інколи і уступати їх. Принцип збереження даних колонкового типу показано на рисунку 1.2.

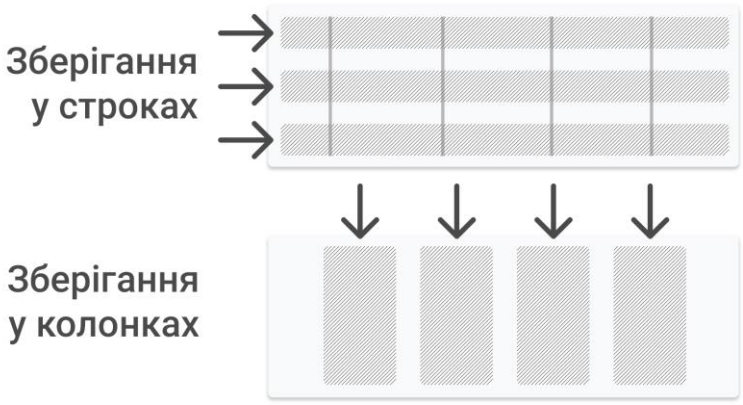

Рисунок 1.2 – Принцип різних типів зберігання даних

При обробці запиту у строковій базі даних – алгоритм пройде усі строки та потім відфільтрує їх за параметрами запиту. У колонковій базі – будуть вибрані лише ті колонки з даними, які відповідають запиту. Тобто при наявності ста мільйонів записів і лише п’яти, які відповідають запиту користувача, строкові бази даних пройдуть всі записи, а колонкові – лише п’ять потрібних. Різниця в швидкості виконання запиту та трафіку, що передається між базою даних та конечним користувачем, очевидна.

Перевагою зберігання даних у строках є більша варіативність обробки та зміни даних в базі. Строкові таблиці швидше працюють на запис даних, особливо, коли запис проходить у потоці. Саме тому велика кількість підключень до колонкової бази даних може призвести до помилок в роботі системи. Виконання операцій «UPDATE» та «DELETE» відбувається швидше, так як строку легше ідентифікувати та не після виконання операції її індекс не змінюється. Також

об’єднання декількох операцій або таблиць під час запиту швидше виконується у строкових типах таблиць, особливо у разі точного налаштування роботи СУБД.

Таким чином, колонкові бази даних ефективні лише для проведення швидкої аналітики та збору фактичних даних з системи. Для складної системи, яка працює з різнорідними даними, як правило неструктурованими, такий тип зберігання даних не підходить. Використовуючи SQL мову запитів, можливо маніпулювати стовпцями, але не настільки ефективно, як строками. Для збору не корисної інформації, а наприклад складної статистики, потрібно переглядати усі записи в базі даних, незалежно від її типу. В такому випадку строкові базі матимуть значну перевагу у гнучкості обробки даних і видачі релевантного результату.

Зважаючи на це, в роботі буде розглянуто два типи баз даних: реляційні та не реляційні з метою визначення переваг та не достатків кожної з моделей в умовах використання у сучасних системах.

Більшість систем змушені працювати у режимі реального часу, тобто працювати з потоками даних, які зазвичай виконуються паралельно в один момент часу. Коли потоків виникає велика кількість, сервер бази даних може не встигати виконувати всі запити, із-за чого виникають черги або відмови. Якщо паралелізм потоків виконання легко реалізувати на стороні програми, то з база даних не може виконати більшість алгоритмів, які використовуються у мовах програмування [5][6]. Таким чином, надійним способом захистити сховище даних – це створити проміжний етап накопичення даних та їх подальша обробка і запис до сховища, як це реалізовано у AWS S3 сервісі [7]. Завдяки проміжному етапу усі зібрані дані сортуються та зберігаються у відповідне місце на сховищі, дозволяючи системі очистити тимчасове сховище для прийому нових даних. Такий підхід дозволяє мінімізувати вірогідність відмови системи та забезпечити належну швидкість обробки даних навіть при великих об’ємах вхідної інформації. Завдяки онлайн сервісам – існує можливість швидко та дешево інтегрувати будь-який інструмент для створення конвеєру по роботі з даними (delivery pipeline) у системі, яка розробляється на будь-якій мові програмування. Такий підхід дозволяє використовувати дані для оновлення всіх елементів системи на етапі, коли вони ще не потрапили до сховища, таким чином підтримуючи релевантність інформаційного поля в системі, що дозволяє їй оновлюватися у реальному часі, з моменту як тільки дані отримані від зовнішнього джерела і до моменту збереження їх у сховищі.

За статистикою використання запитів у системі онлайн бронювання квитків, яку показано на рисунку 1.3, найпопулярнішими є «SELECT» та «INSERT». Хоча така статистика залежить від типу системи та її алгоритму обробки даних, у більшості випадків взаємодії «людина-система», видача та оновлення даних будуть найпопулярнішими типами запитів.

Параметры	#	ø в час	%
select	7,705 k	4,467.2	50.24
insert	4,217 k	2,444.9	27.50
begin	1,513 k	876.9	9.86
commit	1,512 k	876.6	9.86
set option	196 k	113.7	1.28
update	57,344	33.2	0.37
show fields	43,931	25.5	0.29
show tables	42,027	24.4	0.27
show processlist	34,440	20	0.22
delete	4,321	2.5	0.03
delete multi	4,274	2.5	0.03
change db	2,154	1.2	0.01
show create table	2,079	1.2	0.01
create temporary table	966	0.6	0.01
rollback	480	0.3	<0.01
show databases	114	0.1	<0.01
show grants	52	<0.1	<0.01
lock tables	42	<0.1	<0.01
unlock tables	42	<0.1	<0.01
show create trigger	21	<0.1	<0.01
show variables	19	<0.1	<0.01
show binlogs	7	<0.1	<0.01
show slave status	6	<0.1	<0.01
show master status	6	<0.1	<0.01
show create proc	4	<0.1	<0.01
show status	2	<0.1	<0.01
show warnings	1	<0.1	<0.01

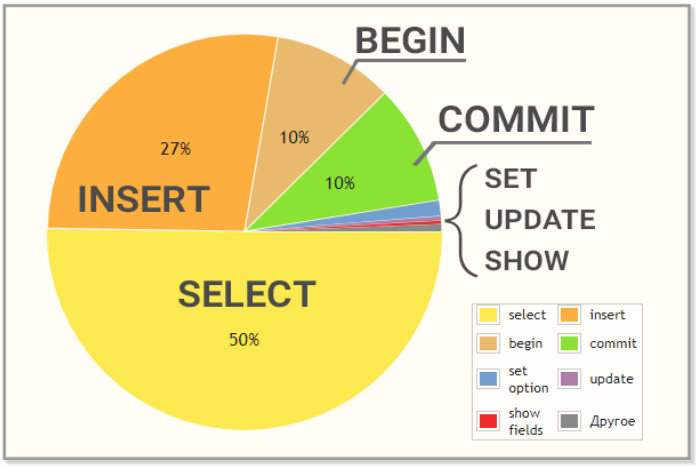

Рисунок 1.3 – Статистика запитів у системі, яку наповнюють та складають люди

Статистика показує лише елементарні операції, які виконала СУБД, але не вказує, чи вони виконувались у складі одного запиту або декількох окремих. Найшвидше елементарні операції вставки та вибірки даних виконуються у документно-орієнтованих базах даних завдяки структурі збереження даних. Так як NoSQL бази зберігають дані не у таблицях, а у об’єктах, які називаються колекції, звернення до них та обробка даних, які вони зберігають, значно швидша [8].

В незалежності від типу системи – завжди найбільше операцій буде саме вибірки даних. Зважаючи мультизадачність сучасних систем, інформація може використовуватися як користувачами, так і іншими системами або компонентами системи. Наприклад, системи API використовуються виключно іншими системами. Це означає, що важко виділити пряме призначення сучасної системи, тому заздалегідь не зрозуміло, як саме буде використана база даних. Аналітики та

інженери баз даних можуть лише припустити, яким буде навантаження на сервер виходячи зі статистики використання певного класу систем, орієнтованої кількості користувачів та складності функціоналу.

Саме тому бази даних існують розподілені та нерозподілені. Якщо виникає надвисоке навантаження на сервер та починають з'являтися технічні затримки при обробці даних – як правило розподіляють один сервер БД на декілька. Існує декілька популярних рішень створення розподілених баз даних – це повна реплікація бази даних та звернення до них за допомогою автоматичного сервісу пере направлення запитів або використання різних баз даних для окремих частин системи.

Реплікація баз даних має свої переваги та недоліки. Коли у системі існує декілька однакових баз даних – це означає більшу доступність та меншу вірогідність відказу системи. Якщо одна з баз даних перестає відповідати – завжди є її повна копія, яка може бути використана системою. Але проблема виникає при спробі синхронізації даних між розподіленими базами, щоб підтримувати релевантність інформації. Це означає подвійну роботу при записі, оновленні та видалені даних з будь-якої бази. Якщо система проектується таким чином, що цих операцій буде багато – краще одразу відмовитися від реплікацій та створити резервну базу даних, яка буде оновлюватися автоматично у певний проміжок часу.

Створення окремих баз даних у розподіленій системі ефективне за умови використання сервісної або мікросервісної архітектури [9][10]. Таким чином кожен окремий сервіс буде використовувати свою незалежну базу даних, що пришвидшує обробку даних цим сервісом та роботу системи в цілому. Проблеми використання такого підходу виникають коли необхідно проаналізувати роботу системи в цілому або вирахувати статистичні показники. Для виконання цього завдання необхідно збирати дані з кожної бази та проводити аналіз. Більше того, у випадку, коли сервісу необхідно отримати динамічні або статичні дані іншого сервісу – програміст повинен передбачити підключення до іншої бази даних по її назві, що заздалегідь може бути невідомо та спричинить помилку у програмному коді.

Окремим рішенням задачі реплікації бази даних – використання хмарових сервісів по збереженню та обробці даних. На даний момент вони є найефективнішими та дешевими рішеннями проблем не тільки реплікації, а й покращення відмовостійкісті системи, безпеки та доступності систем.

Для невеликих систем розподілення баз даних не потрібне та призведе лише до ускладнення роботи з даними та програмним забезпеченням [11]. Як показано на рисунку 1.4, аркуш 17, розподілення будь-якої прикладної програми на окремі частини (сервіси), в тому числі і розподілені бази даних, оправдано зі збільшенням її складності. Як правило, складність для баз даних можна відстежити за показниками виконання запитів. Якщо запити виконуються за невеликий проміжок часу та база даних може без затримок та черг обробляти всі вхідні запити, то реплікація лише ускладнить систему без видимих переваг для користувача чи розробника.

Рисунок 1.4 – Графік зміни продуктивності системи з ростом її складності

Більшість систем створюється за допомогою готових конструкторів або певної архітектури і як правило не включає в собі можливості розподіляти базу даних. У більшості прикладних програм достатньо використання однієї бази даних та додатково системи пошуку типу ElasticSearch або Redis. Така комбінація технологій дозволяє швидко працювати з даними та продуктивно використовувати всі можливості СУБД. Як правило, якщо потік даних не

перевищує 1 терабайт у день, тоді подібних рішень достатньо. Це і є та сама точка перетину двох графіків, яка сигналізує, що продуктивність системи може різко впасти, що призведе до втрати користувачів, коштів та статистики. Падіння продуктивності не залежить від моделі бази даних, яка використовується.

Більшість баз даних, які розробляються для різного типу систем, де сервери опрацюють мільйони запитів та постійно знаходяться у робочому режимі, очікують, що ви встановите стійке з'єднання із сервером бази даних, перш ніж зможете виконувати запити. Але зараз це не є єдиним рішенням, яке повинно бути присутнє у програмному додатку. Зі старим підходом робити безсерверний спосіб (serverless) ніколи не було проблемою. Встановлюється єдине стійке з'єднання (на приклад у системі), і воно буде повторно використовуватися для подальших запитів.

Однак без сервера, який приймає та обробляє запити, кожен окремий запит є власною ефемерною обчислювальною інстанцією. Для кожного випадку витрачати час на встановлення стійкого підключення до бази даних є непродуктивною, небезпечною та затратною дією, особливо коли кошти витрачаються за процесорні хвилини, або скільки часу працює програмна інструкція. Крім того, оскільки кожен запит потребує власного підключення до бази даних, це також створює значне навантаження на саму базу даних. Якщо у системі в будь-який момент є більше ста відвідувачів вашого, це сто нових підключень до бази даних одночасно.

Бази даних, які потребують таких постійних з'єднань, ніколи не адаптувалися або налаштовувалися для обробки одного підключення до бази даних для єдиного запиту. Насправді більшість баз даних взагалі обмежує кількість з'єднань, з якими вони можуть працювати в певний момент часу. Фактична кількість підключень залежить від різноманітних факторів, і хоча можна припустити, що достатньо високий ліміт повинен відповідати очікуваному трафіку у системі, все, що потрібно, – це банальний стрибок руху, щоб вичерпати цю межу.

При використанні баз даних, наприклад AWS RDS у парі з AWS Lambda, виникає наступна проблема. Найкращі практики показують, що база даних повинна знаходитися у віртуальній приватній хмарі (VPC), щоб захистити базу даних від небажаного публічного трафіку. Щоб отримати доступ до бази даних у такому випадку, програмні додатки, що використовують AWS Lambda, також повинні знаходитись у тому самому VPC. Проблема полягає в тому, що функції

Lambda позаду VPC можуть мати «холодний запуск» (старт виконання програмної інструкції із запуском всього набору бібліотек та сторонніх програмних засобів перш ніж виконати код інструкції) до 10 секунд, що робить їх майже повністю непридатними для більшості програм, орієнтованих на користувачів.

У сучасній архітектурі баз даних, які працюють у хмарі, без серверів, можливо, використовувати сучасні технології хмарних баз даних, які були розроблені з урахуванням хмарних та серверних архітектур. Такий підхід дозволяє економити час на розгортання бази даних та налаштування серверу, так як ці послуги надаються за замовчуванням. Основне, на що потрібно звернути увагу, – це бази даних, які не потребують постійних зв’язків для спілкування. Натомість ці бази даних надають спеціальний API бездротових програм REST для зв'язку, тобто немає обмежень підключення та немає проблем з комунікацією через VPC.

Незалежно від того, чи потрібно використовувати традиційні РСУБД або готові до випробувань та новіших сміливих рішень NoSQL, є чудові варіанти, які представляють як сховища типу «ключ-значення», стовпчасті бази даних або змішані варіанти. Більше того, такі бази даних надають свої переваги у використанні та підтримці. Основним моментом безсерверної технології є виставлення системою рахунків на основі використання ресурсів, які надаються хмарою. Оплата проходить лише за серверні хвилини, які задіяні у виконанні запитів базою даних. Це також є недоліком у порівнянні з використанням відкритих (open source) технологій баз даних, які можуть безкоштовно використовуватися у комерційних цілях. Зазвичай ціноутворення безсерверних баз даних ґрунтується на обсягу сховища та числі обчислень. Деталізація метрик цих обчислень ґрунтується на різних даних та постачальниках цих даних для аналізу, при цьому деякі провайдери дозволяють користувачам переходити до передбаченої потужності при необхідності.

Додатковим бонусом до serverless технології є можливість автоматичного масштабування. Це здатність швидко і без будь-якого втручання людини проводити зміни у використанні технічних характеристик серверу бази даних, що використовується. У попередню еру приватних серверів дизайнерам систем доводилося вибирати певний розмір жорстких дисків, що відповідав проекту системи, процесора та пам’яті. У разі помилки з’являлися проблеми, як: недостача місця у базі даних, або погіршення якості послуг зі збільшенням

кількості користувачів. Або навпаки, при помилці та переплаті за сервери, використання стало би економічно не допустимим, так як машини не використовувалися за призначенням. Безперервне автоматичне масштабування – це засіб вирішення описаних раніше питань. Завдяки технології безсерверних баз даних можна масштабувати практично нескінченний простір на диску і обчислювати за лічені секунди, коли до бази даних потрапляють великі та непередбачувані скачки трафіку. Аналогічно є можливість зменшити масштаб, якщо всі користувачі не в режимі офлайн. Треба пам'ятати, що автоматичне масштабування в поєднанні з розрахунком на основі споживання може мати негативні наслідки, якщо не налаштувати цю функцію належним чином. Одна з головних дискусій навколо безсерверних відбувається навколо холодних стартів. Холодний старт відбувається, коли доступ до ресурсу без сервера вперше або після того, як до нього не можна було скористатися більш тривалий період. Холодні старти мають дуже високу затримку, яка помітна як для користувачів системи, так і для розробників. Холодний старт не є особливою проблемою без наявності сервера, але є мінусом при використанні функцій як платформ для послуг (FaaS). Наприклад, безсерверні бази даних не мають холодних стартів і можуть виконувати запити та відправляти відповіді з низькою затримкою навіть після того, як перебувають у режимі очікування протягом довгого періоду часу.

Використовуючи serverless технології баз даних не потрібно думати про операції, які не пов'язані з цією технологією, а які є частиною хмарних та керованих сервісів. За допомогою керованих сервісів не потрібно купувати додаткове обладнання, якщо воно застаріло або вийшло з ладу, і не потрібно оновлювати програмне забезпечення бази даних, коли випускаються оновлення та виправлення помилок. У просторі без сервера багато сервісів пропонують автоматичне резервне копіювання даних, тому вони не втрачається, якщо будь-що виходить з ладу або хтось із користувачів викликав помилку виконання чи відправив невірний запит, що призводить до втрати даних. Ще один важливий аспект, який багато розробників та проектувальників ігнорують, це коли мова йде про безпеку бази даних. Можна звикнути, що у сучасному світі всі мови програмування драйвери, фреймворки та навіть самі СУБД застосовують захист до даних та запитів між серверами, але цього не достатньо для забезпечення належного рівня безпеки даних. Як і технічне обслуговування, воно не пов'язане безпосередньо з продуктом, який розробляється, але може така помилка може виявитися дорожче, якщо хтось викраде дані користувачів [12]. Більшість

безсерверних баз даних захищена за замовчуванням і постачаються із шифруванням під час передачі даних по мережі, деякі з вибраними розробниками шифруванням, яке може бути використано на даних до того, як вони будуть передані до кінцевої точки обробки даних (клієнт або сервер). Такий підхід дозволяє отримати належний рівень безпеки даних без перевірки запитів, або експертизи професіональних архітекторів.

Хоча більшість безсерверних баз даних забезпечують деяку відповідність ACID, що дозволяє транзакціям, що охоплюють цілий кластер баз даних, виконуватися згідно зі стандартів забезпечення безпечного виконання транзакцій. Вони часто мають значні компроміси, особливо з точки зору продуктивності. Виконання запитів та перевірок у фоновому режимі, щоб згодом дозволити відкат, може перешкодити іншим клієнтам отримувати доступ або змінювати дані. Використовуючи алгоритм Кальвіна, компроміси зводяться до мінімуму, щоб клієнти не отримали несвіжих даних і не раптом побачили падіння продуктивності СУБД, оскільки інші клієнти розпочали транзакцію.

Усі функції баз даних без сервера надають розробникам будь-якої частини системи велику потужність, не потребуючи навичок розробки. Ніколи не було простіше створити повнорозмірні програми. Базу даних можна налаштувати одним натисканням кнопки і почати завантажувати в неї дані. Вся СУБД без марних та довгих зусиль буде масштабуватися та архівуватися. Якщо у системи немає користувачів, ресурси не використовуються і безкоштовно чекають на появу запитів. І у той самий час можливе різке масштабування бази даних у випадку великого напливу користувачів. Існує поняття найнижчого навантаження, оптимального навантаження та надзвичайного. У випадку з технологією serverless ситуація відмови працездатності бази даних через високі навантаження ніколи не станеться. Так як незважаючи на розрахунки проектувальника бази даних, масштабування відбувається автоматично та здатне витримати майже будь-яке навантаження, яке досі виникало у великих системах. Найголовніше, що не потрібно витрачати цикли на виправлення помилок та зламане обладнання, тоді як можна впроваджувати наступну функцію до системи. Таким чином, без серверні технології варто використовувати для заощадження часу та ресурсів при створенні або експлуатації систем, вони підходять для проектів будь-якого масштабу та швидко набирають популярність перед товариств розробників.

Отже, існує проблема вибору СУБД на етапі проектування нової системи, так як заздалегідь невідомі масштаби даних, які будуть оброблятись. Більшість сучасних СУБД мають схожі характеристики та можуть однаково швидко виконувати базові CRUD операції на невеликих обсягах даних, що зберігаються. З ростом бази даних з'являються додаткові проблеми масштабування, безпеки та швидкості передачі даних, можливість СУБД обробляти потоки даних. Так як існує велика кількість технік та парадигм створення систем, сервер бази даних повинен підтримувати хоча б найпопулярніші із них. Також для більшості систем, які працюють по технології «сервер-сервер», тобто API, необхідно використовувати базу даних, як сервіс, або навіть безсерверну базу даних, яка знаходиться у хмарі.

Велика кількість показників застосовується для прийняття рішення, що призводить до уповільнення процесу розробки та можливих описаних проблем у майбутньому, що призводить до значних втрат часу, коштів або навіть даних. Помилки зі зберіганням або обробкою даних є найкритичнішими та затратними у всьому процесі створення та роботи будь-якої системи. Зважаючи на те, що заздалегідь невідомо, які технології будуть використовуватися в проекті, складно спроектувати модуль взаємодії СУБД та системи. Враховуючи той факт, що продуктивність тієї чи іншої бази даних залежить від типу та об'єму даних, що зберігаються, архітекторам баз даних необхідно розуміти модуль поведінки бази даних при різних навантаженнях.

Так як для роботи з великими даними часто використовують розподілені системи, з'являються розподілені бази даних, які об'єднують у собі декілька типів СУБД для різних типів даних. Це новий підхід, який не отримує достатньої уваги та його ефективність не тестувалася. Тому при проектуванні бази даних, архітектори часто йдуть на ризик, приймаючи рішення, які повинні бути актуальними на протязі всього функціонування майбутньої системи. Для отримання чітких показників продуктивності різнорідних СУБД при різних обсягах даних, що зберігаються, необхідно провести поглиблений порівняльний аналіз баз даних, їх характеристик та властивостей, типів та додатків, а також можливостей та продуктивності обробки запитів різного рівня складності. Експеримент оснований на попередніх роботах за даною тематикою [13][14][15][16], доповнює їх завдяки визначенню та проведенню налаштувань СУБД та обсягу даних у базах.

2 ПОСТАНОВКА ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Мета: сформулювати, вибрати та визначити критерії продуктивності виконання СУБД запитів різної складності, від простих CRUD операцій до об’єднання запитів. Вивести способи оцінки продуктивності СУБД на основі обраних критеріїв. Визначити налаштування бази даних, кешування та оптимізатору запитів, які впливають на виконання запитів. Обрати дві найбільш поширені моделі баз даних та провести тестування СУБД з проведенням налаштувань та за їх відсутності. Виходячи з особливостей звернення до кожного з типів СУБД – перевірити, наскільки впливає використання тих чи інших типів запитів і налаштувань на їх роботу, а також на швидкість обробки запитів. Завдяки системному підходу [17][18] визначити наскільки налаштування СУБД та сховищ даних впливає на час виконання запитів різної складності. Проаналізувати, як впливає на виконання запитів використання серверу бази даних, або сервіс бази даних у хмарі (DBaaS). На основі проведеного експерименту дати практичні рекомендації щодо підвищення продуктивності роботи СУБД та виконання запитів до великих даних за умов розподілених баз даних в залежності від різних умов експлуатації системи,а також використання певної моделі бази даних, або комбінації різних моделей всередині системи.

Вихідними умовами є розгорнутий сервер бази даних на локальній машині з характеристиками: Intel Core I7-8550U 1x8 cores, 8GB DDR3 RAM, PM961 SSD 256 GB (читання: 2800 MB/s, запис: 1100 MB/s). У якості програмного забезпечення були використані CentOS Linux 7.6.1810 (Core), MariaDB 10.3 X64, Studio 3T, Navicat for MySQL, MySQL Workbench, MongoDB 4.0 X64, Mongo Shell.

Обмеженнями при виконанні роботи можна вважати терміни роботи, програмне забезпечення та характеристики машини, на якій проходять тести, та версії СУБД, які використовуються.

Критерії ефективності розробки – це виведення і відмінність числових показників швидкості виконання запитів різної складності до розподілених СУБД різного типу із застосуванням параметрів налаштування та за їх відсутності. Основним критерієм є можливість зробити висновки щодо доцільності використання певної моделі бази даних, або їх комбінації, для роботи з великими даними на основі запитів, які будуть використані.

3 ЗМІСТОВИЙ ОПИС ТА АНАЛІЗ РІЗНИХ ТИПІВ СУБД І ОСОБЛИВОСТІ ЇХ ЗАСТОСУВАННЯ

Популярними моделями СУБД є реляційна та нереляційна, яка може бути у вигляді графа, «ключ-значення», але частіше всього – документно-орієнтована. Нереляційний підхід з’явився з підвищенням гнучкості систем та мов програмування з метою зменшити час обробки даних та підвищення зручності її використання у програмному коді. Зазвичай замість таблиць такі моделі використовують об’єкти, які схожі за структурою на об’єкти у мові програмування, таким чином їх використання не потребує додаткової конвертації, що інколи виникає при роботі з реляційними базами. Більше того, об’єкти мають властивість описувати себе самостійно, тобто вони не містять метаданих, які зберігаються окремо для кожної таблиці в РСУБД.

Не випадково ці дві моделі конкурують між собою, так як вони пропонують схожі інструменти для роботи з даними. Яскравими представниками цих моделей баз даних є MariaDB та MongoDB, так як кожна з них стала проривною у галузі баз даних. У більшості проектів так чи інакше використовується один з описаних вище підходів, в залежності від поставленої задачі. Розглянемо ці СУБД детальніше.

Реляційну MariaDB використовують у разі необхідності побудови системи з чітким розподілом даних, навіть великого об’єму. Ця СУБД являє собою заміну стандартного MySQL більш інноваційним, швидким двигуном, який дозволяє зберігати та обробляти більшу різноманітність даних [19].

Нереляційну MongoDB варто використовувати у випадках, коли реляційні моделі занадто важкі для побудови архітектури або система має занадто багато параметрів для користувачів, які вони можуть змінювати, додавати, видаляти або копіювати. Як правило, це системи для створення оригінального контенту, такий як сайти, шаблони, графіки або діаграми, тощо. Другим і найважливішим є те, що не реляційні бази даних легко масштабуються. У порівняні з реляційною моделлю, не потрібно додавати жодних зусиль, а лише додавати дані [20]. Але якщо у розробника бракує досвіду у створенні масштабних систем, то може з’явитися проблема перенасичення даних, або навпаки пустих значень у колекціях, що значно знижує ефективність СУБД при використанні на великих даних.

Кожна система має свої архітектурні особливості, які стають очевидні на етапі проектування. Саме тоді слід вибирати яку СУБД буде використовувати система. Кожна система потребує свій тип бази даних, унікального не існує. Перед інженером стає вибір, який вирішує наскільки просто чи складно буде масштабуватися система, як буде проходити процес обміну даним між клієнтом і сервером, який тип даних буде використовуватися та як дані будуть оброблятися. Це означає, що на вибір СУБД впливає багато факторів і треба досконало розуміти сильні та слабкі сторони хоча б декількох із них [21].

Розглядаючи MariaDB слід розуміти різницю між нею та MySQL. MariaDB підтримує більшу кількість двигунів для таблиць даних, має багато налаштувань та параметрів покращення продуктивності, які відсутні в оригінальній СУБД. Ще більше додаткових покращень та оновлень стандартних функцій було добавлено у останню на даний момент версію 10.4 [22]. Завдяки спеціальним двигунам та типам даних, MariaDB здатна ефективно зберігати та працювати з даними будь-якого існуючого типу, обробляти різнорідні дані всередині навіть однієї системи, не втрачаючи у швидкості виконання запитів. Програмні покращення у СУБД вносилися розробниками MySQL та товариством інженерів баз даних.

Однак використання MariaDB не є достатнім для значного покращення продуктивності системи. СУБД має велику кількість налаштувань під кожну складову баз даних, запитів, внутрішніх функцій та обробки даних. Для таблиць можуть бути використані різні технологічні рішення, в залежності від типу даних, які в них зберігаються. За замовчуванням, всі таблиці створюються з двигуном InnoDB, який є оптимальним в середньому випадку. Такий вибір робиться за рахунок того, що у InnoDB швидше виконуються операції вставки та оновлення даних, доступні транзакції, інкрементація індексу вбудована за замовчуванням і підтримка можливості задати зовнішній ключ (foreign key). На вибір також доступні MyISAM, Memory, CSV і інші типи двигунів, які використовують занадто рідко. До основних двигунів також долучається MyISAM, так як він дозоляє індексувати FULLTEXT поля, що значно пришвидшує пошук по тексту, і взагалі продуктивність роботи цього двигуна більше за стандартного InnoDB, тому MyISAM вибирають для невеликих проектів. Це пов’язано з тим, що MyISAM двигун простіший, він не підтримує модель ACID, але зберігає на рівні таблиць, тобто запити щодо інформації про таблиці зберігаються у метаданих та не потребують повторного сканування всієї таблиці [23].

СУБД підтримує всі стандартні механізми зберігання даних, які доступні у MySQL, але має і ряд нових, серед яких: ColumnStore, MyRocks (механізм з відмінною компресією даних), Aria (є заміною MyISAM з рядом переваг і оновлень), FederatedX, OQGRAPH, SphinxSE, TokuDB, CONNECT, SEQUENCE, Spider, Cassandra [24]. Всі ці механізми роблять із MariaDB мульти-модельну базу даних, яка є універсальним інструментом для розробки будь-якої системи.

У базі даних можливо зберігати всі існуючі типи даних. Дані зберігаються у таблицях, які можуть бути логічно пов'язані між собою за допомогою ключів. Поля таблиць можуть мати індекси, унікальні значення, бути зовнішніми ключами до інших таблиць. Реляційна модель передбачає наявність зв'язків типів «1-1», «1-many» або «many-many». Завдяки зв'язкам можна логічно описати як саму базу даних, так і таблиці, відносини між ними та спроектувати модель потоку даних, який визначає принципи роботи базі даних.

Більш важливою стає умова використання бази даних не як невід'ємної частини системи, а як сервісу. MariaDB може бути використана в якості сервісу та розміщена навіть на сторонніх хмарових сервісах. Різниця між базою даних як сервіс (DBaaS) і просто базою полягає у розміщені, управлінні та побудові бази даних для систем. У випадку, коли база даних є лише сервісом – всі операції проводить провайдер, тоді як розробники лише користуються базою даних. Таким чином усі зміни до бази даних, архітектури чи таблиць, можуть бути виконані лише за участі третіх осіб, що не завжди грає позитивну роль як під час розробки, так і у разі виникнення критичних ситуацій надалі.

До бази даних можливо додати тригери та події, які будуть виконуватися за приписаними правилами. Правило пишуться на мові SQL та виконуються, коли відбувається певна подія, настає дата та час, або за інших умов, які задасть інженер бази даних. Вони корисні для підтримання логіки системи, ведення статистики чи аналізу, проведення очищень таблиць або їх копіювання, підбиття проміжних підсумків. Зазвичай, більшість таких операцій можливо реалізувати на стороні мови програмування, але більш продуктивними вони завжди будуть на сервері бази даних, так як споживають менше ресурсів.

Найкращою NoSQL СУБД на даний момент є MongoDB. Це документно-орієнтована СУБД з відкритим вихідним кодом, яка розроблялася для спрощення процесу розробки та розуміння бази даних взагалі. Вона використовує документну модель зберігання даних, тобто схожу на JSON структуру об'єктів (BSON, який згадувався раніше), які потім легко перетворювати у об'єкти для

більшості мов програмування. Такий підхід надає можливість легше розуміти та читати дані, які зберігаються у базі, а також маніпулювати ними використовуючи елементарні запити. Крім того, JSON дозволяє передавати дані між сервером і системою з використанням зручного формату, який швидко обробляється мовами програмування і імплементується користувачу [25]. Такий формат даних також є гарним вибором з точки зору розміру, швидкості обробки даних, оскільки він забезпечує більшу ефективність і надійність, ніж більш складні типи. З'явився формат на початку двохтисячних років і активно розвивається досі за рахунок відкритої роботи товариств розробників, закріпивши за собою статус одного з найбільш широко використовуваного формату даних для передачі по мережі, обігнавши у популярності XML з виходом стандарту Web 2.0, що показано на рисунку 3.1. Дані для аналізу взято за останні 5 років в період з 2015-2019 роки.

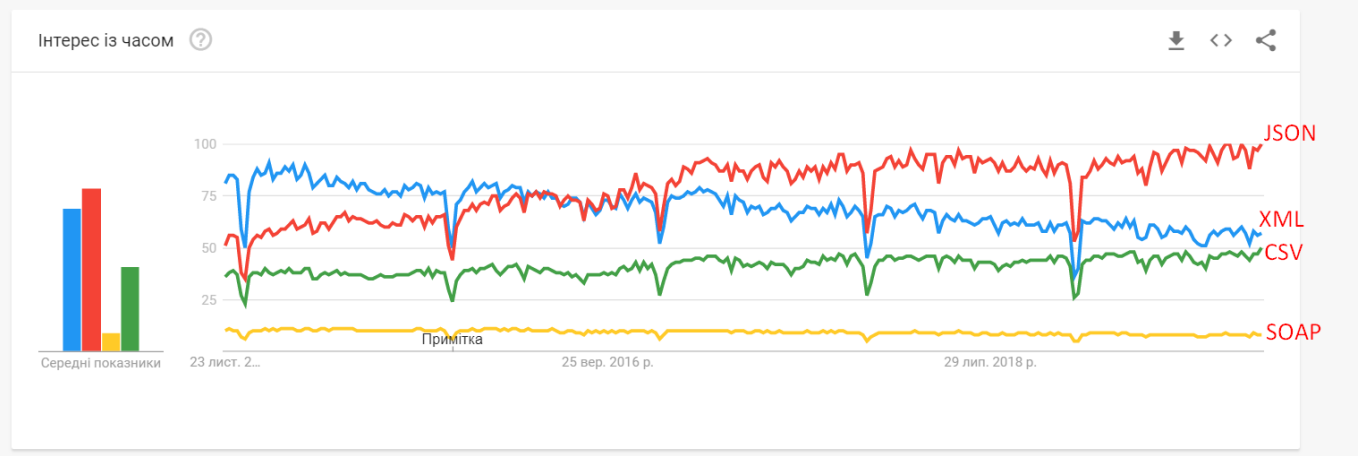

Рисунок 3.1 – Зміна популярності формату даних для передачі по мережі

У базі даних формуються колекції, які зберігаються замість таблиць, як у реляційних моделях. Перевагою колекцій є те, що вони описують самі себе, тобто не зберігають додаткової інформації про себе та типи даних, що вона зберігає. Таким чином, колекції займають рівно стільки місця, скільки даних зберігають. У форматі JSON можна зберігати більшість необхідної інформації, яка може оброблятися сучасними програмами. Завдяки єдиному формату збереження даних, мінімізується вірогідність допустити помилку в процесах обробки даних, що підвищує стійкість СУБД до помилок у потоках даних. Незважаючи на це, на відміну від MariaDB, це не є мульти-модельна база даних, тому як вона може зберігати та працювати лишу з колекціями. Лише використовуючи MongoDB як

DBaaS є можливість використовувати індекси, що показує наявність іншої модулі у базі даних, яка можливо буде додана у наступних версіях.

Передаючи та обробляючи дані у форматі JSON, сервер бази даних не витрачає час на перетворення даних. Одним з головних переваг, запропонованих MongoDB, є використання динамічних схем, які усувають необхідність попередньо визначати структуру, наприклад поля або типи значень. Така модель забезпечує подання ієрархічних відносин, зберігання масивів і можливість змінювати структуру записів, просто додаючи або видаляючи поля. Це NoSQL-рішення поставляється зі вставкою, автоматичним розподілом і вбудованою реплікацією для кращої масштабованості і високої доступності. Саме такі фактори роблять MonogDB потужною, гнучкою і легко масштабованою базою даних загального призначення. MongoDB поєднує в собі вторинні індекси, запити з діапазонами і сортуванням, агрегацією та геопозиційні запити, які можуть бути використані у високонавантажених системах і при великому обсязі даних, що зберігаються. При цьому розробник не звертає пильної уваги на те, чи дані будуть зберігатися або оброблятися всередині бази даних. Оптимізація запитів проводиться оптимізатором (програмою) та проходить у фоновому режимі, але може бути налаштована у СУБД.

Так як база СУБД використовує популярний формат даних – вона може взаємодіяти з більшою кількістю мов програмування, ніж реляційний аналог MariaDB. Більше того, для взаємодії не потрібні жодні інтерфейси або модулі, так як воно відбувається на базовому рівні. Перевагою MongoDB є те, що процедури на стороні сервера можуть відбуватися за написаним Javascript кодом.

Незважаючи на те, що MongoDB – нереляційна база даних, вона підтримує свою мову виконання запитів. Вона дозволяє фільтрувати, сортувати або робити вибірку по будь-яких даних, які зберігаються у базі, незважаючи на розмір та тип об’єктів. Підтримує агрегації та інші сучасні методи використання даних, такі як пошук на основі географії, пошук графіків та пошук тексту. Запити самі по собі JSON, і тому їх легко компонувати, не використовуються також операції об’єднання запитів та рядків таблиць, які були причиною затримок у виконанні запитів у SQL.

Сама СУБД додає багато корисних інструментів по роботі з даними. Вона являє собою справжню платформу даних, яка має всебічний набір інструментів, щоб зробити роботу з даними простіше та швидше. Наприклад вбудовані інструменти аналітики дозволяють збирати статистику, будувати графіки,

візуалізовувати дані у реальному часі та реагувати дані через візуальні інтерфейси.

Використовуючи MongoDB: Atlas можна створити DBaaS систему, яка буде розподіленою та доступною у будь-якій точці світу завдяки хмаровим технологіям сервісів, які вона використовує [26]. Сервіс дозволяє використовувати мережу доставки контенту (CDN серверів) для підвищення надійності, швидкості та спрощення роботи з даними.

Масштабування бази даних у MongoDB виконується простіше за рахунок обмеження можливостей як шардінгу, так і реплікації. У цій СУБД доступна лише реплікація типу Master-Slave, тоді як у реляційній MariaDB – додатково доступна реплікація Master-Master, а шардінг дозволяє створювати не лише окремі сервера, а й кластери, розбиваючи дані за ключами, хешем, списками або зонами. Способи створення розподілених баз даних показано на рисунку 3.2. При створені розподіленої бази даних реляційної моделі – інженеру треба допускати вірогідність розриву зовнішніх ключів, зміну структури таблиць або навіть архітектури бази даних, що зазвичай займає багато часу та зусиль. У не реляційній моделі достатньо лише розподілити колекції по різних зонах, що можна зробити навіть без спеціальних знань та технічної підготовки.

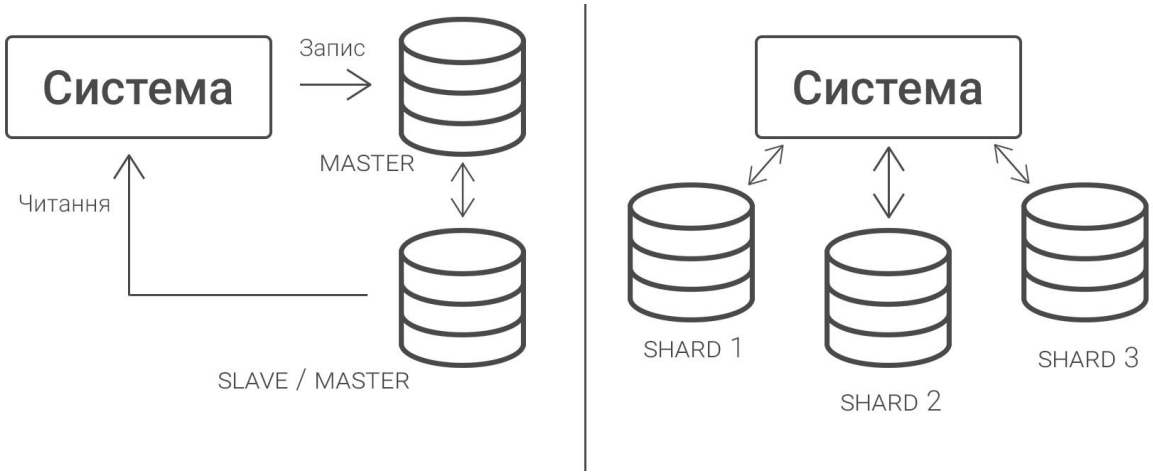

Рисунок 3.2 – Методи створення розподіленої бази даних

Реплікація буває декількох типів: Master-Slave та Master-Master. У першому випадку створюється повна копія бази даних, в яку постійно копіюються всі дані, які записуються чи оновлюються. Читання даних системою відбувається з копії бази даних, таким чином роз’єднуючи навантаження по категоріям операцій.

Копія бази даних не виконує жодних операцій та ніяк не пов’язана з активною базою даних. Другий тип реплікації дозволяє базі даних, яку скопіювали, виконувати ті самі операції, що і основна база. Використовуючи такий підхід – збільшується число операцій між базами даних, що уповільнює роботу системи. Запити між системою і базами даних можуть дублюватися програмно або на стороні серверу СУБД. Але у результаті отримуємо більш надійну систему, яка швидше обробляє запити. Треба звернути увагу на те, що реплікація – це слабкий інструмент для масштабування, так синхронізація та підтримання даних у актуальному стані – це ресурсномісткі операції. Тому зазвичай реплікація використовується разом із шардінгом.

Шардінг дозволяє розподіляти базу даних по частинах так, щоб кожну з них можна було розмістити на окремому сервері. Розподілення таблиць або колекцій залежить від структури бази даних та виконується у самій системі. Кожна новоутворена база даних має запити на запис/читання, та може самостійно відповідати системі. Відрізняють горизонтальний та вертикальний шардінг. Горизонтальний полягає у розподіленні таблиці даних на окремі сервери, при чому на них утворюється одна й та сама таблиця, але доступ до неї редагується архітектурою системи. Наприклад, запити непарних користувачів обробляються одним сервером бази даних, а парних – іншим. Таким чином на різних серверах зберігаються різні дані, що у разі відмови серверу призведе до втрати даних. Як правило, горизонтальний шардінг застосовується тільки до таблиць, які мають від 10 мільйонів записів. Вертикальний шардінг – це виділення таблиці або групи таблиць на окремий сервер для зниження навантаження за умови великої кількості запитів в одному потоці. Допускається сумісне використання шардінгу різних типів.

При роботі з великими даними – розподілення бази даних є невід’ємною частиною побудови архітектури системи ще на етапі проектування. Зважаючи на зріст кількості запитів та потоків даних, налаштування сховища та можливість масштабування набувають критичної важливості. Таким чином з’являється можливість не втрачати швидкість обробки запитів та час взаємодії системи с базою даних, так як потужність та кількість серверів зростає. Використовуючи хмарні рішення – існує можливість автоматичного росту кластеру серверів баз даних в залежності від навантаження, яке створюють користувачі. Автоматичне масштабування заощаджує трафік, використання матеріальних ресурсів, серверів, а автоматичний режим реплікацій і шардінгу створює сприятливі умови

зберігання даних. Такий підхід підвищує надійність системи в цілому, так як будь-яка різка зміна кількості запитів у певний момент часу не зможе вплинути на здатність системи швидко функціонувати. Після стрибків трафіку сервера баз даних продовжать функціонувати.

Основні результати порівняння та ключові характеристики двох моделей баз даних внесено до таблиці 3.1. В умовних позначеннях таблиці SSP розшифровується як скрипти, що виконуються на стороні серверу (Server Side Scripts), тобто інструменти, які виконуються для адміністрування та управління внутрішніми процесами у СУБД. Також приведена кількість мов програмування, які підтримують різні бази даних (Supported Programming Languages) без встановлення додаткового програмного забезпечення.

Таблиця 3.1 – Порівняльні характеристики СУБД MariaDB і MongoDB

	Тригери	Індекси	DBaaS	APIs	SSP	SPL
MariaDB	Так	Так	Azure, MS	4	PL\SQL	19
MongoDB	Ні	Так	Atlas	1	JS	27

За підсумковою таблицею наочно видно, що порівнювані СУБД обробляють запити по-різному, так як використовують різні технології. MariaDB більш функціональна для обробки інформації при будь-яких подіях, які відбуваються у базі даних, у той час як у MongoDB ти самі дії можна виконати лише програмно, завдяки підтримці великої кількості мов програмування. Для автоматизації роботи РСУБД достатньо знати єдину мову запитів – SQL, на якій пишеться усе, що стосується бази даних, а у випадку з MongoDB потрібно додатково знати Javascript для написання того самого функціоналу.

Значною відмінністю цих систем управління базами даних полягає у взаємодії з мовами програмування. Для кожного типу СУБД, що розглядаються, у кожній мові, які їх підтримують, мається спеціальний драйвер, який відповідний за підключення, взаємодію та виконання програмних інтерфейсів. У якості приклада візьмемо мову PHP, тому як вона підтримує обидві СУБД. Кожний драйвер для взаємодії програми та бази даних реалізовує підключення, сесію для використання команд або інтерфейсів взаємодії, такі як кеш, помилки, оновлення даних, тощо. Для MariaDB розроблено значно більше методів, так як вона дозволяє управляти не тільки запитами та потоками даних, а й самою СУБД, підключенням та даними [27]. Для використання у останніх версіях мови

програмування – необхідно обов’язково мати також останні версії MySQL. У той час як драйвер для документно-орієнтованої СУБД значно простіший, для встановлення не потребує контролю версій, та для налаштування має лише один параметр логу помилок. Перетворення програмних команд у команди для СУБД виконується однаково швидко в обох випадках, так як програмне забезпечення завантажує усі драйвери при першому запуску програми. В подальшому використанні між ними немає суттєвої різниці з точки зору використання ресурсів машини.

Важливою характеристикою є безпека запитів, які передаються між системою та базою даних. Так як типи запитів та дані, які передаються, різні в обох випадках – захист даних також відрізняється. В іншому випадку – міняється виконавчий скрипт у програмі, який міняє структуру даних, які прийшли з серверу бази даних. Якщо з’єднання відкрите та стабільне, що у загальному випадку завжди буває при використанні будь-якої мови програмування за винятком випадків, коли розробник самостійно закриває з’єднання, тоді є можливість виконання шкідливого скрипту. У мові програмування PHP є можливість захисту від подібних типів атак, яка полягає у використанні програмних інтерфейсів, які перевіряють стан об’єкту, який зберігає дані з бази із копією структури об’єкта, що зберігається у захищеному сховищі. Перевірка проходить у режимі реального часу, тобто використовуючи додаткові функції драйверу для кожної СУБД можливо міняти структуру запитів лише програмно на стороні сервера. Але для написання безпечного коду слід використовувати певні структури, які можна знайти у документації. Більше практик по захисту даних при використані СУБД можна знайти на офіціальних сайтах та у документації самих баз даних, де прописані всі відомі випадки втрати даних та злому та прописані інструкції, як цього запобігти при розробці систем різного типу.

Зважаючи на те, що MySQL з’явилася раніше, до неї було написано більше інтерфейсів для взаємодії програмного коду та самої бази даних. Завдяки постійному з’єднанню, можливо спроектувати систему таким чином, що вся СУБД буде управлятися за допомогою програмного забезпечення, яке розробляється. Тобто, розробнику доступні всі можливості СУБД з програмного коду, які застосовуються до самої бази даних. Недоступна опція – це налаштування самої бази даних, та усіх внутрішніх процесів, у тому числі оптимізатора запитів.

4 ОСОБЛИВОСТІ ПРОЕКТУВАННЯ РІЗНИХ ТИПІВ БАЗ ДАНИХ І МЕТОДИ ЇХ ЗАПОВНЕННЯ

Проектування реляційних бази даних суттєво відрізняється від документно-орієнтованих. Це пов’язано з підтримкою забезпечення цілісності даних після неатомних маніпуляцій з даними, які зберігаються у базі даних. У реляційних моделях використовується поняття ACID (Atomicity, Consistency, Isolation, Durability), яке означає наявність атомарності, узгодженості, ізольованості і довговічності даних. Схожа модель використовується і для NoSQL баз даних, яка називається мульти-документні ACID транзакції, але вона модифікована для роботи з різними документами. Так як сутність документів – це логічне розподілення різнорідної інформації для швидкого доступу до неї, спочатку не передбачалося одночасне опрацювання даних з різних документів. Хоча ця функція присутня і використовується скоріше як виключення, так як колекції проектуються з оглядом на незалежність кожного документа, як носія унікальної для бази даних інформації.

Принципи атомарності при проектуванні баз даних дозволяють бути впевненим у цільності даних, які зберігаються та обробляються. Під цим мається на увазі, що при виконанні запиту до бази даних або вся транзакція виконується одразу, або взагалі не проходить. Не може бути середнього значення, тобто транзакції не проходять частково, база даних не приймає тимчасових значень чи змін. Кожна транзакція розглядається як єдине ціле і обробляється до завершення, або взагалі не виконується. Атомарність передбачає лише дві прості операції: припинення і коміт (abort & commit). Якщо транзакція припиняється програмно чи із-за з’єднання між базою даних та сервером програми, зміни в базі даних не відбуваються. У випадку коли транзакція успішно завершується та виконується остання команда коміт, тоді зміни з’являються у базі даних.

Узгодженість даних також є фундаментальним принципом роботи реляційної бази даних. Вона означає, що обмеження цілісності повинні підтримуватися таким чином, що база даних була узгодженою до та після вхідної транзакції. Це стосується правильності побудови бази даних. Враховуючи атомарність запитів та транзакцій до бази даних, стан даних у базі повинен бути однаковий до і після транзакції. Якщо хоча б одна за них не виконується, транзакція вважається незавершеною.

Забезпечення неушкодженості даних під час паралельних запитів можливе завдяки властивості ізольованості. Це є важливою властивістю вважаючи факт використання потоків даних та великої кількості одночасних підключень до сучасних систем. Ізоляція гарантує, що кілька транзакцій можуть відбуватися одночасно, не призводячи до стану невідповідності бази даних реальним даним, які змінюються чи оновляється запитами користувачів. Запити виконуються СУБД самостійно, без втручання між собою. Зміни, що відбуваються в певній транзакції, не будуть помітні для будь-якої іншої транзакції, поки ця конкретна зміна цієї транзакції не буде записана у базу або не буде здійснена завдяки операції коміт. Така властивість гарантує як користувачам, так і розробникам, що виконання операцій одночасно не призведе до ситуації, коли виконання запиту принесе неочікуваний результат. Всі операції проводяться на основі даних, які були взяті з бази у момент початку транзакції.

Довговічність даних має на увазі гарантію того, що при успішному завершені будь-якого запиту – база даних збереже ці зміни. Така властивість гарантує, що після завершення транзакції оновлення та модифікації – усі зміни зберігаються та записуються на диск, навіть якщо трапляється збій системи. Будь-які оновлення бази даних стають постійними і зберігаються в енергонезалежній пам’яті. Ефекти транзакції, таким чином, ніколи не втрачаються.

Властивості ACID, у сукупності, забезпечують механізм забезпечення коректності та узгодженості бази даних таким чином, що кожна транзакція є групою операцій, яка діє єдиним механізмом опрацювання даних у базі, дає стійкі результати, діє ізольовано від інших операцій та у результаті записує зміни до сховища, які точно зберігаються там при будь-яких умовах. Ці властивості дуже важливі в умовах використання серверів баз даних в умовах високонавантажених систем. Коли велика кількість користувачів одночасно робить більше тисячі запитів до бази даних – вона повинна встигнути їх обробити та забезпечити актуальність даних, які передаються користувачам, або змінюються користувачами. Більше того, завдяки ACID принципам можливо будувати системи з великою точністю, наприклад для аеропортів, космічних програм, тощо, де необхідно бути впевненим у достовірності даних у певний момент часу, за умови, що ці дані безперервно змінюються різними джерелами. Ці принципи діють незмінно навіть в умовах розподілених реляційних баз даних, що дає можливість швидкого масштабування завдяки шардінгу чи реплікаціям.

Документно-орієнтовані бази даних використовують схожу модель для своїх транзакцій, які проходять декілька документів. Як правило, це використовується у рідких випадках за рахунок архітектури такого типу баз даних. Якщо реляційні бази даних моделюють дані певного суб'єкта у кількох записах та таблицях з відношенням батьківська таблиця–дочірня таблиця, тоді транзакцію потрібно розширити, щоб охопити ці записи та таблиці. Таким чином, дані нормалізуються у декількох таблицях, а запит до бази даних набуває значної складності та виконується за більший час, ніж потрібно для зміни невеликої кількості даних. У випадку, якщо дані у базі будь-яким чином змінюються, наприклад, якщо користувач оновив їх у системі, тоді необхідно буде оновити кілька таблиць у транзакції «все або нічого», яка повинна торкатися кількох таблиць. Часті або одночасні такі операції займають багато часу на виконання у реляційних базах даних. NoSQL бази даних відрізняються у підході до обробки різнорідних даних. Замість того, щоб розбивати пов'язані дані та розділяти їх на декілька таблиць, колекції можуть зберігати пов'язані дані разом у легкодоступній, типізованій, ієрархічній структурі, включаючи піддокументи та навіть масиви.

У MongoDB не використовуються принципи ACID, так як по іншому побудована обробка запитів на рівні СУБД. Вона надає існуючі властивості транзакцій, які розміщені на рівні колекції. За одну операцію одне або більше полів можуть бути записані з оновленнями для декількох піддокументів та елементів будь-якого масиву, включаючи вкладені масиви. Гарантії, які надані СУБД, забезпечують повну ізоляцію по мірі оновлення документа, а будь-які помилки викликають відміну операції та повернення документа у стан, який був до початку операції. Таким чином користувачі отримують послідовний перегляд документа та релевантні дані. Завдяки такому дизайну власники систем отримують ті ж гарантії цілісності даних, що і системи, які використовують реляційні бази даних.

Проблеми обробки великих даних з'являється з ростом кількості баз даних, їх реплікацій та необхідності підтримки оновленості даних в усіх джерелах. Так як ACID застосовується для всіх операцій у реляційній моделі, проблем з масштабуванням не виникає. У NoSQL підході швидкість обробки даних значно знижується. Більшість реплікацій баз даних повинні обробити запити та застосувати зміну даних до того, як основна база даних обробить та відправить повідомлення про успіх запису для користувача. Транзакція залишатиметься

заблокованою, поки щонайменше одна з реплікацій не витягне оновлення з основної бази даних, використовуючи асинхронну реплікацію, сприйнятливу до непередбачуваного відставання реплікації, особливо при сильному навантаженні системи взагалі. Це робить будь-який запис транзакцій у MongoDB повільніше, так як не використовується синхронна реплікація. Будь-яка операція, яка повинна затонути більше одного документа – виконується на порядок повільніше, ніж обробки такої ж кількості таблиць у РСУБД.

Таким чином, документні бази даних мають підвищену вартість мульти-транзакцій. У MongoDB в останній версії 4.0 з’явилися одноразові транзакції, які були описані вище. Однак, зберігаючи оригінальну архітектуру заточування та реплікації даних, недоторканою вона зменшила придатність своїх транзакцій до високопродуктивних систем із суворими затримками та пропускною спроможністю. Будь-яка система, що використовує одноразові транзакції MongoDB 4.0, має вирішити проблеми: горизонтального масштабування, високої затримки між початком обробки запиту та отриманням відповіді у розподілених системах та низької пропускної здатності, що призводить до додаткових затримок при обробці потоків даних, особливо великого розміру.

Враховуючи різницю у обробці запитів, необхідно створити бази даних, які б мали схожу за складністю структуру. Існують практичні випадки, коли документно-орієнтована база працює швидше за реляційну, та при реплікації баз даних, може бути навпаки. За основу візьмемо простий приклад веб-сервісу по залишанню коментарів, який часто зустрічається як складова великої веб-системи, що використовує мікросервісну архітектуру. Структурно, база даних буде складатися з трьох окремих елементів (таблиць та колекцій): користувачі, коментарі та роль у системі, яка визначає, чи може користувач залишити коментар. Для того, щоб визначити якомога точніші результати, будуть використовуватися найпростіші типи даних, які підтримуються у обох СУБД. Також, ці типи даних найпоширеніші у використанні на практиці, що робить тестування приближеним до спостереження за реальною системою [28].

Для реляційної бази даних спроектовано 3 таблиці: clients, clients_comments та clients_access, як показано на рисунку 4.1, сторінка 37. Таблиця clients зберігає в собі: первинний ключ ID (primary key), поле типу big int, ім'я користувача в системі у форматі varchar(80) і його E-mail, також в varchar(80). Тип varchar найчастіше використовується для зберігання текстових даних невеликого розміру до 255 символів та може приймати велику кількість кодувань строк. Статус

користувача в системі показується завдяки простим числам, де 0 – акаунт вважається неактивним, а 1, коли користувач їм користується. Це поле створюється лише для приближення структури бази даних до реальної та проведення більшої кількості тестів. Зберігається ця інформація у форматі tinyint(2), щоб опрацьовувати числові значення при тестуванні. У реляційній базі даних існують більш адаптовані під цю задачу типи полів, наприклад binary, яке зберігає лише 0 чи 1, але такий тип відсутній у NoSQL базах, тому для коректності порівняння цих двох СУБД використовуємо лише прості типи даних. Таблиця clients_comments зберігає інформацію про коментарі користувачів в системі та містить лише два поля – зовнішній ключ UID (Foreign Key), який зв'язує таблицю користувачів і таблицю коментарів та поле самого коментарю, також у varchar(255). У таблиці clients_access представлений теж зовнішній ключ та такий самий прапорець доступу до можливості залишати коментарі, який приймає значення нуля чи одиниці.

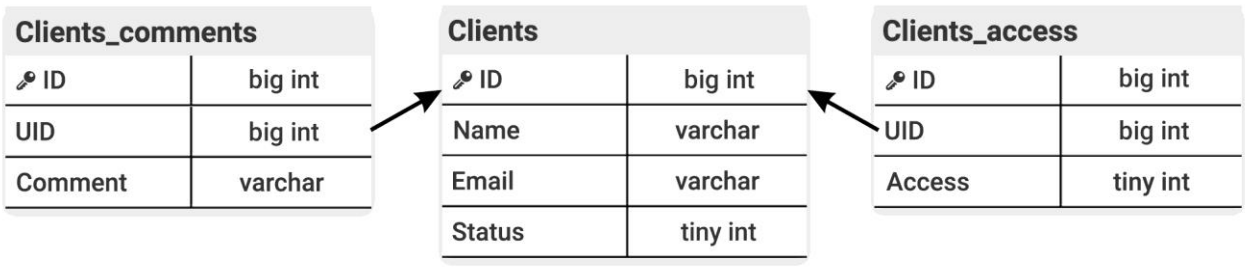

Рисунок 4.1 – Логічна схема реляційної бази даних

Для відтворення схожої структури у нереляційній базі даних – створимо три різні колекції, які будуть містити ту саму інформацію. У MongoDB доступні такі типи даних як числа та строки, а при створенні колекції можливо одразу вказати доступні для запису значення. Інкремент, який за замовчуванням встановлюється у MariaDB до первинного ключа, також можна вказати формулою у колекції. Структура трьох створених колекцій показана на рисунку 4.2.

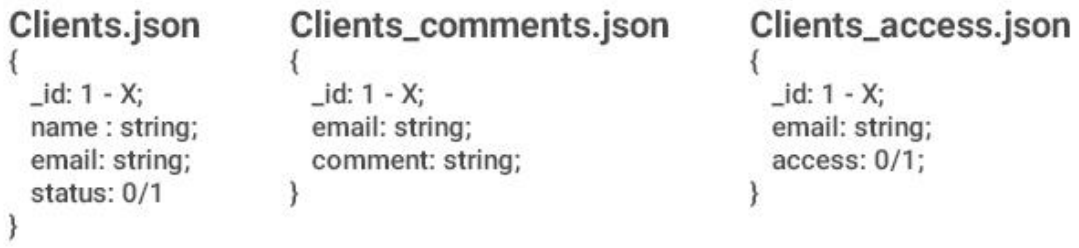

Рисунок 4.2 – Структурна схема колекцій нереляційної бази даних

Як можна побачити, замість зовнішніх ключів було використано унікальне поле Email, яке присутнє у головній колекції користувачів. Воно буде використовуватися при тестуванні складних запитів до баз даних. Також, вказуючи строковий тип даних не йде уточнення довжини строки, так як у форматі JSON може зберігатися строчка будь-якої довжини. Так як у колекції не містять у собі метаінформації, як таблиці, збереження такого типу інформації про поля не є необхідністю. Поле ідентифікатору запису створюється автоматично є захищеним у більшості мов програмування, які використовують драйвер цієї СУБД, так як при запиті користувачі хочуть отримати самі дані, а не їх ідентифікатор у базі. Так як така інформація вважається чутливою та повинна зберігатися у секреті для запобігання ін’єкцій до бази даних.

Для створення потоку даних, який буде взаємодіяти с базами, необхідно розробити генератор синтетичних даних, які будуть записуватися у таблиці та колекції. Для придання експерименту практичного значення, необхідно генерувати такі дані, які найбільше співпадають з реальними даними, які користувачі зазвичай залишають у системі. За структурою, такими даними є ім’я, яке повинно попадати у 80 символів та адрес електронної пошти. Коментарі будуть генеруватися із зібраної бази реальних коментарів у випадковому порядку.

Існує велика кількість бібліотек та готових рішень по генерації шаблонних даних для заповнення бази даних. Але у них є декілька недоліків. По-перше, як правило дані генеруються у випадковій послідовності, тобто одразу можна побачити, що дані були згенеровані комп’ютером. Як правило, це випадковий набір символів (якщо генерується строчка), або неймовірна у реальному житті послідовність чисел. По-друге, більшість генераторів орієнтується на заповнення даних за певними умовами, якими найчастіше є модель бази даних. Саме тому інформація, яка буде міститися у базі даних буде фактично відповідати вимогам тестування, а на практиці, не буде відповідати тим даним, що обробляються.

Саме тому було прийняте рішення зробити власний генератор синтетичних даних, який закривав би недоліки описані вище.

Проаналізувавши базу даних реального проекту, а наповнювалась більше року і містить більше 360 тисяч схожих за структурою записів, були виведені наступні послідовності, які можна використати при генерації даних для тестування баз даних. Так, середня довжина назви поштової скриньки склала 12 символів, без врахування символів, що йдуть після знаку «@». Була виведена база з більше ніж п’ятидесяти тисяч різних імен, як чоловічих, так і жіночих. Була створена база різноманітних коментарів (більше 500 різних повідомлень) користувачів. Спираючись на створені структури даних та корисну інформацію, на мові PHP напишемо три функції для генерації даних для баз даних, які будуть проходити тестування. Вибір між записом 0 та 1 буде відбуватися у момент запису та не потребує аналізу. Так як система-аналог, з якої зібрана статистика, використовується у режимі реального часу, то більшість акантів має статус активних, але це ніяк не впливає на хід експерименту, так як фактично ці числа несуть однакову кількість байт даних. Функції генерації даних до таблиць представлені на рисунку 4.3.

```
function generate_email() {
    $chars = "qazxswedcyfrtgbnhyuimkiolp1234567890";
    $max = 12;
    $domain = "@vindict.com.ua";
    while($max--) {
        $email .= $chars[rand(0,35)];
    }
    $email .= $domain;
    return $email;
}
function generate_name($names) {
    $name = $names[rand(0,51529)];
    return $name;
}
function generate_comment() {
    $comment = $comments[rand(0,535)];
    return $comment;
}
```

Рисунок 4.3 – Функції генерації синтетичних даних

Виведені функції кращі за існуючі аналоги завдяки своїй простоті та незалежності від будь-яких елементів або класів. Як правило, використовуючи бібліотеку, яка має свої генератори даних, наприклад Laravel, для генерації даних необхідно підключити певні класи, які завантажуються разом із запуском системи. Це створює більше навантаження на етапі запуску системи до першого кешування. Використовуючи функціональний стиль програмування, можливо досягти найбільш можливої швидкості, яка не займає стільки часу, щоб вважати генерацію даних – похибкою в обчисленні продуктивності роботи СУБД. Дані генеруються безперервно безпосередньо перед транзакцією до бази даних, імітуючи запит користувача. Завдяки однаковому типу даних – написані функції підходять до обох типів баз даних, які тестуються.

Дані будуть генеруватися послідовними етапами на протязі всього експерименту. Контрольними точками, коли буде замірятися продуктивність бази даних стануть: 150 тисяч, 750 тисяч і 3, 6, 15 мільйонів записів в одній таблиці. Таким чином, зрозуміло, як СУБД обробляють запити в умовах великих даних у порівняні з малою кількістю даних у базі. Більше того, можна скласти порівняльний аналіз швидкості виконання запитів, як за умов проведеного налаштування, так і без неї. Максимальний об’єм бази даних показано на рисунку 4.4, та складає він майже 4 гігабайти на три таблиці за умов зберігання 15 мільйонів записів.

	Таблица	Действие	Строки	Тип	Сравнение	Размер	Фрагментировано
☐	clients		15,000,000	InnoDB	utf8_general_ci	1.7 ГиБ	-
☐	clients_access		15,000,000	InnoDB	utf8_general_ci	961.7 МБ	-
☐	clients_comments		15,000,000	InnoDB	utf8_general_ci	1.1 ГиБ	-
	3 таблиц(а)	Всего	45,000,000	InnoDB	utf8_general_ci	3.8 ГиБ	0 Байт

Рисунок 4.4 – Максимальний об’єм бази даних у експерименті

За замовчуванням після кожного запиту СУБД створює кеш відповіді і запиту для забезпечення швидкої обробки запитів. Для того, щоб провести тесутвання за різних умов об’єму даних, що зберігаються, для кожного етапу експерименту та кожного кроку – всі дані генерувалися заново. Тобто таблиці очищалися та заповнювалися заново для уникнення похибки за раунок попереднього кешування, що значно прискорювало отримання відповіді від даних. У обох типах СУБД кешування не вимикається при будь-якому

налаштуванні як серверу, так і запитів. Такий захисний механізм довзоляє СУБД економити використання процесорного часу та збільшити кількість операцій, які можуть бути паралельно виконані під час обробки потоку запитів. Всі сучасні бази даних використовують кешування запитів на програмному рівні, тоді як інший кеш можна налаштувати.

Таким чином, з точки зору збереження та обробки даних, представлені до порівняння СУБД відрізняються лише на певних етапах розвитку системи, де вони використовуються. У разі, коли об’єми даних невеликі, продуктивність обох баз даних однакова. За рахунок формату BSON, нереляційна база даних швидше буде зправлятися із запитами, але це не буде помітно користувачам. Навіть якщо дані різнорідні та використовуються у потоці, їх оброка буде однаково швидкою, незалежно від типу СУБД. Проектування нереляційної бази має на увазі урахування структури даних, що будуть зберігатися, та компонування колекцій таким чином, щоб розбити ці дані за місцем використання та не розподіляти їх між різними колекціями.

Тобто ще до проведення тестування зрозуміло, що NoSQL бази даних більше підходять або для зберігання неструктурованих даних, або у випадку невеликих чи простих систем. Для складної архітектури великої системи все ж таки слід використовувати реляційні бази даних за їх надійність у роботі з даними, зрозумілість у використанні, масштабованість з використанням сучасних алгоритмів та при збереження властивостей швидкості і доступності. Незважаючи на це, проектувати систему легче саме нереляційну за рахунок того, що це можливо зробити без відомостей щодо даних, які будуть оброблятися. JSON дозволяє легко змінити тип даних на будь-якому етапі проектування чи створення системи, у той час як в реляційній базі зміна типу колонки може призвести до повної заміни структури однієї чи кількох зв’язаних таблиць. Більше того, нереляційні бази даних займають менше простору на сервері, так як не зберігають мета-інформації про себе. Вважається, що колекції можна прочитати самостійно, та додаткові системні пояснення не потрібні. Хоча це не кординально впливає на роботу СУБД в цілому, але мета-інформація таблиць в реляційних базах використувється рідко, в основному для валідації даних під час запитів. У сучасних системах, інформація валідується на стороні клієнта, сервера та додатково бази даних у тілі запиту (оптимізатором), тому подібні порівняння у явному вигляді втрачають свій сенс.

5 НАЛАШТУВАННЯ СУБД ТА ОПТИМІЗАЦІЯ ЗАПИТІВ

Оптимізація запитів – це частина процесу виконання запитів, в якій СУБД у реальному часі після отримання запиту, як є, від користувача, порівнює різні стратегії оброки запиту. У результаті аналізу запиту, вона вибирає ту, яка має найменші очікувані витрати ресурсів та часу на виконання. Оптимізатор запитів, який виконує цю функцію у кожній СУБД, є ключовою частиною бази даних і визначає найбільш ефективний спосіб доступу до даних, що зберігаються, зважаючи на всі фактори виконання запиту на момент його надходження до бази даних. Це дає можливість користувачу запитувати дані, не вказуючи, як ці дані слід отримати.

Вартість доступу до запиту визначається зваженістю на комбінацію вводу-виводу та витрат на обробку команд та даних. Вартість вводу/виводу являє собою вартість доступу до індексу, якщо він існує, та положення даних на диску серверу бази даних. Вартість обробки оцінюється шляхом присвоєння кількості інструкцій кожному кроку при обчисленні результату запиту. Існує два основних підходи до оптимізації: на основі витрат та евристичний. У першому випадку Оптимізатор оцінює вартість кожного методу обробки запиту і вибирає той, який має найнижчу оцінку. Сьогодні, більшість оптимізаторів запитів в СУБД використовує саме такий підхід. Другий підхід використовує правила засновані на формі запиту.

Оптимізація запиту досягається декількома методами: зміна структури сховищ даних, налаштування СУБД, зміна операторів всередині запиту, наприклад заміна операцій у SQL. Особливо, коли виконуються складні або багато розмірні запити – варто проаналізувати, чи є сенс розбити запити на декілька послідовних операцій, як це вплине на результат та завантаженість серверу або каналу зв’язку з базою даних. Також зміна логіки запиту іноді дозволяє машині обробляти інструкції скоріше, хоча логіка запиту залишається тією самою. Іноді оптимізатор запитів змінює структуру запиту, але для користувача і СУБД важливо надати правильно відповідь відповідно до запиту. Саме це ставить за мету оптимізація запитів – це скорочення часу виконання запиту та споживання ресурсів системи для отримання точної вибірки даних, яка очікується за логікою запиту.

Для виконання експерименту необхідно зрозуміти, наскільки оптимізація запитів та налаштування СУБД впливають на час обробки даних. Тому після встановлення обох СУБД буде проведено два тести – з використанням налаштованої та не налаштованої бази даних. За замовчуванням, встановлено автоматичну оптимізацію всіх запитів, які потрапляють до обох СУБД, та деякі налаштування для роботи з даними, такі як типи таблиць, параметри обробки даних, взаємодія сервера бази даних, тощо. Налаштування за замовчуванням дають переваги у використанні конкретної СУБД, так як вони рідко змінюються. Детальне налаштування потрібне і грає значну роль при роботі з великими даними, у іншому випадку різниця у продуктивності не буде помітна.

Перш ніж визначати налаштування бази даних, визначено поняття продуктивної роботи СУБД взагалі, та зокрема обробки запитів. Враховуючи, що СУБД працює в умовах невизначеності, продуктивним будемо вважати таку роботу бази даних, при якій витрачається мінімальний час (t) на обробку певної кількості запитів (q). Апріорна невизначеність не є суттєвою при розгляданні максимальної кількості запитів, які можуть оброблятися за певну кількість часу, так як архітектура системи визначає чіткі правила роботи з даними. Графік зміни часу обробки запитів в залежності від їх кількості показано на рисунку 5.1.

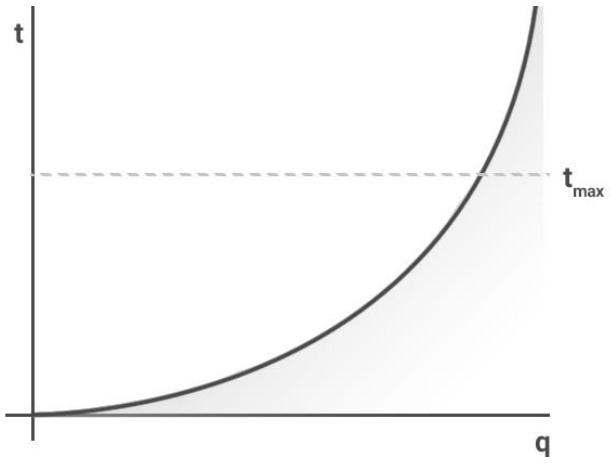

Рисунок 5.1 – Графік залежності часу виконання запиту від кількості запитів

При чому, як видно на рисунку 5.1, є деякий час t_{max}, який означає граничний час виконання запиту чи запитів до бази даних, після чого з’єднання обривається автоматично, так як вважається, що запит займає забагато часу та ресурсів для виконання. Зазвичай, жоден запит не досягає цієї відмітки у часі

виконання, та цей механізм потрібен для підвищення відмовостійкісті СУБД. Такий параметр можна налаштувати, зважаючи на інші фактори, які впливають безпосередньо на виконання одного запиту у один момент часу.

Окрім самого поняття продуктивної роботи бази даних є ще параметри продуктивності [29]. Це набір властивостей, які безпосередньо впливають на швидкість обробки запитів у базі даних. Завдяки параметрам продуктивності буде можливо порівнювати зміни між експериментами та робити висновки щодо можливості виконання поставленої задачі СУБД у різних експлуатаційних умовах.

Час виконання запиту характеризує продуктивність роботи всієї СУБД у зв'язку із системою, що використовується, та залежить від таких параметрів продуктивності, як: складність запиту, складність структури бази даних, об'єм даних, що зберігається у базі, налаштування СУБД та оптимізатору запитів. Більше того, параметри продуктивності можуть змінюватися під час роботи системи як безперервно, так і задаватися користувачами. Наприклад, від користувача системи залежить складність запиту, який він робить до системи, але він не може вплинути на кількість записів у базі, які обробляються СУБД до того, як користувач отримує відповідь. Розглянемо як можливо вплинути на параметри продуктивності у різних СУБД та порівняємо, як це впливає на швидкість виконання запитів.

Для проведення тестування будуть використані дві версії кожної з СУБД – з проведенням налаштувань та без будь-яких додаткових параметрів, що впливають на швидкість роботи. Тобто усі описані у цьому розділі налаштування проводилися два рази – для підвищення продуктивності та її зниження до рівня, коли у повній мірі працюють всі складові бази даних. Це було необхідно для дослідження можливостей самої системи обробки даних без програмних додатків, таким чином можна зрозуміти властивості кожної СУБД, які будуть приведені у розділі 6.

Оптимізація запитів, особливо у мові SQL, необхідна при використанні складних структур об'єднання запитів (JOIN, UNION), пошуку по тексту (LIKE) або групуванню результатів пошуку. Завдяки виразу EXPLAIN у SQL можна побачити яким чином СУБД виконала запит, який поступив від користувача. Аналізуючи ланцюг виконаних операцій – можливо прийняти рішення, який з пунктів може бути вилучений чи покращенний для досягнення більшої продуктивності обробки даних. По-перше, це установка індексів на поля, за якими

шукають у таблиці найчастіше, або групують дані. Індекси у SQL можуть зайняти більше місця та знизити продуктивність для вставок, видалень та оновлень даних у таблицях. Однак якщо у вашій таблиці більше 10 рядків, вони можуть значно зменшити час виконання певних запитів, які використовують найбільш відповідні поля. Завдяки індексам, СУБД не проходить всю таблицю а лише знаходить необхідні рядки, що особливо необхідно при роботі з великими даними. У базі даних, яку було створено для тестування, такими полями є clients.name, clients.status, clients_comments.uid. Вони будуть використовуватися для складних та простих запитів найбільше, тому індекси дадуть перевагу у швидкості виконання запитів.

Коли виконуються запити, де використовуються оператори порівняння LIKE у різних полях або стовпцях певної таблиці, СУБД необхідно не тільки аналізувати поля, а й те, що у них зберігається. Коли ключове слово LIKE використовується занадто часто, оптимізатор запитів може помилятися при скануванні полів, тому рекомендується використовувати UNION у складних запитах, як це показано на рисунку 5.2. Це може зробити запит швидшим, особливо якщо у поля є індекс, який може прискорити виконання основного запиту та інший індекс для прискорення виконання запиту, який приєднано до основного. Треба пам’ятати, що РСУБД не в змозі використовувати індекси, коли в запиті є параметр підбору «%», який використовується на початку пошукової фрази. Якщо потрібно використовувати пошук по тексту, коли необхідно вказувати невизначеність символом «%», при цьому не допустити, щоб база даних виконувала малоефективні операції, слід розглянути можливість використання повнотекстового пошуку (FTS), оскільки він виконується набагато швидше, ніж запити з використанням символів підстановки. Крім того, FTS також може принести кращі та релевантні результати, коли пошук відбувається в мовах роботи з великими даними (більше 10 мільйонів записів). Приклад FTS показано на рисунку 5.2. Якщо використовувати у базі даних пошук за коментарями clients_comments.comment, то можна застосувати повнотекстовий пошук для зменшення часу обробки всієї таблиці коментарів.

```
SELECT * FROM clients_comments WHERE comment LIKE 'Hello%'
UNION ALL SELECT * FROM clients_comments WHERE name 'Vlad%';

ALTER TABLE clients_comments ADD FULLTEXT (comment);
```

```
SELECT * FROM clients_comments WHERE MATCH(comment) AGAINST
('Hello');
```

Рисунок 5.2 – Приклади оптимізації запитів до бази даних

MongoDB також має можливість використовувати оператор explain() для виведення логів виконання запиту. Для оптимізації запитів можна використовувати індекси та складні індекси. Для поширених запитів створюються індекси тих полів, які скануються частіше інших. Якщо запит здійснює пошук у кількох полях, СУБД дозволяє створити складений індекс на декілька полів одночасно, що зручніше індивідуального індексу для кожного поля. Сканування за допомогою індеків у рази швидше, ніж сканування звичайної колекції, особливо коли кількість записів у ній занадто велика. Структури індексів менше, ніж посилання на документи, і зберігаються вони упорядковано.

Використовуючи будь-який тип СУБД користувач у більшості випадків сортує видачу по своєму запиту, тому індексування таких полів важливе і у документно-орієнтованій базі даних. Створюючи сучасну систему, використовуючи MongoDB, можна підвищити продуктивність за допомогою одного стійкого підключення до бази даних, яке повторно використовується для всіх запитів та оновлень, які до неї надходять від всіх користувачів системи.

СУБД виконує всі команди в тому порядку, у якому вони до неї надійшли. Незважаючи на те, що система може робити асинхронні звернення до бази даних, кожний набір команд синхронізується в черзі на стороні серверу і повинен завершитися до виконання наступного завдання, яке знаходиться у черзі на виконання. Якщо до бази даних надається складний запит, який потребує багато часу для виконання, будь-який інший запит більше не може одночасно взаємодіяти з системою використовуючи одне й те саме з’єднання з базою даних, незалежно від мови програмування, або типу системи. Продуктивність бази даних може бути покращена за рахунок ініціалізації декількох підключень до бази даних. Таким чином, на етапі проектування системи, можливо розподілити всі запити до бази даних на різні категорії, наприклад по складності або частоті використання. Створивши для кожної категорії своє з’єднання, проблема черг зменшується. Кожне з’єднання трактується СУБД як окремий клієнт бази даних і не затримує обробку інших клієнтів та запитів, які на них виконуються.

За замовчуванням, СУБД обробляє запити стільки часу, скільки потрібно для того, щоб надати результат користувачу. Запит, який виконується повільно, може затримати інші запити у черзі, що було розглянуто раніше, тоді система може згодом втрачати більше часу на обробку навіть простих запитів. Це може викликати різнорідні проблеми та навіть нестабільність у програмах, які чекають від бази даних асинхронного зворотного виклику, продовжуючи виконувати програмні інструкції. Вказавши обмеження часу в мілісекундах, використовуючи інструмент maxTimeMS(), можна дозволити запиту виконуватися деякий максимальний час, який задається числом мілісекунд. Використовувати таку функцію можна до складних запитів, які імовірно довго обробляються, але значення потрібно підбирати експериментальним шляхом. Таким чином, виникає гарантія відповіді бази даних за будь-яких обставин та параметрів запиту. Однак, СУБД не дозволяє визначити глобальне значення тайм-ауту запитів, тому час повинен бути встановлений індивідуально для окремих запитів (хоча використовуючи деякі програмні бібліотеки на стороні системи, можна застосовувати час виконання запиту за замовчуванням). Окрім часу, можна також обмежити кількість записів, які повинна повернути база даних після запиту. Таким чином, коли набирається вказана кількість записів, пошук по колекції завершується, навіть якщо ще залишилися об’єкти для сканування. Якщо визначена точна кількість потрібних результатів, можливо зменшити попит на мережеві ресурси, видавши метод limit() у MongoDB. Зазвичай це використовується в поєднанні з операціями сортування, так як системи використовують велику кількість інформації, яку користувач не зможе охопити за одну операцію з системою (перегляд, дія, тощо). Гарною практикою зараз вважається поетапна подача контенту користувачу, який завантажується у систему поступово в залежності від дій користувача.

Коли необхідно отримати лише підмножину полів з документів у нереляційній базі, можна досягти кращої продуктивності за рахунок обробки та видачі лише потрібних полів. Тоді такі поля можуть передаватися у вигляді об’єкту разом із основним запитом. Також завдяки використанню функції hint() можна вказати конкретні поля, які мають індекси які потрібно використовувати СУБД, так як за замовчуванням, вона використовує оптимальний індекс для кожної операції окремо. Оптимальний індекс визначається базою даних автоматично.

Підказки (HINT) дозволяють приймати рішення, зазвичай прийняті оптимізатором запитів. ЗА допомогою цієї операції можна подивитися корисну інформацію про дані, що зберігаються, яку не використовує оптимізатор запитів. Підказки забезпечують механізм, який впливає на виконання інструкцій оптимізатором, та змінює його на вибір певного плану виконання запитів на основі конкретних критеріїв. Наприклад, архітектор бази даних знає, що певний індекс є більш вибірковим для певних запитів. Виходячи з цієї інформації, можливо, зробити висновок і вибрати більш ефективний план виконання, ніж оптимізатор запитів. У такому випадку використовуйте підказки, щоб змусити оптимізатор використовувати оптимальний план виконання.

Потрібно розуміти, що велика кількість індексів не зробить базу даних надто швидкою. Кожні індекси використовують окрему ділянку пам’яті, тому індексувати потрібно лише необхідні поля. Існує інструмент ре індексації колекцій, який допомагає встановити індекси заново. Але при створенні архітектури бази даних, необхідно зважати на те, що лише деякі найважливіші поля будуть мати індекси, а інші більшість буде слугувати сховищем для даних будь-яких розмірів.

У MongoDB є унікальний оператор $inc, який використовується для збільшення або зменшення значень у документах. Оператор збільшує значення поля на стороні сервера, як альтернативу вибору документа, або внесення простих модифікацій на стороні користувача, а потім запис всього документа на сервер. Оператор $inc також може бути корисним при усуненні умов перегонів, коли два екземпляри програми запитують один й той самий документ. Оператор самостійно збільшує поле та зберігає весь документ у базі даних одночасно.

При налаштуванні самої бази даних – необхідно розуміти, які області СУБД використовуються у більшості операцій з базою даних, які операції база даних виконує найчастіше та дізнатися загальні параметри роботи системи управління базою даних. Все це можливо лише завдяки системним логам, які необхідно вести при роботі з базою даних. Вони також можуть бути корисні для аналізу роботу системи, навантаженню в залежності від часу, дня тижня, події, тощо. Особливо корисно використовувати логи як системи, так і бази даних, для виявлення помилок, слабких місць або для виявлення причин низької продуктивності. Окрім них існує невеликий набір стандартних інструментів, які покращують продуктивність роботи СУБД, які будуть розглянуті окремо для кожного типу баз даних.

Під час розробки та керування системами, які використовують MongoDB у якості СУБД, вірогідно доведеться аналізувати продуктивність системи та її бази даних. Коли розробники стикаються із зниженою продуктивністю, часто проблемами стають: варіації доступу до бази даних, доступності обладнання та кількості підключень до СУБД у один момент часу. Деякі користувачі системи можуть побачити наслідки таких у проблем у вигляді непродуктивності, які також можуть виникати внаслідок неадекватних або невідповідних стратегій індексації, що може бути пов’язано з поганими схемами проектування колекцій. Ефективність блокування обговорює, як це може вплинути на внутрішнє блокування у MongoDB.

Проблеми з ефективністю можуть вказувати на те, що база даних працює в умовах недостатньої потужності сервера, яку можна додавати для прискорення виконання операцій або збільшення паралельних обробок запитів. Зокрема, робочий набір програми повинен вміщуватися у наявну фізичну пам’ять.

У деяких випадках проблеми з роботою можуть бути тимчасовими та пов’язані з великою кількістю трафіку до серверу бази даних, або завантаженням каналу обробки потоку даних. Завдяки розподіленню бази даних, ця проблема може бути швидко вирішена без втрати даних. Щоб виявляти ці або подібні помилки, можна використовувати профілювання бази даних може допомогти зрозуміти, які операції викликають заниження продуктивності.

Основним джерелом підвищення швидкості роботи будь-якої СУБД є кешування. Воно дозволяє не виконувати запити, якщо вони буди оброблені раніше, не індексувати всю таблицю та максимально зменшити час відгуку бази даних. У MongoDB з’явився новий двигун для зберігання тимчасових даних WiredTiger. Ця система зберігання суттєво покращила продуктивність бази даних та кількість паралельних підключень завдяки швидкості відповіді, порівняно зі старим MMAPv1. Він також пропонує переваги у стисненні та шифруванні даних, якщо це необхідно, але за замовчуванням ці функції вимкнено для економії використання ресурсів СУБД. За замовчуванням MongoDB зарезервує 50 відсотків доступної пам’яті для кешу даних WiredTiger. Розмір цього кешу важливий для забезпечення належної роботи як сховища кешу, так і бази даних в цілому. Доброю практикою вважається, коли розмір кешу повинен бути досить великим, щоб вмістити в себе увесь робочий набір даних для функціонування системи.

Запуск РСУБД із застосуванням налаштувань для конкретних ресурсів допомагає обробляти більші навантаження на сервера баз даних та запобігає уповільненням роботи сервера. Як правило, після додаткового налаштування клієнтського серверу для обробки великих навантажень – великий приріст у продуктивності надає налаштування бази даних на додаткові з'єднання. Великі бази даних реляційного типу можуть вимагати значного обсягу пам'яті, особливо при роботі з великими даними. З цієї причини ми рекомендуємо використовувати Linode з високою пам'яттю для таких налаштувань. Як і у MongoDB, у MariaDB існує сценарій MySQLTuner, який виводить пропозиції щодо підвищення продуктивності та стабільності сервера бази даних. Змінюючи конфігурацію СУБД, треба насамперед розуміти, як це вплине на базу даних та обробку даних по запитах. Навіть дотримуючись вказівок програм, таких як MySQLTuner, найкраще мати певне розуміння процесу налаштування, а саме з'єднання з базою, роботою з кешем, потоки виконання, кеш потоків. Для отримання значень змінних налаштувань виконується команда SHOW, але самі зміні можуть бути змінені у конфігураційному файлі бази даних. Приклад перегляду змінних показано на рисунку 5.3, сторінка 50. Таким чином можна отримувати навіть змінні, назву яких можна доповнювати символом «%». Ці команди виконують функції аналітики таблиць реляційних баз даних, яка допомагає швидко зрозуміти стан СУБД. Доповненням для аналітики, яке часто використовується, стає функція HINT(), яка описувалася раніше.

```
SHOW variables LIKE 'have_query_cache';
SHOW variables LIKE 'query_cache_%';

query_cache_type=1
query_cache_size=10M
query_cache_limit=256k
```

Рисунок 5.3 – Перегляд та зміна конфігурації СУБД

Завдяки великій кількості параметрів у реляційних базах даних, що регулюються, існують практичні поради щодо налаштування роботи СУБД. Так як у більшості випадків використовується InnoDB у якості двигуна таблиць бази даних, його роботу можна покращити використовуючи innodb_buffer_pool_size,

який повинен дорівнювати від п’ятдесяти до семи десяти п’яти відсотків усієї оперативної пам’яті серверу бази даних. Параметр innodb_flush_log_at_trx_commit грає роль компромісу між продуктивністю та надійністю. Якщо його встановлено у 0, то продуктивність бази даних значно збільшується. Однак у разі збою у роботі бази даних можуть бути втрачені до 1 секунди всіх транзакцій, які проходили у базі даних, що складає еквівалент мільйона простих запитів. Значення за замовчуванням для цього параметра становить 1, що забезпечує повну відповідність моделі ACID. Параметр innodb_flush_method створює додаткові буфери з даними, що обробляється. Його можна встановити у O_DIRECT, що призведе до зупинки повторної буферизації за економить час виконання запиту. Так як на сервері бази даних зберігаються файли таблиць, які представлені розширенням .ibd, звернення до них про обробці запитів також можна прискорити. Задавши параметр innodb_file_per_table рівний одиниці, зазначається, що для кожної таблиці буде використано один файл у файловій системи, що поліпшує процедури вводу та виводу файлів, а операції типу TRUNCATE будуть виконуватися швидше. Вказавши параметр wait_timeout у повну кількість секунд, наприклад 60, такі з’єднання будуть перериватися, збільшуючи об’єм пам’яті на виконання інших задач. Дані, що передаються по мережі, СУБД розбиває на окремі пакети. Зазвичай одним пакетом вважається рядок, який надсилається клієнту на його запит. Директива max_allowed_packet визначає максимальний розмір пакета, який база можна надіслати. Якщо встановити це значення занадто низьким, це призведе до зупинки запиту, так як не всі пакети вдасться відправити і виникне помилка роботи СУБД. Тому було встановлене значення розміру пакету, яке відповідає найбільшому пакету, який тільки може бути сформований у базі даних. У разі, якщо база даних працює деякий час з системою, корисно використовувати команди --auto-repair --check --optimize --all-databases, які дозоляють автоматично усовувати проблеми, що моуть виникати унаслідок непередбачуваних збоїв у роботі серверів чи з’єднанні. Команди дозволяють виявити структурні недоліки та помилки у базі даних та виправити їх. Якщо необхідно лише проаналізувати базу даних, використовується mysqlcheck, яка виводить адміністратору бази даних звіт про стан СУБД.

Останніми параметрами, які впливають на виконання запитів СУБД є типи побудови сховищ даних, а саме таблиць і колекцій. Існують різні моделі, або схеми даних, які вдало функціонують в залежності від типу СУБД. Взагалі, дві найбільш розповсюджені схеми – це один до одного та багато до одного. Остання

схема показує себе як найбільш ефективна як для зберігання та обробки даних, так і для розуміння та читання людиною. Така схема полягає у зв'язку даних з одного сховища з даними у іншому.

Методи підвищення продуктивності роботи бази даних включають декілька практик побудови сховищ даних. По-перше, це використання вбудовуваного документа та спрощення його структури, оскільки це зменшує кількість запитів, які потрібно виконати для отримання набору даних, які запрошує користувач. По-друге, не треба застосовувати денормалізацію для документів, які часто оновлюються. Якщо поля у сховищах будуть часто оновлюватися, тоді СУБД буде знаходити всі екземпляри полів, які потрібно оновити. Це призведе до повільної обробки запитів, так як оброблятися буде все сховище даних. По-третє, якщо потрібно отримати дані окремо від іншої інформації, що зберігається, тоді немає необхідності використовувати вбудовування, оскільки для виконання складних запитів потрібно більше часу. У разі коли масив документів, які потрібно запросити з бази даних, досить великий, не треба його вкладати. Зростання масиву має принаймні мати обмеження, так як сам сервер бази даних має обмежені ресурси на його обробку.

Для налаштування роботи таблиць та полів даних у реляційних базах існують додаткові рекомендації, які були враховані при проведенні тестування. Були нормалізовані всі таблиці бази даних, хоча спочатку була обрана відповідна структура. При нормалізації реляційної бази даних необхідно представляти будь-який факт появи даних лише один раз у всій базі даних, щоб зберігати унікальність та не допустити повторів. При формуванні колекцій у NoSQL на цей принцип також має сенс звернути увагу, так як він покращує показники продуктивності не тільки базу даних, а й простір на сервері. Помилкою буде повторення окремих полів, які виступають у ролі вторинного ключу. Замість цього, гарною практикою вважається використання ідентифікаторів, які з використанням індексу дозволяють найшвидше знайти необхідні дані у таблицях. У РСУБД підтримуються різні типи даних, включаючи цілі числа, з плаваючою крапкою, великі числа, дати, час, строки та цілі тексти. Створюючи або налаштовуючи таблиці, необхідно застосовувати принцип «коротше завжди краще». Наприклад, якщо поле у таблиці очікує значення дати, використання типу даних date_time буде ідеальним, оскільки не потрібно запускати складні функції для перетворення поля в дату під час отримання записів за допомогою SQL або мови програмування. Хоча з точки зору зберігання такої інформації, найкращим

рішенням буде штамп часу у форматі UNIX часу, так як він є лише набором чисел. Правильне використання типу даних не тільки зменшує об’єм бази даних на сервері, а й додає швидкості у обробці таких даних, так як СУБД виконує інструкції відповідно до типу даних, що використовуються. Таким чином, коли мова заходить про обчислення, реляційні бази даних можуть краще працювати з цілими значеннями порівняно з типами текстових даних, такими як varchar, що призводить до заміни деяких параметрів у базі даних – числами, які перетворюються на відповідний текст всередині системи при обробці відповіді від бази даних.

Швидкість роботи з даними також залежить від того, чи взагалі зберігаються дані у таблиці. Коли значення відсутнє, зазвичай колонка містить значення NULL. Такого типу даних за замовчуванням слід уникати, коли це можливо, оскільки вони можуть завдати шкоди продуктивності СУБД. Наприклад, якщо необхідно отримати суму чисел з таблиці, але деякий запис має NULL замість числа, очікуваний результат може неправильно сформований, якщо не використовується оператор ifnull для повернення альтернативного значення, у разі коли запис недійсний. У деяких випадках може знадобитися визначити значення за замовчуванням для поля, якщо колонки не повинні обов’язково бути заповненими одразу. Тоді, при можливості, краще вказувати логічне значення за замовчуванням, наприклад 0 для числових полів, або потомну дату для полів дати та часу, щоб уникнути уповільнення обробки пустих значень. До речі, оптимізатор запитів підставляє всі необхідні оператори самостійно, тому ця проблема не набуває значного поширення.

Робота з великими даними має на увазі використання великих таблиць, які можуть бути мати велику кількість типів даних і вимагають більше процесорного часу на обробку запитів. Якщо можливо, рекомендується не перевищувати більше ста різних полів, навіть коли бізнес-логіка системи вимагає цього. Замість збільшення однієї таблиці, слід розбивати її на окремі логічні структури. Процес розподілення таблиць не впливає на архітектуру та може бути виконаний на будь-якому етапі роботи системи. Потрібно також включати менше таблиць у запитах з використанням JOIN або UNION, а краще взагалі уникати їх. Оператор SQL з погано розробленою структурою, який передбачає багато об’єднань, може обробляти запити та шукати записи в базі даних занадто повільно, особливо коли об’єм таблиць бази даних перевищує мільйони записів.

6 ПОРІВНЯЛЬНИЙ АНАЛІЗ ПРОДУКТИВНОСТІ РІЗНИХ СУБД

Для визначення продуктивності, будемо проводити порівняння часу виконання запитів додавання, зміни, пошуку і видалення значень з таблиці (колекції). Складність запитів буде зростати з кожним новим тестом, що проводиться, починаючи з вибірки значень з умовами, закінчуючи обробкою даних і різних джерел, що найбільше наближує умови проведення експерименту до практичного випадку, коли дані, що зберігаються у базі, необхідно згрупувати всередині запиту та повернути користувачу системи.

На момент проведення експерименту, в ньому будуть використані останні версії СУБД, які отримали більшу можливість конфігурації показників серверу та сховищ даних, покращили взаємодію із запитами, а також ліпше стали працювати в Інтернеті (або іншій мережі, наприклад локальній). Використання найновіших версій дозволяє зробити точний структурований аналіз продуктивності СУБД, а також випробовувати нові можливості налаштування.

З огляду на той фактор, що MariaDB є представником реляційних баз даних [30], необхідно спроектувати такі умови, при яких будуть відносини (зв’язки) кількох таблиць для того, щоб застосувати LEFT JOIN по атрибутам для отримання єдиного результату. У MongoDB для виконання аналогічної операції буде використовуватися агрегація даних, тобто угруповання значення і використання змінної $lookup, операції об’єднання. Варто зауважити, що бази даних типу «ключ-значення» проектуються не так, як реляційні, тому використання $lookup небажано. У NoSQL базах даних два окремі запити до різних документів виконуються швидше, ніж одна об’єднана операція, яка захватує більше ніж одну колекцію. Однак в рамках тестування вона буде використана для приведення СУБД, що проходять тестування, до однакових умов експлуатації за рахунок відсутності в MongoDB операції, подібної JOIN у MariaDB. Вся необхідна для нестандартних запитів логіка буде поміщена у змінну $lookup і використана для тестування в нереляційних СУБД.

Під збільшенням складності запитів мається на увазі збільшення обсягу даних в базах даних, які необхідно обробити. Починатися тестування зі 150 тисяч доданих записів, після чого зростає до 750 тисяч, і послідовно до 3, 6 і 15 (сумарно з трьох таблиць 45) мільйонів записів. Дані генеруються самостійно у ході виконання програми для тестування, використовуючи генератори, які були

описані раніше. Цей процес повністю імітує застосування СУБД у реальних веб-додатках, які працюють в робочому режимі. Послідовна генерація дозволяє оцінити швидкість виконання запитів при різних навантаженнях і типах CRUD операцій, що дозволяє в повній мірі порівняти продуктивність СУБД. Дані, отримані при навантажувальному тестуванні, можна використовувати при розрахунку показників продуктивності у разі масштабування бази даних будь-яким з представлених способів.

Для кожного типу запитів буде проводитися окреме тестування на певній кількості записів. Фактично, тестування проводиться без похибок, так як проводиться на локальній машині. Використовуючи розподілені сервери баз даних виникають додаткові часові витрати на передачу даних по мережі. Такі похибки залежать від типу та швидкості підключення, розташування клієнта і сервера, типу серверів та пристрою клієнта. Так чи інакше, при налаштуванні підключення та серверу СУБД, такі затримки не перевищують 300мс [31]. Запити, які використовуються для тестування, показані на рисунку 6.1.

CRUD + JOIN	MariaDB	MongoDB
CREATE	INSERT INTO clients VALUES (?, ?, ?) INSERT INTO clients_comments VALUES (?, ?) INSERT INTO clients_access VALUES (?, ?)	db.getCollection("clients"). insertOne({'_id': '?', 'name': '?', 'email: '?', 'status': '?'}) .....
READ	SELECT SQL_NO_CACHE * FROM clients WHERE name = ? AND status = ?	db.getCollection("clients"). find({'name': '?', 'status': '?'})
UPDATE	UPDATE clients SET status = 1 WHERE name = ? and status = ?	db.getCollection("clients"). updateMany({'name': '?', 'status': '?'} {'$set': {'status': 1}})
DELETE	DELETE FROM clients WHERE name = ?	db.getCollection("clients"). deleteMany({'name': '?', 'status': '?'})
JOIN	SELECT SQL_NO_CACHE * FROM clients LEFT JOIN clients_comments ON clients.id = clients_comments.uid WHERE name = ?	db.getCollection("clients").aggregate([{"$lookup" : { "from" : "clients_comments", "localField" : "email", "foreignField" : "email", "as" : "comments" }}],);

Рисунок 6.1 – Запити для тестування СУБД

Використовуються чотири простих запити моделі CRUD (Create – Read – Update – Delete) з додатковими умовами обробки даних у базах. Більшість запитів використовує просту структуру, що значно прискорює обробку даних та отримання відповіді. Для порівняння швидкості виконання складних запитів (поєднання двох або більше взаємопов’язаних запитів у один), які можуть

використовуватися для аналізу великих даних або спричинені неправильною архітектурою бази. У тестуванні такі запити представлені оператором JOIN у мові SQL та змінною $lookup у NoSQL базі даних. Обидві операції не рекомендується використовувати у СУБД, так як розробники пишуть про заниження показників продуктивності [32].

Виконання запитів у тестовому режимі та в реальній системі відрізняється також частотою звернень до бази даних. Експеримент допускає умовність, що запити виконуються у потоці, який генерується синтетично. Тобто навантаження на СУБД завершується з виконанням усіх запитів для проведення тестування. У випадку подібних систем на практиці – запити виконуються за умов деякого закону розподілу, тобто поетапно. Таким чином навантаження на базу даних менше за рахунок неодночасних запитів до системи, а таких, що розподілені за часом. Так як метою даної роботи є виявлення показників продуктивності кожної СУБД, усі запити будуть виконуватися одночасно, як при умовах стресового тестування.

Першим етапом тестування буде відключення будь-якого налаштування як СУБД, так і обробника запитів. Кожен новий запит відбувається за умов відсутності кешування. Проведення тестування без налаштувань БД дозволить виявити дійсну продуктивність кожної бази даних, її можливості обробки запитів різної складності.

Операція додавання даних до таблиці (колекції) використовує синтетичні дані, які генеруються автоматично перед створенням запису у базі даних. Додавання (INSERT) відбувається одночасно у три таблиці. Результати тестування показані на рисунку 6.2. Тисячі на рисунку умовно позначаються буквою k, а мільйони – великою M. По осі Y зазначено час виконання запиту у секундах.

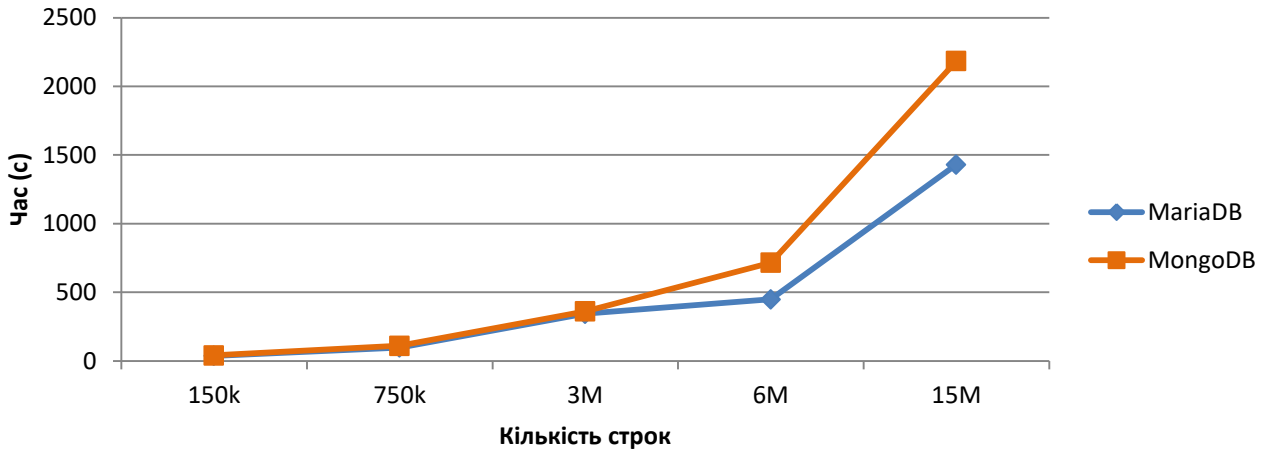

Рисунок 6.2 – Виконання запиту додавання даних без налаштувань БД

Завдяки структурі колекцій, вони здатні швидше зберігати дані, які поступають. Так як у таблицях присутні ключі, по яким проходить індексація значень, їх перевірка та обробка займає деякий час, особливо коли записів у таблиці більше мільйона. Враховуючи можливості шардінгу та реплікації, можна зробити висновок, що обидві СУБД гарно виконують цю задачу. Зазвичай, не допускається зберігання надвеликої кількості даних у одній базі, так як потужності для обробки та аналізу будуть коштувати занадто дорого. Додаючи також умовність розподілу виконання операцій, можна сказати, що результати 37 секунд та 41 секунда для 150 тисяч записів – не суттєво відрізняються між собою.

Операція читання даних, яка статистично визнана найпопулярнішою, використовує додаткові умови для вибірки даних. Зазвичай, архітектори бази даних створюють такі умови, що вибірка даних відбувається або аз числовими, або бінарними полями. Це підвищує швидкість роботи СУБД та зменшує об’єм пам’яті, яка необхідна для її функціонування. Інколи необхідно опрацьовувати і символьні типи даних, наприклад строки чи навіть текст. У запиті, що тестується, використовується обидва типа полів. Результати тестування показані на рисунку 6.3.

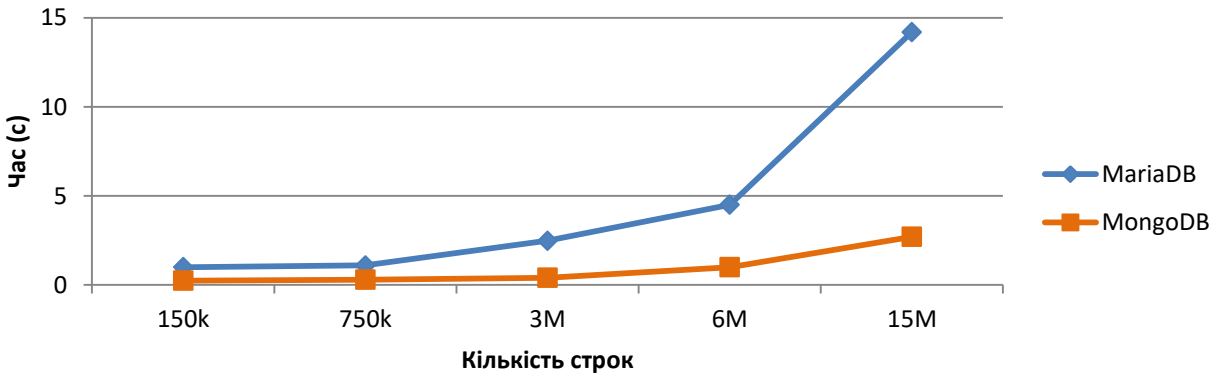

Рисунок 6.3 – Виконання запиту читання даних без налаштувань БД

Читання даних (SELECT) відбувається швидше усіх інших типів запитів в усіх СУБД. Швидкість обробки даних у випадку відсутності налаштувань залежить у більшості від технічних характеристик сервера. У зв'язку з тим, що MariaDB використовує таблиці, зі збільшенням кількості запитів – швидше зростає потреба у використанні пам'яті. Так як у запиті використовуються умови, по яким СУБД необхідно відфільтрувати дані, необхідно переглянути всі запити та перевірити відповідність. Так як у MongoDB не потрібно переглядати колекції на виконання умов по окремих параметрах, а можна зробити це у потоковому режимі, навіть без налаштувань можна отримати значну кількість запитів без суттєвої затримки для кінцевого користувача. На практиці, завдяки правильній архітектурі системи, що використовується, вибірка даних у результаті дає не більше 100 записів. Як правило, це пов'язано з передачею результатів по мережі. Але вибірка може відбуватися і більше, ніж з 15 мільйонів записів у базі даних. Якщо використовуються великі дані, які необхідно обробляти, обов'язково використовуються параметри продуктивності та великі потужності серверу бази даних. Для досягнення великої швидкості отримання результатів саме вибірки даних, наприклад у експертних системах, використовуються колонкові бази даних, які розроблені для вирішення таких задач. Використовуючи хмарні сервіси, можна перетворювати рядкові таблиці у колонкові, або навпаки. На рисунку 6.4 зображені результати тестування зміни даних у базі.

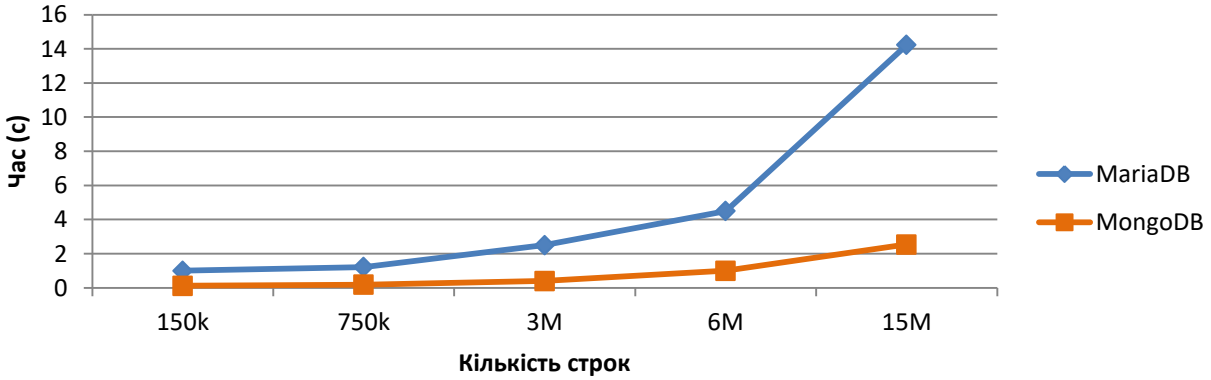

Рисунок 6.4 – Виконання запиту оновлення даних без налаштувань БД

Оновлення даних по швидкості посідає друге місце після читання. Так як у MariaDB враховуються ключі рядків, які не налаштовані, їх обробка займає більше часу, ніж пошук і зміна даних у колекціях MongoDB. Кожен запис у колекції легко змінюється без додаткових операцій, які необхідні SQL, а інколи і принципами ACID. У результаті NoSQL база даних виявилася у три рази швидшою за реляційний аналог у двох найпопулярніших типах запитів. Ідеєю створення цього типу СУБД була прискорення цих операцій, спрощення взаємодії розробників з базою даних та роботи проектування як самої бази даних, так і її взаємодії з системою. Подібні показники гарантують, що будь-хто, використовуючи нереляційну СУБД, отримає значно кращий результат у швидкості обробки запитів читання та оновлення даних, ніж у аналогічної реляційної бази, на створення якої йде більше ресурсів. На рисунку 6.5 показані результати виконання операції видалення даних з бази (DELETE). З появою великого простору для зберігання даних, навіть тих, що не використовуються, ця операція втратила свою минулу важливість, але все ще залишається невід'ємною частиною роботи будь-якої системи. Обидві СУБД виконують видалення за однаковим алгоритмом, але MongoDB, так як немає інформації про свої колекції, як MariaDB про таблиці, виконує видалення повільніше. Це можна бачити за динамікою росту часу виконання запиту з кількістю записів у базі даних у порівнянні з графіком реляційної бази даних.

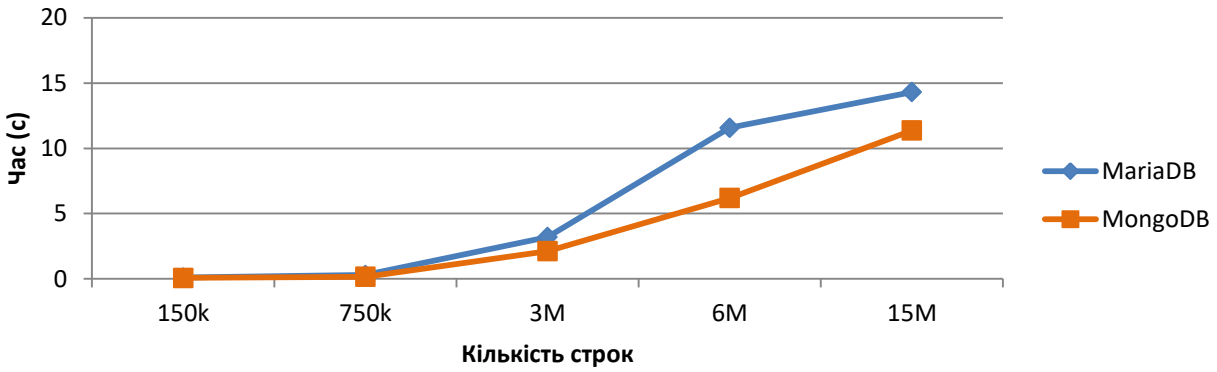

Рисунок 6.5 – Виконання запиту видалення даних без налаштувань БД

Незважаючи на те, що MariaDB без налаштувань має гірші результати, в масштабах великих даних MongoDB програє по показникам продуктивності. Це пов’язано з тим, що у таблицях в реляційних базах є поле AUTO_INCREMENT, яке автоматично визначає порядковий номер запису, який додається до таблиці. Цей параметр зростає на одиницю з кожним новим записом у базі, зазвичай його присвоюють полю з назвою ID, тобто ідентифікатор запису. NoSQL база даних також має такі ідентифікатори, як і масиви, або об’єкти у мовах програмування. Але різниця полягає в тому, що при видаленні певного запису з колекції – необхідно проіндексувати заново всі існуючи записи та присвоїти їм нове значення ідентифікатора. Цей процес вимагає великої кількості ресурсів СУБД, тому зростання часу виконання залежить напряму від кількості записів, а також зокрема від запису, що видаляється.

Останнім тестом виконується об’єднання двох запитів у один та виведення відповіді у єдиному форматі. Результатами цього тестування стають дані, які зберігаються у різних таблицях (колекціях), але вилучені за певних умов, які прописано у запиті. Такі типи запитів виконуються найдовше за будь-яких типів сховищ даних, типу СУБД або даних, що зберігаються. Незважаючи на перевагу тієї чи іншої СУБД, використання таких типів запитів часто є наслідком поганого проектування або неправильного масштабування системи. Результати тестування показані на рисунку 6.6.

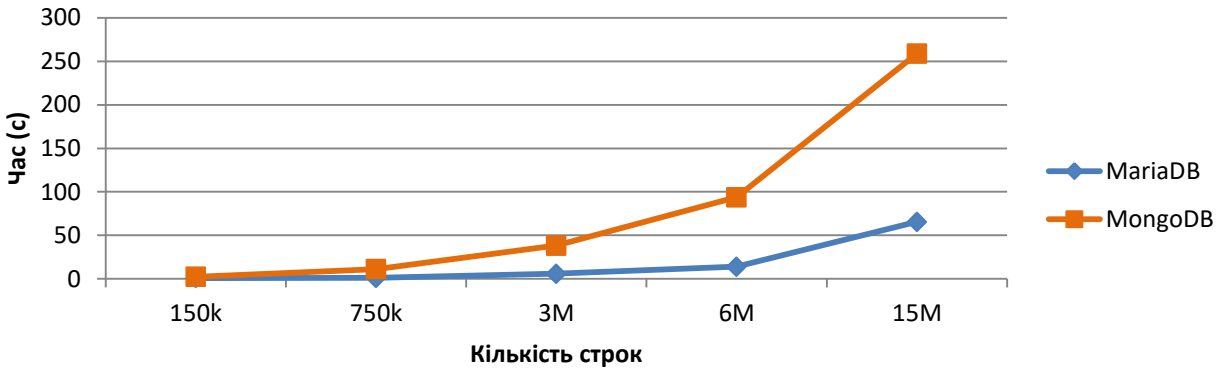

Рисунок 6.6 – Виконання запиту об’єднання даних з різних сховищ без налаштувань БД

Значна для СУБД різниця виникає навіть до позначки в один мільйон записів. Це визвано тим, що однією з цілей створення нереляційних баз було уникнення ситуацій об’єднаних запитів. У будь-якому випадку складні запити потребують більше пам’яті та потужності серверу, саме цю проблему і вирішують швидкі колекції замість повільних таблиць. Таким чином, MariaDB архітектурно створювалася для виконання запитів об’єднання, тому навіть на невеликій кількості записів позувала кращі результати. Так як об’єднання майже завжди відбувається по ключу, який представляє собою значення типу int, тому пошук по таблицях відбувається швидко. Зважаючи на те, що з використанням навіть базових налаштувань за замовчуванням, такі поля в реляційних базах даних індексуються, то швидкість операцій типу JOIN або UNION може конкурувати з простими операціями зі списку CRUD (до п’ятиста тисяч записів). Так як у NoSQL базах даних зберігаються колекції, вони повністю проглядаються при операції об’єднання та для видачі результату об’єднують дані із різних колекцій в третю, що також займає певний час, особливо при роботі з великими даними.

Отже, порівнюючи бази даних без оптимізації виконання запитів і СУБД, можна зробити висновок, що кращою за продуктивністю є нереляційна база даних. Вона показала кращі результати виконання базових CRUD операцій, що є основною роботою СУБД у будь-якій системі. При виконанні складних запитів з об’єднанням, лідером стала реляційна база даних завдяки використанню таблиць, як сховищ даних. Швидкість додавання записів до таблиці не може оцінюватися, як важлива характеристика СУБД, так як дані у реальних системах додаються не безперервним потоком, а тому виконуються майже однаково з несуттєвою

різницею у сторону MongoDB навіть при відсутності налаштувань СУБД. Підсумкові результати тестування СУБД без налаштувань представлені у таблиці 6.1. Позначеннями «к» позначено тисячі, «М» – мільйони і «с» – секунди виконання запитів.

Таблиця 6.1 – Результати тестування СУБД без налаштувань БД

	MariaDB, с					MongoDB, с				
	150 т	750 т	3 М	6 М	15 М	150 т	750 т	3 М	6 М	15 М
INSERT	37.25	97.2	344.2	448.15	1430.1	40.85	111	360.7	715.2	2184.1
SELECT	1	1.1	2.78	4.1	7.52	0.24	0.3	0.4	1	2.73
UPDATE	1	1.22	2.48	4.5	14.23	0.12	0.19	0.42	1	2.54
DELETE	0.1	0.3	3.2	11.58	14.31	0.08	0.17	2.13	6.2	11.37
JOIN	1	1.27	5.96	13.93	65.61	2.53	11.26	38.4	93.9	258.7

У разі використання СУБД з налаштуваннями за замовчуванням, різниця між часом виконання запитів до бази зі 150 або менше тисячами записів виконуються однаково швидко з точки зору користувача, який використовує систему, що працює з базою даних, як сервіс (DBaaS). Без налаштувань, нереляційна СУБД у середньому в 3 рази швидше обробляє запити та дає відповідь, ніж реляційний аналог. Важливішими результатами є обробка невеликих даних і великих, так як можливо проаналізувати поведінку обох СУБД за абсолютно різних навантажень. У більшості випадків на практиці, великі дані будуть розподілені між собою для збереження властивості доступності даних.

Тестування без налаштувань дозволяє зрозуміти наскільки продуктивними є СУБД при виконанні запитів. Під продуктивністю розуміється не тільки можливість бази даних виконати запит користувача за максимально короткий час, а й виконати повторний запит без затримок та виконання операцій, що займають ресурси сервера. Таким чином, продуктивною СУБД можна вважати таку, що швидко обробляє запити користувачів, здатна відтворювати результати попередніх запитів та витрачати якого менше ресурсів при конанні запиту. Підвищення продуктивності баз даних відбувається за рахунок налаштування серверу, самої СУБД, сховищ даних і оптимізації запитів, які можуть виконувати користувачі до того, як вони потраплять до обробки. Не існує інструкції або гарних практик щодо підвищення продуктивності бази даних. Кожен випадок є унікальним в залежності від типу даних, що зберігаються, а також кількості таблиць та розподілення навантаження на систему. Налаштування кожної СУБД – це універсальна робота по пошуку слабких точок, частіше за інше, практичним

шляхом. Не рідко буває, коли спроби підвищити продуктивність приводять до повного краху бази даних, тому проводити налаштування слід поступово, маючи резервні копії даних та конфігурації СУБД.

В обох СУБД були виконані налаштування серверу та сховищ даних, які детально описані у розділі 5. Так як метою експерименту є порівняння продуктивності бази даних, запити залишаються однакові, як і у попередньому тестуванні. Покращення у продуктивності виникають за рахунок індексів, кешування (запитів, ключів, індексів, результатів), мінімізації ресурсів для виконання запитів. Розраховується, що СУБД на цьому етапі експерименту покажуть свої найкращі можливі показники.

При повторному тестуванні не буде проводитися аналіз операції додавання даних (INSERT), так як у потоковому режимі він працює повільно [33]. Це пов’язано з тим, що створення нового індексу потребує опрацювання інших, вже існуючих індексів, перезапис їх до пам’яті, тому виникають додаткові операції, на які витрачається час. Такий алгоритм дій відбувається в обох СУБД, але MongoDB справляється з ним краще. Зважаючи на описані раніше причини некоректності порівняння цього типу запиту, він взагалі не буде враховуватись при проведенні наступних тестів з налаштованими СУБД. Саме з цієї причини на практиці використовують пакетні операції, тобто об’єднання запитів користувачів при зверненні до бази даних, та їх обробка у рамках однієї транзакції. Виконуватися такі транзакції повинні на невеликих об’ємах даних, коли затримка перед поверненням відповіді користувачу – мінімальна. У цілях підвищення скорості роботи розподіленої СУБД використовуються системи розподілених серверів (Content Delivery Network, або CDN), які локально розташовуються у різних місцях. Для користувача вибирається той сервер, який знаходиться до нього найближче. Таким чином затримка передачі даних буде мінімальна, зазвичай у гарно налаштованих системах вона не перевищує 60 мілісекунд, але може доходити і до 200 мілісекунд.

Для отримання результатів роботи безпосередньо СУБД, а не її кешу, результати, які отримуються у ході виконання запитів, не зберігаються. Це означає, що база даних кожен запит виконує заново, у повному обсязі проходячи всі необхідні операції. Більш того, використовуючи кешування, оптимізатор запитів може урізати оригінальні запити у MariaDB, що призведе до виконання різних запитів у тестуванні та невірні результати.

Додаткову затримку при виконанні запитів може створювати мова програмування, яка використовується у системі. В залежності від того, чи встановлені драйвери конкретної СУБД, та який тип даних використовується, результати MongoDB та MariaDB можуть створювати різну затримку. Тому усі запити створюються і обробляються безпосередньо всередині СУБД, отже затримки між виконанням запиту та отриманням відповіді не існує. Деякі мови програмування можуть використовувати додаткові засоби кешування на стороні самої системи, що також створює інші умови експлуатації та може призвести до похибок у розрахунках. Комбінації налаштувань СУБД та кешування на стороні сервера дозволять опрацьовувати великі дані без бачимої для користувача затримки.

Так як усі тести запускаються на локальній машині, жодних затримок не виникає, тому результати, які отримані у ході тестування, повністю відповідають показникам роботи виключно СУБД. Умови виконання запитів з оптимізацією не змінилися. Так, результати читання даних показано на рисунку 6.7.

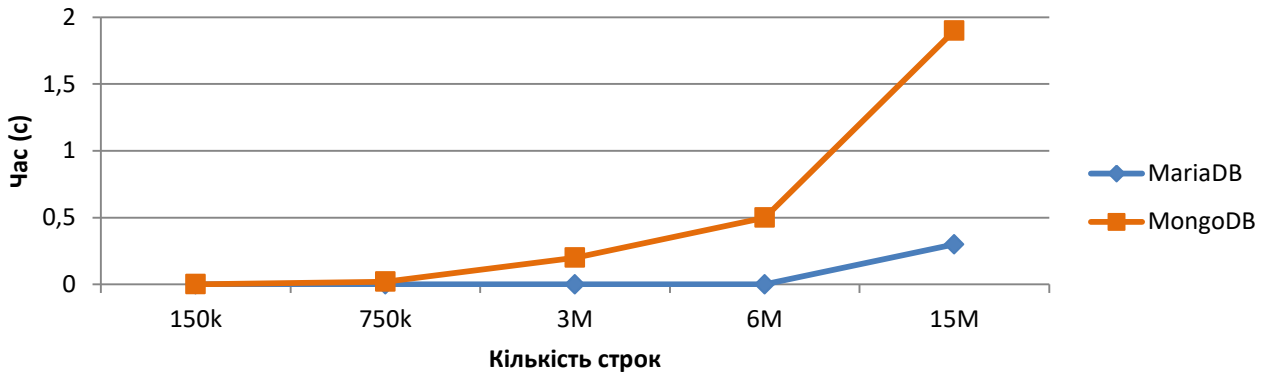

Рисунок 6.7 – Виконання запиту читання даних з налаштуваннями БД

Після налаштувань обох СУБД, показники читання даних покращилися в обох випадках. Але найкращих результатів вдалося досягти реляційній базі даних, яка показала результат у три тисячних секунди для обробки та видачі шести мільйонів запитів. Незважаючи на те, що NoSQL база даних також виконує цей запит з непомітною для людини швидкістю, результати відрізняються небагато менше ніж в тисячу разів. По фінальному тестуванню на п’ятнадцяти мільйонах запитів, відрив між СУБД склав небагато більше, ніж повтори секунди, що значить, MariaDB виявилася у сім разів швидшою за MongoDB у читанні даних.

Вже після одного мільйона записів у колекції, без кешування нереляційній базі складніше обробляти об’єкти, що зберігаються. При неконтрольованому зростанні обсягу колекції, втрата швидкості у виконанні запитів може виявитися критичною для працездатності системи. Завдяки індексам полів у реляційній базі – пошук йде по ним, а не по самій таблиці. Таким чином процес обробки запита прискорюється.

Результати тестування отримано завдяки індексації полів cilents.name і clients.status, по яким йде пошук записів у сховищі. Індексація відбувається у момент додавання запису до таблиці і використовується на протязі всього існування даних у таблиці. При додаванні чи видаленні даних відбувається переіндексація полів, таким чином індекси завжди знаходяться у актуальному стані. Такий вид оптимізації дозволяє швидко опрацьовувати великі дані, але займає певну частину пам’яті на сервері. Тому індексація зазвичай використовується тільки на найважливіших полях бази даних, які часто використовуються у запитах або є зовнішніми ключами для інших таблиць. Гарною практикою вважається індексація полів типу int або різновидів цілих чисел, так як вони займають наймеш можливу кількість пам’яті та швидше за інші обробляються СУБД. У MongoDB також використовуються складові індекси, які застосовуються до декількох колонок (або полів) у рамках одного сховища даних. Такі індекси використовуються, коли заздалегідь відомо, що поля будуть використовуватися у складних запитах. По складових індексах також проводять сортування. Використовуються вони як в усіх реляційних базах даних, так і у нереляційних аналогах.

Оновлення даних також залежить від налаштування полів сховищ. Коли виконується операція оновлення даних – індекси не змінюються, так як прив’язані до конкретного запису. Тому операції оновлення можуть виконуватися швидше за додавання та видалення, але трохи повільніше звичайного читання, так як відбуваються додаткові операції по заміні даних, що зберігаються. Незважаючи на це, у нереляційній базі даних, операції оновлення виконувалися швидше за читання, що може бути завдяки пам’яті СУБД або кешуванню індексів. Результати тестування оновлення даних показані на рисунку 6.8. Виконання операції оновлення даних не залежить від типу даних, які передаються до бази, так як основним завданням СУБД є пошук строчки, до якої потрібно внести зміни. Зазвичай об’єм даних, що передаються до бази, настільки невеликий, що часом

обробки цих даних можна знехтувати при розрахунку часу виконання запиту до БД.

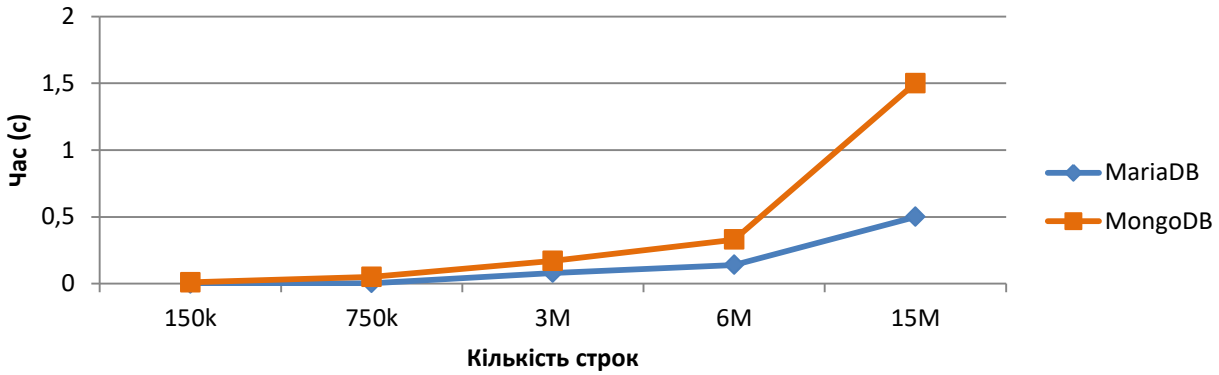

Рисунок 6.8 – Виконання запиту оновлення даних з налаштуваннями БД

На відміну від читання даних, операція оновлення показує гіршу тенденцію зросту часу з кількістю запитів. Після обробки трьох мільйонів записів в таблиці, час на виконання запитів UPDATE в обох СУБД починає різко зростати. Завдяки індексації полів clients.status та clients.name запит виконується найшвидше. При оновлені даних також важливо вказати правильний буфер пам’яті, який може використати СУБД для обробки запиту. Це дозволить підвищити пропускну здатність бази даних під час обробки потоку великих даних.

Не зважаючи на той факт, що не реляційні бази даних були розроблені для заміни класичної реляційної моделі саме в області читання та оновлення даних, кожен з тестів показав, що це можливо лише за умов невдалого налаштування обох СУБД, або його відсутності. Якщо використовувати всі можливі налаштування РСУБД, а також гнучкі можливості роботи із запитами та таблицями, то у результаті отримаємо те, що реляційна база даних у три рази швидше оновлює дані, що зберігаються. Вважаючи на те, що колекції легко піддаються шардінгу, що було описано раніше, такої проблеми на практиці не виникає. Завдяки розподіленню колекцій, які зберігають дані користувачів, їх обробка не займає значного часу. Який би міг вплинути на працездатність всієї системи. Однак при тестуванні з навантаженням, тенденція зростання асу виконання запиту у MongoDB набагато вища, що призведе до швидкого отримання помилок. Такий темп зростання часу не дозволяє заздалегідь продумати плани проведення налаштувань, та у разі виявлення такої затримки на

практиці – скоріше за всього збитки будуть великими. Тем не менш, найвищий результат у півтори секунди є допустимим при роботі по мережі. З точки зору використання запитів, немає переваг у використанні потоків даних, чи окремих запитів. Так як скоріше за всього будуть використовуватися різні поля, обробка запитів оновлення буде виконана за однакову кількість часу.

Видалення даних з бази виконується рідше, ніж попередні дві операції. Сучасні системи можуть собі дозволити не видаляти дані, так як кількість пам'яті зростає, а сама пам'ять стає дешевшою. Тем не менш, дані видаляються, із юридичних причин, або у випадку невеликих систем заради економії простору на сервері, особливо коли у базі даних зберігаються конфігураційні дані. Результати тестування запитів видалення даних показані на рисунку 6.9.

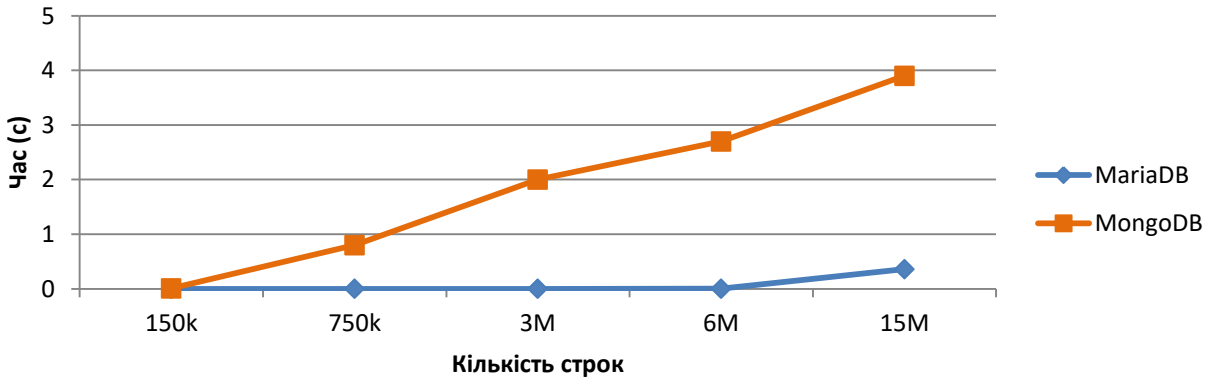

Рисунок 6.9 – Виконання запиту видалення даних з налаштуваннями БД

З рисунку 6.9 видно, що запит на видалення даних у нереляційній базі навіть з відмітки сто п'ятдесят тисяч записів почав швидко зростати. Це обумовлено типом роботи колекції, як механізму зберігання даних. Так як вона схожа на об'єкт або масив у мові програмування – кожний запис має свій внутрішній ідентифікатор, або індекс. Так як один із записів зникає – ітеративність індексу також змінюється. Для того, щоб відновити правильність індексування записів у колекції – вони перезаписуються кожного разу, як змінюється їх послідовність. Фактично, з кількістю записів, зростає кількість перезаписів, що відбуваються на кожній ітерації видалення даних з таблиці. Саме завдяки цій механіці обробки даних MongoDB має такий високий час виконання записів видалення даних. Навіть налаштування пам'яті чи кешування не допоможе досягти кращих показників часу для виконання цього запиту. Одним із

значних мінусів нереляційної бази є така поведінка при видалені даних, хоча на невеликих об’ємах записів, що зберігаються, це помітити складно, так як час виконання ненабагато більший, а рівний реляційному аналогу.

При використанні операції видалення у не потоковому режимі, на обсягах даних до ста п’ятдесяти тисяч, швидкість завдяки налаштуванням буде однакова швидка в обох СУБД. Хоча слід зауважити, що так як і оновлення, видалення даних частіше інших типів запитів виконується саме у потоці.

Індексація полів значно полегшує пошук записів для видалення у реляційних таблицях. Завдяки відокремленню індексів від даних, які зберігаються, їх видалення або пошкодження не впливають один на одного. Переіндексація відбувається тільки у випадках додавання даних до таблиць, або індексування за SQL командами (у ручному режимі). Інкремент ідентифікаторів таблиць також не впливає на обробку даних у таблиці, тому інтенсивність росту часу виконання запитів невелика і виконується на одному часовому рівні в порівнянні з усіма CRUD операціями, що роздивляються у рамках цього експерименту.

Останнім етапом тестування є перевірка продуктивності СУБД при виконанні об’єднаних запитів. Незважаючи на складність виконання запитів, маніпуляцій з даними, що зберігаються, не виникає, тому швидкість обробки даних повинна дозволити виконати вибірку даних якомога швидше. Сам запит представляє собою читання даних, але за умов використання декількох таблиць (або колекцій). Кожне поле, яке використовується СУБД у якості зовнішнього ключа або ідентифікатора, – індексовано, а саме clients_comments.uid. Більшу складність запиту одає пошук по імені користувача, тобто символьний пошук. Хоча поле clients.name також проіндексовано, символьний пошук виконується довше за рахунок порівняння більшої кількості даних, ніж у випадку з числами. Символьний пошук у нереляційній базі виконується швидше, ніж у РСУБД, завдяки формату BSON, який використовується для зберігання даних. Зважаючи на цей факт, у якості зовнішнього атрибута у колекціях в MongoDB було обране унікальне поле email користувача. Він є локальним та зовнішнім ключем у колекціях, по якому можна проводити агрегації. Такий варіант вирішує проблему об’єднання даних та не створює у колекціях зайві поля, які не несуть корисної інформації. Так як колекції не мають метаінформації та читаються так, як є (as-is), зайві технічні поля, наприклад ідентифікатори, можуть заплутати інженерів, особливо коли їх з’являється більше одного для різних колекцій. Унікальні поля

використовуються без втрати швидкості, більше того, після додавання складеного індексу – по швидкості пошуку вони рівносильні числовим ідентифікаторам, які також мають індекси. Результати виконання запитів з об'єднанням запитів показані на рисунку 6.10.

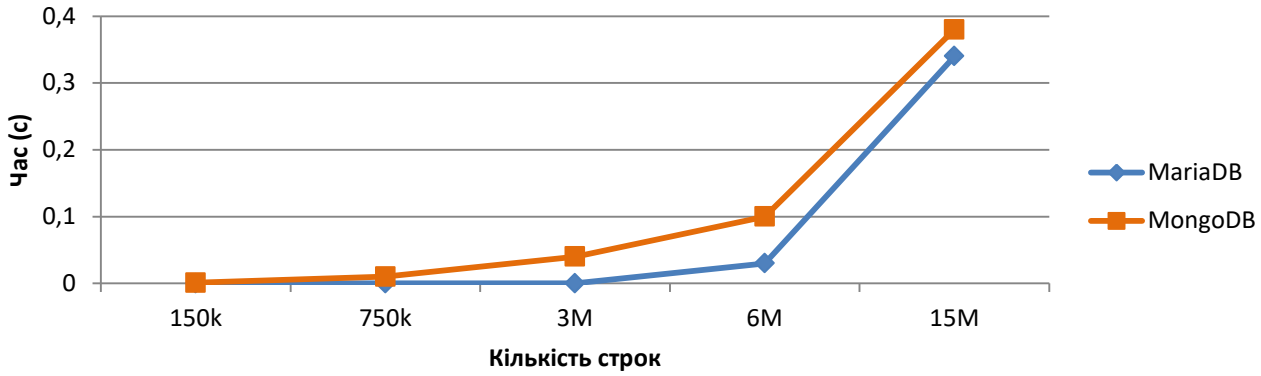

Рисунок 6.10 – Виконання запиту об'єднання даних з різних сховищ з налаштуваннями БД

Динаміка росту часу виконання запитів показує, що виконання операції об'єднання для багатьох таблиць може призвести до значного сповільнення роботи СУБД, тобто втрати продуктивності. Завдяки індексам, результати тестування запиту об'єднання лише на декілька сотих краще, ніж читання даних. За умов використання строкових індексів або більше, ніж трьох таблиць у одному запиті – час виконання може показувати погіршення у три чи більше разів.

За умови розподіленої бази даних, запит об'єднання ніяк не оптимізується, на відміну від звичайного читання чи оновлення даних. Будуть відбуватися однакові процеси об'єднання таблиць, що призведе до тих самих результатів, що і тестування в умовах єдиної бази даних. Усі оператори об'єднання даних, що доступні у мовах SQL та обробниках запитів NoSQL баз даних, мають на увазі збільшення часу виконання запиту та допускаються до використання лише за умов, коли запит не взаємодіє напряму з користувачем, а потрібен для аналізу або аналітики. Таким чином, використання такого типу запитів може бути непродуктивним для СУБД при певних умовах. Кожного разу розробникам необхідно тестувати виконання об'єднаного запиту, або простих CRUD операцій окремо. Якщо результат запиту до бази даних містить в собі декілька операцій їх можна помістити до блоку COMMIT та виконати послідовно, додавши певні

обробники даних на стороні системи за логікою роботи запиту або проектним рішенням. Як правило, більшість таких запитів виявляється більш продуктивною, ніж використання JOIN у реляційній базі та $lookup у MongoDB. Для окремих запитів також використовується менше пам'яті СУБД та вона скоріше звільнюється у порівнянні з використанням операцій об'єднання, що дозволяє збільшити відказостійкість СУБД та обробляти більшу кількість запитів у певний момент часу. Загальні результати тестування з налаштуваннями баз даних показані у таблиці 6.2.

Таблиця 6.2 – Результати тестування СУБД з налаштуваннями БД

	MariaDB, с					MongoDB, с				
	150 т	750 т	3 М	6 М	15 М	150 т	750 т	3 М	6 М	15 М
INSERT	10^{-5}	10^{-5}	10^{-5}	$3 \cdot 10^{-5}$	0.3	0.001	0.02	0.2	0.5	1.9
SELECT	10^{-5}	0.003	0.08	0.14	0.5	0.01	0.05	0.17	0.33	1.5
UPDATE	10^{-5}	10^{-5}	10^{-5}	0.004	0.36	0.01	0.8	2	2.7	3.9
DELETE	10^{-5}	10^{-5}	10^{-5}	0.03	0.34	0.001	0.01	0.04	0.1	0.38
JOIN	10^{-5}	10^{-5}	10^{-5}	$3 \cdot 10^{-5}$	0.3	0.001	0.02	0.2	0.5	1.9

Результати тестування показують, що на невеликих обсягах даних, усі запити однаково продуктивно можуть бути опрацьовані СУБД. Але це не означає однакову продуктивність при виконанні запитів на великих даних. Реплікації та шардінг, як вказувалося раніше, ефективно працюють на великому обсязі даних, тобто в кожному окремому сховищі може бути навіть більше п'ятнадцяти мільйонів записів. Саме тому налаштування роботи бази даних має свої переваги та тестування проводилося на різних об'ємах даних для порівняння продуктивності СУБД.

У підсумках експерименту можна однозначно стверджувати, що більш продуктивною СУБД є реляційна MariaDB. Вона показала кращі результати за всіма тестуваннями, за умов налаштування бази даних. Завдяки правильно вказаним індексам та структурі таблиць, виконання запитів займає мінімальну кількість часу та ресурсів сервера, а також дозволяє кешувати результати для наступних запитів, як ідентичних, так і запитів по обраних полях. Реляційні бази даних виникли і розвиваються вже не першу декаду, що дозволяє більш точно налаштовувати СУБД під конкретні нужди системи, яка використовує базу даних. Саме налаштування серверу бази даних, правильне збереження даних у таблицях з використанням найбільш відповідних типів даних, а також по можливості оптимізації запитів від системи дозволяють РСУБД зберігати перевагу при роботі

з великими даними. Незважаючи на те, що більшість запитів виконується майже на одному рівні с не реляційною базою, видалення даних та операції об'єднання ще залишаються невід'ємною частиною сучасних систем, тому їх треба враховувати при виборі СУБД. При використанні баз даних без додаткових драйверів та модулів, реляційна база даних показує кращий результат по виконанню запиту, але враховуючи той факт, що на практиці відповіді на запити обробляються у системі, витрачається додатковий час на приведення даних з СУБД у об'єкт, який використовується у вибраній мові програмування. Тем не менш, така затримка може стати вирішальною при виконанні запитів до великих даних, так як розмір об'єкту, який приходить у відповідь на запит, може бути більшим, ніж очікується. При рівних умовах використання обмеженої кількості пам'яті, обидві СУБД можуть рівнозначно гарно обробляти потоки даних, навіть від групи клієнтів.

Отже СУБД реляційного типу має однозначні переваги, але нереляційні бази даних також популярні та розвивають свій потенціал. Це відбувається за рахунок того, що у сучасних системах недостатньо використовувати лишу гарно налаштовану розподілену базу даних, але й використовувати програмні оптимізатори, так як бібліотеки для мов програмування, кешування на стороні системи, мінімізація кількості та підвищення якості запитів, правильна архітектура бази даних, особливо при масштабуванні та розподіленні.

Значно більшої продуктивності СУБД досягають за рахунок кешування на стороні бази даних, яке було описано раніше, та на стороні системи, що використовує базу даних. Такими програмними доповненнями можна створювати «in-memory» кеш, який зберігає корисну інформацію щодо запитів у оперативній пам'яті та підставляє значення замість запитів, які можна замінити. Таким чином спадає навантаження на базу даних, а користувачі отримують відповіді миттєво. Актуальність даних синхронізується з базою даних для підтримки актуальності даних. Такі системи також називають пошуковими двигунами.

Порівнюючи СУБД та пошукові двигуни, варто зазначити, що вони мають однакові для виконання запитів користувачів. Але для організації та отримання даних, кожна технологія покладається на різні парадигми. Хоча в базах даних можна зберігати та обробляти велику кількість структурованих даних, пошукові системи можуть шукати неструктурований текст, навіть ефективніше, ніж не реляційні бази даних, які не мають уповільнення пошуку по строковим значенням.

Пошукові двигуни відрізняються від баз даних за обсягом. Вони застосовують менші за розмірами програмні засоби повнотекстового пошуку, такі як онлайн-публікації, інтерактивні каталоги, класифікована реклама, управління цифровими активами, аналіз та пошук в Інтернеті. Програмне забезпечення пошукового двигуна значно спрощує процес пошуку відповідної інформації, дозволяючи користувачам шукати інформацію текстом на природній мові. Потім результати будуть організовані за релевантністю, яка може включати текстові документи, географічну інформацію, зображення, відео, аудіо та інші дані корисного навантаження, тобто дані, які несуть практичну користь для користувача, що їх запитує.

База даних з пошуковим двигуном – це різновид нереляційної бази даних, який присвячений пошуку змісту даних у сховищах різних типів, в тому числі і розподілених. Пошукові двигуни використовують індекси для класифікації схожих між собою характеристик серед усіх даних, що зберігаються, для зменшення часу на пошук та підвищення продуктивності пошуку. Вони призначені для роботи з даними, особливо великим даними, напів-структурованими або неструктурованими, і вони, як правило, пропонують спеціалізовані методи для обробки сирих даних, тобто будь-якого формату. Прикладом одного с кращих пошукових двигунів є ElasticSearch, який посідає перше місце серед двигунів на ресурсі DB Engines. У неї використовується документно-подібна структура зберігання даних, а також доступна версія для хмарного використання. Використовуючи SQL мову для запитів, можливість використання тригерів або методи реплікації та шардінгу, робить цей двигун продуктивнішим у зв’язці з реляційною базою даних. Нажаль, напряму підтримується небагато мов програмування, але більшість конфігурацій та дій по обробці запитів відбувається з основною базою даних, яка може взаємодіяти з пошуковими двигунами, або деякі частини системи можуть також напряму звертатися до пошукових двигунів, отримуючи такі самі відповіді, як і у випадку бази даних, але для цього потрібно проводити додаткову конфігурацію у системі.

На практиці частіше використовується «in-memory» програмний кеш, який використовує оперативну та статичну пам’ять на сервері для збереження інформації щодо виконання та запуску запитів від системи до бази даних. У мові програмування PHP найчастіше використовується Redis або Memcached, але їх застосування повинно бути обґрунтованим з точки зору обсягу даних, що зберігаються. Гарним тоном вважається запуск цих програмних додатків в умовах

сильної завантаженості серверу та наявності його достатніх обчислювальних потужностей. У іншому випадку, використовуючи ці сховища типу «ключ-значення», виконання запитів лишу уповільниться, а система буде вимагати більше пам’яті та обчислювальних ресурсів. Не існує однозначної цифри, коли потрібно починати використовувати кешування. Більшість думок основана на практичному досвіді спеціалістів, хоча й на пряму залежить від типу даних, які зберігаються. Усіма признаний факт, що кращим значенням у середньому випадку, починати використовувати програмне кешування можна з відмітки у десять мільйонів записів.

Перевагами таких додатків до основної бази даних є створення великих пар типу «ключ-значення», які можуть зберігати до 512 мегабайт даних для кожного поля, тобто для одного запиту може бути збережено до одного гігабайту даних у кеші. Таким чином, навіть складний запит до великих даних може бути оптимізований до видачі результатів за лічені секунди. Більше того, Redis використовує власний алгоритм хешування, тобто збереження пар у власному сховищі у найбільш доступному для подальшого використання стані. Окремо від основних бази даних, можливо створювати програмні реплікації, що дозволять виконувати запити на менших об’ємах даних, відповідно покращити рівень продуктивності використання СУБД. Реплікації доступні лише типу «Master-Slave». Оскільки програмний кеш можна використовувати для налаштування ефективної реплікації, в будь-який момент часу служба кешування буде працювати і працює, навіть якщо будь-який з підлеглих вузлів не працює. Таким чином зберігається постійна доступність даних. Так як програмні кеші різного рівня встановлюються для більшості мов програмування, їх підтримка та налаштування не займає багато часу та коштів. Зазвичай, Redis встановлюється за замовчуванням навіть у популярні фреймворки, але вимкнений по замовчуванню. Сховище «ключ-значення» має систему передачі повідомлень із програмним кодом, який його використовує, тобто можливо швидко отримувати необхідні дані. Незважаючи також на обмеженість пам’яті, яка використовується для кешування, додавати і видаляти з пам’яті дані, запити або відповіді сервера дуже швидко і легко завдяки функції масової вставки значень. Так як Redis підтримує транзакції, команди можуть бути виконані послідовно, а не одночасно, як за їх відсутності. Це дозволяє підвищити безпеку та надійність виконання запитів у рамках будь-якого типу систем або додатків.

При використанні будь-якої СУБД, описані вище методи будуть покращувати продуктивність виконання запитів. При виборі певних переваг кожного типу баз даних, необхідно мати на увазі, яка буде більшість запитів у системі та який тип даних використовується найчастіше. Це дозволить спроектувати найкраще рішення з використанням необхідних технологій. Допускається, що навіть в рамках однієї системи можливе використання декількох типів СУБД, які розподілені між собою. Наприклад, маючи базу клієнтів, або базу знань, краще використовувати не реляційну модуль СУБД та сховища у вигляді документів, так як з даними, які можна отримати по запитах, легше працювати на наступних етапах роботи системи. Також, швидко доступними у документно-подібних базах даних вважаються тексти, конфігурації, аналітичні дані та інша статика, яка має якості накопичування та здебільшого використовуються у запитах читаннях, рідше – оновлення і дуже рідко – видалення даних. З іншого боку, динамічні дані, що безпосередньо використовуються системою та користувачами, продуктивніше обробляти у реляційних базах даних за рахунок структури сховищ даних та зв’язків між ними.

Об’єднувати декілька типів СУБД у одній системі варто з огляду на архітектуру бази даних та їх обсягу. Якщо структурно зручніше зберігати дані у єдиному місці, або це обумовлено програмними обмеженнями системи, додавання різних типів СУБД та їх конфігурації призведе лише до втрати продуктивності. Якщо ж база даних не має великих даних, а використовується для обробки даних до 7-8 гігабайт, то підключення різнорідних СУБД також не дасть значного покращення у часі виконання запитів. Ця перевага дозволяє створювати великі проекти командою новачків або у обмежений строк, так як масштабування і налаштування такої СУБД може бути виконана пізніше. РСУБД потрібно на етапі проектування проаналізувати та тільки після цього створювати. Так чи інакше, у більшості випадків СУБД встановлюють із фреймворком, які використовуються занадто часто, як платформи для створення нових програмних продуктів. Вони надають на вибір не тільки реляційні СУБД, як було раніше, а повний список популярних рішень, що сприяє росту популярності NoSQL баз даних та підходу в цілому. Зважаючи на постійне вдосконалення обох типів баз даних, не виключено, що вони стануть прямими конкурентами одне одному, замість конкуренції між реляційними базами даних.

ВИСНОВКИ

У ході виконання роботи були вирішені задачі обзору існуючих робот за даною тематикою, огляд предметної області та виявлення невирішених проблем, підготовка баз даних та синтетичних даних, визначені та застосовані налаштування до СУБД та сховищ даних. Аналіз результатів виконання простих CRUD запитів та операцій об’єднання запитів до налаштованої та не налаштованої бази даних привів до наступних висновків:

1. за умов роботи з обсягом даних до 1 мільйона записів – обидва типи СУБД, реляційна та NoSQL, обробляють запити за менш ніж 0,001 секунди, навіть за умов оброблення потоку даних. Різниця між результатами існує, але не настільки велика, щоб робити висновки щодо переваг того чи іншого типу СУБД, таким чином при використанні шардінгу обидві моделі баз даних працюють однаково добре;
2. за відсутності навіть мінімального налаштування кешування та буферу пам’яті, полей таблиць та колекцій, оптимізатору запитів і його параметрів, нереляційна СУБД виконує базові CRUD операції швидше, ніж реляційна, у випадку побудови непов’язаних між собою колекцій, коли не виконуються складні запити з об’єднанням. Тобто, NoSQL СУБД не вимагає від розробників додаткових знань для роботи з даними, не потребує написання додаткових скриптів для підтримки швидкості роботи з даними і може бути створена значно швидше за рахунок використання базових налаштувань СУБД;
3. налаштування СУБД суттєво впливають на швидкість виконання запитів. Зважуючи на тип даних, що обробляються, та кількість запитів, можна підвищити продуктивність роботи системи з базою даних та підвищити швидкість обробки запитів з 7-14 секунд до 0,3-0,5 секунд.

При роботі з великими даними, індекси та кешування реляційних баз дозволяють значно швидше виконувати запити будь-якої складності. Хоча документно-орієнтовані бази даних легше та швидше масштабуються, така перевага перекривається швидкістю виконання запитів за умов використання таблиць. Так як MySQL бази даних старіші за новий NoSQL підхід, можливо з додаванням більшої кількості налаштувань до СУБД, згодом можливо бути використовувати лише їх.

Використання різнорідних баз даних в рамках однієї системи може бути оправдано лише її архітектурою. Для складних систем рекомендується використання сховища типу «ключ-значення» в оперативній пам’яті для зберігання динамічних даних. У якості основної бази даних – використовувати реляційну СУБД, так як по результатам експериментів вона виявилася найпродуктивнішою. Більше того, РСУБД використовує найпопулярніший язик запитів SQL, який є потужним інструментом роботи з даними. Для сховищ різнорідних даних, особливо неструктурованих, краще використовувати нереляційні бази даних, так як у них легше опрацювати великі дані для аналізу, статистики і звітів.

При використанні бази даних, як сервісу (DBaaS), в тому числі при створенні serverless програмних додатків, краще використовувати колонкові бази даних у парі з хмаровими сервісами. Так як сервер обробки інформації відсутній – найважливішим параметром є швидкість виконання запитів та мінімальний час роботи процесору.

Для виконання запитів з прийнятною швидкістю на різних обсягах даних необхідно виконувати налаштування параметрів буферу пам’яті, кешування та оптимізатору запитів як самої СУБД, так і сховищ даних. Без налаштувань та масштабування бази даних, будь-які запити будуть виконуватися неприпустимо довго. Для реляційних баз даних найшвидшим та найпростішим варіантом масштабування є створення різних шардів для окремих груп користувачів. Нереляційну бази даних можуть масштабуватися легше, в тому числі з використанням реплікацій, але запити не можуть об’єднати декілька розподілених колекцій даних, в той час як у SQL це можливо.

Більшості сучасних систем, які працюють з великими даними, потрібно мати встановлений пошуковий двигун (наприклад ElasticSearch) для підвищення швидкості виконання запитів. Щоб зменшити навантаження на базу даних за умов потокової обробки запитів, слід використовувати кеш запитів і даних на стороні системи, який постійно оновлюється, наприклад Redis. Проміжні висновки та матеріали роботи були опубліковані автором у якості матеріалів тез та доповідей на наукових конференціях та статей у наукових журналах [34][35]. Описані у даній роботі приклади практичного використання та проектування розподілених баз даних та систем з сервісною архітектурою перевірялися при створенні реальних систем в рамках написання автором наукових робіт [36][37][38][39].

ПЕРЕЛІК ПОСИЛАНЬ

1. Strategy Analytics: Internet of Things Now Numbers 22 Billion Devices But Where Is The Revenue?. URL: https://news.strategyanalytics.com/press-release/iot-ecosystem/strategy-analytics-internet-things-now-numbers-22-billion-devices-where (дата звернення 21.10.2019).

2. Jackson B. Google Cloud vs AWS in 2019 (Comparing the Giants). URL: https://kinsta.com/blog/google-cloud-vs-aws (дата звернення: 21.10.2019).

3. Tale S. SQL: The Ultimate Beginners Guide: Learn SQL Today. Бостон : O'Reilly Media, 2016. 199с.

4. Macedo T. Redis Cookbook: Practical Techniques for Fast Data Manipulation. Бостон : O'Reilly Media, 2011. 74с.

5. Kunigk J., Buss I., Wilkinson P., George L. Architecting Modern Data Platforms: A Guide to Enterprise Hadoop at Scale. Бостон : O'Rielly Media, 2019. 636с.

6. DB-Engines Ranking – popularity ranking of database management systems. URL: https://db-engines.com/en/ranking (дата звернення: 21.10.2019).

7. Cloud Object Storage | Store & Retrieve Data Anywhere | Amazon Web Services. URL: https://aws.amazon.com/s3 (дата звернення 22.10.2019).

8. BSON (Binary JSON) Serialization. URL: http://bsonspec.org (дата звернення 21.10.2019).

9. Lilienthal C. Sustainable Software Architecture: Analyze and Reduce Technical Debt. Каліфорнія : Rocky Nook, 2019. 307с.

10. Гавриленко С. Ю., Иванов В. Г. Разработка системы графического описания и моделирования распределенных программных объектов при проектировании информационных систем. *В сборнике научных трудов «Системы обработки информации»*. 2014. № 1(117). С. 10-13.

11. Гребенник И.В., Иванов В.Г., Иванов Д.В., Урняева И.А. Математическая модель задачи планирования передачи файла от нескольких источников потребителю. *В сборнике научных трудов «Системы обработки нформации»*. 2015. № 10(135). С. 82-85.

12. Ambler W. Scott The Object Primer: Agile Model-Driven Development With Uml 2.0. Кембрідж : Cambridge University Press, 2004. 572с.

13. C. Gyorodi, R. Gyorody, G. Pecherle, A. Olah A Comparative Study: MongoDB vs. MySQL. *The 13th International Conference on Engineering of Modern Electric Systems* : матеріали міжнар. наук.-практ. конф., м. Орадя, 11-12 червня 2015р. Орадя, 2015. С. 192-198.

14. C. Gyorodi, R. Gyorody, I. Andrada, B. Livia A Comparative Study Between the Capabilities of MySQl Vs. MongoDB as a Back-End for an Online Platform. *IJACSA*. 2016. №1. С. 73-38.

15. E. Andersson, Z. Berggren A Comparison Between MongoDB and MySQL Document Store Considering Performance : атестаційна робота бакалавра комп'ютерних наук : 122 / Umeo University. Умео, 2017. 65с.

16. D. Damodaran, S. Salim, S. M. Vargese PERFORMANCE EVALUATION OF MYSQL AND MONGODB DATABASES. *IJCI*. 2016. №5. С. 187-194.

17. В. Джонс К. Дж. Методы проектирования. Москва : МИР, 1986. 326с.

18. Клир Дж. Системология. Автоматизация решения системних задач. Москва : Радио и связь, 1990. 534с.

19. Storage Engines - MariaDB Knowledge Base. URL: https://mariadb.com/kb/en/library/storage-engines (дата звернення: 21.10.2019).

20. Hills T. NoSQL and SQL Data Modeling: Bringing Together Data, Semantics, and Software. Вестфілд : Technics Publications, 2016. 258с.

21. Top 11 Open Source Databases for Your Next Project. URL: https://geekflare.com/open-source-database (дата звернення: 21.10.2019).

22. Release Notes – MariaDB Knowledge Base. URL: https://mariadb.com/kb/en/library/release-notes (дата звернення: 23.03.2019).

23. Potter J. MySQL Performance: MyISAM vs InnoDB. URL: https://www.liquidweb.com/kb/mysql-performance-myisam-vs-innodb (дата звернення 23.05.2019).

24. Kelner E., Razzoli F. MariaDB Essentials. New York : Packt, 2015. 206с.

25. Difference Between JSON vs SOAP. URL: https://www.educba.com/json-vs-soap (дата звернення 23.05.2019).

26. Managed MongoDB Hosting | Database-as-a-Service | MongoDB. URL: https://www.mongodb.com/cloud/atlas (дата звернення 21.10.2019).

27. PHP: MySQLi – Manual. URL: https://www.php.net/manual/ru/book.mysqli.php (дата звернення: 26.10.2019).

28. Иванов В.Г., Ныцик А.М. Разработка среды дореализационного моделирования и тестирования программного обеспечения. *В сборнике научных трудов «Системы обработки информации»*. 2010. № 9(90). С. 40-43.

29. Tips for SQL Database Tuning and Performance | Toptal. URL: https:// toptal.com/sql-server/sql-database-tuning-for-developers (дата звернення: 28.10.2019).

30. R. Duyer Learning MySQL and MariaDB. Бостон : O'Rielly, 2015. 408с.

31. How Network Latency Impacts User Experience. URL: https://www.citrix.com/blogs/2017/09/25/how-network-latency-impacts-user-experience (дата звернення: 28.10.2019).

32. $lookup (aggregation) | MongoDB Manual. URL: https://docs.mongodb.com/manual/reference/operator/aggregation/lookup (дата звернення 28.10.2019).

33. B. Schwartz, P. Zaitsev, V. Tkachenko, D. J. Balling High Performance MySQL: Optimization, Backups & Replication. Бостон : O'Rielly Media, 2008. 826с.

34. В. Стьопін, Б. Горбатенко, В. Іванов Автоматизована система обліку занять і генерації звітів. *Міжнародна весняна школа з верифікації та штучного інтелекту* : матеріали міжнар. практ. конф., м. Харків, 25 квітня 2018 р. Харків, 2018. С. 15–22.

35. В. Стьопін, Б. Горбатенко, В. Іванов Парсинг та інтеграція даних в сучасних розподілених web-системах. *XXIII Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті»* : матеріали наук.-практ. конф., м. Харків, 16 квітня, 2019 р. Харків, 2019. С. 7-8.

36. В. Стьопін, Б. Горбатенко, В. Іванов Впровадження мікросервісної архітектури у високонавантажені сучасні web-системи. *Інформаційні технології в культурі, мистецтві, освіті, науці, економіці та бізнесі* : матеріали між нар. наук.-практ. конф., м. Київ, 18 квітня, 2019 р. Київ, 2019. С. 15-18.

37. В. Стёпин, Б. Горбатенко, В. Иванов Сервис автоматизации проверки посещения занятий учащимися в учебных заведениях. *Наукове мислення*. 2017. №2. С. 31-33.

38. В. Стёпин, Б. Горбатенко, В. Иванов Способы автоматизации процесса учёта студентов на занятиях. *Актуальные научные исследования в современном мире*. 2017. №12. С. 33-39.

39. V. Stopin, B. Horbatenko, V. Sayenko Bug Trace service for IBM Bluemix. *Proceedings of the 21st Conference of Open Innovations FRUCT* : матеріали між нар. наук.-практ. конф., м. Гельсінки, 7 листопада, 2017 р. Гельсінки, 2017. С. 503-507.