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 проблем розміщення різнорідних даних великого розміру в розподілених базах даних
(тема)

Виконав:
Студент 2 курсу, групи *СПРм-18-1*
Спеціальність *122 – Комп'ютерні науки*
(код і повна назва напряму)
Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Системне проектування*
(повна назва освітньої програми)

Горбатенко Б. В
(прізвище, ініціали)

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 *Горбатенко Богдану Володимировичу*
(прізвище, ім'я, по батькові)

1. Тема роботи (проекту) Дослідження проблем розміщення різнорідних даних великого розміру в розподілених базах даних
затверджена наказом по університету від « » _____ 2019 р. № _____
2. Термін подання студентом роботи (проекту) _____
3. Вихідні дані до роботи (проекту) Функція: Розробка окремих елементів системи по автоматизації обліку відвідування занять учнями в навчальних закладах. Перелік використаних програмних засобів: OS Windows 10, OS CentOS Linux 7.6.1810 (Core), Sublime Text 3, PHPMyAdmin, MongoDB Server 4.0 X64, Studio 3T, MariaDB 10.3 X64, Navicat for MySQL, MySQL Workbench, Mongo Shell, Apache Server, Google Chrome Browser, Microsoft Office 2016. Технічне забезпечення: комп'ютер, підключений до Інтернету, зі встановленим середовищем розробки та локальним сервером для роботи з базою даних.
4. Зміст пояснювальної записки (перелік питань, що потрібно розробити) 4.1 Вступ, 4.2 Огляд стану проблеми до зберігання та обробки великих обсягів даних, 4.3 Постановка задачі кваліфікаційної роботи, 4.4 Аналіз сучасних методів зберігання різнородних даних, 4.5 Особливості проектування серверних та хмарових баз даних, 4.6 Робота з даними за умов використання розподіленої бази даних, 4.7 Порівняльний аналіз швидкості роботи різних СУБД за умов використання різнородних типів даних, 4.8 Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій

1.1 Логіка роботи обмеження цілісності, 1.2 Обмеження посилальної цілісності і заборона на невизначені значення, 1.3 Узагальнена логіка роботи транзакцій, 1.4 Узагальнена логіка роботи транзакцій, 1.5 Оптимізатор запитів, 1.6 Класифікація нових систем за теоремою CAP, 1.7 Порівняння ACID та BASE, 1.8 Логіка роботи технології MapReduce, 3.1 Популярність використання СУБД за DB Engines, 3.2 Принципи побудови розподіленої системи, 3.3 Різниця між JSON та BSON, 3.4 Одночасне зберігання структурованих та напівструктурованих даних, 4.1 Приклад взаємодії з серверною базою даних, 4.2 Приклад взаємодії з базою даних у хмарі, 5.1 Логічна схема проектованої реляційної бази даних, 5.2 Логічна схема проектованої реляційної бази даних, 5.3 Вихідний код основних алгоритмів генерації даних, 5.4 Приклад даних MariaDB, 5.5 Приклад даних MongoDB, 5.7 Підсумковий обсяг бази даних MariaDB

6. Консультанти розділів роботи

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, наукових статей та існуючих аналогів з теми дипломної роботи	*03.09 — 17.09.17*	
3.	Структурний аналіз та тестування існуючих СУБД на синтетичних наборах даних без вдосконалення	*18.09 — 10.10.19*	
4.	Вибір параметрів вдосконалення різних СУБД та їх вплив на оцінку тестування	*11.10 — 18.10.19*	
5.	Тестування запитів на оптимізованих СУБД	*19.10 — 28.10.19*	
6.	Проектування структури СУБД для тестування складних запитів	*29.10 — 11.11.19*	
7.	Оформлення пояснювальної записки та графічних матеріалів	*12.11— 26.11.19*	
8.	Оформлення презентаційних матеріалів	*27.11 — 04.12.19*	
9.	Представлення на рецензування	*06.12.19*	
10.	Представлення дипломної роботи в дек	*10.12.19*	

Дата видачі завдання _____ _____20__ р.

Студент Горбатеко Б. В.

(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 8
1 Огляд стану проблеми до зберігання та обробки великих обсягів даних 9
2 Постановка задачі кваліфікаційної роботи 21
3 Аналіз сучасних методів зберігання різнородних даних 23
4 Особливості проектування серверних та хмарових баз даних 33
5 Робота з даними за умов використання розподіленої бази даних 41
6 Порівняльний аналіз швидкості роботи різних СУБД за умов використання різнородних типів даних 53
Висновки 64
Перелік посилань 66
Додаток А 69
Додаток Б 103

ВСТУП

Система управління базами даних (СУБД) – це програмний інструмент або іншими словами інтерфейс між кінцевим користувачем та програмою, і природно, самої базою даних, на якій виконуються завдання. За допомогою СУБД легше і зручніше працювати з інформацією. Наприклад, створювати, оновлювати, шукати, видаляти і відновлювати дані в базах даних (БД), а також визначати взаємозв'язки між її компонентами (таблицями для реляційного типу або колекціями для NoSQL рішень).

Сьогодні, при розробці більш-менш серйозних веб додатків або прикладних рішень гостро постає питання у виборі системи управління базами даних для виконання покладених на неї завдань. База даних повинна відповідати ряду вимог, серед яких: надійність, розширюваність, продуктивність і можливість витримувати великі навантаження. У різних областях діяльності накопичено величезну кількість даних, що веде до посилення вимог до їх обробки та зберігання, зокрема до продуктивності систем управління базами даними. Дана проблема особливо актуальна для даних, що вимагають глибокого аналізу. У зв'язку з цією ситуацією з'являються нові підходи до побудови таких систем, які повинні подолати недоліки існуючих.

Мета дослідження – виявлення найбільш продуктивної СУБД або комбінації різнорідних СУБД та вдосконалення розміщення різнорідних даних великого розміру. Мета досягається за рахунок проведення за методом аналогії та методами структурного та порівняльного аналізів існуючих СУБД, їх властивостей та характеристик, проведення робіт по поліпшенню взаємодії баз даних, виведення практичних порад для коректного розміщення різних обсягів даних.

Об’єктом дослідження є різнорідні СУБД великих обсягів, як складові частини єдиної розподіленої системи. Предметом дослідження є визначення якісних та кількісних показників при розміщенні різнорідних даних великого розміру в розподілених базах даних.

Проміжні висновки і результати цієї роботи розглянуті і представлені в 2 публікаціях.

1 ОГЛЯД СТАНУ ПРОБЛЕМИ ДО ЗБЕРІГАННЯ ТА ОБРОБКИ ВЕЛИКИХ ОБСЯГІВ ДАНИХ

Розробка системи управління базами даних почалася з того, що розробники, яким довелося створювати багато додатків, поступово стали звертати увагу на те, що в різних за своєю суттю додатках їм доводиться програмувати одну і ту ж функціональність, пов'язану з накопиченням, пошуком і аналізом даних. Тенденції роботи з великими обсягами даних росли з кожним роком і прикладних задач, що вимагають обробки цих даних відповідно теж росли. Крім того, багаторазова реалізація дуже схожою функціональності (причому не завжди вдала) значно здорожувала кінцевий програмний продукт. На цьому тлі і виникла ідея розмежування функцій програми і функцій обробки даних. З'явилася ідея про створення централізованих систем, які забезпечують доступ до даних, тобто ідея про створення високоефективних і якісних універсальних систем, орієнтованих нема на потреби окремих додатків, а виключно на обробку даних. Такого роду системи можуть бути створені один раз. Для їх створення можна використовувати програмістів вищої кваліфікації, а самі системи в подальшому використовувати багаторазово при створенні різноманітних додатків. Ця ідея дозволяла істотно підвищити якість і знизити вартість подальшої розробки додатків. Було запроновано наступні ідеї принципів для систем, керуючих базами даних [1]:

- централізація управління зберіганням і доступом до даних;
- зниження вартості розробки і підвищення якості додатків за рахунок винесення загальної функціональності в СУБД;
- підтримка складних логічних структур;
- забезпечення незалежності даних і додатків;
- забезпечення цілісності даних (Integrity);
- підтримка узгодженості даних (Consistency);
- захист від несанкціонованого доступу;
- розмежування прав доступу;
- підтримка високорівневих ефективних мов запитів.

В сучасних системах управління даними значення і зміст багатьох з цих функцій істотно змінилося. Деякі з цих функцій повністю або частково перейшли до інших типів прикладних систем, важливість і наявність інших систем залежать від архітектури прикладної інформаційної системи і типу використовуваної

СУБД. Однак в цілому не можна не відзначити, що цей список дійсно визначив напрямок розвитку систем баз даних на багато років вперед. Спробуємо перерахувати основні функції сучасних СУБД і почнемо з функціональності, передбаченої цим списком.

Забезпечення незалежності даних і додатків спочатку розглядалося як найважливіший елемент систем управління. Ті принципи, які розуміються під незалежністю даних і додатків, розбиваються на три можливі групи:

- одні і ті ж дані можуть використовуватися для різних додатків;
- поява нових вимог до даних (наприклад, додавання нових полів, таблиць, бізнес логікі поведінки даних, тощо) не повинно впливати на роботу існуючих додатків;
- припустимо асинхронне впровадження нових версій програм (останній принцип особливо актуальний для крупних клієнтів).

Для підтримки відповідних вимог були, зокрема, введено мови, які декларують опис структури і бізнес логікі поведінки даних. Словники, що представляють відповідну інформацію, що зберігаються в системах управління даними, тощо. Словники даних і мови опису структури і бізнес логікі даних в тому чи іншому вигляді включені майже в усі сучасні традиційні СУБД. Більшість систем строго дотримуються принципів незалежності даних, проте слід зазначити, що останнім часом стали з'являтися СУБД, які відступають від цих, здавалося б, беззастережно корисних принципів. Порушення принципів незалежності першої зі згаданих вище груп означає, що з'являються сховища даних, вузько орієнтовані на специфіку даних і функціональність однієї програми. Це могло б здатися дуже нераціональним, якби не та обставина, що такі додатки носять ексклюзивний характер і затребувані мільйонами користувачів. Порушення принципів незалежності з другої і третьої груп очевидним чином позначається на розробці, супроводі та впровадженні програм, які працюють з такими системами.

Крім відповідності структурі описаних даних, система управління даними повинна забезпечувати виконання ряду умов. Цим умовам повинні задовольняти збережені дані (наприклад, квартира в якій живе людина повинна бути від 1 до 100; квартира, по якій визначається її нумерація повинна бути в списку будинків вулиці, тощо). Такого роду умови називають обмеженнями цілісності (Integrity), і вони також описуються за допомогою спеціальних мов опису даних. За перевірку виконання заданих обмежень відповідає СУБД. Будь-які операції, що намагається

порушити ці обмеження, відкидаються. На рисунку 1.1 зображена логіка роботи обмеження цілісності.

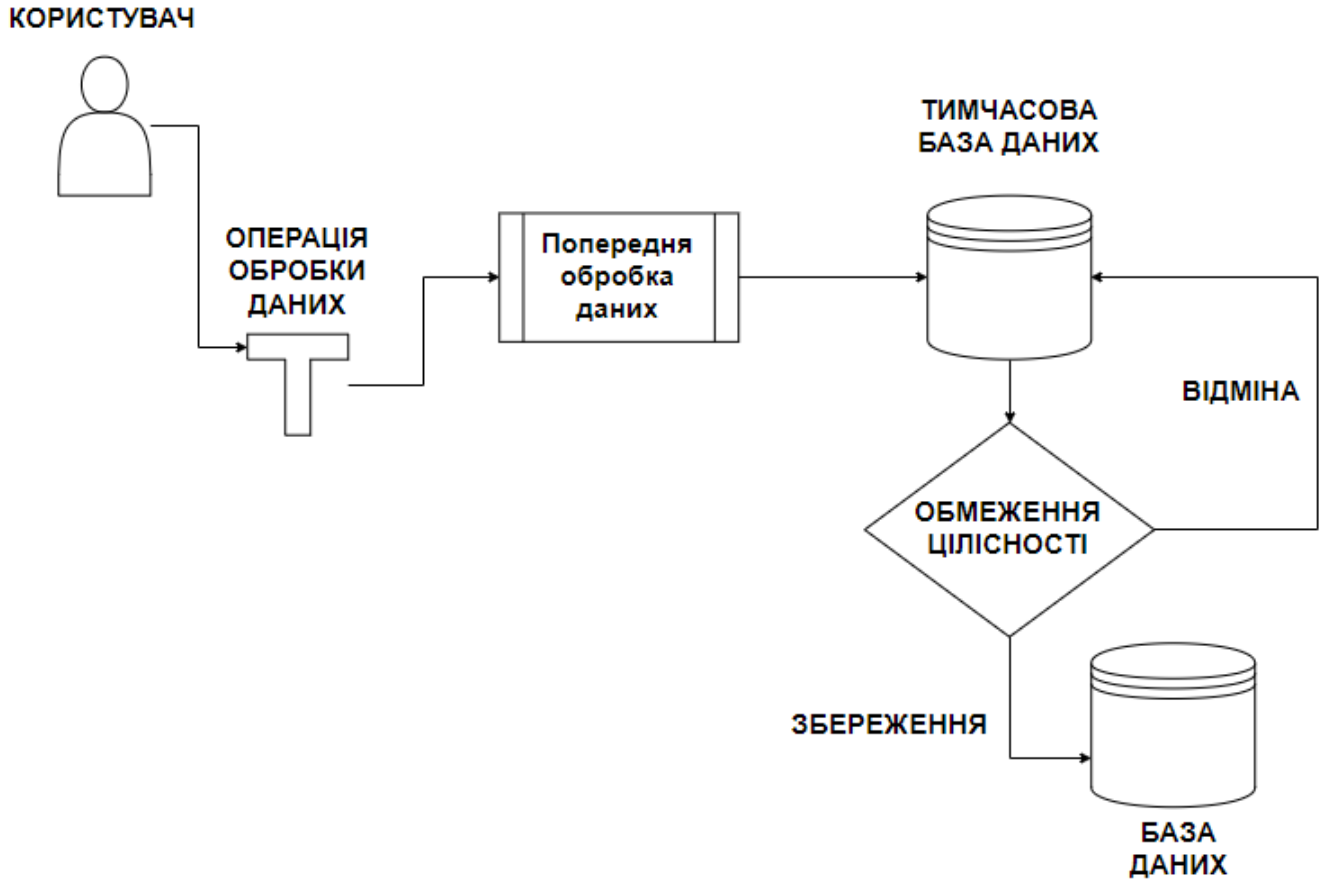

Рисунок 1.1 – Логіка роботи обмеження цілісності

Типовими обмеженнями цілісності, які на сьогоднішній день присутні практично в більшості великих СУБД, є обмеження посилальної цілісності, заборона на невизначені значення, заборона повторюваних значень і контроль можливого діапазону значень атрибутів. Приклад обмеження посилальної цілісності та заборона на невизначені значення наведена на рисунку 1.2.

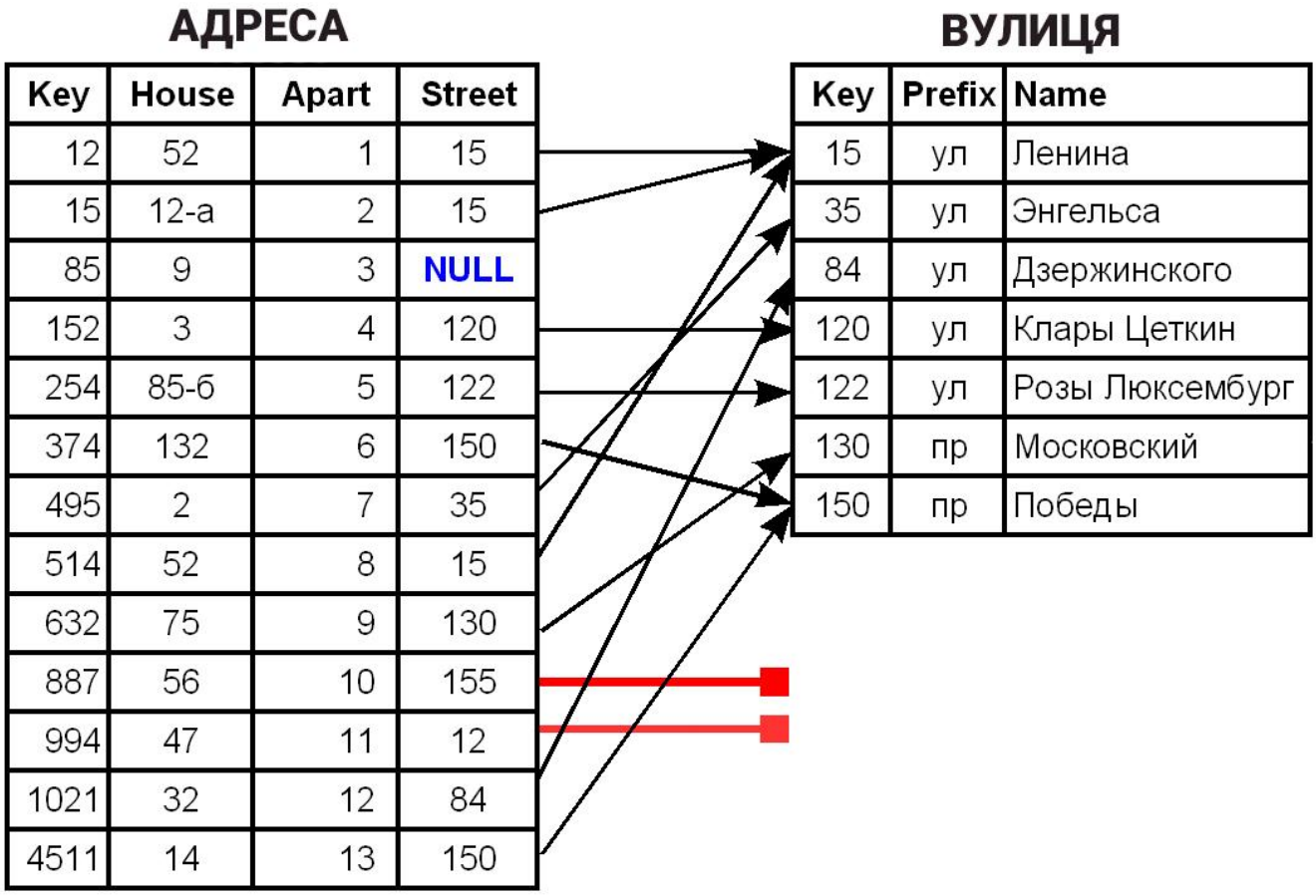

Рисунок 1.2 – Обмеження посилальної цілісності і заборона на невизначені значення

Однак існують СУБД, в яких правила цілісності або взагалі відсутні, або присутні лише частково. При цьому слід зазначити, що якщо сама природа даних така, що вимагає врахування якихось обмежень, то ці обмеження все одно доведеться враховувати на будь-якому рівні, і якщо немає можливості врахувати їх на рівні СУБД, то розробники будуть змушені їх враховувати на рівні створюваного додатка.

Підтримка узгодженості (Consistency) пов'язана зі статками даних. Деякі стану бази даних називаються узгодженими. Завдання СУБД - забезпечення механізмів, що дозволяють переводити дані з одного узгодженого стану в інше. Набір операцій, які переводять базу з одного узгодженого стану в інше, прийнято називати транзакцією. Узагальнена логіка роботи транзакції зображена на рисунку 1.3.

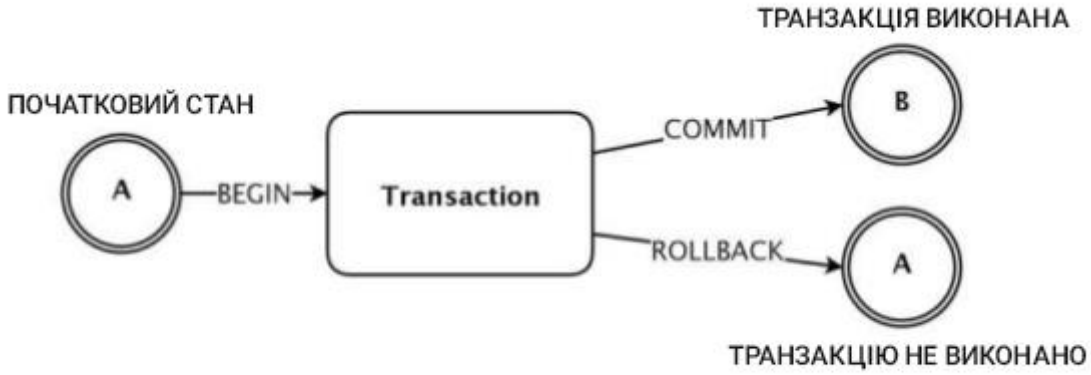

Рисунок 1.3 – Узагальнена логіка роботи транзакцій

Підтримка узгодженості даних з боку СУБД означає, що після вдалого або невдалого завершення роботи програми база даних у будь-якому випадку виявиться в узгодженому стані, тобто всі транзакції додатки або будуть виконані повністю, або не залишать жодних слідів в даних (тобто СУБД забезпечить відкидання не повністю завершених транзакцій). Ця вимога називають атомарний транзакцій (Atomicity). Після фіксації транзакції зміни в даних стають постійними. Така властивість називають тривалістю транзакцій (Durability).

Підтримка узгодженості суттєво ускладняється, якщо СУБД допускає паралельне (або конкурентну) виконання транзакцій. Для забезпечення цього завдання в традиційних СУБД були розроблені різноманітні витончені алгоритми виконання транзакцій, що створюють для користувачів ілюзію ізольованості (Isolation) роботи з даними. Про СУБД, що підтримують всі ці властивості, кажуть, що вони мають ACID властивості. Проте насправді за цими поняттями можуть ховатися зовсім інші рівні ізольованості. Розрізняють такі рівні ізольованості транзакцій (детальніше на рисунку 1.4):

- read uncommitted;
- read committed;
- read commited using row versioning;
- repeatable read;
- snapshot;
- serializable.

РІВЕНЬ ІЗОЛЯЦІЇ	БРУДНЕ ЧИТАННЯ	НЕПОВТОРНЕ ЧИТАННЯ	ФАНТОМНЕ ЧИТАННЯ
Read uncommitted	ТАК	ТАК	ТАК
Read committed	НІ	ТАК	ТАК
Read commited using row versioning	НІ	ТАК	ТАК
Repeatable read	НІ	НІ	ТАК
Snapshot	НІ	НІ	НІ
Serializable	НІ	НІ	НІ

Рисунок 1.4 – Рівні ізольованості транзакцій

Read uncommitted забезпечує найслабший рівень ізоляції, що допускає читання даних незафіксованих транзакцій - так зване «брудне» читання. Serializable забезпечує найсильніший рівень ізоляції, кожний наступний включає всі попередні. Більшість традиційних СУБД за замовчуванням використовують рівень Read commited. На цьому рівні ізоляції забезпечується захист від «брудного» читання, однак при повторному читанні одного набору даних результати можуть виявитися різними, якщо в проміжку між читаннями дані були змінені успішно зафіксованої транзакцією. Рівень ізоляції на основі управління версіями рядків (Read commited using row versioning) підтримує версії кожної змінної рядка. Додатки можуть вказати, що транзакція використовує версії рядків для перегляду даних в тому вигляді, в якому вони існували на початку транзакції або запиту, замість того, щоб захищати всі читання за допомогою блокувань. При використанні контролю версій рядків ймовірність того, що операція читання заблокує інші транзакції, значно зменшується. Рівень Repeatable read не дозволяє змінювати дані, лічені активною транзакцією, але не перешкоджає додаванню нових даних іншими транзакціями. Найсильніша рівень ізоляції (Serializable) згідно зі стандартом, повинен забезпечувати абсолютну ілюзію автономної роботи

(як ніби інших транзакцій не існує). Тільки цей рівень ізоляції забезпечує захист від, так званого, «фантомного читання». Насправді деякі промислові СУБД замість рівня Serializable використовують рівень Snapshot Isolation, який дещо слабше Serializable, але сильніше Repeatable read. Транзакція, що працює на даному рівні, бачить тільки ті зміни даних, які були зафіксовані до її запуску, тобто поводиться так, начебто отримала при запуску моментальний знімок бази даних і працює з ним. Очевидно, що підтримка рівня ізоляції Serializable найбільш складна, і саме тому за замовчуванням в більшості систем використовуються більш низькі рівні, хоча системи з рівнем ізоляції Serializable і конкурентним виконанням мільярдів транзакцій також існують.

Алгоритми, що забезпечують підтримку згаданих вище рівнів ізольованості, блискуче справляються зі своїми завданнями. Важко уявити, що колись такі завдання потрібно було вирішувати силами розробників різноманітних додатків. Однак останнім часом стали з'являтися СУБД, які намагаються себе протиставити традиційним СУБД і в якості від одного з принципів висувають відмова від рівня ізоляції Serilizable [2]. Така відмова виглядає принаймні дивним на тлі того, що лише незначна частина додатків, що використовують традиційні СУБД, дійсно використовує рівень ізоляції Serializable.

Функції захисту даних від несанкціонованого доступу і розмежування доступу є перманентною завданням практично будь-який СУБД. З цією метою були розроблені різноманітні прийоми, що дозволяють приховувати реальні дані від користувачів (наприклад, уявлення), розмежовувати права доступу (ролі, привілеї, мандатний доступ), а в останніх версіях деяких промислових СУБД і різноманітні прийоми для відображення, а іноді і навмисного спотвореного відображення даних (ORACLE 12C). Ще один перспективний напрям розвитку захисту даних - ізоляція адміністраторів серверів баз від власне даних. У більшості сьогоднішніх СУБД адміністратори серверів бази можуть бачити і змінювати всі дані в БД. Нові засоби захисту дозволяють адміністраторам виконувати всі операції по адмініструванню БД, але не дозволяють бачити і змінювати дані (ORACLE 12C). Однак слід зазначити, що, незважаючи на достаток механізмів захисту даних в рамках СУБД, у багатьох сьогодні створюваних додатках реалізована дворівнева захист даних, сенс якої зводиться до того, що на першому рівні перевіряються права кінцевих користувачів на рівні додатку, і тільки на наступному рівні - права уніфікованого користувача на рівні СУБД. Більш того, останнім часом намітилася тенденція по створенню

трирівневої системи захисту даних, яка починається з перевірки користувачів на рівні операційної системи. Винос механізмів управління користувачами з СУБД і додатків в єдину централізовану систему організації в більшості випадків дійсно спрощує управління користувачами в масштабах організації і дозволяє в значній мірі автоматизувати весь процес управління

Наявність високорівневої та ефективної мови запитів є, мабуть, найбільш важливою відмінністю СУБД. З цією метою в рамках різноманітних систем управління були реалізовані високорівневі декларативні мови для маніпуляцій над даними, з яких на особливу увагу заслуговує SQL. Елегантність і незалежність від специфіки СУБД, а також його підтримка провідними виробниками баз даних зробили мову SQL основним стандартом для обробки даних. Але однієї елегантності мови явно недостатньо для забезпечення його ефективності (швидкого і не дуже ресурсу необхідного виконання запитів). Тому практично у всіх СУБД, з'явилися так звані оптимізатори запитів (рисунок 1.5).

Рисунок 1.5 – Оптимізатор запитів

Основна функція оптимізатора запитів полягає в побудові можливих планів виконання запитів і виборі оптимального з них. Вхідні дані для оптимізатора

запитів включають запит, схему бази даних (визначення таблиць і індексів) і статистику бази даних. План виконання - це результат роботи оптимізатора запитів. План виконання визначає послідовність доступу до вихідних таблиць і те, які методи використовувати для отримання даних з кожної таблиці. Оптимізація – це процес вибору одного плану виконання з багатьох потенційно можливих планів. У кожного з можливих планів виконання є супутні витрати в розмірі вартості використаних обчислювальних ресурсів. Як правило, найкращим планом є той, на виконання якого необхідно менше часу, проте в деяких випадках критерії можуть залежати від вимог додатка. Наприклад, потрібно мінімізувати час отримання перших рядків запиту або час повного виконання запиту. Оптимізатори вибирають плани на основі явно або неявно певної функції вартості. Залежно від того, як визначена функція вартості, розрізняють два режими роботи оптимізаторів: оптимізатор, який працює на основі правил, і оптимізатор, який працює на основі статистичних показників фактично збережених даних. Сховище запитів дає уявлення про вибір плану виконання і продуктивності. Це спрощує вирішення проблем з продуктивністю, допомагаючи швидко знаходити відмінності в продуктивності, викликані змінами в плані виконання. Сховище запитів збирає дані телеметрії, такі як журнал запитів, плани, статистику виконання і статистику очікування. В даний час алгоритми оптимізації досить добре опрацьовані і реалізовані в промислових СУБД, і тільки в досить рідкісних випадках потрібно ручна настройка (за допомогою підказок оптимізатора) або повне переписування запитів з метою поліпшення їх продуктивності. Однак в останнє десятиліття стали активно з'являтися так звані NoSQL-системи, які відмовляються від багатьох принципів, закладених в основі більш класичних СУБД, а деякі з них відмовляються навіть від декларативних мов запитів [3]. Відмова від використання високорівневих мов запитів призводить до досить очевидних наслідків: розробники додатків змушені формулювати низькорівневі запити, що вимагають значно більше часу на етапі розробки, що, в кінцевому рахунку, призводить до розтягування і подорожчання всього періоду створення програми. Розробник, який опинився в такому становищі, нагадує автогонщика, якому замість швидкісного боліда запропонували самокат. По суті справи у розробників додатків є два шляхи: використовувати дорогу СУБД з декларативною мовою запитів і швидко створити додаток або використовувати недорогу (або взагалі безкоштовну) СУБД з низькорівневою мовою запитів і витратити значно більше часу (і коштів) на створення програми. У кожного з цих

шляхів є свої плюси. Перший забезпечує гарантовану якість запитів додатка при невеликих витратах на розробку, другий - зайнятість значної частини розробників. Як би це не здалося дивним, перший зі згаданих вище плюсів не для всіх компаній є досить переконливим. Сьогодні в IT-індустрії є ряд компаній, послугами яких користуються мільйони користувачів у всьому світі. Такі компанії можуть дозволити собі величезний штат висококваліфікованих розробників. Додатки, що створюються такими компаніями, часто використовують системи управління даними з низькорівневими мовами запитів. Успіх, супутній цим системам, створює ілюзію простоти створення додатків. Але насправді за цими програмами стоїть праця висококваліфікованих розробників, тестувальників, адміністраторів, аналітиків і так далі. Розробникам з невеликих компаній слід віддавати собі звіт в тому, що формула створення таких додатків носить ексклюзивний характер і не масштабується.

Важливим аспектом обробки даних виступає вид обробки, тобто які одиниці даних можуть бути оброблені за допомогою однієї операції. Розрізняють два види обробки: обробка окремих об'єктів та масова обробка. Наприклад, мова SQL з самого початку орієнтована на масову обробку даних [4]. За допомогою однієї операції можна обробити велику кількість рядків. Більш того, особливістю всіх популярних реалізацій SQL в рамках традиційних СУБД є та обставина, що масовані операції обробки виконуються значно ефективніше обробки одиничних об'єктів, тобто ефективніше обробити за допомогою однієї операції 5000 об'єктів бази даних, ніж 5000 раз виконувати обробку окремих об'єктів. Але слід зазначити, що системи, в яких мови запитів орієнтовані на обробку окремих елементів, також існують і, більш того, активно розвиваються і обговорюються.

Підтримка транзакцій і суворої узгодженості даних в умовах розподілених систем вимагає значного числа синхронних взаємодій. Ця обставина не тільки знижує час відгуку системи, а й призводить до зменшення її відмовостійкості. Виходячи з цих проблем люди задумалися над тим, що не завжди потрібна сувора узгодженість даних і власне підтримка транзакцій, таким чином з'явилося таке поняття як теорема CAP. Теорема CAP (Consistence, Availability, Partition tolerance - узгодженість, доступність, стійкість до поділу) - це твердження, яке говорить, що розподілена система не може одночасно володіти більш ніж двома з наступних трьох характеристик - це доступність (availability), узгодженість (consistency) і стійкість до розривів мережі (partition tolerance). На основі теореми CAP з'явилась класифікація нових систем (рисунок 1.6) [5], CA – виконує вимоги по

узгодженості і доступності, CP – виконує вимоги по узгодженості і стійкості до поділу, AP – виконує вимоги по доступності та стійкості до поділу.

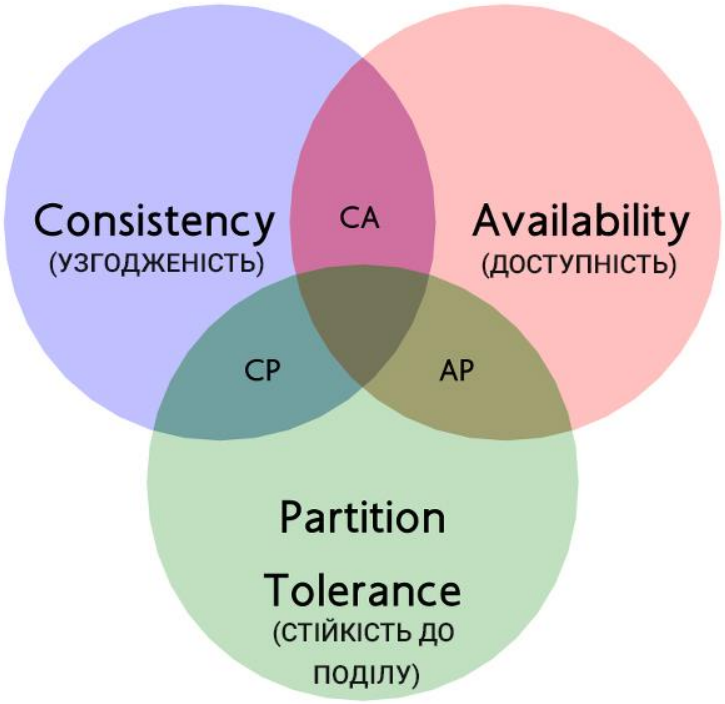

Рисунок 1.6 – Класифікація нових систем за теоремою CAP

NoSQL системи позиціонують себе як альтернативні системи, які в змозі зберігати і обробляти гігантські обсяги даних [6]. Більшість систем NoSQL створювалися з розрахунком на легке масштабування і роботу на кластерах з недорогих комп'ютерів в хмарному середовищі. Системи NoSQL, як правило, не підтримують ACID транзакцій, як це передбачено в традиційних СУБД. Їх іноді називають BASE-системами (Basically Available, Soft state, Eventually consistent) [7]. У цій назві Basically Available означає, що дані доступні завжди, коли до них відбувається звернення, навіть якщо частина з них зараз не доступна; Softstate означає, що дані можуть знаходитися в неузгоджені стані якийсь період часу, а Eventually consistent означає, що, в кінцевому рахунку, після деякого періоду часу дані в сховищі опиняться в узгодженому стані. На рисунку 1.7 наведено порівняння ACID та BASE систем.

Рисунок 1.7 – Порівняння ACID та BASE

Для багатьох NoSQL систем при обробці даних характерно використання технології MapReduce [8]. Відповідно до цієї технології, обробка даних складається з двох кроків - Map і Reduce (рисунок 1.8).

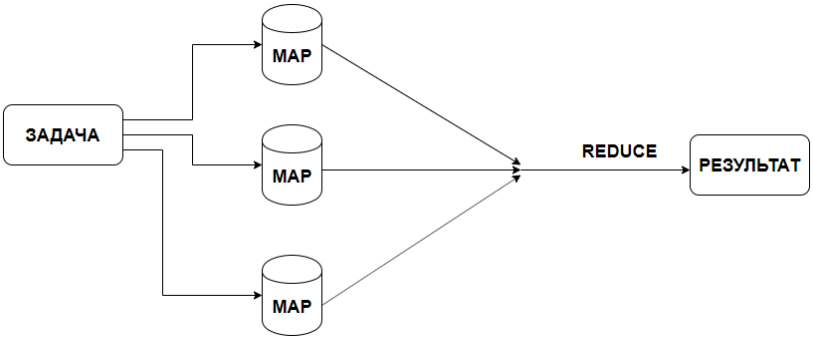

Рисунок 1.8 – Логіка роботи технології MapReduce

Отже, у підходах до збереження та обробки даних існує багато рішень та підходів для прискорення видачі результату користувачу. Різні моделі баз даних об’єднують у собі лише частину існуючих методів по забезпеченню швидкого, надійного та коректного доступу до даних, що зберігаються. Сучасні СУБД можуть комбінувати ці методи, або використовувати окремі переваги кожного з них. В залежності від типу даних, що зберігаються, та моделі бази даних, будуть використані різні методи збереження даних, які підтримуються конкретною СУБД.

2 ПОСТАНОВКА ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Предметом дослідження є визначення та формулювання якісних та кількісних показників типів даних, що зберігаються, при розміщенні різнорідної інформації великого розміру в розподілених базах даних.

Об’єктом дослідження є різниця у взаємодії, зберіганні та обробці різнорідних СУБД, реляційної та нереляційної, з великими обсягами даних, як складової частини єдиної розподіленої системи.

Метою дипломної роботи є знаходження, формулювання та визначення показників оптимального розміщення різнорідних даних великого об’єму. Отримати характеристики експлуатації різних типів даних при створенні таблиць та колекцій, визначити метрики по яким можна провести порівняльний аналіз. Порівняти швидкість роботи з даними при використанні реляційної та NoSQL бази даних, або їх комбінації, та отримати порівняльні характеристики СУБД при використанні однакових типів даних, чисел та строк. Визначити способи вдосконалення розміщення різнорідних даних великого розміру, привести на їх основі практичні рекомендації щодо зберігання та обробки даних великого обсягу. Мета досягається за рахунок проведення за методом аналогії та методами структурного та порівняльного аналізів існуючих СУБД, їх програмних властивостей та технічних характеристик, формування та проведення робіт по поліпшенню взаємодії баз даних з системою за рахунок проведення налаштувань окремих програмних компонентів СУБД, зміни програмних констант, визначення та проведення заходів оптимізації бази даних великого обсягу завдяки реплікації чи шардінгу. У результаті виконання роботи сформулювати практичні поради проектувальникам та архітекторам баз даних щодо коректного розміщення різних обсягів даних на всіх етапах життєвого циклу системи та способи підвищення продуктивності роботи СУБД.

Для досягнення поставленої мети буде обґрунтовано та використано два різних типи СУБД – MongoDB та MariaDB. На основі цих двох СУБД будуть проводитися ряд порівняльних експериментів, суть яких полягає у розміщені в них великих обсягів даних і виконання запитів користувачів до бази. Метою експериментів є виявлення різниці у часі обробки різних типів даних

різнорідними СУБД. Під методами підвищення продуктивності бази даних розуміється установка індексів на часто використовувані атрибути.

У роботі необхідно відтворити та проаналізувати різні умови використання порівнюваних СУБД, змінюючи обсяг даних та запити до баз даних, і зробити висновки щодо доцільності застосування розподілених баз даних при різних зовнішніх факторах, таких як об’єм даних, що зберігаються, локальне місце їх зберігання, на сервері чи у хмарі, і метод взаємодії бази даних і програмного забезпечення (локально, онлайн, напряму з сервером або через сторонні сервіси). Необхідно визначити та спроектувати структуру бази даних, враховуючи той фактор, що реляційні і NoSQL бази даних різні по своїй класифікації і відповідно підходи в проектуванні завжди відрізняються. Для заповнення баз даних в рамках атестаційної роботи, необхідно реалізувати генератор синтетичних даних, який буде створювати дані, наближені до реальних, використовуючи готові шаблони. Після розміщення даних у базах, визначити способи та провести роботи по поліпшенню взаємодії з даними, зокрема впровадженню масштабування різними способами. Фінальним етапом є визначення критеріїв оцінки обробки різнорідних даних різними СУБД та проведення тестування швидкості роботи баз даних. Критерієм ефективності буде час виконання запитів.

Вихідними умовами є локальний сервер з характеристиками приведеними в таблиці 2.1.

Таблиця 2.1 – Характеристики сервера

Процесор	Intel Core I7-8550U 1x8 cores
RAM	8GB DDR3 RAM
ROM	SSD 256 GB (читання: 2800 MB/s, запис: 1100 MB/s)
ОС	CentOS Linux 7.6.1810 (Core)
ПЗ	MariaDB 10.3 X64, PhpMyAdmin, Studio 3T, Navicat, MySQL Workbench, Apache, MongoDB 4.0 X64, Sublime Text 3, Mongo Shell

3 АНАЛІЗ СУЧАСНИХ МЕТОДІВ ЗБЕРІГАННЯ РІЗНОРОДНИХ ДАНИХ

Більшість даних у сучасних системах зберігається саме у базах даних. Тільки незначні відбитки інформації або конфігурація може бути збережена у пам’яті програми або приладу, який використовується користувачем. За останні тридцять років багато раз змінювалися типи як баз даних, так і сховищ даних. Але сучасними підходами прийнято вважати реляційний та NoSQL підхід до збереження даних. Слід зауважити, що різниця між виникненням цих підходів складає 25 років. Графік популярності сучасних СУБД приведено на рисунку 3.1 [9]. На ньому видно, що реляційна база даних MySQL займає лідерство. Після нього по полярності йде документно-орієнтована MongoDB та реляційна MariaDB, яка показує швидкий темп росту популярності за рахунок своїх можливостей.

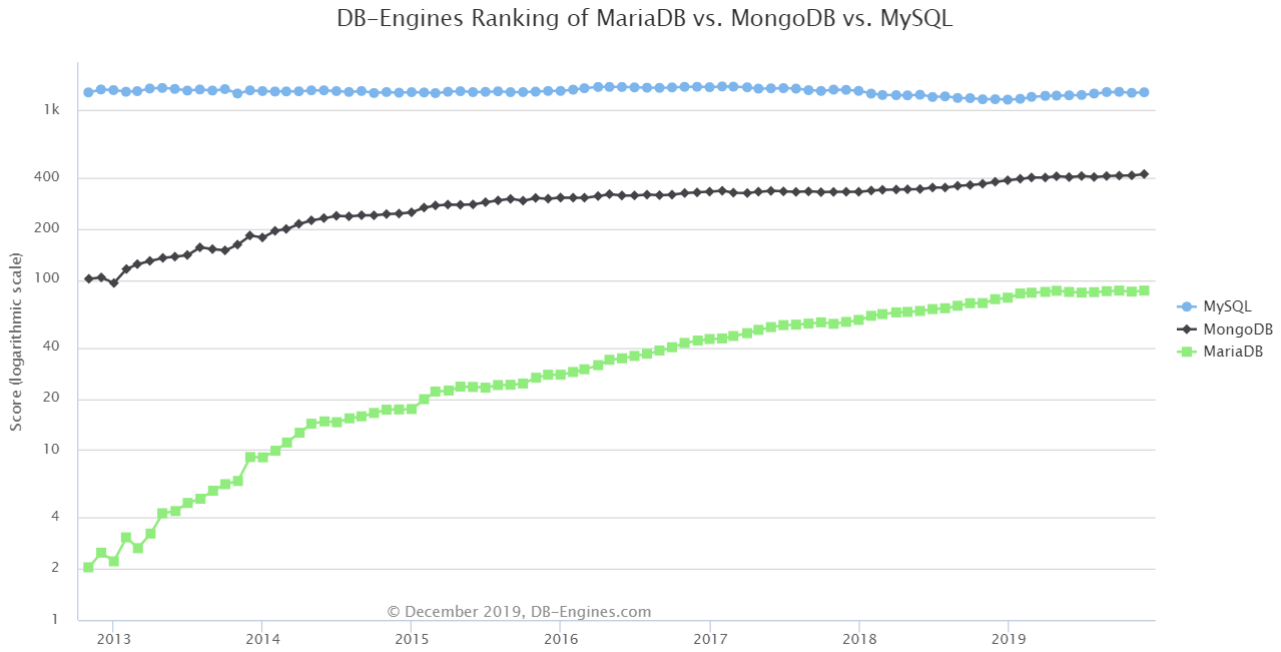

Рисунок 3.1 – Популярність використання СУБД за DB Engines

Реляційні бази даних мають на увазі збереження даних у таблицях, які пов’язані між собою кортежами даних. Кортежі поєднуються завдяки ключам: зовнішніми та основним. До реляційних баз даних застосовується як реляційна алгебра, так і зведення бази даних до нормальних форм, що значно оптимізує

збереження даних, та обчислення у СУБД. Мова виконання запитів SQL дозволяє не тільки управляти даними всередині бази, а й проводити аналітику чи вибірку навіть із сирих даних. Більше того, ця мова дуже розповсюджена і підходить для виконання запитів на більшості РСУБД. Реляційні моделі дуже прості, вони містять у собі лише базові концепти, тоді як зв'язки лише симулюються. Також у таблицях реляційних баз даних можна використовувати лише певні типи даних, що пропонуються. Мінусами реляційного типу СУБД можна вважати недостатню семантику моделювання і наявність складних типів та структур даних, які потрібно моделювати для збереження іноді базової інформації. Також у бази даних мала підтримка як самих типів даних, так і покращення таблиць для їх збереження. Часто виникають складні зв'язки між об'єктами, завдяки яким необхідно створювати додаткові таблиці з ключами («many-to-many»), що призводить до уповільнення роботи бази даних.

Реляційні моделі баз даних мають проблеми, які обумовлені їх структурою. По-перше, це використання моделі Атомарність-Послідовність-Ізоляція-Довговічність (ACID) кожним запитом, навіть коли це не потрібно [10]. По-друге, таку базу даних складно масштабувати, так як зв'язки між таблицями уповільнюють цей процес. Також недоліки помічаються у доступності та гнучкості такого виду СУБД. Традиційні методи масштабування мають на увазі розширення можливостей сервера бази даних, управління пам'ятью та кешуванням (запитів, відповідей), а також можливостей реплікації та зберігання у пам'яті запитів для їх моментального відтворення у випадку, коли це потрібно. Можливостями розширення баз даних також вважають створення додаткових масивів даних на диску та шардінг.

Модель ACID підтримує можливість безпечно та надійно виконувати запити, особливо складні запити до великих даних за умов високою завантаженості бази даних. Атомарність – дає гарантію, що всі операції в рамках однієї транзакції будуть виконані, або жодна не виконається. Послідовність гарантує, що база даних буде контролювати початок і кінець всіх транзакцій, що до неї надходять. Ізоляція дозволяє транзакції виконуватися так, начебто вона єдина на всю базу даних, тобто має доступ до всіх ресурсів і вони незмінні у час виконання транзакції. Довговічність має на увазі неможливість відмінити транзакцію, коли вона була успішно виконана у базі даних.

Окрім того, реляційні моделі даних важкі для зберігання та обробки, логування, установки блокування та буферу, або потоків. У більшості випадків,

неможливо опрацювати прості операції з даними різних типів, так як вони зберігаються у різних таблицях або базах даних [11]. У розподілених базах даних неможливо зробити об’єднаний запит до декількох баз даних, а при операції оновлення, вставки чи видалення даних – зміни необхідно послідовно заносити до всіх баз даних, які існують у системі. Можуть статися випадки, коли одна з баз даних перестає працювати, що призводить до зростання навантаження у системі та іноді до втрати даних. Усі переваги реляційних моделей у зв’язках між таблицями, але така структура має свої складності, у проектуванні, розробці та експлуатації. Особливо важко прискорити обробку даних, що зберігаються, при роботі з великими даними, так як час виконання операцій значно зростає.

Через недоліки реляційної моделі виникають інші, нетрадиційні бази даних. Кожна з них повинна була зберігати дані для певного кола задач, так XML бази даних зберігали XML-файли, бази даних у вигляді графів – невеликі обсяги даних. Але найбільш популярними та багатофункціональними стали документні бази даних, сховища типу «ключ-значення» та колоночні бази даних. Їх перевагами були відсутність схеми та структури, вони могли бути розповсюджені на території та легко масштабувалися за рахунок типу даних, який зберігали [12]. Легке розширення додаванням машин до серверу бази даних дозволяло при проектуванні системи забувати про проблеми зі зберіганням даних. Кожна колекція зберігала певну частину даних, яка логічно розміщувалася у системі. Швидкість обробки даних можна збільшувати на ходу, додаючи нові машини до кластеру, або збільшуючи потужність єдиного серверу бази даних (горизонтальне або вертикальне масштабування підтримується однаково). Більше того, нереляційні бази даних дозволяють швидко обробляти запити читання та оновлення, які розповсюджені найбільше серед усіх інших типів.

Нереляційна база даних використовує принцип BASE (Basic Availability, Soft-state, Eventual consistency), який відповідає більшості вимог до сучасних систем. Базова доступність припускає, що база даних повинна працювати переважну більшість часу. На практиці, доступність серверу повинна бути не менше, ніж 99.9999%, що дозволяє йому називатися надійним. Під м’яким станом мається на увазі, що сховища даних не повинні бути послідовні для запису, а обробка даних може йти паралельно. Подійна послідовність також може бути синхронізована. Також до принципу Base входить правило, що комп’ютер не може одночасно надати легку можливість для розширення, доступність та узгодженість. Завжди з цих трьох властивостей можна реалізувати лише дві у

рамках однієї системи. Доступність дозволяє будь-якому користувачу системи вносити зміни до бази даних одночасно, та будь-які інші користувачі будуть бачити ці зміни. Узгодженість має на увазі, що всі користувачі бази даних завжди бачать одні й ті самі дані, за будь-яких обставин. А розширення дозволяє системі працювати розподілено на декількох машинах, та продовжити працювати у випадку, якщо одна з них стане недоступна. Ці принципи та взаємодія користувача с сервером бази даних зображено на рисунку 3.2.

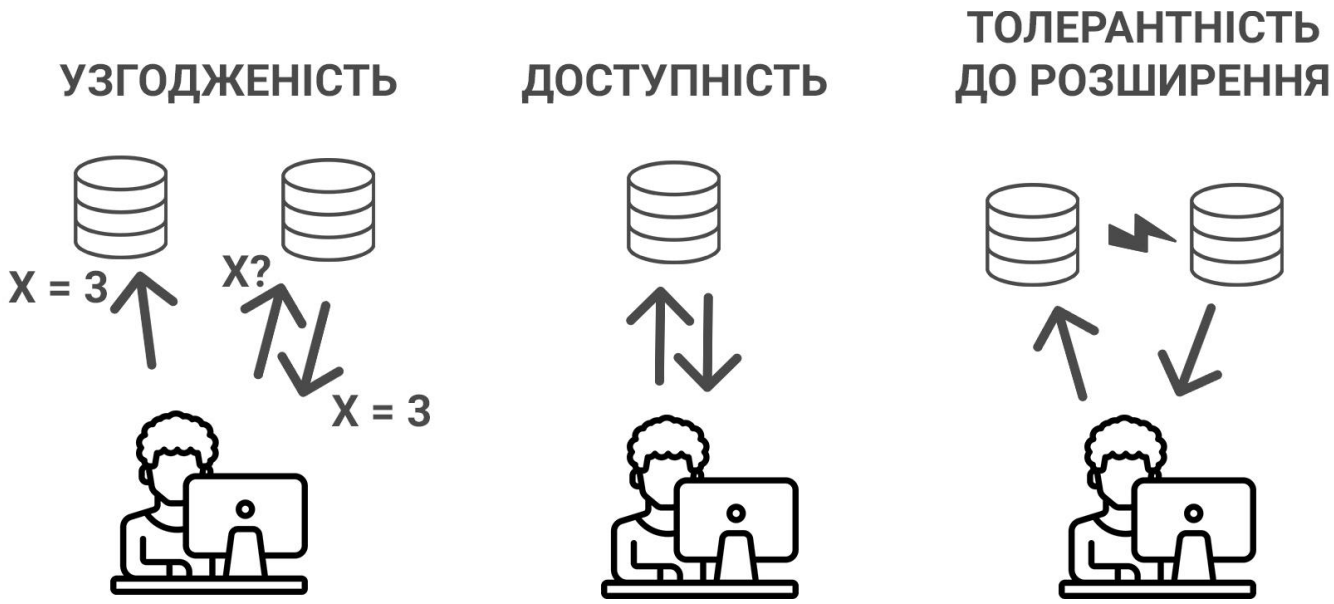

Рисунок 3.2 – Принципи побудови розподіленої системи

Тобто, за теоремою Брьюера [13], неможна одночасно забезпечити три з наведених можливостей. ЗА цією теоремою, усі розподілені бази даних можуть поділятися на три основних класи – це:

– CA – розподілена система, з узгодженістю та доступністю, але нехтується толерантність до розширення, що надто властиво реляційним базам даних, так як вони дотримуються принципу ACID;
– AP – розподілна система без гарантії узгодженості даних, що на момент створення теореми було від сумнівом, але більшість NoSQL баз даних побудовано саме на цьому принципі;
– CP – це система, яка не гарантує доступність, але надто точна в узгодженості та розподіленні. Наприклад, банкомати працюють по такій схемі, так як узгодженість важливіша за доступність.

Завдяки цій теорії проектуються навіть сучасні системи, так як підтримувати всі три принципи виявляється занадто дорого. Тому розглянемо їх основні переваги, щодо створення розподіленої системи бази даних.

Принципи ACID дозволяють побудувати стійку, узгоджену базу даних, яка буде виконувати ізольовані транзакції з фокусом на операцію COMMIT, яка буде центральною операцією в будь-якій РСУБД. Дозволяє послідовно виконувати операції з послідовною зміною даних у базі. Хоча база даних і вважається гарно-доступною, треба розуміти, що більшість випадків непрацездатності викликано сервером, тому говорити про надійність постійного підключення варто лише Щодо локальної бази даних. Але ускладнюється цей принцип труднішим створенням і експлуатацією, високими затратами на роботу та зв’язки між даними.

BASE дозволяє поставити доступність на перше місце та навіть не у розподіленій системі можливо звернутися до різних колекцій та отримати дані. База даних, побудована за цим принципом, легко розширюється, швидка та легка в обслуговуванні, не навантажена зв’язками та складними запитами. Нереляційні бази даних мають декілька категорій, які використовуються для вирішення різних задач.

Сховища типу «ключ-значення» використовується як швидко-доступний контейнер з простими даними, які доступні по певному ключу. Не потребує схем та структур, а також має простий інтерфейс взаємодії та короткий час відповіді на запити. Дані зберігаються зазвичай у строковому вигляді. Може використовуватися як сховище тимчасових корисних даних користувача, або для зберігання статичної інформації, як на сервері, так і в пам’яті. Сховище типу «ключ-значення» може бути використано як додаткова база даних у системі, або як швидко доступне сховище на стороні користувача. У сучасних системах може бути використано у безсерверній (serverless) системі, як один з сервісів, що підключаються [14]. Підтримуються більшістю сучасних мов програмування, та мають можливість працювати асинхронно у пам’яті.

Другим розповсюдженим типом сховища є документні бази даних. Вони найчастіше використовуються у web-додатках, так як мають високу стійкість до неповноти даних. Для підключення документ них баз не потребується значної кваліфікації, проектування та тестування. Завдяки колекціям (документам), у яких зберігаються дані, навіть без налаштувань СУБД будуть швидко виконуватися операції вставки, оновлення та читання даних. У документно-орієнтованих базах

дані зберігаються у форматі бінарного JSON файлу. База даних, наприклад MongoDB, представляє документи JSON у двійковому кодованому форматі під назвою BSON [15]. BSON розширює модель JSON, щоб забезпечити додаткові типи даних, упорядковані поля та підтримувати ефективність для кодування та декодування на різних мовах. На рисунку 3.3 представлено різницю між JSON та BSON.

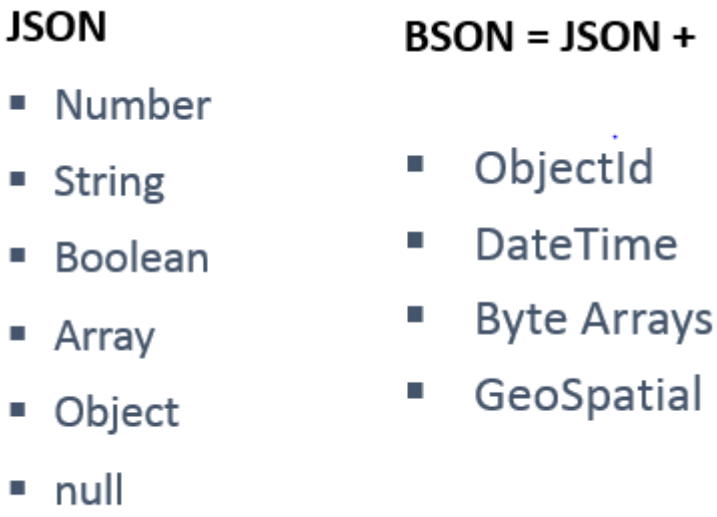

Рисунок 3.3 – Різниця між JSON та BSON

Впровадження та робота з BSON форматом легка для будь-якої системи. Як і JSON, реалізація BSON підтримує вбудовування об’єктів та масивів у інші об’єкти та масиви, таким чином база даних навіть може "дістатися до" об’єктів BSON для створення індексів та порівняння об’єктів із виразами запитів як на рівні сховища даних, так і вкладених клавішах BSON. Це означає, що документна база даних надає користувачам простоту використання та гнучкість документів JSON разом зі швидкістю обробки двійкового формату. JSON легкий і надійний формат для передачі даних по мережі, а також у перетворені даних, які відправляються із СУБД, до об’єкту, який буде оброблятися системою. Зазвичай, більшість мов програмування використовують об’єкти даних, схожі на JSON, тому передача даних між базою та системою займає мінімально можливий час.

Сховища даних типу графів дозволяють створити стійку структуру з великою кількістю зв’язків, яка використовує алгоритми графів. Таким чином, для отримання відповіді від такого сховища, потрібно, щоб запит пройшов увесь граф

у пошуках правильної відповіді, що сильно уповільнює взаємодію з такою базою даних. Більше того, при масштабуванні або створені кластеру цей недолік робить майже неможливим якісне та швидке розширення, що є основною вимогою до сучасних баз даних, так як кількість неструктурованої інформації зростає. Тому це рішення використовується для аналітики, коли потрібно зробити аналіз ситуації та динаміки зміни даних, а також виконати певні дії в залежності від зміни статистичних даних. Як правило, використовуються як другорядні бази даних, які використовуються окремо для виконання певної функції системи або програмного додатка.

З ростом популярності web-додатків та розширенням Інтернету, деякі ресурси в мережі почали володіти неабиякою кількістю даних, які потрібно передавати по мережі. Тому були створені окремі XML бази даних, які найбільше підходять для видавництв, а також інших систем, які зберігають велику кількість подібних між собою даних Такі сховища мають швидкий пошук, валідацію формату даних, що зберігаються та можуть швидко читати навіть велику кількість даних. Але окрім читання, інші CRUD операції виконуються повільно [16]. Так, швидше буде перезаписати увесь XML, аніж оновлювати деякі поля. Завдяки таким сховищам даних, швидко робити вибірки даних за певними критеріями та передавати їх на обробку до системи, так як XML – це один з найпоширеніших форматів даних для передачі по мережі, який визнаний як безпечний, легкий для читання та розуміння та надійній у передачі [17]. Зазвичай цей тип сховищ використовується при розробці web-додатку для зберігання та читання статичної інформації.

Таким чином, NoSQL бази даних мають певні переваги перед реляційною моделлю. Найголовнішим є легкість масштабування будь-якого сховища даних, у порівнянні зі складною структурою зв’язків, які присутні між таблицями. Завдяки можливості зберігання різнорідних даних в рамках одного документа, виключається необхідність використовувати операції об’єдняня типу JOIN або UNION. Дані, які приходять до бази, не повинні бути валідовані відносно схеми даних, яка присутня у сховищі. Це підвищує відказостійкість всієї СУБД. Завдяки легкості у проектуванні та експлуатації, а також схожості об’єктів мов програмування та даних, що зберігаються, NoSQL бази даних є найпростішими для створення нових програмних додатків. Також цей тип баз даних є більш стійким до помилок, має невелику затримку обробки мережевих повідомлень та швидко обробляє дані, що зберігаються (окрім операції видалення). Кешування

інтегроване у такі бази даних, та його можна навіть не налаштовувати, тому більшість вважає, що документні бази даних працюють швидше реляційних за однакових умов, але насправді у NoSQL базах лише більше налаштувань за замовчуванням.

У документних баз даних також є недоліки. Це обробка кожного ключа у сховищі, непрямі або слабкі зв'язки між ключами, читання повної строки інформації по ключу, навіть коли потрібно запитати лише деяку інформацію, не настільки надійна система користувачів та аутентифікації, як у реляційних базах даних. Так як дані зберігаються в шарі сховища, вони не можуть бути ефективно використані для аналітики, створення звітів та динаміки змін даних, агрегації або вилучання певних значень за складним набором параметрів.

Отже, основними типами даних, які порівнюються, є реляційна модель та JSON документ. Так як документи гнучкі, прості у використанні та зберіганні, а також розповсюджені в мовах програмування, це робить їх простішими у використанні, ніж реляційні таблиці [18]. Хоча таблиці в свою чергу надійні, підтримують транзакції та цілісність даних. Різниця в тому, що структуровані дані описуються метаданими таблиць, які зберігаються окремо. Напівструктуровані дані, такі як NoSQL документи, описують самі себе, так як добре читаються людиною без будь-якої додаткової інформації. Сучасна модель даних потребує обидва підходи.

Модель даних у сучасних системах часто складається зі структурованих та напівструктурованих даних, так як кількість різнорідної інформації, яку можна зберігати і обробляти зростає, насамперед завдяки технологічному прогресу. Так, дані з пристрою користувача можуть передаватися навіть без його прямої участі, а потім використовуватися у аналізі його поведінки чи вподобань. Більшість неструктурованих даних несе статичний характер та не уточнюється, так як кількість параметрів лише буде навантажувати базу даних. Часто, неструктуровані дані зберігають користувацькі налаштування у програмних додатках, деталі місць на мапі, або розгорнутий опис об'єктів. Зазвичай, така інформація надається для ознайомлення, або інтерактивно завантажується до системи для початку роботи програми. Таким чином, напряму дані не фільтруються і не сортуються при видачі, тому легше зберігати їх напів-структурованими. Приклад зберігання обох типів даних в рамках однієї бази даних показано на рисунку 3.4.

Структуровані дані (за схемою)			Напівструктуровані дані (за самими даними)
name	latitude	longitude	attr
Lou Malnati's	42.0021628	-87.7255662	{ "foodType": "Pizza", "menu": "loumalnatis.com/our-menu" }

Рисунок 3.4 – Одночасне зберігання структурованих та напівструктурованих даних

На рисунку 3.4 також показано, що стовпці описують дані, які зберігаються як структуровані, у той час як «attr» не дає нам будь-якої корисної інформації. Але прочитавши JSON об'єкт, можна легко зрозуміти інформацію, яку він зберігає.

Таким чином, реляційна база даних може створювати гібридні сховища, які зберігають різні типи даних. Мова SQL дозволяє обробляти такі дані та вибирати окремі поля з JSON об'єкта напряму до відповіді на запит. Для цього використовується команда JSON_VALUE(attr, '$.details.foodType'), яка у даному прикладі поверне тип їжі, тобто «Pizza» для сховища даних, яке представлено на рисунку 3.4. Так як JSON може неповністю зберігати дані всіх полів, а деякі залишати пустими, тому й запити до них будуть повертати NULL. Гібридні моделі також дозволяють використовувати індекси, змінювати JSON структуру на льоту при виконанні запитів, вставляти масиви даних та працювати з його елементами, а також навіть створювати нові документи із структурованих даних.

Якщо необхідно зменшити час перетворення об'єкта даних до того, який міг би оброблятися напряму мовою програмування, існує можливість створювати JSON документи у запитах до реляційної бази даних. Використовуючи команду JSON_OBJECT та комбінуючи її з іншими, можна створювати будь-які об'єкти, навіть використовуючи інші об'єкти. Більше того, такі документи будуть надійними та перевірятися при виконанні SQL операцій [19]. Тобто при операції оновлення даних кожне значення може проходити валідацію, якщо структурувати дані перед записом до бази даних. Ця функція може бути важлива при збережені важливих конфігураційних даних, або налаштувань, які потім швидко зчитуються для прямого використання у системі.

Таким чином, дані різного типу можна зберігати та продуктивно використовувати як в реляційній базі даних, так і NoSQL, але різниця у підходах

полягає у пріоритетах проектувальника бази даних. Якщо дані повинні бути швидко доступні та узгоджені між собою як всередині бази, так і декількох баз, тоді необхідно використовувати реляційну модуль бази даних. Так як вона може бути гібридною та зберігати JSON документи з можливістю застосування до них SQL запитів, це не відрізняє напівструктуровані дані від звичайних, які зберігаються за структурами таблиць. Реляційні бази даних надійні, а транзакції завжди виконуються, як очікується, завдяки принципам ACID. Їх використання необхідне у великих системах, яким необхідна надійна та узгоджена база даних. Використання тригерів та операцій додає більше можливостей управління даними.

Нереляційні бази даних використовуються для швидкості роботи з даними, особливо різнорідними, так як зберігають їх у загальновизнаному форматі та легко передають по мережі. Підтримується більшістю мов програмування без додаткових програмних модулів. Легко масштабується та може не містити значення до деяких полів. Використовуючи принцип BASE системи можуть використовувати базу даних без проектування, а так, як це вимагається зовнішніми умовами. Використання бази даних може знайти місце в будь-якій системі, як основна чи другорядна модель зберігання даних.

Дані можуть зберігатися як на сервері, так і у локальному сховищі на стороні клієнта, у пам’яті або у хмарі. Так, динамічні або конфігураційні дані можуть використовувати сховища типу «ключ-значення» та зберігатися на стороні користувача, а великі дані слід зберігати у хмаровому сховищі або на сервері. Не рекомендується зберігати важливі дані у хмарі, наприклад власні дані користувачів, або банківські акаунти, треба завжди зберігати на власному сервері, що підтверджено на законодавчому рівні [20]. Хмарові сховища краще підходять до розміщення основної бази даних або кластеру розподілених баз даних, так як мають високий рівень доступності, стійкості до відказів та автоматичну масштабованість у випадку, коли це необхідно. У випадку, коли база даних використовується у якості сервісу (DBaaS), можливо використати додаткові сервіси валідації та збереження даних по категоріям, сортування або фільтрації даних, розподілу потоків даних та контроль за виконанням запитів. При розробці serverless додатка, краще використовувати колонкові бази даних, так як збереження даних не відрізняється від реляційних баз даних, але пошук по базі відбувається, що дозволяє використати менше процесорних хвилин.

4 ОСОБЛИВОСТІ ПРОЕКТУВАННЯ СЕРВЕРНИХ ТА ХМАРОВИХ БАЗ ДАНИХ

Бази даних залишаються невід'ємною частиною кожної комп'ютерної системи. З переходом значної кількості програмних додатків у он-лайн, деякі архітектурні особливості баз даних втратили свою актуальність. Автоматизація процесу створення програмного продукту набула великих змін з появою хмарових сервісів та серверів, а також популяризації мікросервісної архітектури програмних додатків. СУБД є основним елементом будь-якої програми, вони зберігають та обробляють усі важливі і другорядні дані, необхідні для функціонування програм та підтримки користувацького інтерфейсу. Завжди є хоча б одна база даних, або в деяких випадках безліч баз даних, особливо при збере гані великих даних, які працюють у фоновому режимі, синхронно, щоб підтримувати додатки функціональними. Бази даних можна вважати сховищами інформації, де вона пов'язана між собою, сортується та зберігається структурно або напівструктурно, що робить її легкою для пошуку та доступною у використанні.

Проектування структури баз даних є важливим кроком розробки програмного забезпечення, і важливо, щоб архітектор на першому етапі заклав основні властивості: доступність, надійність, швидкість роботи з даними. Структура бази даних має важливе значення, оскільки це важливо для створення програмних додатків, які матимуть можливість масштабування та здатність працювати за умов великих навантажень. Для роботи над структурою баз даних важливо спочатку вибрати програмне забезпечення і тип бази даних [21]. Основними типами є реляційні або нереляційні бази даних, які розглядалися раніше. Також важливою рисою при проектуванні є розміщення бази даних, яка може бути розгорнута на сервері або у хмарі. Різниця полягає у використанні ресурсів при роботі з базою даних.

Існує кілька типів баз даних, а вибір правильного типу є важливою частиною проектування бази даних. Можна класифікувати типи баз даних двома способами. По-перше, на основі мови запитів, яку використовує база даних для визначення та маніпулювання даними. Бази даних, які використовують SQL, є найбільш популярними для структурованих даних. Тоді як бази даних NoSQL

краще для машинного навчання, веб-аналітики та звітів, IoT (Інтернету речей) через такі переваги, як краща масштабованість, гнучкість та швидкість. Інший спосіб класифікації баз даних базується на моделях даних. Існує 4 типи баз даних: реляційна база даних, ієрархічна база даних, мережева база даних та об'єктно-орієнтована база даних. Вивчення різних типів баз даних та вибір типу, який найкраще відповідає потребам програми, що розробляється, це важливий попередній крок для проектування структури бази даних.

Важливу роль при проектуванні також має управління майбутньою базою даних. Воно охоплює будь-які дії, вжиті з метою організації, підтримки та аналізу процесів даних. Управління базами даних допомагає організаціям контролювати дані та надавати їм значення на будь-якій стадії життєвого циклу даних. Сьогодні системи управління базами даних охоплюють багато завдань і комплімент загальній інфраструктурі даних організації. Організації можуть вибрати вже існуючу систему управління базами даних під свої потреби та вдосконалювати її або створити індивідуальне рішення відповідно до унікальних вимог. Завдання управління базою даних часто охоплюють деякі або всі наступні цілі:

– продуктивність, яка дозволяє контролювати потоки даних;

– оптимізація зберігання завдяки якій можна планувати та проводити моніторинг і аналіз сховщиа для зберігання даних;

– ефективність, що сприяє відстежуванню даних на всіх етапах їх обробки від запиту до запису, та оптимізувати цій процес;

– безпека, що застосовується як засіб захисту даних, що використовує шифрування та токенізацію, як основні інструменти;

– конфіденційність та захист конфіденційних даних та забезпечення дотримання правил роботи з даними;

– аналіз, що дозволяє провести інтеграцію з інструментами великих даних для створення звітів та бізнес-аналітики.

Щоб створити продуктивну базу даних, яка буде використовуватися та масштабуватися, важливо пам’ятати про перспективність кожної частини системи та користувачів, що з нею взаємодіють. Зберіть дані від них перед тим, як створити базу даних, щоб знати, що вони очікують від бази даних та наскільки кваліфіковані вони в роботі з нею. Це дасть уявлення про те, наскільки складною повинна бути структура бази даних, що створюється, та чи доведеться навчати користувачів щодо особливостей взаємодії з нею [22]. У більшості випадків, сучасна система використовує сервісну або мікросервісну архітектуру, тому

кожна окрема частина може звертатися до бази даних. При використанні серверної бази даних, зробити канал зв'язку легче, так як окремий сервер має власні характеристики та функціонує незалежно. Схема звернення користувача до серверу бази даних показана на рисунку 4.1. Зв'язок між базою даних, як і кешем Memcache, та backend сервером є двостороннім. Кешування бази даних або запитів може зберігатися окремо, або всередині серверу бази даних, в залежності від об'єму даних, що кешуються. Клієнт звертається до спеціального сервісу розподілу навантаження у системі, який переміщує його до користувацького інтерфейсу системи. Завдяки функціоналу, інтерфейс пов'язується з сервісами для виконання програмних інструкцій, які вже використовують або кеш, або пряме підключення до серверу бази даних.

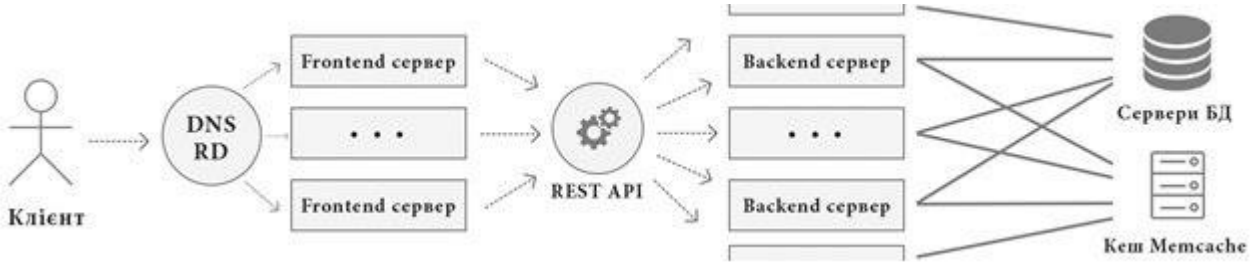

Рисунок 4.1 – Приклад взаємодії з серверною базою даних

При розробці серверних баз даних, реляційних та нереляційних, гарними практиками вважаються наступні ключові кроки. Визначаючи елементи даних та маркування таблиць і стовпців, важливо дотримуватися послідовного методу. Це допоможе краще зрозуміти дані, що зберігаються, для всіх, хто працює з базою даних. Один з найкращих методів назви таблиць та стовпців – дати їм просту та лаконічну назву, яка описує ті дані, які вони містять. Наприклад, позначити стовпчик, що містить назву ресторану як "НазваРесторану". Також слід уникати використання множинних імен, скорочень, і не слід використовувати пробіли. Якщо дотримуватися цих правил, майбутнім користувачам буде легше використовувати базу даних. Нормалізація бази даних грає значну роль при створенні структури і означає процес організації всієї інформації в базі даних, щоб уникнути тиражування та надмірності даних [23]. Простіше кажучи, нормалізація означає розподілення даних у декілька менших за обсягом, але пов'язаних таблиць, замість зберігання різнорідних даних в одній великій таблиці. Це стандартна у багатьох випадках практика проектування баз даних для

нормалізації даних, яка підвищує простоту використання бази, зменшує ймовірність дуплікації даних та приводить до однозначності інформації, що зберігається. Однак треба переконуватися, що у процесі нормалізації залишається первинна логічна структура бази даних, оскільки дані можуть бути поширені в занадто багато окремих таблиць.

Документація є важливою для побудови кращої структури бази даних будь-якого типу, оскільки вона відстежує всі дрібні деталі та зв'язки. Структура бази даних повинна бути задокументованою інструкціями, діаграмами взаємодії (Entity Relationship або ER), збереженими процедурами та всією іншою відповідною інформацією. Документація повинна надавати достатню кількість інформації програмістам та кінцевим користувачам, щоб вони могли зрозуміти, як можна користуватися базою даних. Одним з найважливіших правил є безпека даних, що зберігаються в базі. Конфіденційність даних є важливою проблемою сучасного світу інформаційних технологій. Для максимальної безпеки слід зашифрувати паролі, обмежити доступ до бази даних за допомогою автентифікації та використовувати окремі канали для зв'язку з базою даних. Це забезпечить збереження даних від нападу третіх осіб та порушення конфіденційності.

Правильна структура бази даних полягає у можливості швидкої та простої масштабованості, а це означає, що база може витримати велику завантаженість і тримати роботу програми, навіть якщо кількість одночасних з'єднань збільшується з часом. Для підприємств, які очікують зміни навантаження, важливо пам'ятати про цей принцип під час проектування бази даних. Не використання збережених процедур є однією з найпоширеніших помилок у розробці структури бази даних. Збережені процедури надають можливість використовувати заздалегідь підготовлений код, який стане зручним, коли доведеться обробляти дані. Наприклад, якщо надходить SQL запит, який часто виконується за логікою системи, то для спрощення завдання для бази даних можна закодувати такий запит у збережену процедуру замість того, щоб виконувати його знову і знову. Коли у базі є збережена процедура, можна лише виконати її для запуску SQL запиту за один крок. Кодування декількох збережених процедур заздалегідь може показатися складним завданням, та декілька процедур можуть не використовуватися взагалі, але якщо витратити час на проектування архітектури та взаємодії бази з користувачами, і документувати їх, то можна полегшити використання бази даних для кінцевих користувачів та спростити логіку взаємодії бази і системи.

При створені реляційної бази даних, часто буває, що колонка з ідентифікатором є єдиним ключем. Перша нормальна форма передбачає, що всі рядки таблиці повинні бути однозначно ідентифікованими [24]. Отже, кожна таблиця повинна мати первинний ключ. SQL Server дозволяє визначити числовий стовпець як стовпець ID, а потім автоматично генерує унікальне значення для кожного рядка. Ці типи значень, використовувані як ключі, називають сурогатними ключами. Слово сурогат означає «те, що замінює», і в цьому випадку сурогатний ключ повинен бути резервним для природного, оригінального ключа. Проблема полягає в тому, що занадто багато проектувальників використовують сурогатний стовпчик ключових даних як єдиний стовпець ключових даних в даній таблиці. Сурогатні ключові значення не мають фактичного значення в реальному світі, вони існують лише для однозначної ідентифікації кожного рядка. Тому, як правило, кожна таблиця бази даних повинна мати природний ключ, який має значення для користувача, і може однозначно ідентифікувати кожен рядок таблиці. У деякому випадку, коли не можна знайти або вивести природний ключ, тоді використовується сурогатний ключ.

Основні правила перевірки на NULL, довжину рядка, присвоєння сторонніх ключів, тощо, повинні бути визначені в базі даних. Існує багато різних способів імпорту даних на сервери. Якщо базові правила визначені в самій базі даних, це гарантує, що вони завжди будуть правильними та не пошкодяться, і користувач зможете відправляти запити, не турбуючись, чи дотримуються базові правила коректності типів даних. Принаймні, можна бути впевненим, що ці дані відповідають тим самим основним правилам, які закладені в архітектурі системи.

Хмарна база даних – це сукупність даних, структурованих або неструктурованих, які знаходяться на приватній, державній або гібридній інфраструктурі хмарних обчислень. З точки зору структури та потоків даних, хмарна база відрізняється від серверної лише обробкою запитів та деякими архітектурними особливостями. Критична різниця полягає в тому, де знаходиться база даних. Якщо серверна база даних підключена до місцевих користувачів через внутрішню мережу, то хмарна база даних знаходиться на серверах і сховищах, обладнаних хмарою або базою даних, як постачальник послуг (DBaaS). Наприклад, програмному додатку, який звертається до бази даних SQL, що знаходиться на сервері або у хмарі, необхідно виконати одні й ті самі дії, в той час як база даних може по-різному обробити запит. Поведінка бази даних має бути однаковою, будь то доступ до них через прямі запити, такі як оператори SQL, або

через виклики API. Однак можливо виявити невеликі відмінності у часі відповіді, який залежить від багатьох факторів, такі як затримка у мережі, розташування серверу та швидкість мережі.

У традиційній хмарній моделі база даних працює на багаторівневій інфраструктурі з окремими типами доступу та функціоналом. Завдання контролю та управління базами даних покладаються на працівників серверного центру, який зобов'язується супроводжувати фізичні машини. Модель використання бази даних, як сервісу (DBaaS), надає можливість використовувати вже існуючого постачальника сервісів для створення повноцінної системи. У класичній компоновці DBaaS провайдер підтримує фізичну інфраструктуру та базу даних, залишаючи користувачу лише впливати на дані та порядок роботи з базою даних. Альтернативно, архітектор може створити керований окремий сервер бази даних, в якій провайдер надає обслуговування та можливість управління базами даних.

У порівнянні з роботою з серверною базою даних на фізичному сервері та архітектурою зберігання даних, хмарна база даних надає деякі переваги. По-перше, усунення фізичної інфраструктури. У середовищі хмарних баз даних постачальник хмарних обчислень серверів, сховищ та іншої інфраструктури відповідає за обслуговування та підтримання високої доступності. Організація, яка володіє базою даних та керує нею, несе відповідальність лише за підтримку та підтримку програмного забезпечення бази даних та даних, які вона зберігає. У середовищі DBaaS постачальник послуг відповідає за керування та експлуатацію програмного забезпечення бази даних, залишаючи користувачів DBaaS відповідальним лише за власні дані [25]. По-друге, це значна економія витрат завдяки усуненню фізичної інфраструктури, що забезпечується окремою командою спеціалістів. Використовуючи сервіси, основними перевагами СУБД вважається миттєва масштабованість, гарантії ефективності, підтримку відмови, зниження цін та виключення необхідності у технічному досвіді для керування базою даних.

При створенні бази даних у хмарі декілька змінюється стратегія її використання та управління. Використовується інший алгоритм доступу до даних, який може бети змінений проектувальником на його розсуд. Як правило, додаються додаткові сервіси обробки та валідації даних, управління потоком даних та поєднання з необхідними сервісами. Так як сама СУБД також використовується як сервіс, існує багато варіацій взаємодії користувача з базою

або кешем, і вона визначається архітектурою самої системи. Приклад такої структури показаний на рисунку 4.2.

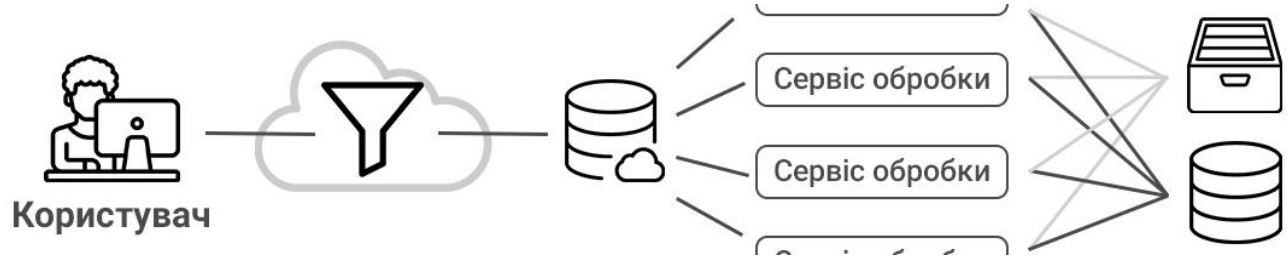

Рисунок 4.2 – Приклад взаємодії з базою даних у хмарі

Такий тип архітектури використовується найчастіше і вважається стандартним для взаємодії бази даних і сервісів у хмарі. При зверненні до хмарової бази даних – запис користувача проходить декілька етапів. По-перше, він фільтрується як вхідний трафік, так і використовуючи певні функції, які вказані розробниками. Це може бути перевірка вхідних даних, або будь-яка інша програмна маніпуляція. Після чого дані потрапляють до тимчасового сховища, яке зберігає усі дані, які надійшли від системи. Періодично, тимчасова база даних очищається, та всі дані, які у ній зберігались підлягають обробці у інших сервісах у хмарі. Обробка дозволяє відокремити корисні дані, від другорядних помилкові та успішні запити, та потім розподілити всю інформацію або у архів для подальшого аналізу, або у хмарову базу даних системи, яка використовується як основна.

Поширення хмарових сервісів, а також хмарних баз даних призвели до переосмислення того, як створюється структура баз даних для мікросервісних систем та чи створюється вона взагалі. Хоча багато хто дотримується думки, що бази даних у хмарі є такими самими, як бази даних на сервері, на практиці і модель використання, і платформа відрізняються, що робить певні зміни у процесі проектування. По-перше, база даних зазвичай розміщується віддалено. Користувачі, які споживають дані, рідко взаємодіють з цими даними. Здатність даних ефективно пересуватись між фізичною базою даних та додатком чи користувачем – це найголовніше, що потрібно враховувати при розробці структури бази даних та дизайну програмних додатків, які використовують цю СУБД. По-друге, база даних може бути широко розповсюджена. Це означає, що аспекти даних можуть існувати в хмарі, у користувача, в кеші або в декількох

хмарах. Ці типи складних мульти-хмарних архітектур стають нормою, і бази даних повинні бути розроблені на основі використання мульти-хмари, включаючи спосіб розподілу даних на кожній платформі (шардінг та реплікація).

Таким чином, до бази даних зазвичай надається доступ за допомогою сервісів передачі даних. Використання хмари означає використання хмарних API чи сервісів, і база даних не відрізняється у цьому підході. При розгляді нових систем, що будуються на хмарних платформах, а також існуючих систем, які будуть мігрувати до хмари, використання сервісної орієнтації є набагато більш «архітектурно правильним» та продуктивним рішенням. Таким чином, збільшується можливість повторного використання цих служб даних серед багатьох типів додатків, порівняно з більш традиційними підходами до створення структур баз даних. Це еволюція в застосуванні баз даних на хмарних платформах.

Основними правилами вважаються проектування структури бази даних як автономного об’єкта, який не пов’язаний з конкретною програмою чи системою. Закладання безпеки та управління у базі даних э важливим етапом проектування, та розглядається, як правило, окремо. Безпеку слід враховувати, коли база даних розроблена, і розширяти її можливості, у ході використання. Термін «дані в спокої» відноситься до неактивних даних, які не використовуються у сховищі. Хоча неактивні дані можуть здатися недостатньо важливими для забезпечення витрат на інформаційну безпеку, дані, що знаходяться в спокої, часто є цілями для кіберзлочинців. Тому слід використовувати токенізацію, яка заміняє важливі дані лише набором символів, які не несуть сенсу та шифрування для безпечної передачі важливих даних по мережі та використання їх тільки всередині системи.

Розробка та впровадження баз даних є однією з найважливіших задач при проектуванні будь-якої системи, яка орієнтована на дані, а також під час розвитку та масштабування. Слід правильно створити план використання бази даних, використовувати належну нормалізацію, використовувати чіткі стандарти іменування та документувати сховищ та полів даних, що є гарною практикою для всіх проектувальників баз даних.

Таким чином, при створенні структури бази даних для хмари чи сервера потрібно враховувати основні принципи роботи та взаємодії з даними, але звертати увагу на різницю у обробці даних.

5 РОБОТА З ДАНИМИ ЗА УМОВ ВИКОРИСТАННЯ РОЗПОДІЛЕНОЇ БАЗИ ДАНИХ

Для проведення роботи використані останні стабільні версії СУБД (на момент проведення експериментів), це MariaDB 10.3 і MongoDB 4.0 [26][27]. Для забезпечення рівності результатів порівняльного аналізу, тестований об'єкт повинен мати структуру, наближену до тієї, яка поширена на практиці, і при цьому бути схожою за складністю зв'язків для забезпечення об'єктивності результатів. Таким чином, проектується база даних, яка представляє список користувачів веб-сервісу, де будуть зберігатися: інформація про користувача, його рівні доступу в системі і коментарі, які він в ній залишав.

Для реляційної бази даних спроектовано 3 таблиці (рисунок 5.1): clients, clients_comments, clients_access.

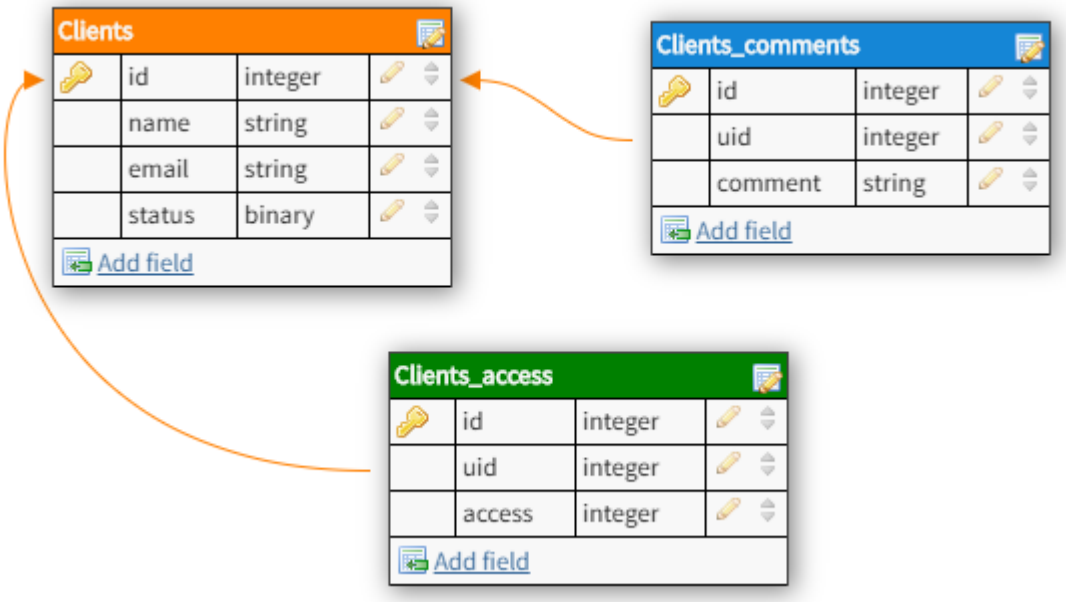

Рисунок 5.1 – Логічна схема проектованої реляційної бази даних

Таблиця clients зберігає в собі основні дані про клієнта, та має наступні атрибути:

- id, первинний ключ (Primary key), тип medium int;
- name, ім'я клієнта, тип varchar (80);
- email, e-mail клієнта, тип varchar (80);

– status, статус клієнта в системі (0 – не активний / 1 – активний), тип tinyint (1).

Таблиця clients_comments зберігає інформацію про коментарі користувача в системі, та має наступні атрибути:

– id, первинний ключ (Primary key), тип medium int;
– uid, id клієнта у системі, являє собою зовнішній ключ (Foreign Key) для таблиці clients, тип medium int;
– comment, коментар клієнта у системі (текст), тип varchar (512).

Таблиця clients_access зберігає інформацію про рівень доступу користувача в системі, та має наступні атрибути:

– id, первинний ключ (Primary key), тип medium int;
– uid, ідентифікатор користувача у системі, являє собою зовнішній ключ (Foreign Key) для таблиці clients, тип medium int;
– access, рівень доступу користувача у системі, тип tinyint (4).

Використовуючи NoSQL модель немає необхідності впроваджувати велику кількість різноманітних типів даних, так як колекції будуть займати рівно стільки місця, скільки даних вони містять, без додаткової мета-інформації.

Такі типи даних вибрані тому, що вони зберігаються в базах найчастіше, особливо коли предметна область пов'язана з web-технологіями. Кращим рішенням було б зберігання численних даних в зв'язці з базою знань, де б коди могли бути розшифровані. Однак такий підхід виправданий при наявності великої кількості даних, які не можуть оброблятися СУБД без видимої затримки більше 2 секунд.

Спроектуємо аналогічну за складністю зв'язків і кількості інформації, що зберігається схему для NoSQL бази даних (рисунок 5.2).

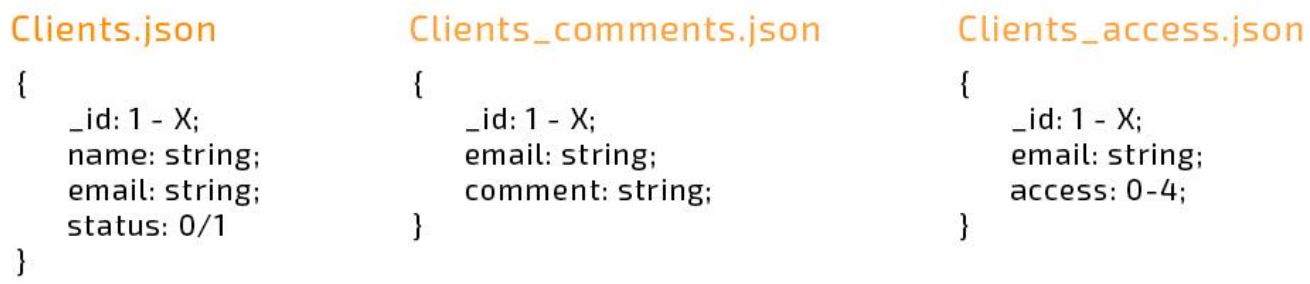

Рисунок 5.2 – Логічна схема проектованої NoSQL бази даних

Структура NoSQL бази даних також спроектована з допущенням, що текстові та числові дані зберігаються набагато частіше за інших типів і їх обробка дає однозначні результати при навантажувальні тестуванні, на відміну від типів даних, які можуть не бути присутнім в певній СУБД, наприклад, просторові.

Для додавання тестового набору даних в базу даних були використані власні генератори значень на мові PHP. Для підвищення швидкості генерації великої кількості записів була використана імітація потоків виконання в PHP. Для підвищення релевантної інформації, що зберігається, генерації клієнтів використовується база справжніх імен, для генерації e-mail - алгоритм випадкового вибору символів, таким чином створюються унікальні значення. Це дозволяє проводити тестування продуктивності в умовах функціонуючого веб-додатки на сервері БД. Вихідний код основних функцій генератора представлений на рисунку 5.3.

```
function generate_email() {
    $chars = "qazxswedcvfrtgbnhyujmkiolp1234567890";
    $max   = 12;
    $vindict = "@vindict.com.ua";
  while($max--) {
      $email .= $chars[rand(0,35)];
    }
    $email .= $vindict;
    return $email;
}

function generate_name($names) {
    $name = $names[rand(0,51529)];
    return $name;
}

  function generate_comment($comments) {
      $comment = $comments[rand(0,535)];
      return $comment;
  }
```

Рисунок 5.3 – Вихідний код основних алгоритмів генерації даних

Схеми різних БД схожі між собою в тому, що інформація логічно структурована та зберігатися окремо один від одного. Використання автоматичних генераторів дає схожі результати, однак може тривати довше на запис генерованої інформації безпосередньо в БД. На додаток до цього не всі генератори можуть створити умовно-реальні синтетичні дані, що вплинуло б на

похибка результатів тестування. Приклад створюваних даних для СУБД MariaDB представлений на рисунку 5.4

Clients

id	name	email	status
1	Айнара	te5req6ofj87@vindict.com.ua	1
2	Жанбаур	l6l3l4yvbl9i@vindict.com.ua	1
3	Халбуви	4jqo147mv6ex@vindict.com.ua	0
4	Лукриция	tk7m48yi8byj@vindict.com.ua	0
5	Девятко	trch4nupof7t@vindict.com.ua	0
6	Акзере	fzb8y6ehfvun@vindict.com.ua	1
7	Калор	phahu3qz8ywd@vindict.com.ua	1
8	Бриттани	z5yjrjs69srn@vindict.com.ua	0
9	Аникита	j0929d2ayhwg@vindict.com.ua	1
10	Иеремия	jcev5cx3lizd@vindict.com.ua	0
11	Джакоб	fw5jr0e1mand@vindict.com.ua	0
12	Гирмиона	vr1fx4um7rrf@vindict.com.ua	1
13	Пекола	ckxc2riob9p2@vindict.com.ua	0

Clients_access

id	uid	access
1	1	0
2	2	1
3	3	1
4	4	4
5	5	1
6	6	3
7	7	2
8	8	2
9	9	0
10	10	1
11	11	4
12	12	1
13	13	4

Clients_comments

id	uid	comment
1	1	Павловна
2	2	Павловна
3	3	приближенная
4	4	верный
5	5	как
6	6	изысканном
7	7	Павловны,
8	8	que
9	9	ses
10	10	быть
11	11	Qu'a-t-on
12	12	он
13	13	можно

Рисунок 5.4 – Приклад даних MariaDB

Дані для MongoDB згенеровано аналогічним чином. Приклад генерованих значень для СУБД MongoDB представлено на рисунку 5.5.

Clients

```
{
    "_id" : NumberInt(1),
    "name" : "Лаодика",
    "email" : "j8khhtcwb7za@vindict.com.ua",
    "status" : NumberInt(0)
}
// ------------------------------------------------
{
    "_id" : NumberInt(2),
    "name" : "Курьяк",
    "email" : "1esuj9dbmu6a@vindict.com.ua",
    "status" : NumberInt(1)
}
// ------------------------------------------------
{
    "_id" : NumberInt(3),
    "name" : "Юрата",
    "email" : "9n3f59dcw3bt@vindict.com.ua",
    "status" : NumberInt(1)
}
```

Clients_access

```
{
    "_id" : NumberInt(1),
    "email" : "j8khhtcwb7za@vindict.com.ua",
    "access" : NumberInt(2)
}
// ------------------------------------------------
{
    "_id" : NumberInt(2),
    "email" : "1esuj9dbmu6a@vindict.com.ua",
    "access" : NumberInt(1)
}
// ------------------------------------------------
{
    "_id" : NumberInt(3),
    "email" : "9n3f59dcw3bt@vindict.com.ua",
    "access" : NumberInt(3)
}
```

Clients_comments

```
{
    "_id" : NumberInt(1),
    "email" : "j8khhtcwb7za@vindict.com.ua",
    "comment" : "dites"
}
// ------------------------------------------------
{
    "_id" : NumberInt(2),
    "email" : "1esuj9dbmu6a@vindict.com.ua",
    "comment" : "bien,"
}
// ------------------------------------------------
{
    "_id" : NumberInt(3),
    "email" : "9n3f59dcw3bt@vindict.com.ua",
    "comment" : "comme"
}
```

Рисунок 5.5 – Приклад даних MongoDB

В рамках роботи над проектуванням структури бази даних для MongoDB, була можливість об'єднати всю інформацію про клієнтів (тестова предметна

область) в одну колекцію і працювати виключно з однією колекцією, приклад розміщення даних в такій колекції представлений на рисунку 5.6.

```
{
    "_id": 1,
    "name": "Laodika",
    "email": "das4adka3sdsa@vindict.com.ua",
    "status": 1,
    "access": 2,
    "comments": [
        { "comment": "Home sweet home"},
        { "comment": "Something new"}
        ...
        N
    ]
}
```

Рисунок 5.6 – Приклад розміщення даних в рамках однієї колекції (MongoDB)

Причому на вкладені елементи можна накласти індекси і шукати по ним. Це вписується в рамки NoSQL і слогана «зберігайте дані як завгодно», проте це не зовсім так. Багато розробників, почувши цей слоган, вирішили повністю відмовитися від реляційної, структурованої системи зберігання даних і писали все в один документ. Суть цієї проблеми добре описано в статті Sarah Mei «Why You Should Never Use MongoDB» (Чому ви ніколи не повинні використовувати MongoDB) [28]. У ній розробники соціальної мережі Diaspora вирішили використовувати MongoDB, як і багато розробників Diaspora вирішили, що NoSQL це легке рішення від продумування архітектури бази даних (а саме всі зв'язки можна зберігати в одному документі), але, на жаль, архітектуру в NoSQL треба продумувати набагато ретельніше , ніж в реляційних базах даних. Через деякий час, у розробників Diaspora почалися проблеми зі зберіганням даних, а саме проблема з пам'яттю, дублювання даних і проблеми з читанням даних, але найголовніша проблема - неможливість зміни структури бази даних через багаторівневу вкладеность даних всередині колекції. Тому навіть в рамках цієї роботи багаторівнева вкладеність застосовуватися не буде.

У зв'язку з різним підходом до зберігання даних, розміри баз даних різні. MongoDB, яка зберігає дані в форматі JSON, займає менше місця, ніж реляційні таблиці MariaDB.

На рисунках 5.7 та 5.8 представлені результати генерації даних, з підсумковим обсягом інформації, що зберігається для MariaDB і MongoDB відповідно. Відмінністю від практичного прикладу є той факт, що, як правило, даних в таблиці коментарів буде більше, ніж користувачів, проте на результати експерименту це не вплине, і таблиці заповнені таким чином для зручності проведення порівнянь для певних даних.

	Таблица	Действие	Строки	Тип	Сравнение	Размер	Фрагментировано
☐	clients		15,000,000	InnoDB	utf8_general_ci	4.7 ГиБ	-
☐	clients_access		15,000,000	InnoDB	utf8_general_ci	2.2 ГиБ	-
☐	clients_comments		15,000,000	InnoDB	utf8_general_ci	3.4 ГиБ	-
	3 таблиц(а)	Всего	45,000,000	InnoDB	utf8_general_ci	10.3 ГиБ	0 Байт

Рисунок 5.7 – Підсумковий обсяг бази даних MariaDB

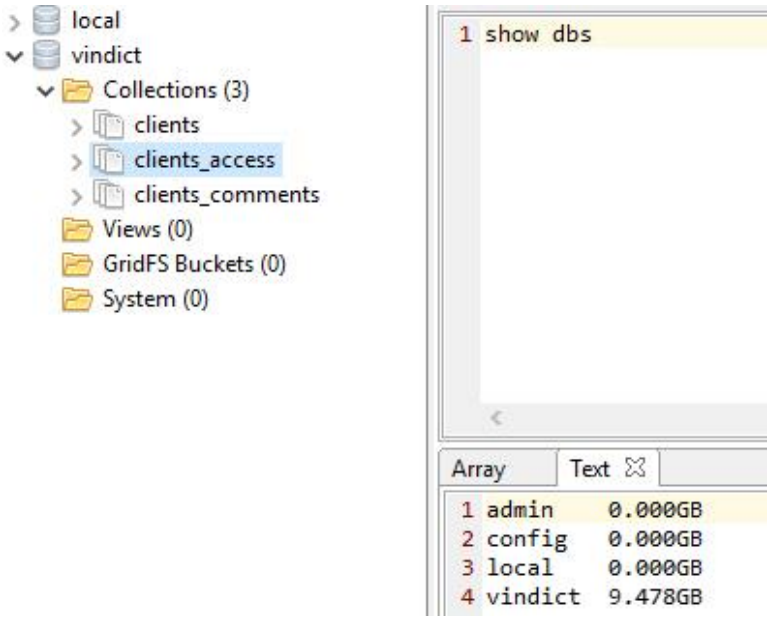

Рисунок 5.8 – Підсумковий обсяг бази даних MongoDB

Тип таблиці відіграє велику роль у визначенні підсумкової швидкості роботи бази даних. MariaDB підтримує два різних типи таблиць: транзакційні (наприклад, InnoDB) і без підтримки транзакцій (наприклад, MyISAM).

Серед безумовних переваг транзакційних таблиць є те, що вони мають високу надійність, тобто якщо станеться збій в роботі MariaDB або виникнуть проблеми з обладнанням, втрачені дані будуть підлягати відновленню – або методом автоматичного відновлення, або за допомогою резервної копії та журналу транзакцій. Також транзакційний тип таблиці має можливість поєднувати кілька операторів і приймати ці оператори однією командою COMMIT. Транзакції дають можливість запуску команди ROLLBACK, що скасовує внесені зміни (якщо робота не проводиться в режимі автоматичної фіксації). При виникненні збою під час оновлення, всі зміни будуть відновлені (в нетранзакціонних таблицях всі внесені зміни не можуть бути скасовані).

Серед переваг таблиць без транзакцій варто означити те, що вони мають більш високу швидкість виконання запитів за рахунок відсутності виконання додаткових транзакцій. Таблиці цього типу займають менше дискового простору через відсутність зберігання додаткових транзакцій. Для оновлення таблиці використовується менше оперативної і дискової пам'яті [29].

На практиці найчастіше використовують транзакційний тип таблиць, тобто InnoDB, тому в рамках цієї роботи також буде використаний цей тип таблиць.

Після того, як було згенеровано дані для MariaDB та MongoDB, постає питання роботи з великими обсягами даних. Компанія Oracle проводила дослідження продуктивності систем з великими обсягами даних (що містять понад 500 мільйонів записів). Ці дослідження, показують, що для баз MariaDB, виконання команди «INSERT» для запису понад 350 тисяч рядків займає трохи більше 6 хвилин, а виконання запиту на читання за допомогою команди «SELECT» для баз, що містять понад 2 млн записів, займає більше 4 хв. Подібні дослідження продуктивності демонструють, що використання баз даних на основі SQL, що містять великі обсяги даних і не використовують технології масштабування, дуже важко. У свою чергу, в MongoDB з самого початку закладений механізм спрощування процесу масштабування.

Для реляційної бази даних MariaDB передбачено такі засоби масштабування і розподілу навантаження як партиціювання, тобто розбиття таблиць на логічні частини за обраними критеріями в рамках однієї машини, і реплікація, яка полягає в побудові розподіленої системи з робочими вузлами, що зберігають ідентичну інформацію і забезпечують, таким чином, горизонтальне масштабування операцій читання. Партиціювання підрозділяється на наступні види:

– По діапазону значень. Даний вид дозволяє явно задати кордону поділу таблиці за значенням числових або логічних значень стовпців;
– По точному списку значень. Застосовується в разі, коли потрібно виділення груп одиничних записів в окремі партіції (підтаблиці);
– По хеш-функції. У цьому випадку користувач не вказує явно кордону розбиття. При використанні цього виду вказується тільки кількість партіцій, на які слід розбити таблицю, і стовпець, від якого береться хеш-функція. На підставі отриманої хеш-функції СУБД сама виробляє розбиття;
– За ключем. Даний вид практично аналогічний способу розбиття по хеш-функції з тією відмінністю, що розбиття завжди відбувається по ключовому стовпцю.

В рамках роботи для СУБД MariaDB буде використано партиціювання по діапазону значень, так як заздалегідь відомий максимальний обсяг записів в таблицях бази даних. Для партиціювання за діапазоном в якості діапазонів використовувалися ідентифікатори (id) згенерованих записів – всього 15 партіцій для кожного з мільйона записів. Всього буде створено 15 партіцій на таблицю (по 1 партиції на кожен мільйон записів). На рисунку 5.9 представлений вихідний код створення таблиці clients з партиціюванням по діапазону значень.

```
CREATE TABLE clients (
    `id` MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT,
    `name` VARCHAR(80) NOT NULL,
    `email` VARCHAR(80) NOT NULL,
    `status` TINYINT(1) UNSIGNED NOT NULL DEFAULT '0',
    PRIMARY KEY (`id`)) ENGINE = InnoDB
    PARTITION BY RANGE (id)
(PARTITION p0 VALUES LESS THAN (1000000) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (2000000) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (3000000) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (4000000) ENGINE = InnoDB,
 PARTITION p4 VALUES LESS THAN (5000000) ENGINE = InnoDB,
 PARTITION p5 VALUES LESS THAN (6000000) ENGINE = InnoDB,
 PARTITION p6 VALUES LESS THAN (7000000) ENGINE = InnoDB,
 PARTITION p7 VALUES LESS THAN (8000000) ENGINE = InnoDB,
 PARTITION p8 VALUES LESS THAN (9000000) ENGINE = InnoDB,
 PARTITION p9 VALUES LESS THAN (10000000) ENGINE = InnoDB,
 PARTITION p10 VALUES LESS THAN (11000000) ENGINE = InnoDB,
 PARTITION p11 VALUES LESS THAN (12000000) ENGINE = InnoDB,
 PARTITION p12 VALUES LESS THAN (13000000) ENGINE = InnoDB,
 PARTITION p13 VALUES LESS THAN (14000000) ENGINE = InnoDB,
 PARTITION p14 VALUES LESS THAN (15000000) ENGINE = InnoDB);
```

Рисунок 5.9 – Вихідний код створення таблиці clients з партиціюванням

СУБД MongoDB з самого початку спроектована з розрахунку на горизонтальне масштабування, яке є одним з найбільш розумних методів підвищення потужності систем зберігання даних. Для цього в ній реалізований

механізм автоматичного шардінгу (autosharding), який управляє розподілом інформації, що зберігається між вузлами системи. При цьому код програми нічого не знає про інфраструктуру і взаємодіє з сегментованим кластером так, як ніби це один вузол.

На кожному з шардів, які є окремими машинами, зберігається деяка частина всього безлічі наявних даних. Завдяки їм і реалізується масштабування операцій запису і читання. Конфігураційний сервер відповідає за зберігання метаінформації про глобальної конфігурації кластера, фізичному місцезнаходження кожної бази даних, колекції та журналі змін, в який записується історія міграції даних між вузлами. Без цієї інформації неможливо отримати узгоджене уявлення сегментованого кластера, тому в виробничих умовах для підвищення відмовостійкості рекомендується мати конфігураційні сервери в кількості, яка дорівнює кількості шардів. Маршрутизатор (він же «майстер») реалізує інтерфейс взаємодії з кластером. Саме йому адресує запити драйвер додатки.

Варто відзначити, що шардінг даних носить, головним чином, логічний характер. Для сегментованої колекції специфікується ключ шардінгу (shard key), який зобов'язаний бути індексом або його частиною і бути присутнім у всіх документах колекції [30]. Далі діапазон всіляких значень ключа ділиться на порції або блоки (chunks). На рисунку 5.10 представлено принцип роботи шардінгу.

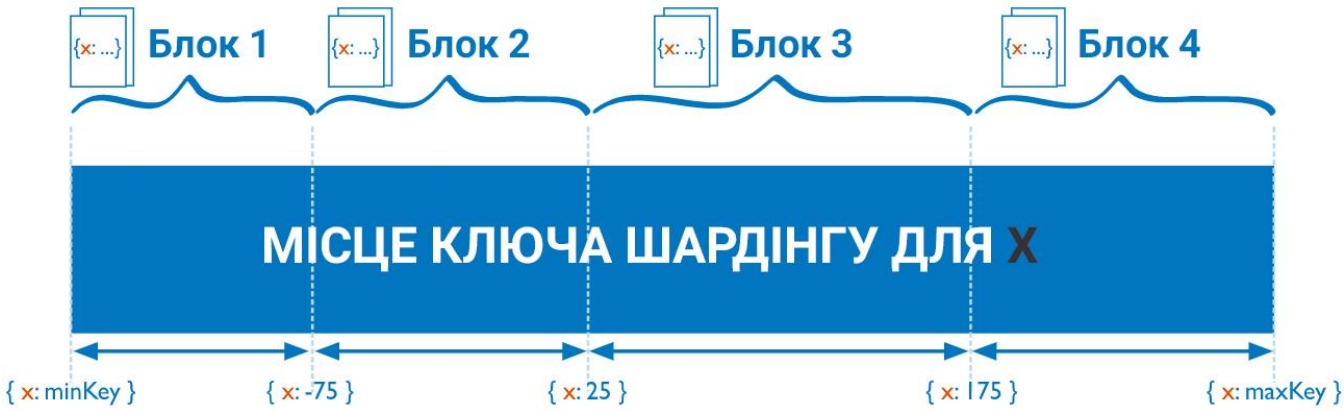

Рисунок 5.10 – Принцип розбиття даних на блоки в MongoDB

Для всіх JSON-документів однозначно визначається, до якої порції вони відносяться. Кожен шард зберігає у себе деяку кількість порцій колекції, причому відповідні їм діапазони значень ключа не обов'язково є суміжними, а порядок

проходження документів в порції нічого не говорить про особливості їх фізичного розташування в шарді.

По досягненню максимального розміру порції (визначається в настройках), відбувається логічне розщеплення порції на два піддіапазону (рисунок 5.11).

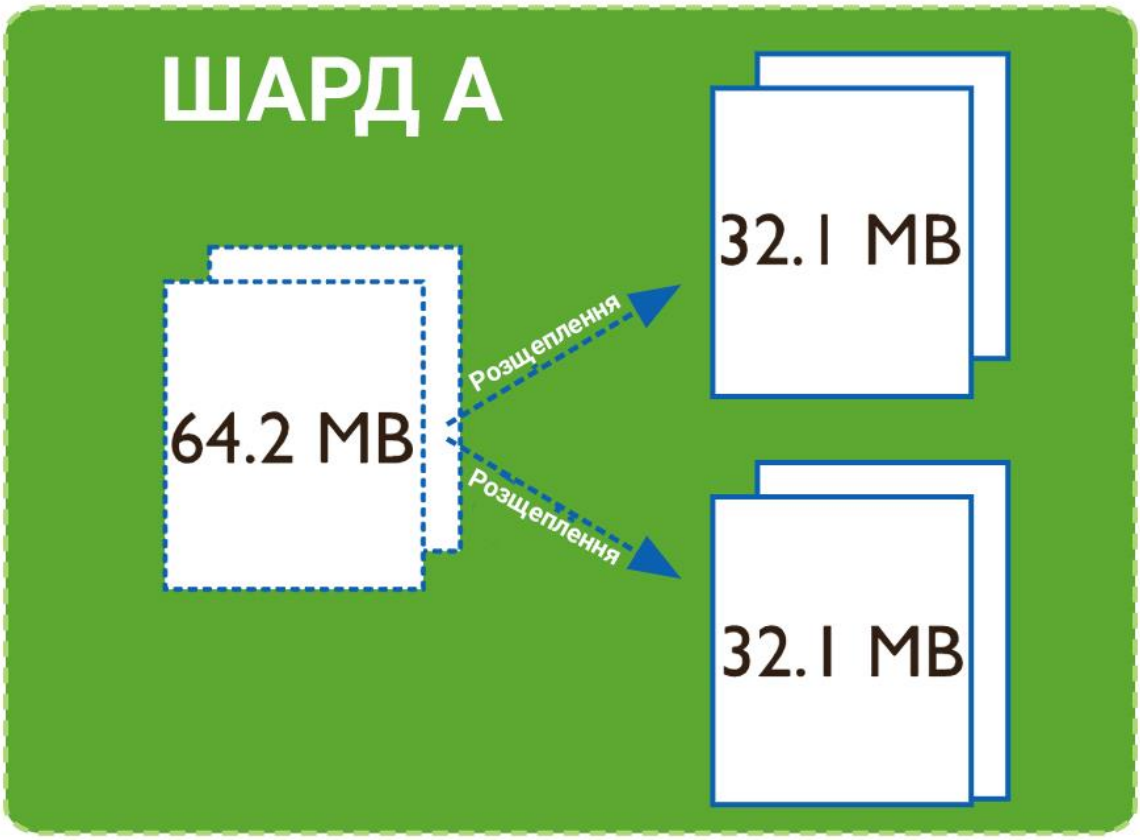

Рисунок 5.10 – Принцип розщеплення при шардінгу в MongoDB

Внутрішня балансувальна функція MongoDB здійснює фізичний процес міграції даних між шардами для управління навантаженням на окремі вузли системи (рисунок 5.11).

Рисунок 5.11 – Принцип міграції даних при шардінгу в MongoDB

Конкретні цифри залежать від загального обсягу даних, але найчастіше міграція здійснюється, коли різниця між максимальним і мінімальною кількістю порцій серед усіх шардів системи стає більше восьми.

Для проведення поліпшення реляційної бази даних була проведена робота з конфігураційними настройками MariaDB [31]. Були внесені корективи представлені на рисунку 5.12.

```
innodb_buffer_pool_size = 6G
innodb_log_file_size = 1G
innodb_log_buffer_size = 1M
innodb_flush_method = O_DSYNC
innodb_flush_log_at_trx_commit = 2
query_cache_size = 0
max_connections = 300
```

Рисунок 5.11 – Зміни в конфігураційних файлах СУБД MariaDB

Розглянемо кожен з параметрів детальніше:

– innodb_buffer_pool_size – це буферна сховище InnoDB, в якому зберігаються хеш-кодування всіх індексів і запитів. Встановлюється в розмірі 80-90% від розміру оперативної пам'яті;
– innodb_log_file_size – цей параметр встановлює розмір логу операцій (так операції спочатку записуються в лог, а потім застосовуються до даних на диску). Чим більше цей лог, тим швидше будуть працювати записи (тому що їх поміститься більше в файл логу). Файлів завжди два, а їх розмір однаковий. Значенням параметра задається розмір одного файлу;
– innodb_log_buffer_size – це розмір буферного сховища транзакцій;
– innodb_flush_method – цей параметр визначає метод скидання даних з пам'яті на диск. Вибір значення – вибір між швидкістю і надійністю. O_DIRECT використовує 4 операції запису на диск на кожен виклик. Всі ці

операції верифицируются, що забезпечує додаткову надійність запису на диск, однак уповільнює виконання. O_DSYNC також використовує 4 операції запису на диск на кожен виклик. Однак при цьому, тільки дві з цих операцій будуть верифіковані (2 операції виконуватися послідовно, а ще 2 паралельно, але без гарантій). Це забезпечить більш ефективну запис, що дозволить збільшити пропускну здатність запису MariaDB.

- innodb_flush_log_at_trx_commit – зміна цього параметра може підвищити пропускну здатність запису даних в базу в сотні разів. Він визначає, чи буде Mysql скидати кожну операцію на диск (в файл логу). 1 використовується для випадків, коли збереження даних - це пріоритет номер один, 2 для випадків, коли невелика втрата даних не критична. В цьому випадку транзакції будуть скидатися в лог на диск тільки раз в секунду;
- query_cache_size – значення цього параметра визначає скільки пам'яті варто використовувати під кеш запитів;
- max_connections – максимальна кількість з'єднань з базою даних за один проміжок часу.

Отже, при використанні розподіленої бази даних, зменшується об’єм кожної з баз, що прискорює виконання запитів. Використання шардінгу дозволяє не втрачати швидкість при зростанні об’єму бази даних. Зміна конфігураційних параметрів дозволила прискорити обробку різнорідних даних, які зберігаються, за рахунок розширення пам’яті, методу кешування та обмеження потоку даних. Необхідно провести порівняльний аналіз та визначити наскільки це впливає на роботу СУБД.

6 ПОРІВНЯЛЬНИЙ АНАЛІЗ ШВИДКОСТІ РОБОТИ РІЗНИХ СУБД ЗА УМОВ ВИКОРИСТАННЯ РІЗНОРОДНИХ ТИПІВ ДАНИХ

Тестування проводилося на наступній конфігурації: одна машина (8-ми ядерний 4GHz Intel Core I7-8550U процесор, 8GB DDR3 RAM, SSD диск розміру 256 GB) для запуску однієї з систем на базі ОС CentOS Linux 7.6.1810 (Core). В якості програмного забезпечення були використані: MariaDB 10.3 (X64), Apache, Navicat for MySQL, MySQL Workbench, PhpMyAdmin, MongoDB 4.0 (X64), Mongo Shell, Studio 3T.

Над базами даних виконувалися наступні операції: вставка запису, читання записи, оновлення записи, читання записи з об'єднаних таблиць (JOIN або агрегація за допомогою $lookup). Операції виконувалися над таблицями (колекціями) базами даних наступних розмірів – 150 тисяч, 750 тисяч, 3 мільйони, 6 мільйонів, 15 мільйонів. Сумарно три таблиці налічували 45 мільйонів записів на фінальному етапі проведення тестів. Для вимірів продуктивності над системою виконувалося 20 операцій і бралося середнє значення. Для аналізу продуктивності вимірювалося час відгуку систем на запит - час між початком запиту й одержанням відповіді.

Для кожного тестування, щоб уникнути хешування результатів, дані в таблицях заповнювалися повторно. Навіть при відключенні оптимізації, як запитів, так і зберігання даних, хешування ключів неминуче. Програмне, оптимізація необхідна для прискорення виконання повторних запитів для CRUD операцій. Структура проведення тестування передбачає використання нестандартних операцій, таких як JOIN, щоб перевірити відміну продуктивності СУБД при різному навантаженні під час виконання складних запитів. Команда з'єднання для NoSQL виглядає наступним чином (рисунок 6.1)

```
db.vindict.aggregate (
[
  {$ lookup:
    {from: 'clients_comments',
     localField: 'email',
     foreignField: 'email',
     as: 'comments'
    }
}])
```

Рисунок 6.1 – Запити для тестування продуктивності роботи СУБД

На рисунку 6.1, змінна from – найменування зовнішньої колекції, localField – поле приєднання з поточної колекції, foreignField – поле приєднання із зовнішньої колекції, as – псевдонім, найменування для отриманого приєднання записів. На момент проведення тестів можна було здійснювати приєднання тільки по одному полю і тільки в форматі left outer join. Таким чином, виходячи з існуючих обмежень і тимчасових характеристик виконання даного запиту під час організації приєднань таблиць в обраному NoSQL-рішенні, слід користуватися іншими методами.

Налаштування позначає таку настройку СУБД, при якій час виконання запитів залежить тільки від алгоритму їх виконання конкретної СУБД. Всі фактори, які могли б вплинути на продуктивність, виключаються.

Для тестування використовуються як прості запити пошуку інформації, так і ускладнені запити вибірки даних. Запити по складності будуть ідентичними для тестованих систем управління базами даних. Передбачувана навантаження на різні СУБД буде однаковою за рахунок використання ідентичних типів і обсягів даних. Виконувані запити для тестування представлені на рисунку 6.2.

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

Рисунок 6.2 – Запити для виявлення продуктивності роботи СУБД

Фактично, час виконання запитів в цьому розділі представлено без похибки. На практиці, швидкість виконання запитів завжди буде вище, так як за замовчуванням в обох СУБД стоять налаштування оптимізації запитів, хешування і пошук по збереженим ключам, особливо в разі складних запитів

Додавання значень в базу даних, індекси, налаштування та масштабування на цьому етапі не використовуються. Для виконання умов тестування, запущений ітераційний процес INSERT до досягнення початкової позначки в 150.000 записів, далі в 750.000 і далі, до останньої ітерації в 15 мільйонів значень. Показники часу виконання операції починаються з моменту додавання першого запису, і закінчується після додавання останньої записи і завершення операції СУБД. Для позначення тисяч використовується буква k, а для мільйона записів - буква M.

Додавання даних відбувається одночасно в 3 таблиці, в разі MariaDB і відповідно в 3 колекції для MongoDB. Результати тестування процесу додавання (INSERT) представлені на рисунку 6.3.

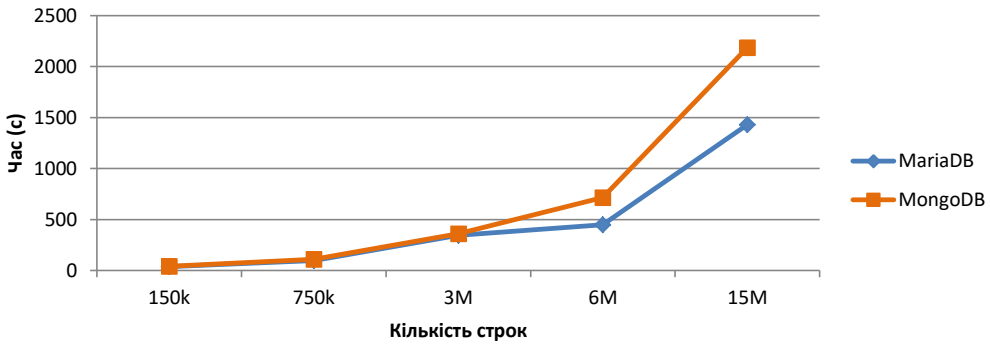

Рисунок 6.3 – Час обробки запиту додавання даних у БД без налаштувань

MariaDB продемонструвала більш високу продуктивність запису, ніж у MongoDB, у якій процес додавання даних займає майже в 2 рази більше часу.

Швидкість читання даних з бази даних (SELECT). Результатом цієї операції буде вибірка певних значень з БД. Результати тестування представлені на рисунку 6.4.

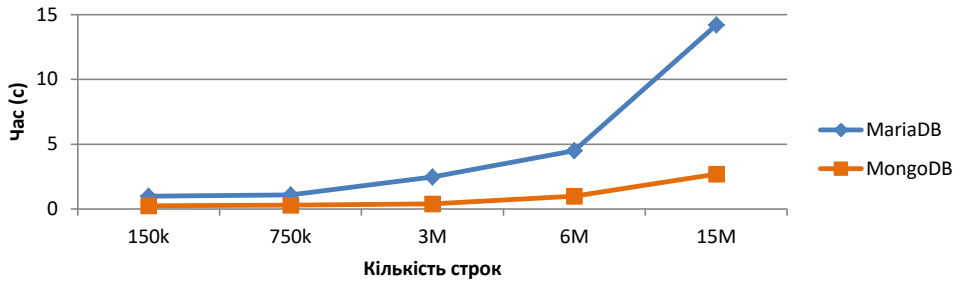

Рисунок 6.4 – Час обробки запиту читання даних у БД без налаштувань

Швидкість оновлення (зміни) даних. Результатом цієї операції буде змінені значення певних полів, які будуть вказані в запиті. Результати цього тестування представлені на рисунку 6.5.

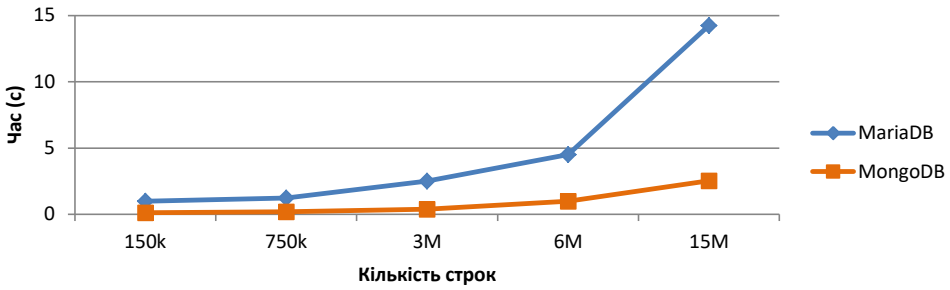

Рисунок 6.5 – Час обробки запиту оновлення даних у БД без налаштувань

MongoDB продемонструвала перевагу в швидкості виконання при оновленні даних, більш, ніж в 3 рази, на всіх етапах тестування.

Швидкість видалення даних (DELETE). Неактуальні дані видаляються для звільнення простору, і швидкість виконання цього процесу відіграє важливу роль в роботі всієї системи. В результаті виконання операції деякі рядки, зазначені в запиті, будуть видалені з БД. Результати тестування процесу видалення даних представлені на рисунку 6.6.

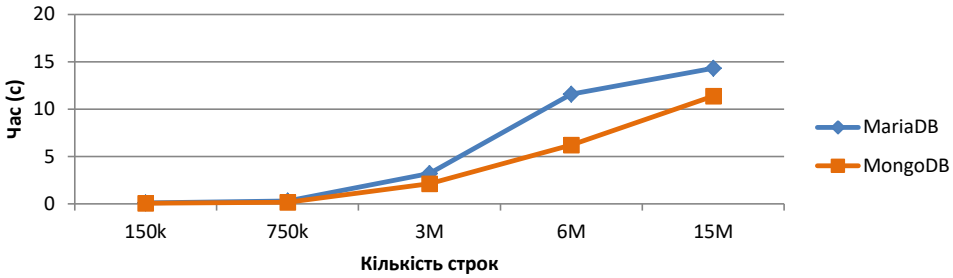

Рисунок 6.6 – Час обробки запиту видалення даних у БД без налаштувань

На малих обсягах даних MariaDB і MongoDB знаходяться на однаковому рівні – процеси видалення виконуються менш секунди, проте при збільшенні даних, MariaDB починає відставати в швидкості виконання і за підсумком програє MongoDB в цьому типі тестуванні.

Фінальним етапом тестування є перевірка швидкості виконання з'єднання двох таблиць (колекцій) для виведення єдиного формату даних. Результатом цієї операції буде висновок даних, взятих з різних джерел (таблиць). Результати тестування з'єднання представлені на рисунку 6.7.

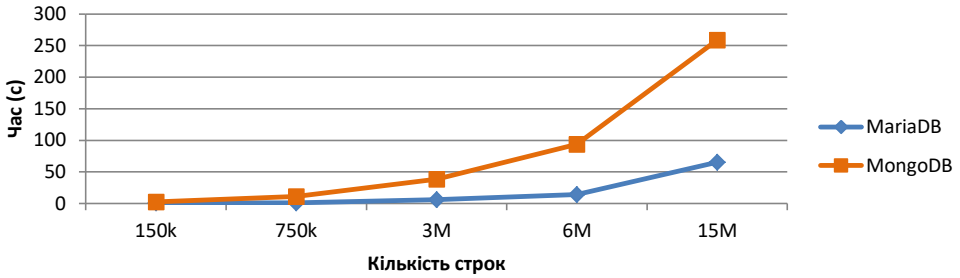

Рисунок 6.7 – Час обробки запиту об’єднання даних у БД без налаштувань

Явним фаворитом в процесі об'єднання результатів є MariaDB, почасти тому що реляційні бази даних спочатку розраховані на це по своїй структурі, а в MongoDB операція з'явилася не так давно.

При тестуванні MongoDB продемонструвала явну ефективність виконання запитів в читанні, зміну і видалення даних, проте в процесах додавання і об'єднання даних лідирує MariaDB. Всі результати винесені в таблиці 6.1, де виділено час в секундах (с) обробки запитів двох СУБД.

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

Для складання повної картини порівняльного аналізу необхідно повторне тестування вже з проведенням робіт по поліпшенню баз даних, установки індексів в БД та процесів масштабування. Налаштування включила в себе: налаштування показників СУБД, включення оптимізатора запитів, настройка хешування

результатів виконання запитів і ключів, а також використання налаштувань СУБД за замовчуванням.

Додавання даних повторно проводитися не буде, так як цей процес є надлишковим з огляду на те, що при установці індексів процес додавання даних значно сповільнюється. Уповільнення відбувається через те, що при кожному додаванні нових даних індекс будується заново і тому відбувається значне уповільнення виконання процесу додавання.

Для оптимізації були виставлені індекси на запитувані дані. Для MariaDB були встановлені індекси на 'name', 'status' в clients, 'uid' в clients_comments і clients_access. Для MongoDB встановлені індекси на 'name', 'status', 'email' в clients, 'email' в clients_comments і clients_access. Результати проведених тестів по порядку (читання, зміни, видалення та об'єднання даних) представлені на рисунках 6.7, 6.8, 6.9, 6.10.

Відповідно, при цьому хешування результатів запитів було навмисно відключено в ході оптимізації роботи СУБД, так як воно не дозволяє повноцінно оцінити швидкість виконання будь-яких запитів. Щоб домогтися «чистих» результатів, обидві бази даних виконують операції повномірного, без скорочення запитів оптимізатором і збереження результатів.

Тестування проводилося безпосередньо, використовуючи оболонку СУБД, так як при виконанні запитів мову програмування створює додаткову затримку. Так, наприклад, PHP при використанні MariaDB завжди буде показувати результат виконання запиту більше однієї секунди, хоча реально це не так. Такі затримки обумовлені роботою внутрішніх скриптів мови програмування і передачею даних навіть невеликого обсягу. Інший цікавий факт, що за допомогою структури, в якій MongoDB зберігає дані і алгоритму хешування, який використовується цій же СУБД, запити на мові PHP можуть виконуватися і оброблятися зі швидкістю до 0.01 секунди, показуючи найвищу продуктивність, навіть при 15 мільйонах записів. Незважаючи на це, MariaDB може досягти таких же результатів, однак хешування даних необхідно проводити на стороні сервера, використовуючи саму мову програмування, а не СУБД. Для цього, як правило, використовуються сторонні бібліотеки і витрачається більше годин на розробку, чого можна уникнути, використовуючи MongoDB, так як вона має вбудований кеш запитів і їх результатів.

Завдяки новим версіями СУБД, була в загальному поліпшена їх продуктивність і взаємодію з даними, а також робота по мережі. Датою оновлення

конфігурації БД, додані типи даних і способи роботи з інформацією всередині БД. Всі ці зміни несуть позитивні зміни при тестуванні і в повній мірі проявляються при детальній настройці кожної СУБД.

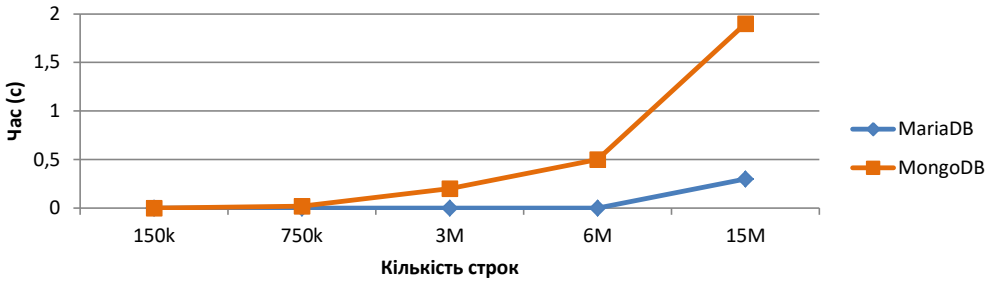

Рисунок 6.7 – Час обробки запиту читання даних у БД з проведеним налаштуванням

Як видно на рисунку 6.7, читання даних в налаштованій MariaDB до 6 мільйонів записів не перевищує 0.0003 секунди і лише до показника в 15 мільйонів записів, швидкість виконання склала 0.26 секунд. У той же самий час, MongoDB прогнозовано стала сповільнюватися в швидкості після 1 мільйона записів, так як читання колекцій більш витратна операція, ніж читання за індексами. Фінальне порівняння показало, що оптимізована MariaDB по операції читання даних в середньому в 7 разів швидше MongoDB.

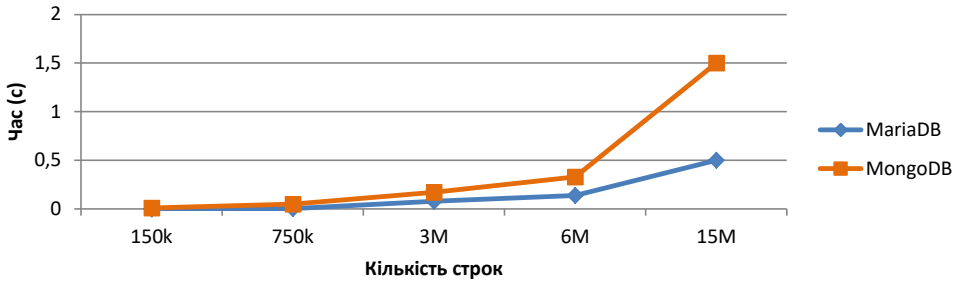

Рисунок 6.8 – Час обробки запиту оновлення даних у БД з проведеним налаштуванням

Оновлення даних – більш витратний процес, навіть при налаштованих індексах. Як видно на рисунку 6.8, MongoDB вже з 750 тисяч записів починає істотно збільшувати витрати на виконання операції оновлення. Хоча MariaDB також показує схожу динаміку зростання, проставлені індекси і внутрішня оптимізація СУБД значно впливають на продуктивність, і при 15 мільйонах час виконання запиту не перевищило 0.5 секунд. В цей же час, MongoDB впоралася з цим завданням в 3 рази довше (1.5 сек).

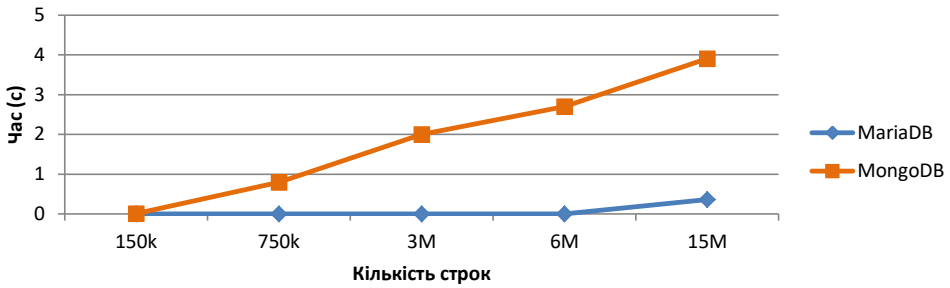

Рисунок 6.9 – Час обробки запиту видалення даних у БД з проведеним налаштуванням

Видалення даних на MongoDB неприємно здивував в зв'язку з самою структурою зберігання даних. Так як структура колекцій і проставлені індекси, пошук необхідної інформації і її видалення, не займають багато часу, однак зсув колекції і переіндексування об'єктів даних займає відчутні в роботі СУБД секунди. Варто зауважити, що такі затримки практично невідчутні для БД маленьких обсягів (до 2 Гб), що розв'язує руки розробникам в невеликих проектах і дозволяє зберігати швидко доступні, невеликі обсяги інформації, наприклад, базу знань або експертну базу.

Під час тестування оптимізація MariaDB показала очікувані результати в 0.004 секунди для 6 мільйонів і майже 0.36 секунд для 15 мільйонів. Швидке знаходження запису і виконання операції видалення не викликає затримок навіть при великій кількості записів.

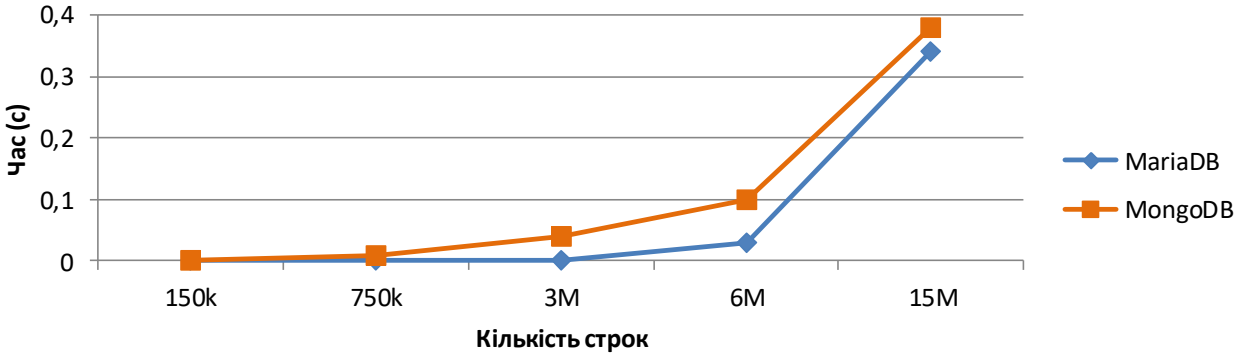

Рисунок 6.10 – Час обробки запиту об’єднання даних у БД з проведеним налаштуванням

Об'єднання – самий складний і витратний по ресурсам, і як наслідок, є самою повільною операцією в тестуванні. У великих або некоректно спроектованих базах виконання цієї операції в значній мірі уповільнює роботу системи, що призводить до труднощів інших розробників. Незначна різниця в часі виконання зберігається при всіх контрольних вимірах часу виконання. Однак варто враховувати, що проводилося виконання об'єднання 2 таблиць, хоча в реальній системі їх кількість може зрости до 8 в складних запитах з численними умовами, сортуванням, угрупованням або використанням додаткових функцій. MariaDB при таких умовах може виконувати запит більш 1.5 секунд, в той час як MongoDB незначно, але довше, не перевищуючи поріг в 2 секунди.

При тестуванні з оптимізацією явним лідером є СУБД MariaDB, яка у всіх етапах тестування проявила високу ефективність і продуктивність. У таблицю 3 зведені остаточні результати тестування з оптимізацією.

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

Проведене дослідження дозволяє заявляти про доцільність використання технології партіціювання в контексті підвищення ефективності взаємодії з

реляційними СУБД. Доцільність підтверджується тим, що використання секціонування таблиць призводить до якісної зменшення часу виконання запитів до реляційних СУБД, що підтверджується проведеними дослідженнями. Але слід зазначити, що цей підхід працює тільки в рамках одного сервера. Тобто, або необхідно налаштовувати партіціроване на кожному сервері окремо, або використовувати виключно на головному сервері, де знаходяться найбільш важливі дані. Хоч MongoDB і поступається в більшості проведених тестів однак не варто стверджувати, що дана СУБД є гірше MariaDB, при наявності хороших кластерів, вона здатна видавати результати рівні або можливо навіть вищі за MariaDB, принаймні в операціях читання і запису даних.

Горизонтальний шардінг (який присутній в MongoDB з коробки) – це одне з найбільш потужних засобів масштабування бази даних. Поділ таблиці на окремі сервера дозволяє масштабувати їх практично нескінченно, рівномірно розподіляючи навантаження між шардами. Для реалізації горизонтального шардінгу в MariaDB необхідно розробити ряд програмних модулів і засобів, які будуть запити від клієнтів направляти через балансувальника навантаження на одну з доступних і найменш завантажені сервера для обробки SQL-запиту. Балансувальник виявляє також сервера які перестали працювати і перенаправляє трафік SQL-запитів на робочі сервера. Таким чином, горизонтальний шардінг для реляційних баз даних є не зовсім простим засобом масштабування.

У підсумку слід позначити, що в сучасному світі немає протистояння між реляційними та нереляційними базами даних. Замість цього варто говорити про їх спільне використання для вирішення завдань, на яких та чи інша технологія показує себе найкраще.

ВИСНОВКИ

В результаті виконання атестаційної магістерської роботи були досліджені питання пов'язані з розмішені різнорідних даних великого обсягу в розподілених базах даних. Був проведений аналіз двох систем управління базами даних – реляційної MariaDB і нереляційної (NoSQL) MongoDB.

В ході роботи були вивчені питання пов'язані з ознайомленням з технічними аспектами роботи з СУБД MariaDB, MongoDB, розподіленими середовищами і обгортками зовнішніх даних та поліпшенням цих баз даних при роботі з великими обсягами даних. Таким чином були розглянуті теми масштабування, серед яких – партіціювання, шардінг і реплікація. Крім цього досліджено теми, які пов'язані з поліпшенням продуктивності бази даних за рахунок коректної розстановки індексів і правильного проектування самої структури бази даних. Після вивчення матеріалів, була спроектована структура для реляційної бази даних і аналогічна за складністю для NoSQL бази даних. Для заповнення даними баз даних синтетичними даними був розроблений генератор даних. Дані, які були створені генератором даних наближені до реальних.

Було проведено тестування, що складаються з різних за складністю та умовами етапів: читання, запис, оновлення, видалення та об'єднання. При цьому тестування проводилося спочатку на базах даних без поліпшень у вигляді розстановки індексів або засобів масштабування, а потім з ними.

Результатом виконання атестаційної роботи є висновки отриманні в ході аналізу проведеного дослідження, а саме те, що при роботі з коробки (без поліпшень) MongoDB майже у всіх етапах дослідження показує себе краще за продуктивністю і відповідно швидкості виконання. При вставці даних і об'єднанні (JOIN в разі MariaDB або $lookup в разі MongoDB) NoSQL система MongoDB показує результати гірше, ніж реляційна MariaDB. Після проведення поліпшення баз даних, результати кардинально змінюються і майже у всіх проведених етапах дослідження MariaDB показувала результати краще MongoDB. Аналіз результатів дозволяє зробити висновки про ефективність кожної СУБД до і після оптимізації, а також зробити висновок про можливість застосування їх на практиці.

Не варто забувати той факт, що в MongoDB JSON-документ є фактично єдиним форматом подання та зберігання даних, таким чином, якби порівняння відбувалося за швидкістю обробки json документів, то реляційна база даних

MariaDB показала результати швидкості обробки даних гірше, ніж MongoDB. Денормализация даних як спосіб превентивного приєднання кортежів в MongoDB є цілком допустимим рішенням, що особливо яскраво проявляється на великих обсягах даних. Однак варто зазначити, що рішення про денормалізацію потрібно приймати виходячи з оцінок витрат на підтримання узгодженості.

Результати дослідження показали, що більш ефективною СУБД є MariaDB. Для досягнення максимальної переваги цієї СУБД необхідна ретельна настройка, опрацювання структури таблиць, ретельна робота с масштабування, поліпшення запитів і ряд додаткових поліпшень самої бази даних, які є важкими для нових користувачів.

MongoDB виявилася гідним конкурентом MariaDB в окремих етапах досліджень з точки зору часу виконання запитів. Цей результат підтверджує існуюче серед фахівців в сфері СУБД думку про те, що MongoDB є одним з найбільш сильних представників сімейства NoSQL систем. У той же самий час експерименти показали, що, незважаючи на запевнення розробників даної СУБД, на сьогоднішній день не можна говорити про неї як про заміннику більш традиційних систем обробки та зберігання даних.

Проміжні дослідження і результати застосування описаних в даній роботі системи управління базами даних були опубліковані в якості наукових публікацій і статей в наукових журналах і в якості доповідей та тез на наукових конференціях [32][33][34]. Багато описаних в цій статті рекомендацій і порад для практичного застосування перевірялися при створенні реальних проектів і систем в рамках написання наукових робіт автором [35][36][37][38].

ПЕРЕЛІК ПОСИЛАНЬ

1. Кузнецов С., Посконин А. Возможно ли сотрудничество SQL и NoSQL?. *Открытые системы*. 2013. №9. С. 38-41.

2. R. Cattell Scalable SQL and NoSQL data stores. *ACM SIGMOD Record*. 2011. № 39 (4). С. 12–27.

3. Data C. J. An introduction to database systems (8^{th} edition). Бостон : Addison-Wesley publ., 2004. 1040с.

4. MySQL – это лучшая NoSQL-система. URL: https://habr.com/ru/company/wix/blog/283158 (дата звернення: 18.10.2019).

5. Changlin H. Survey on NoSQL Database Technology. *Journal of Applied Science and Engineering Innovation*. №2 (2). 2015. С. 50-54.

6. Abramova V., Bernardino J., Furtado P. Which NoSQL Database? A Performance Overview. *Open Journal of Databases (OJDB)*. №1 (2). 2014. С. 17-24.

7. Nayak A., Poriya A., Poojary D. Type of NOSQL Databases and its Comparison with Relational Databases. *International Journal of Applied Information Systems*. №5 (4). 2013. С. 16–19.

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

9. DB-Engines Ranking. URL: http://db-engines.com/en/ranking/ (дата звернення: 19.10.2019).

10. E. Brewer Towards Robust Distributed Systems. *ACM Symposium on the Principles of Distributed Computing* : матеріали XIX міжн. наук.-практ. конф. (м. Ореогн, 16 лип. 2000 р.). Орегон, 2000. С. 7.

11. 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.

12. 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.

13. Лучинкин З. С., Сидоркина И. Г. Проектирование модели данных, слабоструктурированной, нереляционной, документо-ориентированной базы даннях. *Вестник Чувашского университета*. 2014. № 2. С. 103–107.

14. Садаладж П., Фаулер М. NoSQL. Новая методология разработки нереляционных баз данных. Киев : Вильямс, 2015. 192 с.

15. Fowler M., Sadalage P.J. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Бостон : Addison-Wesley, 2013. 192с.

16. Harrison G. Next Generation Databases, NoSQL , NewSQL and BigData. Нью Йорк : Apress, 2014. 235с.

17. Chodorow K. MongoDB: the Definitive Guide. Бостон : O'Reilly Media, 2013. 432с.

18. Vohra D. Pro MongoDB Development. Нью Йорк : Apress, 2015. 504с.

19. 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.

20. DBaaS: базы данных в облаке. URL: https://habr.com/ru/company/technoserv/blog/337860 (дата звернення: 19.10.2019).

21. SQL или NoSQL — вот в чём вопрос. URL: https://habr.com/ru/company/ruvds/blog/324936 (дата звернення: 19.10.2019).

22. Hecht R., Jablonski S. NoSQL evaluation: A use case oriented survey. *International Conference on Cloud and Service* Computing : матеріали міжн. наук.-практ. конф. (м. Гонконг, 12 груд. 2011 р.). Гонконг, 2011. С. 336-341.

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

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

25. 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 2015 р. Орадя, 2015. С. 192-198.

26. Storage Engines - MariaDB Knowledge Base. URL: https://mariadb.com/kb/en/library/storage-engines (дата звернення: 19.10.2019).

27. MongoDB Documentation. URL: https://docs.mongodb.com (дата звернення 19.10.2019).

28. Why You Should Never Use MongoDB. URL: http://www.sarahmei.com/blog/2013/11/11/why-you-should-never-use-mongodb (дата звернення: 19.10.2019).

29. MariaDB в сравнении с MySQL – Особенности. URL: https://mariadb.com/kb/ru/mariadb-vs-mysql-features (дата звернення: 19.10.2019).

30. Sharding — MongoDB Manual. URL: https://docs.mongodb.com/v4.0/sharding (дата звернення: 20.10.2019).

31. Оптимальная настройка Mysql. URL: https://ruhighload.com/Оптимальная+настройка+mysql (дата звернення: 20.10.2019).

32. 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.

33. 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.

34. 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.

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

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

37. 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.