Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

магістра
(назва освітньо-кваліфікаційного рівня)

студента *Усатенко Максима Володимировича*
(ПІБ)

академічної групи *121м-22-2*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Дослідження реляційної бази даних, як відмовостійкого компоненту високонавантаженого бізнесу*

_____М.В.Усатенко

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц.ПриходченкоС.Д.*			
розділів:				
спеціальний	*доц.Приходченко С.Д.*			
Рецензент				
Нормоконтролер	*Доц. Сироткіна О.І.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.А. Алексеєв
(підпис) (прізвище, ініціали)

« » 2022 року

ЗАВДАННЯ
на виконання кваліфікаційної роботи магістра

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

студенту 121м-22-1 Усатенко Максиму Володимировичу
(група) (прізвище та ініціали)

Тема дипломного проекту *Дослідження реляційної бази даних як відмовостійкого компонента високонавантаженого бізнесу*

1 ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Наказ ректора НТУ «Дніпровська політехніка» від 29.11.2018 р. № 2025 -л

2 МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень – стратегії розподіленого збереження даних в реляційних базах даних та їх вплив на відмовостійкість.

Предмет досліджень – реляційні та нереляційні бази даних як компоненти високонавантаженого бізнесу.

Мета роботи – розробка інструментарію та програми з автоматизації щодо прийняття рішень з вибору бази даних.

3 ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна результатів дипломної роботи полягає в удосконаленні інструментарію прийняття рішення про вибір бази даних в залежності від різних факторів впливу.

Практична цінність полягає у тому, що результати роботи, отримані в ході дослідження, можуть застосовуватися при виборі бази даних як для високонавантаженого бізнеса, так і для сучасного IT-середовища.

4 ЕТАПИ ВИКОНАННЯ РОБІТ

Найменування етапів робіт	**Строки виконання робіт (початок – кінець)**
Аналіз теми та постановка задачі	01.09.2023 – 30.09.2023
Розробка інструментарію прийняття рішення про вибір бази даних в залежності від різних факторів впливу	01.10.2023-31.10.2023
Створення на базі розробленого інструментарію програми аналізу баз даних за допомогою мови програмування C# на платформі .NET	01.11.2023-10.12.2023

Завдання видав _____ *Приходченко С.Д.*
(підпис) (прізвище, ініціали)

Завдання прийняв до виконання _____ *Усатенко М.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *01.09.2023 р.*

Термін подання кваліфікаційної роботи до ЕК *11.12.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 76 с., 28 рис., 13 табл., 4 дод., 24 джерела.

Об'єкт дослідження: стратегії розподіленого збереження даних в реляційних базах даних та їх вплив на відмовостійкість.

Предмет дослідження: реляційні та нереляційні бази даних як компоненти високонавантаженого бізнесу.

Мета магістерської роботи: розробка інструментарію та програми з автоматизації щодо прийняття рішень з вибору бази даних.

Методи дослідження. Для виконання поставлених завдань були використані методи літературного аналізу: огляд літератури, критичний аналіз, емпіричні методи: експеримент, аналіз даних, системне та математичне моделювання, експертної оцінки.

Наукова новизна полягає в удосконаленні інструментарію прийняття рішення про вибір бази даних в залежності від різних факторів впливу.

Практичне значення роботи. Результати роботи, отримані в ході дослідження, можуть застосовуватися при виборі бази даних як для високонавантаженого бізнеса, так і для сучасного IT-середовища.

Список ключових слів: бази даних, високонавантаженність, відмовостійкість, зберігання данних, сервер проекту, кількість запитів, проєкт, C#, SQL, NoSQL, MySQL, Performance Schema, MongoDB

ABSTRACT

Explanatory note: 76 pp., 28 pictures, 13 tables, 4 appendices, 24 sources.

Object of research: Strategies of distributed data storage in relational databases and their impact on fault tolerance.

Subject of research: Relational and non-relational databases as components of high-load business.

Purpose of Master's thesis: development of tools and programs for automation of decision-making on the selection of a database.

Research methods. Methods of literary analysis were used to fulfill the tasks: literature review, critical analysis, empirical methods: experiment, data analysis, system and mathematical modeling, expert evaluation.

Originality of research consists in improving the tools for making a decision on the choice of a database depending on various influencing factors.

Practical value of results. The results of the work obtained in the course of the study can be applied when choosing a database both for a highly loaded business and for a modern IT environment.

Keywords: databases, high load, fault tolerance, data storage, project server, number of requests, project, C#, SQL, NoSQL, MySQL, Performance Schema, MongoDB

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СУРБД - система управління базами даних

РБД - реляційні бази даних

ACID - (Atomicity, Consistency, Isolation, Durability)

IoT - (internet of things)

SQL - (Structured Query Language)

NoSQL - (not only SQL)

ЦОД - центр обробки даних

AWS - Amazon Web Services

PMM - Percona Monitoring and Management

ВСТУП

Актуальність дослідження. Дослідження реляційних баз даних як відмовостійкого компонента високонавантаженого бізнесу є дуже актуальною і важливою для сучасного IT-середовища. Дослідженням баз даних приділяли увагу багато вітчизняних та закордонних вчених. Серед них варто визначити: Зінов'єва І.С. та Артемчука В.О. , які досліджували сучасні підходи до подальшої еволюції концепції баз даних [1], Березький О.М., який займався розробленням реляційної бази даних інтелектуальної системи [2], Мулеса О.Ю., визначала методики вивчення реляційних без даних [3], Брацький,В.О., та М'якшило, О.М. досліджували особливості застосування реляційних і нереляційних баз даних на прикладі SQL Server та MongoDB [4], Тафт Р. вивчав стійкі бази даних [5], Ли В. визначав великомасштабний і прагматичний набір даних тексту в SQL [6], Танг П. виявляв SQL-ін'єкції на основі штучної нейронної мережі [7].

Зростання обсягів даних - це одна з ключових викликів, з якими стикаються сучасні організації. За останні роки обсяги даних експоненційно зросли завдяки цифровій трансформації та зростаючій кількості джерел, які генерують дані. Це означає, що бізнеси та організації мають справлятися із надзвичайно великими обсягами інформації, яка включає в себе структуровані дані, такі як бази даних, а також напівструктуровані та неструктуровані дані, наприклад, тексти, фотографії, відео та сенсорні дані від пристроїв IoT. Це викликає потребу в ефективному зберіганні, обробці та аналізі цих даних.

Мета дослідження полягає у розробка інструментарію та програми з автоматизації щодо прийняття рішень з вибору бази даних.

Завдання дослідження. Для досягнення поставленої мети в роботі сформульовані і вирішені такі завдання:

1. Розглянути реляційні бази даних як відмовостійкий компонент високонавантаженого бізнесу;

2. Визначити шкала факторів впливу на бази даних;

3. Встановити хронологію виникнення та розвитку реляційних баз даних;

4. Визначити відмінності реляційних та нереляційних баз даних;

5. Розкрити поняття реплікації баз даних, як важливого аспекту у забезпеченні доступності, надійності та швидкодії систем обробки даних;

6. Дослідити існуючи підходи до збільшення продуктивності і доступності баз даних;

7. Дослідити технічні рішення налаштувань роботи баз даних;

8. Встановити фактори впливу на вибір типу бази даних;

9. Розробити математичний інструментарій прийняття рішення про вибір бази даних;

10. Розробити алгоритм проведення навантажувального тестування та визначення максимальної кількості запитів до сервера MySQL;

11. Проаналізувати ризик вибору реляційної бази;

12. Розробити програму аналізу баз даних за допомогою мови програмування C# на платформі .NET.

Об'єкт дослідження: стратегії розподіленого збереження даних в реляційних базах даних та їх вплив на відмовостійкість.

Предмет дослідження: реляційні та нереляційні бази даних як компоненти високонавантаженого бізнесу.

Методи дослідження. Для виконання поставлених завдань були використані методи літературного аналізу: огляд літератури, критичний аналіз, емпіричні методи: експеримент, аналіз даних, системне та математичне моделювання, експертної оцінки.

Наукова новизна полягає в удосконаленні інструментарію прийняття рішення про вибір бази даних в залежності від різних факторів впливу.

Практичне значення роботи. Результати роботи, отримані в ході дослідження, можуть застосовуватися при виборі бази даних як для високонавантаженого бізнеса, так і для сучасного IT-середовища.

Особистий внесок автора:

1. Наукові результати роботи отримані автором самостійно.

2. Вибір методів досліджень і технологій реалізації;

3. Розробка математичного інструментарію прийняття рішення про вибір бази даних;

4. Розробка теоретичної частини роботи, в якій досліджені і систематизовані знання про реляційні бази даних як відмовостійкого компоненту високонавантаженого бізнесу;

5. Оцінка отриманих результатів.

Структура і обсяг роботи. Робота складається з вступу, трьох розділів і висновків. Містить 94 сторінки, в тому числі 79 сторінок тексту основної частини з 33 рисунками, списку використаних джерел з 70 найменуваннями на 6 сторінках, 4 додатка на 9 сторінках.

РОЗДІЛ 1
ПРИНЦИПИ РОБОТИ РЕЛЯЦІЙНИХ БАЗ ДАНИХ

1.1. Реляційні бази даних як основна технологія зберігання даних

Реляційні бази даних є однією з технологій, що використовуються для зберігання структурованих даних, і вони можуть бути частиною більшої архітектури для роботи з різними типами даних. На Рис. 1.1 узагальнимо важливість дослідження реляційних баз даних як відмовостійкого компонента високонавантаженого бізнесу для сучасного IT-середовища.

Зростання обсягів даних

- Завдяки цифровій трансформації більше бізнесів збирають, обробляють і зберігають великі обсяги даних. Реляційні бази даних залишаються однією з основних технологій для зберігання даних.

Високонавантаженість

- Бізнес-системи, особливо в інтернеті, повинні бути доступними та працездатними в будь-який час. Реляційні бази даних вимагають високої доступності і відмовостійкості, особливо при високому навантаженні.

Важливість даних

- Дані є одним із найцінніших ресурсів для бізнесу. Вони використовуються для прийняття рішень, аналітики та взаємодії з клієнтами. Надійність та цілісність даних мають величезне значення.

Розвиток технологій

- Реляційні бази даних не стоять на місці і постійно розвиваються. Вони отримують нові функції та можливості, щоб відповідати сучасним вимогам.

Архітектурна відмовостійкість

- Для бізнес-систем важливо розглядати реляційні бази даних як один із компонентів архітектури, який може забезпечити відмовостійкість та високу доступність системи.

Рис. 1.1. Важливість дослідження реляційних баз даних як відмовостійкого компонента високонавантаженого бізнесу для сучасного IT-середовища

Важливо розглядати проблеми масштабування, продуктивності та відмовостійкості реляційних баз даних в умовах великого обсягу даних, оскільки вони відіграють важливу роль у забезпеченні доступності та надійності даних для бізнесу.

Високонавантаженість є ключовою характеристикою сучасних бізнес-систем. Ця концепція вказує на здатність системи або додатку ефективно обробляти і відповідати на велику кількість запитів або транзакцій в реальному часі. Існує декілька ключових аспектів високонавантаженості, які важливо розглядати. По-перше, вона впливає на користувацький досвід, оскільки користувачі очікують, що додатки будуть реагувати швидко, навіть при великому навантаженні. По-друге, доступність грає важливу роль, оскільки багато бізнес-систем повинні бути доступними 24/7, і відмова може призвести до серйозних фінансових втрат. Також важлива скалабельність, оскільки система повинна зростати разом зі збільшенням обсягів даних та користувацького трафіку. Нарешті, високонавантаженість може створювати ризики для безпеки та цілісності даних, тому важливо розробити відповідні заходи для їх захисту при високому навантаженні.

Важливість даних в сучасному бізнес-середовищі не може бути переоцінена. Дані є одним з найцінніших ресурсів для організацій у різних галузях. Вони відіграють ключову роль у прийнятті рішень, взаємодії з клієнтами, оптимізації операцій, прогнозуванні та здобутті конкурентної переваги. Тому збереження та обробка даних стали невід'ємною частиною будь-якої успішної компанії. У цьому контексті реляційні бази даних відіграють ключову роль у забезпеченні доступності, цілісності та ефективного використання даних для підтримки бізнес-процесів.

Розвиток технологій є суттєвим аспектом, тому, що сучасні технології швидко розвиваються і змінюють підходи до зберігання та обробки даних. Хмарні технології надають гнучкість та масштабованість для зберігання та обробки великих обсягів даних. З ростом IoT з'являється більше сенсорних даних для обробки. Технології штучного інтелекту та машинного навчання

використовуються для аналізу даних та прийняття рішень. Зростає значення захисту даних, оскільки кіберзагрози стають більш складними. Мікросервісна архітектура впливає на архітектурні рішення для забезпечення високої доступності та масштабованості систем.

Архітектурна відмовостійкість є ключовою характеристикою систем, які працюють в умовах високого навантаження та вимагають надійності. Це означає, що система може продовжувати працювати, навіть якщо виникають проблеми або відмови в окремих її компонентах. Вона враховує можливість відмови апаратного чи програмного забезпечення та надає механізми для забезпечення неперервності роботи системи.

Архітектурна відмовостійкість може включати в себе такі аспекти, як реплікація даних для забезпечення резервних копій, автоматичне виявлення та відновлення відмов, розподілені архітектури для масштабування та ізоляції помилок, а також використання різних центрів обробки даних для забезпечення доступності в разі відмови одного з них. Забезпечення архітектурної відмовостійкості є критичним завданням для систем, які працюють у високонавантажених бізнес-сценаріях, де недоступність може призвести до значних фінансових втрат та втрати довіри клієнтів.

В Табл.1.1 наведемо фактори впливу на бази даних як відмовостійкого компонента високонавантаженого бізнесу за 2-бальною шкалою, де:

2 - висока важливість фактору;

1 - низька важливість фактору.

Таблиця 1.1

Фактори впливу на бази даних як відмовостійкого компонента високонавантаженого бізнесу

№ з/п	Фактор впливу	Оцінка
1	2	3
1	Збереження даних	1

Продовження таблиці 1.1

1	2	3
2	Неперервність роботи	1
3	Втрати при відмові	1
4	Доступність для клієнтів	1
5	Репутаційні ризики	1
6	Витрати на відновлення	2
7	Скасування послуг	2
8	Конкурентна перевага	1

Таблиця 1.1 відображає оцінку важливості різних факторів відмовостійкості для високонавантаженого бізнесу. Більш важливі фактори отримують вищий бал, тоді як менш важливі - нижчий. Така шкала з факторами може бути використана для визначення пріоритетів у впровадженні заходів з підвищення відмовостійкості для бізнесу.

1.2. Бази даних, як фундаментальний інструмент високонавантаженого бізнесу

База даних є фундаментальним інструментом в сучасному інформаційному світі, який дозволяє зберігати, організовувати та отримувати доступ до великих обсягів даних. Це концептуальне утворення, яке дозволяє зберігати різноманітну інформацію і взаємодіяти з нею, надаючи можливість здійснювати аналіз, приймати рішення та підтримувати бізнес-процеси. Бази даних є невід'ємною частиною роботи у сферах бізнесу, науки та технологій, і вони постійно розвиваються, адаптуючись до зростаючих потреб у зберіганні та обробці даних. Вони дозволяють організаціям зберігати та використовувати інформацію з максимальною ефективністю та надійністю, що робить їх незамінним інструментом в цифровому віці.

На Рис.1.2 зазначимо хронологію виникнення та розвитку реляційних баз даних:

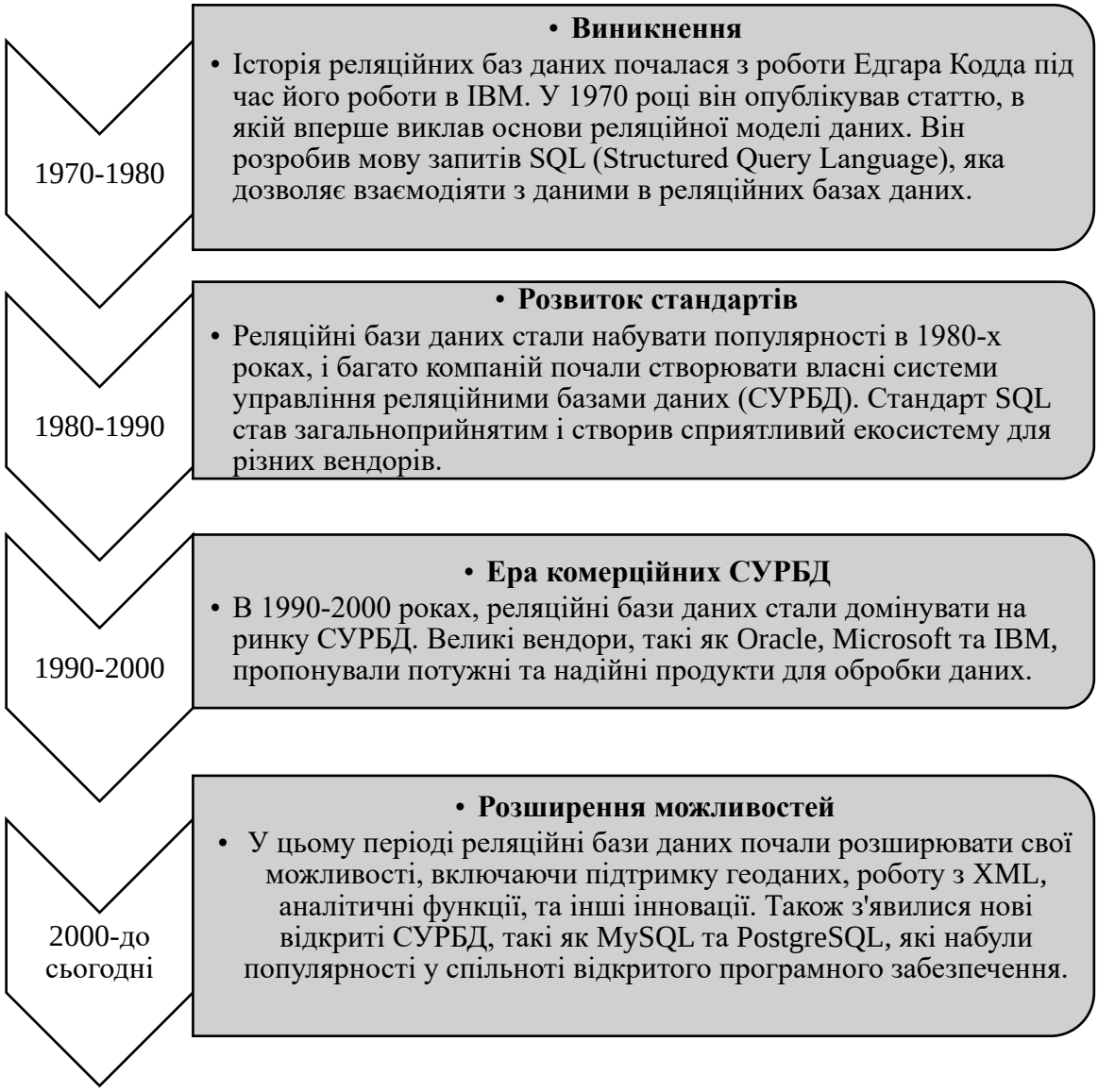

Рис.1.2. Хронологія виникнення та розвитку реляційних баз даних

Реляційні бази даних (РБД) мають довгу і цікаву історію, що розпочалася у 1970-х роках. Спроби створити системи для зберігання та організації даних попереднього покоління були недостатньо ефективними, і справедливою мірою інженери та вчені шукали новий підхід.

Едгар Кодд, британський інженер та вчений-інформатик, відзначається як винахідник реляційної моделі даних, що визначила основи для розробки сучасних реляційних баз даних. В 1970 році він опублікував статтю, в якій вперше презентував цю модель, встановивши принципи організації даних у вигляді таблиць і рядків [8]. Кодд також створив мову запитів SQL, яка стала

стандартом для роботи з реляційними базами даних. Його теоретичні дослідження, такі як нормалізація даних, сприяли покращенню ефективності та цілісності даних у реляційних системах. Внесок Едгара Кодда має велике значення для історії реляційних баз даних і сучасних інформаційних технологій. Сьогодні реляційні бази даних залишаються основою багатьох додатків та систем, а їх історія є свідченням про важливий розвиток технологій у сфері управління даними. Реляційні бази даних (РБД) і нереляціонні бази даних (NoSQL) відрізняються за кількома ключовими характеристиками (Табл.1.2):

Таблиця 1.2

Відмінності реляційних та нереляційних баз даних

Ключова характеристика	Реляційні бази даних	Нереляційні бази даних
1	2	3
Модель даних	Дані організовані у вигляді таблиць (реляцій), де інформація представлена у вигляді рядків і стовпців. Для доступу до даних використовується мова запитів SQL.	Дані можуть бути представлені у різних форматах, таких як дерева, ключ-значення, документи, графи тощо. Це дає більшу гнучкість у зберіганні даних, особливо в нереляційних структурах.
Схема даних	Вимагається фіксована схема даних, де визначені типи даних і структура таблиць.	Бази даних можуть бути безсхемними або мати динамічну схему, що дозволяє додавати або змінювати поля без необхідності переробки всіх даних.
Масштабованість	Зазвичай мають обмежену можливість горизонтального масштабування (розширення на більше обладнання), що може призвести до обмежень у роботі з великими обсягами даних та високою навантаженістю.	Більш схильні до горизонтального масштабування, що робить їх ефективними для роботи з великими та розподіленими даними.

Продовження таблиці 1.2

1	2	3
Транзакції	Бази даних підтримують транзакції з гарантією ACID (Atomicity, Consistency, Isolation, Durability), що забезпечує цілісність даних.	Деякі бази даних можуть підтримувати транзакції, але не завжди з гарантією ACID. Деякі бази системи пропонують більше флексібільний підхід до консистентності даних.
Застосування	Бази даних добре підходять для ситуацій, де схема даних стабільна і не піддається частим змінам. Вони ідеально підходять для транзакційних систем та бізнес-застосунків.	Бази даних часто використовуються для великих обсягів даних, які можуть змінюватися, а також для задач, де потрібна висока продуктивність та можливість масштабування.

Тобто, обираючи між РБД і NoSQL, важливо враховувати вимоги бізнес-проекту, типу даних, що обробляєтеся, та потреби у масштабованості та гнучкості системи.

Вибір бази даних завжди відбувається виходячи з усіма відомої CAP теореми, яка дає загальне уявлення фундаменту зберігання даних. Зазначимо три основні аспекти теореми CAP на Рис. 1.3:

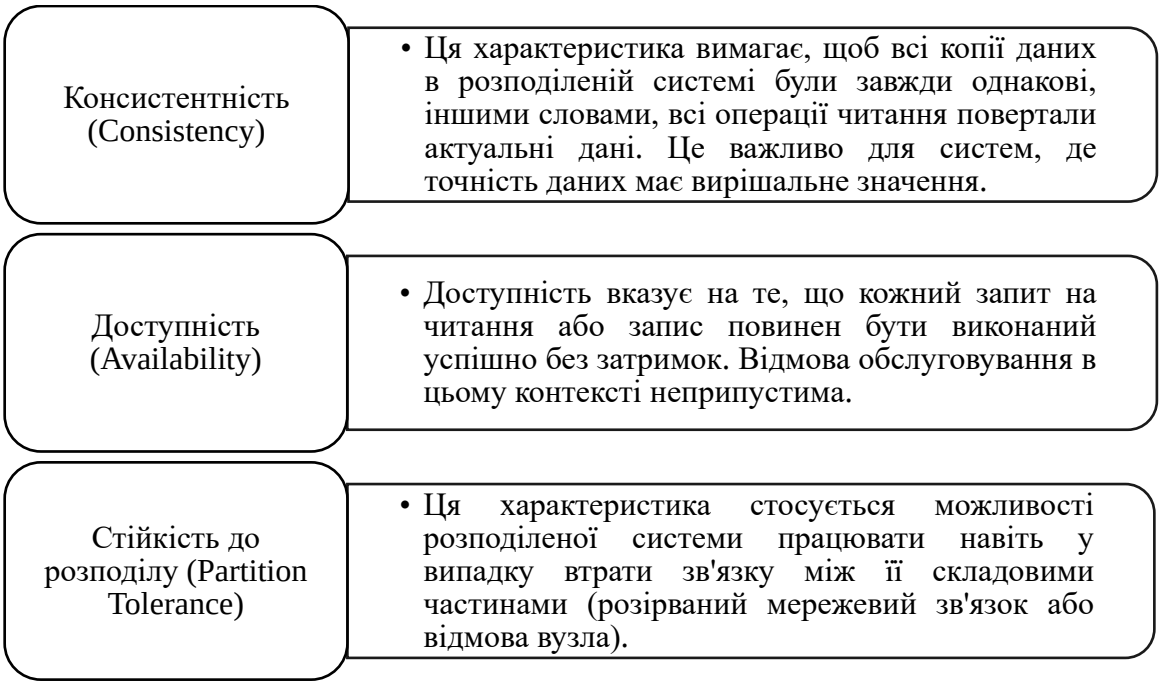

Рис. 1.3. Три основні аспекти теореми CAP

В розподілених системах існує невід'ємна напруга між трьома основними аспектами: Співробітництвом (Consistency), Доступністю (Availability) і Терпимістю до Помилок (Partition Tolerance). Теорема CAP, запропонована Еріком Брюером у 2000 році, формалізує цю напругу і стверджує, що в розподіленій системі можна забезпечити одночасно не більше двох з цих трьох характеристик [9].

Теорема CAP визначає, що у розподіленій системі можна забезпечити одночасно лише дві з цих характеристик, а третю доведеться жертвувати. Вибір між цими характеристиками залежить від конкретних потреб та вимог додатку чи системи. Вибір між характеристиками зазначено на Рис.1.4:

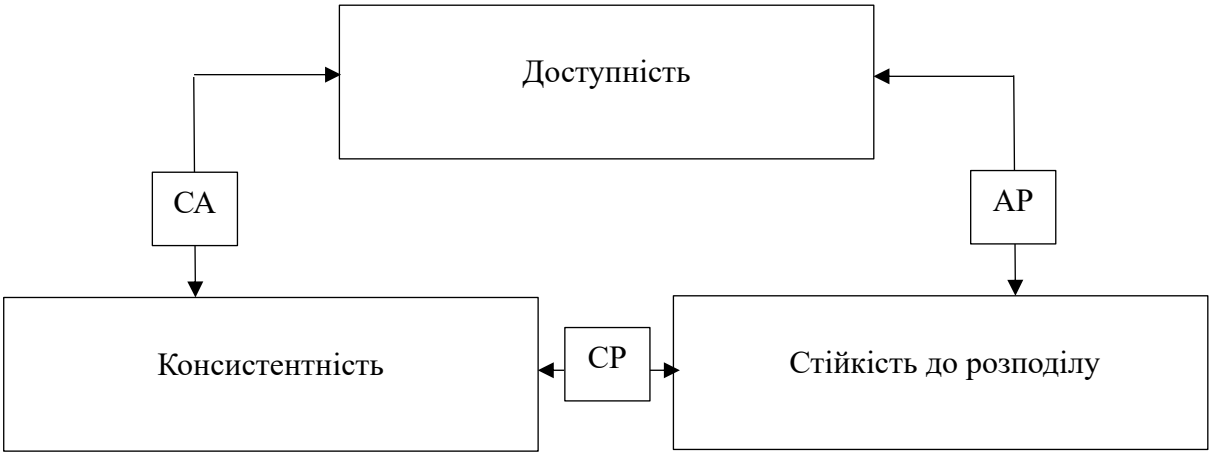

Рис. 1.4. Взаємозв’язки між характеристиками за теоремою CAP

В Україні існують численні високонавантажені бізнеси у різних галузях, які активно використовують інформаційні технології для підтримки та розвитку своєї діяльності. Наведемо перелік таких компаній на Рис.1.5.

Розетка - один із найбільших онлайн-роздрібних торгових майданчиків в Україні, який пропонує широкий асортимент товарів від електроніки до побутової техніки. Компанія регулярно вдосконалює свою інформаційну систему для оптимізації обробки замовлень, відстеження інвентарю та забезпечення високої доступності свого веб-сайту для клієнтів у всі часи.

ПриватБанк - найбільший комерційний банк в Україні, який активно використовує технології для свого фінансового обслуговування та дистанційних

банківських послуг. Великий обсяг транзакцій, доступ до банківських рахунків онлайн та захист від кібератак є важливими аспектами їхнього бізнесу.

Рис. 1.5. Компанії з високонавантаженими бізнесами

Grammarly - це компанія, яка розробляє інноваційний програмний продукт для правописної та граматичної перевірки текстів. Вона відома своєю широкою глобальною аудиторією та високою завантаженістю, оскільки мільйони користувачів в усьому світі використовують їхній сервіс щодня.

WOG - одна з найбільших українських мереж заправних станцій. Вони використовують інформаційні системи для керування постачанням пального, ведення обліку продажів та взаємодії з клієнтами через програми лояльності.

TELEMART.UA - це інтернет-магазин комп'ютерної техніки та електроніки в Україні. Вони спеціалізуються на продажу ноутбуків, смартфонів, комп'ютерних аксесуарів та іншої електроніки. Telemart.ua активно використовує інформаційні технології для обробки замовлень, взаємодії з клієнтами та підтримки широкого асортименту товарів на своєму веб-сайті.

Ці компанії є прикладами успішного високонавантаженого бізнесу в Україні, які впроваджують та використовують сучасні інформаційні технології для забезпечення ефективності та конкурентоспроможності у своїх галузях.

Розглянемо існуючи бази даних за їх типом та використанням у Таблиці 1.3:

Таблиця 1.3

Типи баз даних та їх використання

Назва бази даних	Тип	Використання
1	2	3
MySQL	Реляційна	Веб-додатки, електронна комерція, блоги
MongoDB	Нереляційна	Зберігання та обробка неструктурованих даних
PostgreSQL	Реляційна	Геоінформаційні системи, аналітика
Cassandra	Нереляційна	Інтернет великих обсягів даних, соцмережі
SQLite	Реляційна	Мобільні додатки, вбудовані системи
Redis	Нереляційна	Кешування, сесії, швидкі дані доступу
Oracle	Реляційна	Великі корпоративні системи, фінанси
Neo4j	Нереляційна	Робота з графовими структурами, соцмережі

Ця таблиця надає загальний огляд різних баз даних, вказуючи їхній тип (реляційна або нереляційна) та типові області застосування.

Реплікація баз даних - це важливий аспект у забезпеченні доступності, надійності та швидкодії систем обробки даних. Деякі бази даних мають специфічні характеристики та підходи до реплікації, які дозволяють їм оптимально працювати у розподілених середовищах. Ось кілька типів баз даних, які добре піддаються реплікації (Рис. 1.6, 1.7, 1.8, 1.9):

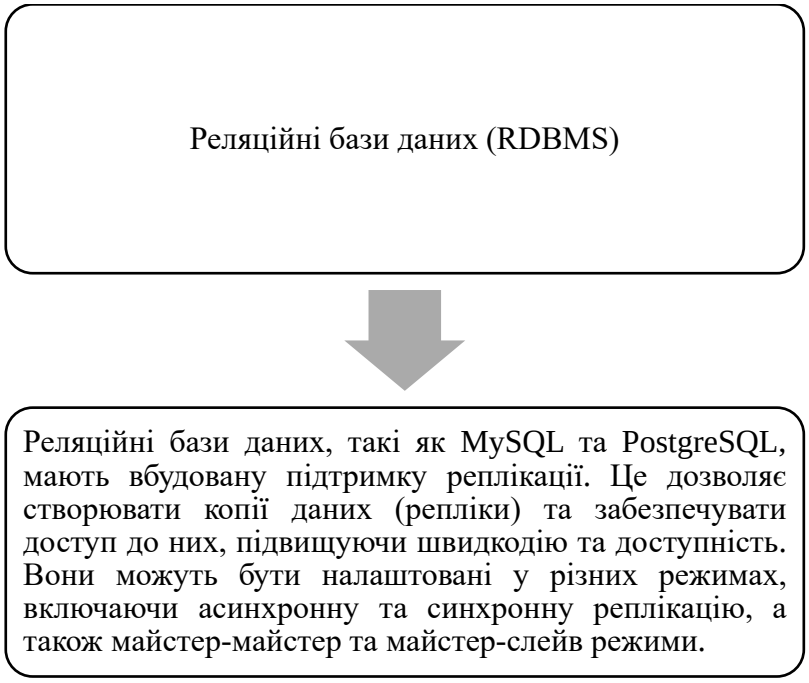

Рис.1.6. Реляційні бази даних (RDBMS), які підлягають реплікації

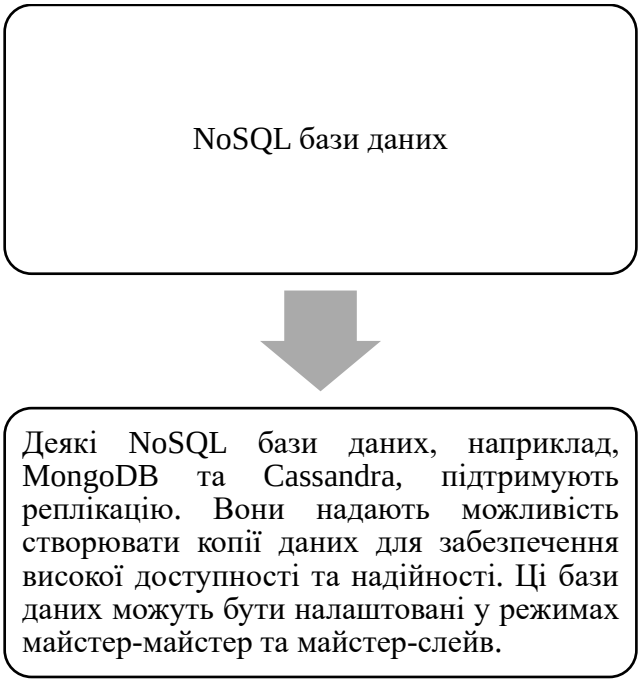

Рис.1.7. NoSQL бази даних, які підлягають реплікації

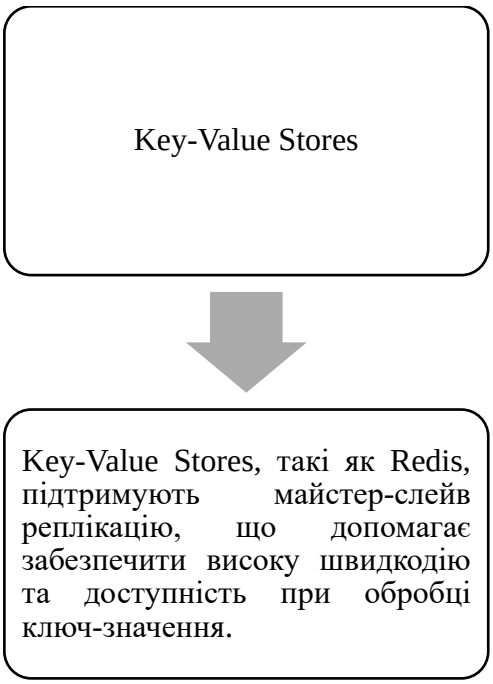

Рис.1.8. Key-Value Stores, які підлягають реплікації

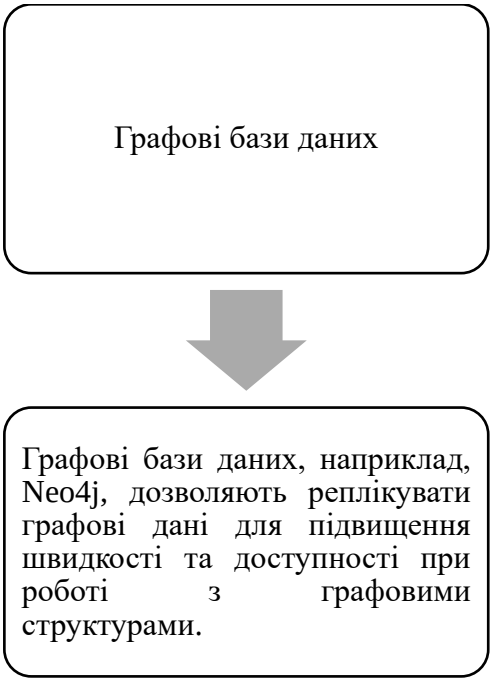

Рис.1.9. Графові бази даних, які підлягають реплікації

Але, вибір конкретної бази даних для реплікації повинен враховувати потреби та вимоги проекту, а також враховувати правильну конфігурацію, управління конфліктами та моніторинг для забезпечення успішної реплікації та високої надійності системи. Наведемо на Рис. 1.10 алгоритм роботи реплікації:

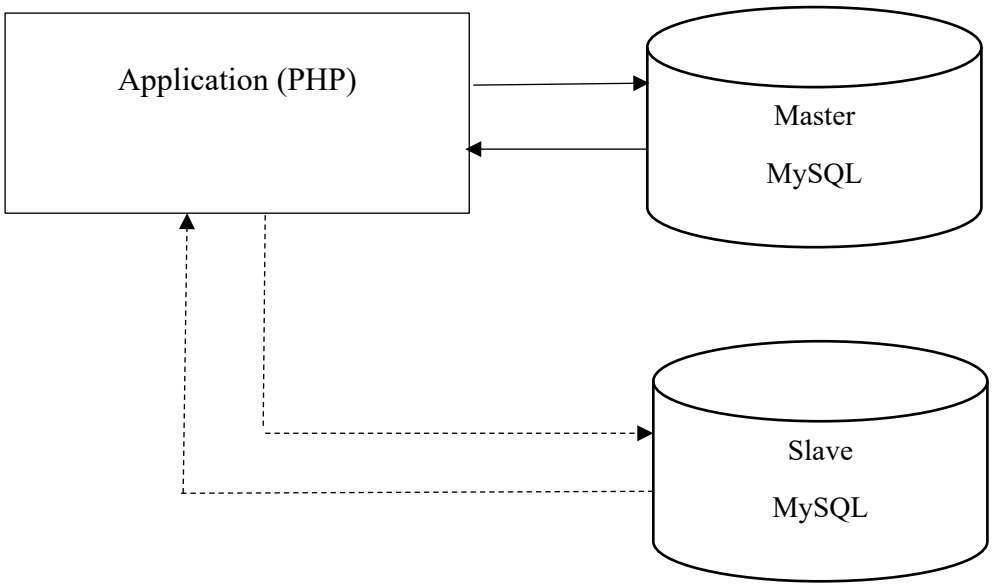

Рис. 1.10. Алгоритм роботи реплікації

Алгоритм роботи реплікації у реляційних базах даних демонструє наступну послідовність дій: Спочатку необхідно налаштувати параметри реплікації. Це включає в себе визначення, які таблиці або дані будуть репліковані, тип реплікації (наприклад, майстер-майстер, майстер-раб) та встановлення параметрів безпеки та надійності. Початковий набір даних передається з майстер-бази до репліки. Цей процес може бути здійснений через резервне копіювання та відновлення або спеціальними механізмами реплікації.

Після ініціалізації репліка бере участь у неперервному процесі реплікації. Кожна зміна, що відбувається на майстер-базі (додавання, оновлення, видалення записів), записується в спеціальний журнал або лог реплікації. Регулярно (за визначеними інтервалами), репліка перевіряє цей журнал реплікації та виконує всі зміни, які ще не були застосовані до своєї бази даних. У випадках, коли та сама зміна відбувається одночасно на майстері та на репліці, може виникнути конфлікт. Існують різні стратегії вирішення конфліктів, включаючи пріоритет майстра чи репліки.

Важливо встановити системи моніторингу для відстеження статусу реплікації та вчасного виявлення проблем. Регулярна підтримка є також важливою частиною процесу. В разі виникнення серйозних проблем, таких як

втрата даних чи пошкодження репліки, важливо мати систему резервного копіювання та відновлення, щоб відновити працездатність системи. Слід систематично аналізувати продуктивність реплікації та при необхідності вдосконалювати її параметри для оптимізації роботи системи.

Цей алгоритм надає загальний огляд того, як працює реплікація в реляційних базах даних. Але, конкретні кроки та налаштування можуть відрізнятися в залежності від конкретної реалізації та використовуваних технологій.

1.3. Підходи до збільшення продуктивності і доступності баз даних

Дослідженнями продуктивності розподілених відмовостійких інформаційних систем та нереляційних сховищ даних, а також аналізу впливу параметрів узгодженості даних на швидкодію та пропускну здатність займалися такі українські вчені, як Карпенко А. С., Тарасюк О.М., Горбенко А.В. [10].

Результатами дослідження стали кількісні показники впливу налаштувань узгодженості на продуктивність баз даних під час різних робочих навантаженнях, в умовах, коли всі репліки розташовані в одному центрі обробки даних (ЦОД), або ж географічно розподілені по різним ЦОД.

Варіанти забезпечення суворої узгодженості в NOSQL досліджував Белоус Р., Нікітін В. [11]. Результатом є опис теорії CAP, щодо існування трьох варіантів забезпечення суворої узгодженості.

Розглянемо різні підходи до збільшення продуктивності і доступності баз даних. Горизонтальне та вертикальне масштабування - це два різних підходи до збільшення продуктивності і доступності системи. Зазначимо основні їх відмінності у Табл.1.4:

Таблиця 1.4

Порівняння підходів до збільшення продуктивності і доступності баз даних

Відмінності	
Вертикальне масштабування (Vertical Scaling)	
Визначення	Використання
Вертикальне масштабування полягає у збільшенні ресурсів (процесора, пам'яті, диска) на існуючому сервері. Це означає покращення апаратного обладнання одного сервера.	Вертикальне масштабування часто використовується, коли система зіштовхується з обмеженнями щодо обсягу ресурсів (наприклад, висока централізована обробка даних або одного додатку).
Горизонтальне масштабування (Horizontal Scaling)	
Визначення	Використання
Горизонтальне масштабування включає в себе додавання нових серверів до існуючої інфраструктури, зазвичай у розподіленому середовищі. Це означає розподіл завдань між багатьма серверами.	Горизонтальне масштабування є відмінним варіантом для систем, де обсяги роботи можуть збільшуватися, ідея полягає у додаванні нових серверів, коли це потрібно, щоб забезпечити більшу продуктивність та доступність.

Головна відмінність між ними полягає в тому, як вони реалізовані. Вертикальне масштабування вимагає апгрейда апаратного обладнання на існуючих серверах, тоді як горизонтальне масштабування вимагає додавання нових серверів для розподіленої обробки завдань. Обидва підходи мають свої переваги і недоліки і можуть бути використані в залежності від конкретних потреб і обмежень проекту.

Зазначимо в Табл.1.5 які типи баз даних зазвичай краще масштабуються горизонтально або вертикально:

Таблиця 1.5

Аналіз баз даних щодо масштабування

Тип бази даних	Горизонтальне масштабування	Вертикальне масштабування
Реляційні бази даних	Важко масштабується	Зазвичай масштабується
NoSQL бази даних	Зазвичай масштабується	Зазвичай масштабується
Key-Value Stores	Зазвичай масштабується	Зазвичай масштабується
Графові бази даних	Зазвичай масштабується	Зазвичай масштабується

Ця таблиця надає загальний огляд тенденцій щодо масштабування різних типів баз даних. Однак важливо враховувати, що ефективність масштабування також залежить від конкретної реалізації бази даних, а також від потреб і обмежень конкретного проекту.

1.4. Висновки до першого розділу

Дослідження реляційних баз даних як відмовостійкого компонента високонавантаженого бізнесу є дуже актуальною і важливою для сучасного IT-середовища. Зростання обсягів даних - це одна з ключових викликів, з якими стикаються сучасні організації. Це викликає потребу в ефективному зберіганні, обробці та аналізі цих даних. Реляційні бази даних є однією з технологій, що використовуються для зберігання структурованих даних, і вони можуть бути частиною більшої архітектури для роботи з різними типами даних. Важливо розглядати проблеми масштабування, продуктивності та відмовостійкості реляційних баз даних в умовах великого обсягу даних, оскільки вони відіграють важливу роль у забезпеченні доступності та надійності даних для бізнесу.

Визначена шкала факторів впливу на бази даних як відмовостійкого компонента високонавантаженого бізнесу. Така шкала з факторами може бути використана для визначення пріоритетів у впровадженні заходів з підвищення відмовостійкості для бізнесу.

База даних є фундаментальним інструментом в сучасному інформаційному світі, який дозволяє зберігати, організовувати та отримувати доступ до великих обсягів даних. Це концептуальне утворення, яке дозволяє зберігати різноманітну інформацію і взаємодіяти з нею, надаючи можливість здійснювати аналіз, приймати рішення та підтримувати бізнес-процеси. Встановлено хронологію виникнення та розвитку реляційних баз даних.

Визначені відмінності реляційних та нереляційних баз даних за такими класифікаційними ознаками як: модель даних, схема даних, масштабованість, транзакції, застосування. Зроблено огляд існуючи бази даних за їх типом та областю застосування. Тобто, обираючи між базами даних, важливо враховувати вимоги бізнес-проекту, типу даних, що обробляєтеся, та потреби у масштабованості та гнучкості системи. Вибір бази даних завжди відбувається виходячи з CAP теореми, яка дає загальне уявлення фундаменту зберігання даних.

Розкрито поняття реплікації баз даних, як важливого аспекту у забезпеченні доступності, надійності та швидкодії систем обробки даних. Через те, що деякі бази даних мають специфічні характеристики та підходи до реплікації, які дозволяють їм оптимально працювати у розподілених середовищах, зроблено аналіз баз даних, які підлягають реплікації. Описано алгоритм роботи реплікації у реляційних базах даних.

Досліджено підходи до збільшення продуктивності і доступності баз даних. Зроблено порівняння підходів до збільшення продуктивності і доступності баз даних, які мають свої переваги і недоліки і можуть бути використані в залежності від конкретних потреб і обмежень проекту. Виконано загальний огляд тенденцій щодо масштабування різних типів баз даних.

РОЗДІЛ 2
ТЕХНОЛОГІЯ ВИБОРУ БАЗИ ДАНИХ ЯК ВІДМОВОСТІЙКОГО КОМПОНЕНТУ ВИСОКОНАВАНТАЖЕНОГО БІЗНЕСУ

2.1. Технічні рішення налаштувань роботи баз даних

2.1.1. Встановлення MySQL на сервер

Представимо комбіновану інструкцію щодо встановлення MySQL на сервері Ubuntu разом з налаштуванням користувача, включаючи команди (Таблиця 2.1):

Таблиця 2.1

Комбіновану інструкцію встановлення MySQL на сервері Ubuntu

Послідовність	Алгоритм дій
1	2
1. Оновлення системи	Необхідно переконатися, що сервер має актуальні оновлення. Відкриваємо термінал і виконуємо такі команди: *sudo apt update* *sudo apt upgrade*
2. Встановлення MySQL	Встановимо сервер MySQL. Виконуємо наступну команду: *sudo apt install mysql-server* Під час встановлення необхідно встановити пароль для користувача root MySQL. Необхідно ввести пароль і підтвердити його.
3. Запуск MySQL	MySQL повинен автоматично запуститися після встановлення. Проте, якщо цього не сталося, необхідно виконати: *sudo systemctl start mysql*
4. Налаштування безпеки MySQL	Для підвищення безпеки MySQL виконуємо таку команду: *sudo mysql_secure_installation* Необхідно ввести пароль користувача root MySQL. Потім відповісти на наступні запити:

Продовження таблиці 2.1

	- Встановити пароль для користувача root? (Y/n): Відповідь «Y» і ввести пароль. - Видалити анонімних користувачів? (Y/n): Рекомендується відповісти «Y». - Заборонити вхід користувачу root зовні? (Y/n): Рекомендується відповісти «Y». - Видалити тестову базу даних? (Y/n): Рекомендується відповісти «Y». - Перезавантажити таблиці привілеїв? (Y/n): Рекомендується відповісти «Y».
5. Створення нового користувача MySQL	Входимо до інтерфейсу MySQL за допомогою команди *mysql* та облікових даних адміністратора (root): *sudo mysql -u root -p* Потім створюємо нового користувача, замінивши «username» та «password» на бажане ім'я користувача та пароль: CREATE USER *«username'@'localhost»* IDENTIFIED BY *«password»;* Тут *«localhost»* вказує на те, що користувач може увійти лише з локального комп'ютера. Якщо потрібно дозволити вхід з віддалених машин, то змінюємо *«localhost»* на *«%»*.
6. Надання користувачу прав	Потрібно надати користувачу права на конкретну базу даних. Наприклад, щоб надати повні права на всі бази даних: GRANT ALL PRIVILEGES ON *.* TO *«username'@'localhost»;* Завершує налаштування прав виконання команди: FLUSH PRIVILEGES
7. Вихід із інтерфейсу MySQL	Після завершення налаштування потрібно вийди з інтерфейсу MySQL: *exit*

Отже, за допомогою комбінованої інструкції може бути встановлено MySQL і є окремий користувач з правами доступу до бази даних на сервері Ubuntu.

2.1.2. Налаштування реляційної бази на приклади MySQL

Для налаштування реляційної бази на приклади MySQL наведемо приклади можливих значень для деяких змінних у конфігураційному файлі MySQL (Рис.2.1):

innodb_buffer_pool_size	• Наприклад, для сервера з великою кількістю доступної пам'яті, можливо встановити значення *«innodb_buffer_pool_size»* на 70% від загального обсягу доступної фізичної пам'яті, наприклад *«innodb_buffer_pool_size = 6G».*
key_buffer_size	• Для таблиць MyISAM можливо встановити значення *«key_buffer_size»*, наприклад, *«key_buffer_size = 512M»*.
max_connections	• Залежно від очікуваної кількості одночасних з'єднань, можливо встановити *«max_connections».* Наприклад, *«max_connections = 200».*
log_error	• Для вказання розташування журналу помилок, можливо встановити *«log_error»,* наприклад, *«log_error = /var/log/mysql/error.log».*
skip-name-resolve	• Якщо не потрібно виконувати DNS-розрішення для з'єднань, можна встановити *«skip-name-resolve»*.
secure_file_priv	• Необхідно встановити *"secure_file_priv",* щоб обмежити місце, звідки користувачі можуть завантажувати файли
long_query_time	• Потрібно налаштувати *"long_query_time"* для виявлення повільних запитів. Наприклад, *"long_query_time = 2"*
innodb_log_file_size	• Для InnoDB таблиць можливо встановити розмір журнальних файлів, наприклад, *"innodb_log_file_size = 256M"*

Рис. 2.1. Значення змінних у конфігураційному файлі MySQL

Значення цих параметрів повинні відповідати потребам сервера, обсягу пам'яті та вимогам до продуктивності. Важливо слідкувати за змінами в продуктивності після внесення змін та виправити їх в разі потреби.

2.1.3. Налаштування реплікації на приклади MySQL

Реплікація в MySQL дозволяє створити копію даних з одного сервера (майстера) на інший (слейв). Наведемо алгоритм дій для налаштування реплікації (Рис.2.2):

Рис.2.2. Алгоритм реплікації в MySQL

Необхідно переконатися, що MySQL встановлено на обох серверах (майстері і слейві) і працює. Налаштовуємо конфігурацію на майстер-сервері:

- відкриваємо конфігураційний файл MySQL на майстер-сервері. Зазвичай це файл «my.cnf» або «my.ini»;
- додаємо або розкоментуємо наступні параметри, де server-id - унікальний ідентифікатор майстера, log_bin - шлях до журналу бінарних журналів, і binlog_do_db - назва бази даних, яку будемо реплікувати.

server-id = 1

log_bin = /var/log/mysql/mysql-bin.log

binlog_do_db = база_даних

- перезапускаємо MySQL після внесення цих змін.

Налаштовуємо конфігурацію на сервері-слейві:

- відкриваємо конфігураційний файл MySQL на сервері-слейві і додаємо наступні параметри, де server-id - унікальний ідентифікатор слейва, replicate-do-db - назва бази даних, яку будемо реплікувати з майстера:

server-id = 2

replicate-do-db = база_даних

- перезапускаємо MySQL після внесення цих змін.

Для завантаження резервної копії на сервер-слейв можемо використовувати команду «*mysqldum»p* або інші засоби для створення резервної копії вашої бази даних на майстер-сервері. Потім завантажуємо цю резервну копію на сервер-слейв і відновлюємо її.

Для запуску реплікації на сервері-слейві виконуємо на сервері-слейві таку команду:

sql

CHANGE MASTER TO

MASTER_HOST = «IP_майстера»,

MASTER_USER = «користувач_реплікації»,

MASTER_PASSWORD = «пароль_реплікації»,

MASTER_LOG_FILE = «mysql-bin.000001»

Виконуємое на сервері-слейві команду:

sql

START SLAVE

Тепер реплікація повинна почати працювати. Можливо перевірити стан реплікації, використовуючи команду «SHOW SLAVE STATUS\G;».

Це базовий опис налаштування реплікації в MySQL, але потрібно бути обережно з налаштуванням і збереженням паролів реплікації у безпеці, оскільки вони дуже важливі для безперебійної роботи реплікації.

2.1.4. Встановлення MongoDb

Представимо комбіновану інструкцію щодо встановлення MongoDB на сервері Ubuntu з налаштуванням користувача (Рис.2.3):

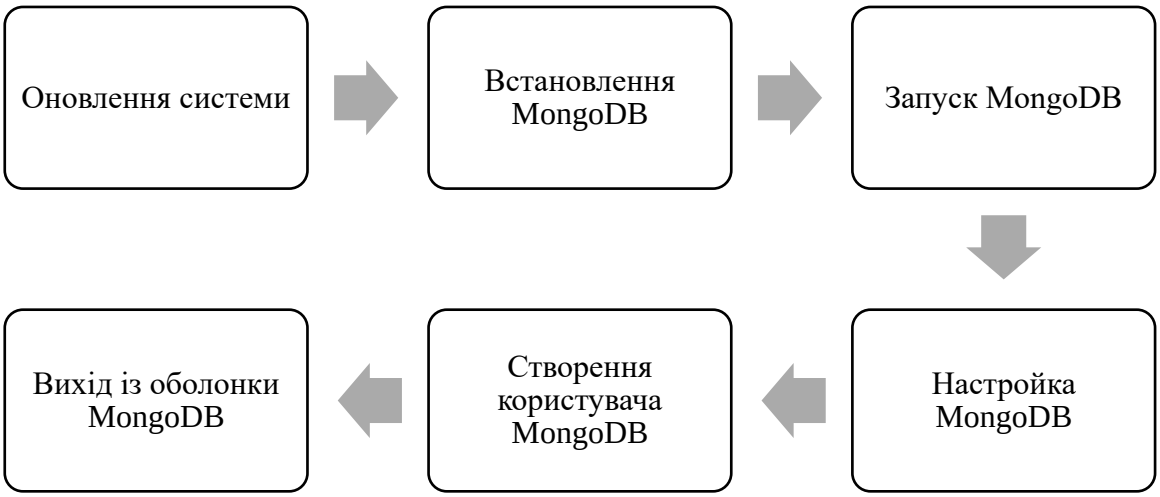

Рис. 2.3 Комбінована інструкція щодо встановлення MongoDB на сервері Ubuntu

Для оновлення системи потрібно переконатися, що сервер має актуальні оновлення. Відкриваємо термінал і виконуємо такі команди:

sudo apt update

sudo apt upgrade

Для встановлення MongoDB виконуємо наступні команди:

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 68818C72E52529D4

echo «deb [arch=amd64] https://repo.mongodb.org/apt/ubuntu bionic/mongodb-org/4.4 multiverse» | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list

sudo apt-get update

sudo apt-get install -y mongodb-org

MongoDB має запуститися автоматично після встановлення. Однак, якщо цього не сталося, виконуємо:

sudo systemctl start mongod

MongoDB доступна на порту 27017 за замовчуванням. Для більшої безпеки встановлюємо фаєрвол або зв'язуємось з іншими методами безпеки, щоб обмежити доступ до сервера MongoDB.

Для створення користувача для доступу до бази даних MongoDB входимо до оболонки MongoDB з командного рядка:

mongo

Потім переходимо до адміністративної бази даних та створюємо нового користувача. Змінюємо *«username»*, *«password»*, та *«database»* на бажані значення:

use admin

db.createUser({ user: «username», pwd: «password», roles: [«readWrite», «dbAdmin»] })

Закривши оболонку MongoDB, повертаємось до командного рядка.

Вихід із оболонки MongoDB робимо командою:

exit

Тепер у нас встановлена MongoDB і є окремий користувач з правами доступу до бази даних на вашому сервері Ubuntu.

2.1.5. Налаштування реляційної бази на приклади MongoDb

Наведемо приклади конкретних значень для деяких параметрів конфігурації MongoDB (Рис.2.4).

Ці значення можуть змінюватися в залежності від конкретних вимог та середовища вашого проекту. Важливо дбайливо налаштовувати параметри, щоб забезпечити безпеку та ефективну роботу MongoDB на dedicated сервері.

storage.engine	• Обираємо тип сховища. Наприклад, для WiredTiger (за замовчуванням) можемо встановити: *"storage.engine = wiredTiger"*
dbPath	• Зазначаємо шлях до каталогу бази даних. Наприклад, *"dbPath = /var/lib/mongodb"*
logPath	• Встановлюємо шлях до журнальних файлів. Наприклад, *"logPath = /var/log/mongodb/mongod.log"*
bindIp	• Зазнаємо IP-адреси та порти, на яких слухає сервер MongoDB. Наприклад, *"bindIp = 127.0.0.1,192.168.0.100*"
port	• Встановлюємо порт сервера MongoDB. Наприклад, *"port = 27017"*
auth	• Вмикаємо або вимкаємо автентифікацію. Наприклад, "auth = true"
sslMode	• Вибераємо режим шифрування (наприклад, *"requireSSL*" для вимоги до SSL). Наприклад, *"sslMode = requireSSL"*
clusterAuthMode	• Встановлюємо метод автентифікації для кластерів Наприклад, *"clusterAuthMode = keyFile"*
setParameter	• Додаткові параметри можуть бути встановлені за потреби, наприклад, *"setParameter = enableLocalhostAuthBypass=0"*
journal	• Включаємо журналювання (journaling) для забезпечення надійності даних. Наприклад, "journal = true".
replication	• Налаштовуємо параметри реплікації, якщо використовуємо реплікаційний кластер

Рис. 2.4. Значення для параметрів конфігурації MongoDB

2.2. Фактори впливу на вибір між реляційним та NoSQL сховищем в залежності від навантаження проекту

Вирішення проблем горизонтального масштабування реляційних баз даних є складним процесом, який супроводжується численними викликами. Однією з основних проблем є спільний доступ до даних. При роботі з

розподіленими даними декілька серверів повинні здійснювати доступ до одних і тих самих даних, що може призвести до конфліктів і несумісності даних.

Іншою важливою аспект є складність запитів. Пересилка запитів між різними фрагментами бази даних може призвести до збільшення складності запитів і сповільнення їх виконання. Крім того, управління транзакціями в умовах розподіленої системи також стає складнішим завданням, і це може вплинути на консистентність даних.

Однією з ключових проблем є також продуктивність. На жаль, горизонтальне масштабування не завжди призводить до лінійного збільшення продуктивності, і деякі операції можуть бути повільними через необхідність співпрацювати з розподіленими даними.

Для вирішення цих проблем і досягнення ефективного горизонтального масштабування реляційних баз даних важливо використовувати правильні стратегії і технології, які відповідають конкретним вимогам проекту. Також слід звертати увагу на синхронізацію, резервне копіювання і налаштування системи, оскільки це також є важливими аспектами у розподіленому середовищі.

Завданням кваліфікаційної роботи є розробка універсальної формули, яка б точно визначала вибір між реляційним та NoSQL сховищем в залежності від навантаження проекту. Вибір між цими двома типами систем зазвичай ґрунтується на кількох факторах, і він є завданням, яке вимагає аналізу конкретних вимог і обставин проекту. Ось деякі фактори, які можуть вплинути на вибір (Рис.2.1).

Тому рекомендації щодо вибору бази даних зазвичай ґрунтуються на вивченні конкретного проекту, його вимог і мети. Важливо провести аналіз, визначити пріоритети і вибрати технологію, яка найкраще задовольнить потреби проекту.

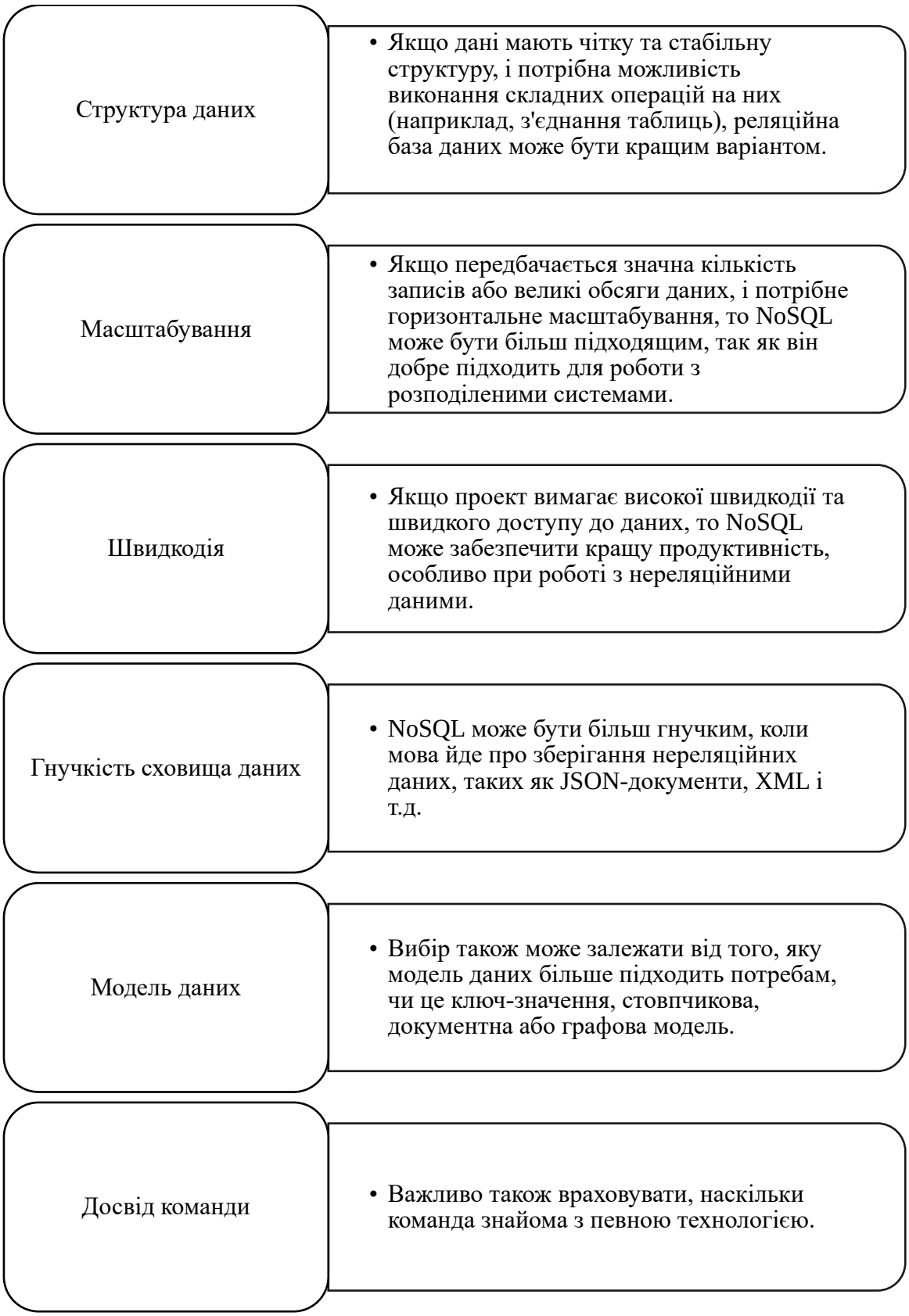

Рис. 2.5. Фактори впливу на вибір між реляційним та NoSQL сховищем

Складено за: [12]

2.3. Математичний інструментарій прийняття рішень про вибір бази даних

У математиці лінійна та нелінійна формули відрізняються за наявністю лінійних або нелінійних операцій над змінними. Лінійна формула включає лише лінійні операції над змінними, такі як додавання, віднімання, множення на константи. Загальний вигляд лінійної формули виглядає так:

$$y = a \times x + b, \text{ де} \quad (2.1),$$

де x - змінна, a і b - константи.

Лінійні формули описують прості залежності та лінійні відношення між змінними. Нелінійна формула включає нелінійні операції, такі як піднесення до ступеня, взяття логарифмів, множення змінних між собою та інші складні математичні операції. Загальний вигляд нелінійної формули може бути досить складним і не має такого простого вигляду, як у лінійних формул.

Вибір між лінійними та нелінійними формулами залежить від конкретного контексту та завдання. Лінійні формули часто використовуються для моделювання простих відношень та залежностей, тоді як для більш складних або нелінійних зв'язків потрібно використовувати нелінійні формули.

Лінійні формули можуть мати певні переваги у деяких областях, оскільки їхні властивості досить добре вивчені, і їхні рішення можуть бути обчислені швидко та надійно. Однак у багатьох випадках потрібно використовувати нелінійні формули для точного опису складних явищ та залежностей. Вибір формули повинен базуватися на конкретних вимогах завдання та реальних даних.

Програмісти регулярно використовують математичні формули у своїй роботі, і частота використання може значно відрізнятися в залежності від конкретної галузі програмування і завдань, які вони вирішують. Математика є важливою для розв'язання складних завдань, створення алгоритмів, аналізу даних та моделювання різних явищ у різних галузях програмування. Як приклад, це використання лямбда-формули, яка представляє собою анонімний блок коду,

який можна використовувати як значення. Лямбди дозволяють створювати короткі, однорядкові функції без необхідності визначати їх окремо.

Лямбда-вирази можна використовувати у виразах, LINQ-запитах, делегатах та в подіях (Рис.2.6). Вони спрощують код і дозволяють писати більш компактні та експресивні вирази [13].

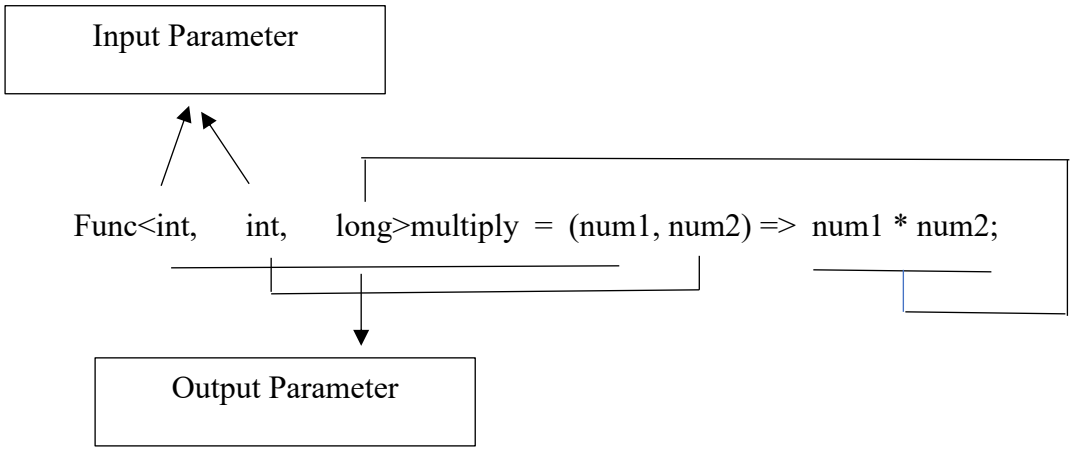

Рис. 2.6. Лямбда-формула мовою C#, яка є наріжним камінням програмування у світі .NET платформи

При виборі бази даних програмісти використовують поняття «селективність». Селективність в базі даних - це показник, який вказує на те, скільки унікальних значень міститься в стовпці порівняно з загальною кількістю записів. Чим вище селективність, тим більше унікальних значень міститься в стовпці [14]. Це важливо при прийнятті рішення про створення індексу, оскільки індекси допомагають прискорити пошук, але вони також займають додаткове місце в базі даних.

Наведемо приклади розрахунку селективності з баз даних високонавантаженого бізнесу:

У стовпці з повними іменами клієнтів (наприклад, «Прізвище Ім'я По батькові»):

Всього записів: 1000

Унікальних імен: 300

$$\text{Селективність} = \frac{\text{Кількість унікальних імен}}{\text{Всього записів}} = \frac{300}{1000} = 0{,}3$$

В цьому прикладі селективність дорівнює 0,3, що означає, що в цьому стовпці багато унікальних імен. У цьому випадку створення індексу на стовпці з повними іменами може бути корисним для прискорення пошуку за іменами клієнтів.

У стовпці з поштовими адресами:

Всього записів: 10000

Унікальних адрес: 50

$$\text{Селективність} = \frac{\text{Кількість унікальних адресів}}{\text{Всього записів}} = \frac{50}{10000} = 0{,}005$$

В цьому прикладі селективність дорівнює 0,005, що свідчить про те, що поштові адреси в цьому стовпці практично унікальні. У цьому випадку створення індексу може бути менш ефективним, оскільки він буде займати багато місця і, можливо, не суттєво прискорить пошук.

Отже, рішення про те, чи створювати індекс, і якого типу, буде залежати від селективності стовпця і вимог до продуктивності запитів.

Селективність грає важливу роль при створенні індексів в базі даних. Вона вказує на те, наскільки унікальними є значення у стовпці порівняно з загальною кількістю записів. Чим вище селективність, тим більш ефективний індекс.

Якщо селективність висока (близько до 1), тобто у стовпці багато унікальних значень, то індекс допоможе ефективно виконувати запити, оскільки він швидко зменшить кількість записів, які потрібно прочитати.

У випадку низької селективності (близько до 0), коли значення практично унікальні, створення індексу може бути менш ефективним, оскільки індекс буде майже так само великим, як сама таблиця.

В Таблиці 2.2 наведемо приклади розрахунку селективності за різними значеннями:

Таблиця 2.2

Приклади розрахунку селективності бази даних

№ запису	Загальна кількість записів	Унікальні значення	Селективність
1	2	3	4
1	1000	300	0,3
2	5000	100	0,02
3	200	50	0,25
4	30000	10	0,0003
5	100	100	1,0
6	1500	30	0,02
7	8000	5000	0,625
8	6000	6000	1,0
9	700	400	0,57
10	40000	200	0,005

Отже, важливо аналізувати селективність перед створенням індексу. Найкращим рішенням може бути створення індексу на стовпцях з високою селективністю, або комбінація індексів для досягнення оптимальної ефективності запитів.

2.4. SELECT та UPDATE запити та навантаження на бази даних

Реплікація в SQL залежить від типу операцій (SELECT та UPDATE) та від самого навантаження [15]. В Таблиці 2.3 зробимо порівняння SELECT та UPDATE запитів:

Таблиця 2.3

Порівняння SELECT та UPDATE запитів

SELECT-запити	UPDATE-запити
1	2

Продовження таблиці 2.3

Легше реплікувати, оскільки зазвичай не змінюють дані у базі даних і, отже, не викликають конфліктів записи між майстром і репліками. SELECT-запити просто зчитують дані з бази даних та передають їх на репліки. Це означає, що вони не вимагають запису до журналу бінарних логів (binary log), який використовується для реплікації даних.	Складніше реплікувати, тому що вони змінюють дані у базі даних. Це може призвести до конфліктів запису, коли одночасно виконуються оновлення на майстрі та репліках. UPDATE-запити мають бути записані до журналу бінарних логів і відтворені на репліках, що потребує додаткових ресурсів.

Вибір того, яке навантаження легше реплікувати, залежить від конкретних вимог та конфігурації системи:

- якщо система виконує багато SELECT-запитів та невелику кількість змінних операцій (наприклад, UPDATE), то реплікація SELECT-запитів буде більш простою та ефективною.

- якщо є великим обсягом операцій, що змінюють, і транзакцій, то управління реплікацією може вимагати додаткових зусиль для забезпечення цілісності даних на всіх репліках.

Отже, реплікація MySQL може бути налаштована для різних цілей, включаючи балансування навантаження, створення резервних копій даних і забезпечення відмовостійкості. Тому вибір того, яке навантаження реплікувати, має відповідати конкретним потребам та бізнес-цілям.

Amazon - це величезна компанія, що надає широкий спектр послуг, включаючи інтернет-торгівлю, хмарні обчислення, потокове відео, музику та багато іншого. Залежно від конкретного сервісу та його призначення, трафік може суттєво відрізнятися. Розглянемо деякі приклади, щоб зрозуміти, якого трафіку у Amazon може бути більше [16]. На Рис. 2.7 наведемо трафік інтернет-торгівлі (Amazon.com):

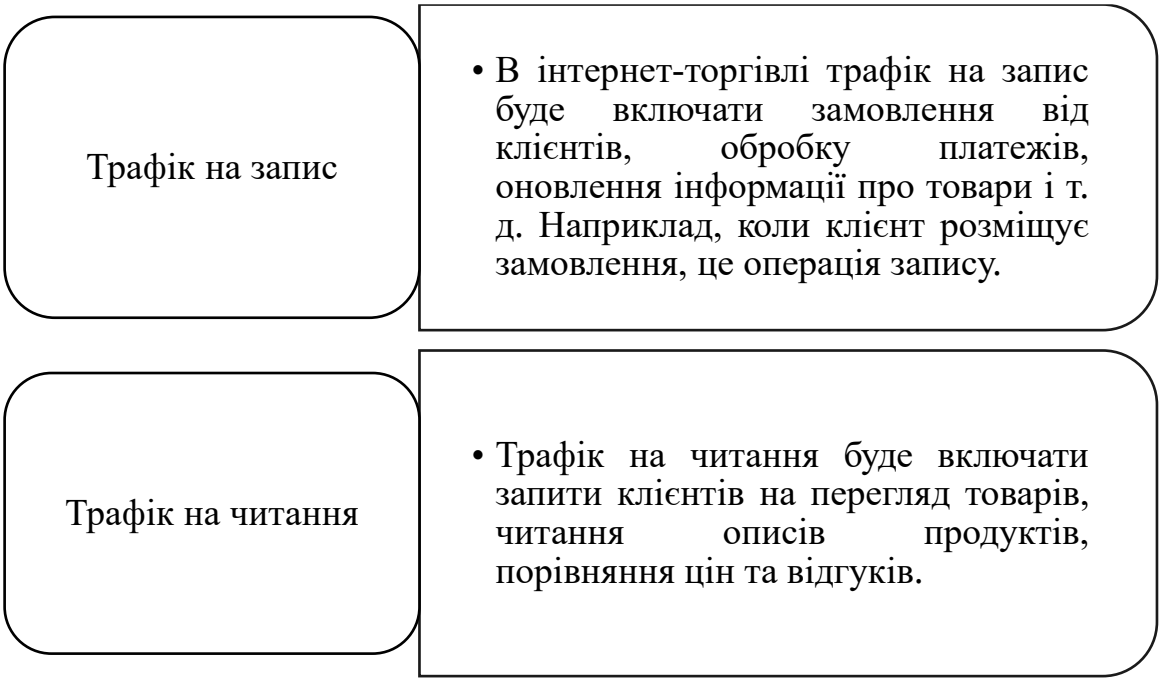

Рис. 2.7. Трафік інтернет-торгівлі (Amazon.com)

Складено за [17]

На Рис. 2.8 наведемо трафік Amazon Web Services (AWS):

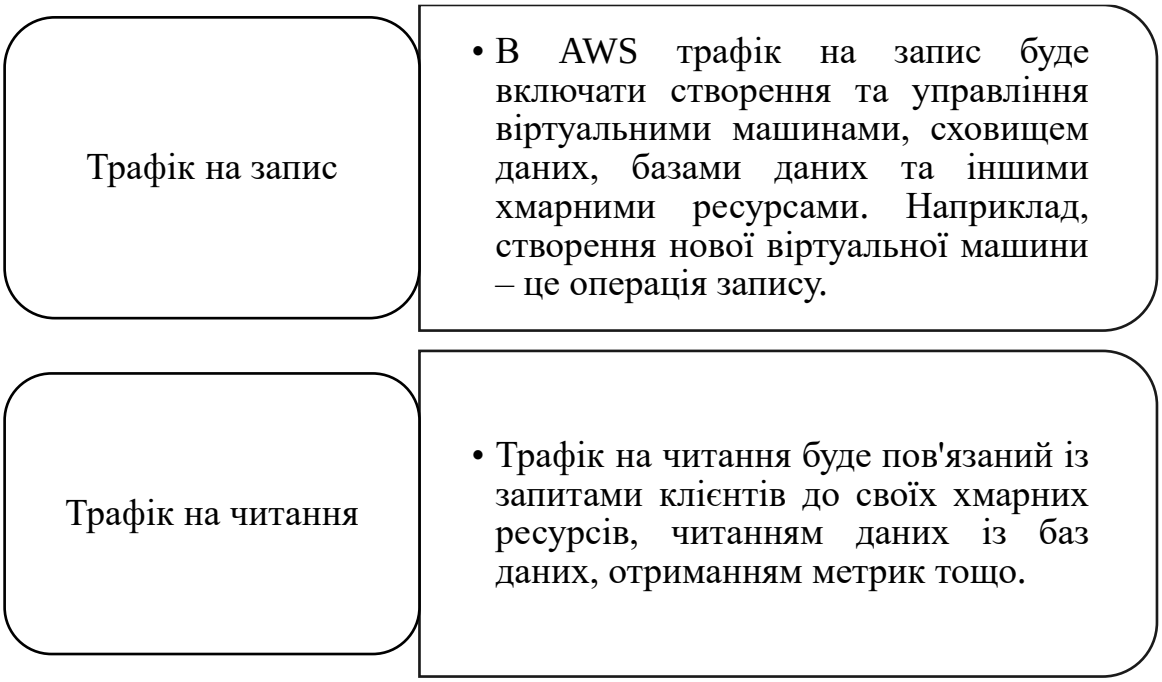

Рис. 2.8. Трафік Amazon Web Services (AWS)

Складено за [18]

На Рис. 2.9 наведемо трафік Amazon Prime Video:

Трафік на запис	• Створення та завантаження нових відео-контенту, оновлення метаданих про фільми та серіали, облік переглядів та оцінок користувачів.
Трафік на читання	• Запити користувачів на перегляд відеоконтенту, навігація за каталогом фільмів, перегляд описів та відгуків.

Рис. 2.9. Трафік Amazon Prime Video

Складено за [19]

Наведемо на Рис. 2.10 навантаження на базу даних Amazon інтернет-торгівля (Amazon.com):

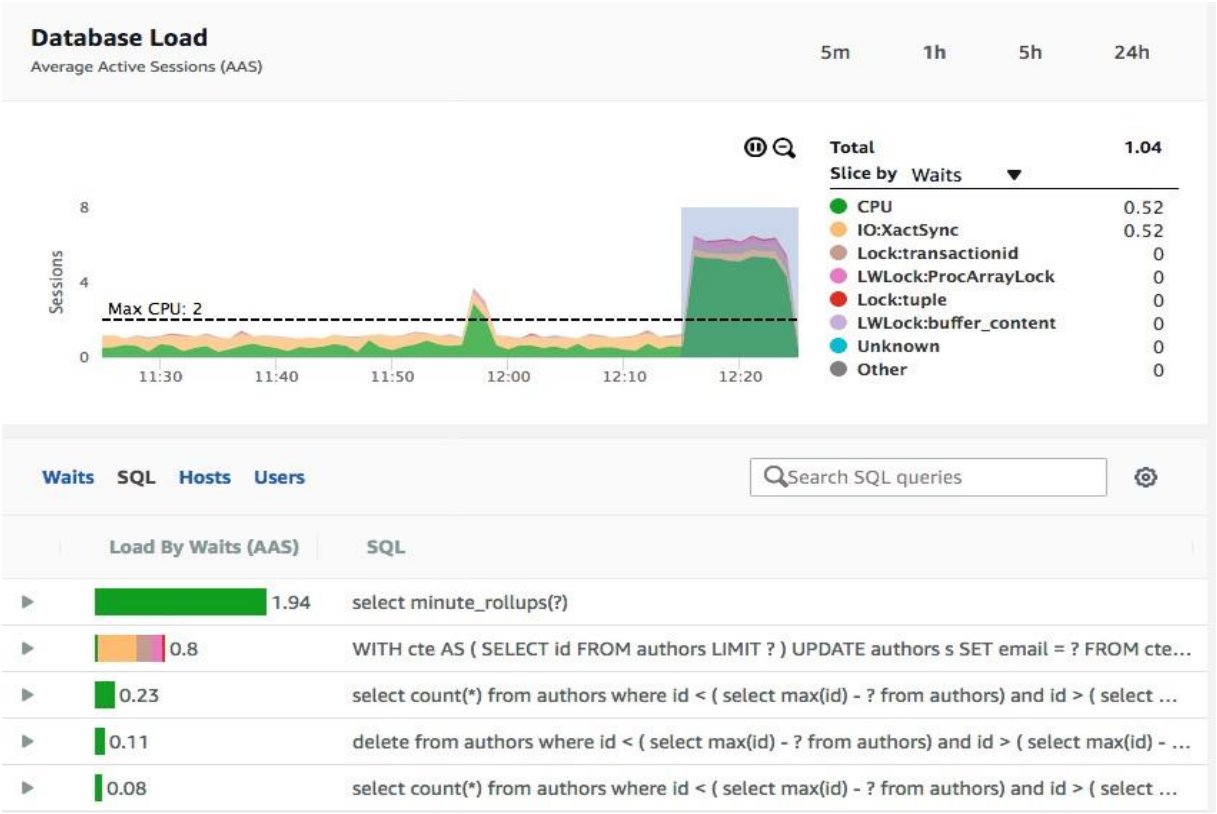

Рис.2.10. Навантаження на базу даних Amazon

Загальна кількість трафіку на читання та запис для Amazon буде залежати від бізнес-потреб та специфіки кожного конкретного сервісу. Наприклад, сервіси AWS можуть мати більш інтенсивний трафік на запис, оскільки клієнти створюють та настроюють свої хмарні ресурси. З іншого боку, інтернет-торгівля Amazon.com матиме інтенсивний трафік на читання, оскільки безліч клієнтів переглядає та купує товари. Але, Amazon як компанія надає різноманітні послуги, і трафік може сильно змінюватись між цими послугами та продуктами. Представимо порівняння навантаження SELECT та UPDATE трафіку в реальному часі (Рис.2.11):

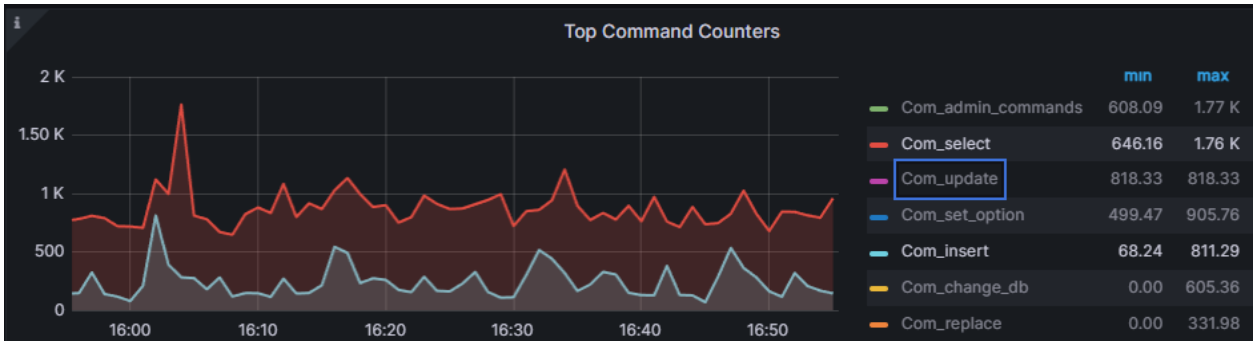

Рис. 2.11. Порівняння навантаження SELECT та UPDATE трафіку в реальному часі

Порівняння навантаження SELECT та UPDATE трафіку в реальному часі надає важливу інформацію про те, які типи операцій в базі даних домінують у конкретний момент часу. Порівняння навантаження SELECT та UPDATE трафіку надає контекст для ефективної роботи з базою даних, допомагає виявляти потреби в оптимізації та реагувати на можливі проблеми.

Отже, навантаження на базу даних MySQL може суттєво змінюватися в залежності від часу доби та дня тижня. Це залежить від безлічі факторів, включаючи активність користувачів, тип вашого веб-додатку або сервісу та географічне розподілення користувачів. Ось як воно може змінюватися (Табл.2.4):

Таблиця 2.4

Зміна навантаження на базу даних MySQL

Зміна навантаження	Характеристика
1	2
Піки активності в робочий час	Зазвичай навантаження на базу даних зростає в робочий час, коли багато користувачів входять у систему. Це може статися вранці та в робочі дні, коли користувачі починають роботу, виконують запити та взаємодіють із додатком.
Знижена активність в нічний час	В нічний час або в вихідні дні, коли більшість користувачів відпочивають або не працюють, активність на сервері може знижуватися. Цей час може використовуватися для виконання резервних копій, оптимізації бази даних та інших завдань обслуговування.
Піки активності в різні дні тижня	Для різних типів додатків активність може відрізнятися залежно від дня тижня. Наприклад, електронна комерція часто має піки активності на вихідних, а корпоративні додатки - в будні.
Глобальні відмінності	Якщо додаток обслуговує користувачів з різних часових поясів або країн, навантаження може змінюватися відповідно до цих глобальних різниць.

Продовження таблиці 2.4

Сезонні варіації	Навантаження також може змінюватися в залежності від сезону та свят. Наприклад, у період свят або розпродажів магазини можуть бачити значний приріст активності.

Для ефективного управління такою динамікою навантаження важливо моніторити сервер, масштабувати його за необхідності, оптимізувати запити та базу даних та планувати обслуговування в періоди з низькою активністю. Ефективне управління часом доби та днями тижня також може допомогти заощадити ресурси та забезпечити стабільну продуктивність сервера.

Ціни на облікові рішення баз даних можуть змінюватися вночі, коли навантаження зазвичай менше. Різниця в навантаження між прайм-тайм та вночі (Рис.2.12):

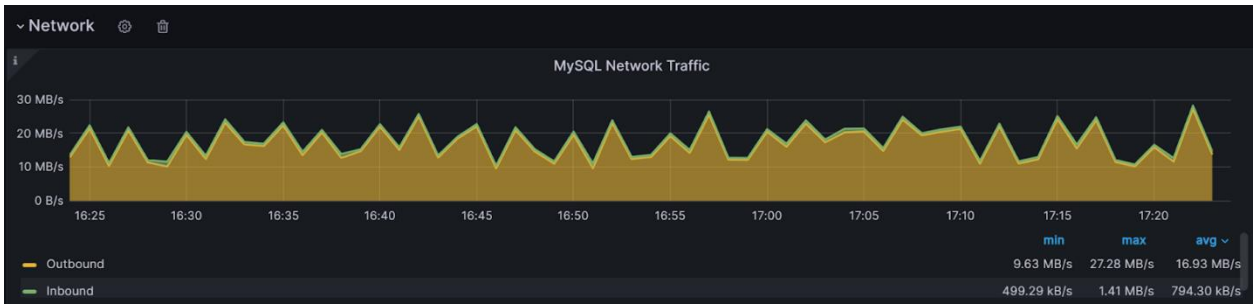

Рис. 2.12. Різниця в навантаження між прайм-тайм та вночі

Це може бути зв'язано з кількістю користувачів, які використовують базу даних в цей час, і попитом на неї. В нічний час, коли менше користувачів активно взаємодіють із системою, попит на ресурси бази даних може бути нижчим. Це може призвести до тимчасового зниження цін на облікові рішення баз даних, оскільки менше користувачів конкурують за ці ресурси. Це може бути вигідно для підприємств, які прагнуть економити на оплатах за облікові послуги. Однак важливо враховувати, що ціни на облікові рішення баз даних можуть

змінюватися в залежності від багатьох факторів, включаючи загальний попит на ресурси та політику постачальника послуг баз даних.

Тому, якщо важливо ефективно використовувати ресурси та знижувати витрати, потрібно розглядати можливості встановлення ресурсів баз даних на пік навантаження, і зменшення їх у ночі або в періоди низької активності.

2.5. Кількість запитів проєкту

Середня кількість запитів на секунду на MySQL сервері вказує на кількість SQL-запитів, які сервер може виконати за одну секунду. Це важливий показник продуктивності для баз даних.

Чим вище QPS, тим більша продуктивність сервера. Висока кількість QPS може свідчити про добре налаштований та оптимізований сервер, що може обробляти великі обсяги запитів в найкоротший час. Але, в реальних умовах продуктивність бази даних може бути залежить від багатьох чинників (Рис.2.13):

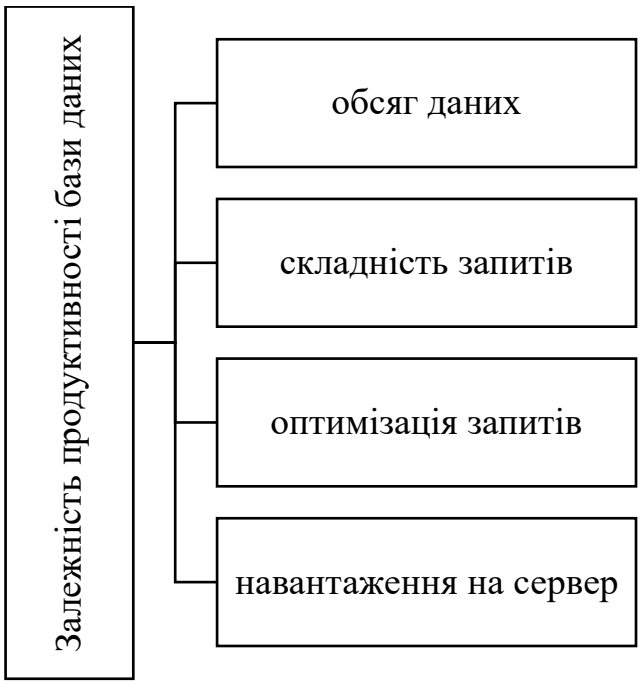

Рис.2.13. Чинники залежності продуктивності бази даних

Також важливо, що показник QPS може коливатися в залежності від часу доби, коли навантаження на базу даних може бути вище (наприклад, під час пікових навантажень) або нижче (у періоди меншої активності).

Отже, середня кількість запитів на секунду є важливим показником продуктивності MySQL сервера, але варто аналізувати його разом із іншими метриками та ураховувати особливості конкретного середовища.

Для відстеження середньої кількості запитів на секунду на MySQL сервері є кілька способів. Можливо використовувати MySQL Workbench, де в розділі «Performance» та «Server Status» можна знайти інформацію про кількість запитів на секунду.

Також можна виконати SQL-запит *SHOW GLOBAL STATUS LIKE «Questions»*;, щоб отримати загальну кількість запитів, виконаних на сервері, а потім розрахувати середню кількість запитів на секунду.

Існують сторонні інструменти моніторингу, такі як Prometheus з експортером MySQL або Percona Monitoring and Management (PMM), які надають докладну інформацію про сервер, включаючи кількість запитів в секунду.

Також можливо налаштувати логування запитів MySQL і аналізувати логи, щоб визначити середню кількість запитів в секунду. Це робиться за допомогою параметра «general_log» у конфігураційному файлі MySQL.

Інший варіант - використовувати графічні інтерфейси сторонніх розробників, такі як Grafana для створення панелей моніторингу з графіками, що відображають кількість запитів на секунду.

Отже, вибір методу залежить від переваг та доступності. Важливо стежити за цією метрикою, особливо в різні часові інтервали, щоб розуміти навантаження на сервер та його зміни.

Команда *SHOW GLOBAL STATUS LIKE «Questions»*; у MySQL надає інформацію про кількість виконаних запитів на сервері. Це включає загальну кількість запитів, включаючи різні типи запитів, такі як SELECT, INSERT, UPDATE та інші. Ця команда повертає два стовпці: «Variable_name» (Назва змінної) і «Value» (значення цієї змінної) (Табл.2.5):

Таблиця 2.5

Використання команди SHOW GLOBAL STATUS LIKE «Questions»; у MySQL

Variable_name	Value
Max_connection_timeout	60
Enable_long_history	1
Internal_buffer_optimization	0
Dedicated_server	0
Max_threads	255

Навантаження на проектах, які працюють в одному часовому поясі та на тих, які працюють в різних часових поясах, може відрізнятися через кілька факторів (Табл.2.6):

Таблиця 2.6

Фактори впливу на навантаження

Фактори	Характеристика
1	2
Активність користувачів	У проектах, які працюють в одному часовому поясі, активність користувачів зазвичай синхронізована з робочим днем в цьому часовому поясі. В проектах з різними часовими поясами активність може змінюватися залежно від часу в кожному поясі, що призводить до більш розподіленого навантаження протягом доби.
Піки навантаження	В проектах з різними часовими поясами можуть виникати «перекриття» годин великої активності, коли користувачі з різних поясів активні одночасно. Це може призвести до піків навантаження, які потребують більшої масштабованості і ресурсів.

Продовження таблиці 2.6

Обробка даних з різних джерел	В проектах з різними часовими поясами може бути необхідно обробляти дані з різних джерел, які працюють в різний час. Це може вплинути на роботу з даними та запитами до бази даних.
Планування обслуговування	У проектах з різними часовими поясами потрібно ретельно планувати час обслуговування і резервне копіювання, оскільки одні користувачі можуть працювати, коли інші відсутні.
Синхронізація інформації	Якщо в проекті існує необхідність синхронізації даних між різними часовими поясами, це може стати додатковим завданням з погляду навантаження на базу даних.

Для успішного управління навантаженням у проектах з різними часовими поясами важливо враховувати ці фактори та вчасно реагувати на піки навантаження, планувати обслуговування та оптимізувати структуру бази даних. Велика роль у цьому також відводиться географічному розподіленню серверів і даних.Вибір між реляційною (SQL) та нереляційною (NoSQL) базами даних для проектів, які працюють в різних часових поясах, залежить від конкретних потреб та характеру проекту. Обидві системи можуть бути використані в таких проектах, але вони мають свої переваги і недоліки (Табл.2.7):

Таблиця 2.7

Переваги та недоліки реляційних (SQL) та нереляційних (NoSQL) баз даних для проектів, які працюють в різних часових поясах

Реляційні бази даних (SQL)	Нереляційні бази даних (NoSQL)
1	2
1. Добре підходять для проектів зі складною структурою даних, яка вимагає детальних зв'язків між таблицями.	1. Добре підходять для проектів, де структура даних може змінюватися динамічно, і коли потрібна гнучкість у роботі з даними.

Продовження таблиці 2.7

2. Забезпечують стандартні можливості для операцій з базами даних, такі як транзакції, цілісність даних та складні запити SQL. 3. Підходять для великих обсягів даних, коли потрібно забезпечити відносну послідовність та синхронізацію між джерелами даних в різних часових поясах.	2. Можуть бути ефективними для швидкості читання та запису даних, що особливо важливо в проектах з великими обсягами навантаження. 3. Забезпечують розподілені можливості та масштабованість, що корисно для глобальних проектів з користувачами у різних часових поясах.

Можливо застосування гібридного підходу, коли використовується як реляційна, так і нереляційна бази даних, в залежності від конкретних вимог та частини проекту. Важливо ретельно проаналізувати потреби проекту, масштабність, структуру даних та вимоги до швидкості перед тим, як приймати остаточне рішення.

2.6. Висновки до другого розділу

Досліджені технічні рішення налаштувань роботи баз даних: представлена комбінована інструкція щодо встановлення MySQL на сервері Ubuntu разом з налаштуванням користувача та командами; для налаштування реляційної бази на прикладі MySQL наведені приклади можливих значень для деяких змінних у конфігураційному файлі MySQL; наведено алгоритм дій для налаштування реплікації, який дозволяє створити копію даних з одного сервера (майстера) на інший (слейв); представлена комбінована інструкцію щодо встановлення MongoDB на сервері Ubuntu з налаштуванням користувача; наведені приклади конкретних значень для деяких параметрів конфігурації MongoDB.

Прийняття рішення щодо вибору бази даних зазвичай ґрунтуються на вивченні конкретного проекту, його вимог і мети. Важливо провести аналіз, визначити пріоритети та обрати технологію, яка найкраще задовольнить потреби

проекту. Встановлені фактори впливу на вибір типу бази даних, які вимагають аналізу конкретних вимог і обставин проекту: структура даних, масштабування, швидкодія, гнучкість сховища даних, модель даних, досвід команди.

Виконано порівняння навантаження SELECT та UPDATE трафіку в реальному часі, що надає важливу інформацію про те, які типи операцій в базі даних домінують у конкретний момент часу. Порівняння навантаження SELECT та UPDATE трафіку надає контекст для ефективної роботи з базою даних, допомагає виявляти потреби в оптимізації та реагувати на можливі проблеми. Встановлено, що навантаження на базу даних MySQL може суттєво змінюватися в залежності від часу доби та дня тижня. Це залежить від безлічі факторів, включаючи активність користувачів, тип веб-додатку або сервісу та географічне розподілення користувачів.

Середня кількість запитів на секунду на MySQL сервері вказує на кількість SQL-запитів, які сервер може виконати за одну секунду. Це важливий показник продуктивності для баз даних. Чим вище QPS, тим більша продуктивність сервера. Висока кількість QPS може свідчити про добре налаштований та оптимізований сервер, що може обробляти великі обсяги запитів в найкоротший час. Визначені чинники, від яких залежить продуктивність бази даних: обсяг даних, складність запитів, оптимізація запитів, навантаження на сервер.

Встановлені фактори впливу на навантаження проектів, які працюють в одному часовому поясі та на тих, які працюють в різних часових поясах: активність користувачів, піки навантаження, обробка даних з різних джерел, планування обслуговування, синхронізація інформації. Для успішного управління навантаженням у проектах з різними часовими поясами важливо враховувати ці фактори та вчасно реагувати на піки навантаження, планувати обслуговування та оптимізувати структуру бази даних.

РОЗДІЛ 3
ОПТИМІЗАЦІЯ ВИБІРУ РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ ЯК ВІДМОВОСТІЙКОГО КОМПОНЕТУ ВИСОКОНАВАНТАЖЕНОГО БІЗНЕСУ

3.1. Розрахунок прийняття рішення про вибір бази даних

Використовуючи фактори впливу запропонуємо комплексну формулу прийняття рішення про вибір бази даних, що має числові та параметричні компоненти. Дана формула допомагає чітко визначити, який тип бази даних вибрати - реляційний чи не реляційний:

$$R = (1 - SEL) \times (SMQPS - QPS) + \frac{QPS}{SMQPS \times (1 - SEL)} \quad (3.1),$$

де:

SEL - (select coefficient) питома вага SELECT запитів серед усіх запитів;

SMQPS - (server max queries per second) максимальна кількість запитів, яка може витримати 1 фізичний сервер проекту;

QPS - (queries per seconds) бажана кількість запитів, яку має тримати проект без лага;

R - (relational coefficient) коефіцієнт, що дає зрозуміти яку базу даних вибрати. Якщо число позитивне, то реляційна база даних буде краще під проект, якщо негативне, значить краще вибирати NoSQL базу даних. Значення за $|10| < 10$, говорять про те, що допустимо обидва варіанти, і немає однозначної відповіді.

Кількість SELECT-запитів може впливати на можливість та ефективність реплікації бази даних у реляційних системах. Реплікація - це процес створення копій даних і їх збереження на різних серверах для покращення доступності, продуктивності та надійності системи [22].

Встановлено, що реплікація добре працює з select навантаженням, оскільки розпаралелює виконання на різних вузлах, але update навантаження не

буде прискорене, тому що всім подібним запитам доведеться виконуватися на всіх вузлах системи.

Математично, ця формула є арифметичним виразом, який обчислює значення R (імовірно, результат) на основі деяких вхідних змінних та констант.

1. (1 - SEL) - це вираз віднімає від 1 питому вагу SELECT реквестів серед всіх запитів. Це представляє обчислення (1 - SEL) в десятковому вигляді.

2. (SMQPS - QPS) - цей вираз обчислює різницю між значеннями максимальної кількості реквестів, які може витримати один фізичний сервер та кількістю запитів на проєкті.

3. (QPS / (SMQPS * (1 - SEL))) - цей вираз обчислює вираз QPS поділити на добуток SMQPS і (1 - SEL).

Загалом, ця формула об'єднує ці вирази і виконує різні операції (додавання, віднімання, множення та ділення) для отримання значення R, яке, як припущено, має специфічне значення або інтерпретацію в контексті, в якому вона використовується.

Коректний розрахунок змінних у формулах є важливим для точності та надійності обчислень. Важливо перевіряти дані на правильність, використовувати відповідні одиниці вимірювання, дотримуватися правильного порядку операцій та правильних правил округлення, уникати ділення на нуль, а також докладно документувати формули. Правильний розрахунок допомагає уникнути помилок та забезпечити точність результатів.

Ця формула є лінійною, оскільки всі операції, які в ній використовуються (додавання, віднімання, множення, ділення), є лінійними операціями. Лінійна формула має вигляд, де кожна змінна має коефіцієнт, і всі змінні підлягають лише лінійним операціям. У цьому випадку, всі змінні (SEL, SMQPS, QPS) з'являються у лінійних операціях або в додаванні/відніманні до та множенні на константи.

Зробимо розрахунок за формулою 3.1 щодо вибору типу бази даних (реляційний чи не реляційний) у Таблиці 3.1:

Таблиця 3.1

Розрахунок вибору між типом бази даних

QPS 1	SEL 2	SMQPS 3	R 4
140	0.9	200	13,00
80	0.1	400	288,22
400	0.7	100	-76,67
100	0.3	800	490,18
600	0.3	700	71,22
20	0.1	500	432,04
40	0.01	600	554,47
150	0.15	100	-40,74
700	0.9	800	18,75
600	0.1	500	-88,67
900	0.2	400	-397,19
1000	0.02	400	-585,45
1000	0.8	400	-107,50
500	0.8	400	-13,75
550	0.3	500	-33,43
550	0.5	500	-22,80
500	0.4	500	1,67
500	0.9	500	10,00
400	0.01	1000	594,40
1000	0.1	100	-798,89
900	0.2	100	-628,75
800	0.1	100	-621,11
700	0.15	700	1,18
1000	0.9	1000	10,00
85	0.4	100	10,42
105	0.2	100	-2,69
95	0.9	100	10,00
5	0.99	100	5,95
600	0.01	100	-488,94
600	0.09	100	-448,41
90	0.9	100	10,00

Наведені розрахунки значення R допоможуть обрати базу даних (реляційний чи не реляційний) (Рис. 3.1):

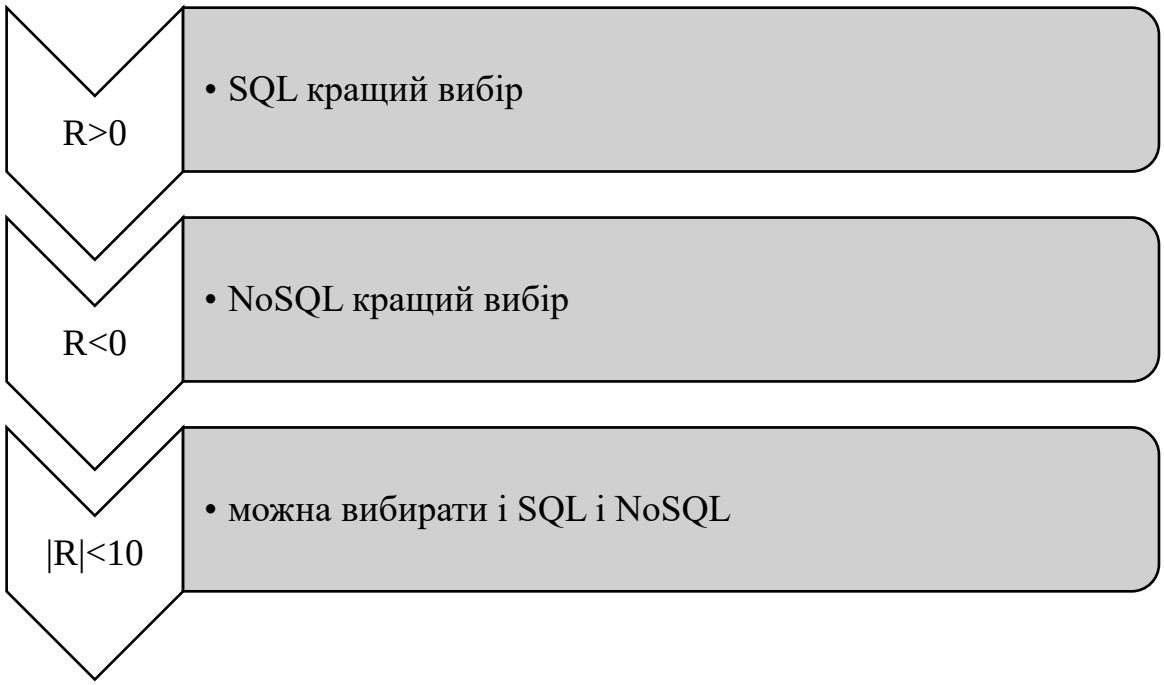

Рис.3.1. Вибір бази даних в залежності від розрахунку R (коефіцієнту, що дає зрозуміти яку базу даних вибрати)

Кількість запитів різного виду до бази даних MySQL в реальному часі вказує на обсяг взаємодії програми з базою даних протягом певного періоду часу (зазвичай в секундах або мілісекундах). Це може бути важливо для моніторингу та оптимізації продуктивності додатків, які використовують базу даних MySQL. На Рис.3.2 показано кількість запитів різного виду до бази даних MySQL в реальному часі:

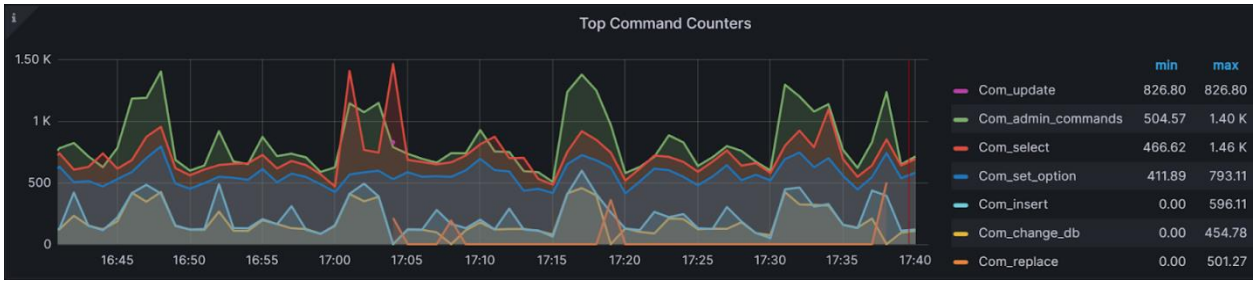

Рис. 3.2. Кількість запитів різного виду до бази даних MySQL в реальному часі [23]

Для врахування різниці в навантаженні в різні часи доби, формулу для прийняття рішення про вибір бази даних, що має числові та параметричні компоненти, буде мати наступний вигляд:

$$R = (1 - SEL) \times (SMQPS - QPS) + (\frac{QPS}{1-SEL}) \times F \qquad (3.2),$$

де:

F - це фактор, який враховує різницю в навантаженні в залежності від часу доби. Причому, цей фактор можливо визначати на основі аналізу навантаження в різні часові періоди.

Наприклад, якщо навантаження зазвичай знижується вночі, то F може бути менше 1, щоб зменшити вагу навантаження в цей період [24].

Таким чином, врахування F фактору допоможе при прийняття рішення про виборі між реляційними та NoSQL базами даних враховувати динаміку навантаження.

Загалом, моніторинг кількості запитів різного виду дозволяє вчасно виявляти та вирішувати можливі проблеми з продуктивністю та надмірним навантаженням на базу даних.

3.2. Моніторинг та налагодження SQL-запитів

Для більш довгострокового моніторингу та збору інформації про запити MySQL, краще використовувати інструменти, такі як Performance Schema. Пропонуємо таблицю «Events_statements_history_long» для аналізу запитів, що виконуються. Щоб розрахувати кількість запитів SELECT від загальної кількості запитів, необхідно виконати SQL-запит наступним чином:

```
SELECT
  SUM(CASE WHEN digest_text LIKE 'SELECT%' THEN 1 ELSE 0 END) AS select_queries,
  COUNT(*) AS total_queries
FROM performance_schema.events_statements_history_long;
```

У цьому запиті використовується таблиця «Events_statements_history_long» із Performance Schema. Спочатку використовуємо функцію SUM з умовою CASE, щоб підрахувати запити, текст яких починається з SELECT, як SELECT-запити. Потім використовуємо COUNT(*) для розрахунку загальної кількості запитів. Результатом будуть два числа: кількість SELECT-запитів та загальна кількість запитів.

Performance Schema надає більш детальну інформацію про запити, і можливо адаптувати цей запит, щоб отримати додаткові відомості про виконання запитів у системі MySQL.

«Events_statements_history_long «- це одна з таблиць у Performance Schema в MySQL, яка надає історичну інформацію про виконувані SQL-запити. Ця таблиця містить записи про запити, які були виконані на сервері MySQL, і надає інформацію про тривалість виконання, використовувані ресурси та текст запиту. Наведемо характеристики таблиці «Events_statements_history_long». У Таблиці 3.2 представимо схему таблиці:

Таблиця 3.2

Схема таблиці «Events_statements_history_long», де зберігається інформація про виконані SQL-запити

Структура	Характеристика
1	2
EVENT_ID	Унікальний ідентифікатор події
END_EVENT_ID	Ідентифікатор завершальної події, пов'язаної з цим запитом
THREAD_ID	Ідентифікатор потоку, який виконав запит
DB	Ім'я бази даних, до якої належить запит
SQL_TEXT	Текст SQL-запиту

Продовження таблиці 3.2

DIGEST_TEXT	Унікальний хеш-код, який представляє текст запиту
CURRENT_SCHEMA	Поточна схема бази даних, у якій виконувався запит
DIGEST	Хеш-код для визначення запиту
DIGEST_HASH	Хеш-код для визначення запиту
FIRST_SEEN	Час першого запиту в форматі «рррр-мм-дд гг:хх:сс»
LAST_SEEN	Час останнього запиту в цьому форматі «рррр-мм-дд гг:хх:сс»

Таблиця «Events_statements_history_long» може бути використана для аналізу продуктивності виконаних SQL-запитів у системі. Можливо визначити, які запити займають найбільше часу або використовують найбільше ресурсів. Також таблиця корисна при профілюванні та оптимізації запитів у MySQL. Щодо термінів зберігання, то старі записи можуть бути автоматично видалені відповідно до налаштувань терміну зберігання даних (наприклад, за допомогою параметра «performance_schema_events_statements_history_long_history_size»).

Для отримання списку найбільш довгих запитів з Таблиці «Events_statements_history_long» можна використовувати SQL-запит, аналогічний наступному:

*SELECT * FROM performance_schema.events_statements_history_long*
ORDER BY TIMER_WAIT DESC;

Цей запит поверне запити, відсортовані за часом очікування (TIMER_WAIT) у порядку зменшення.

Таким чином, таблиця «Events_statements_history_long» - потужний засіб моніторингу та налагодження SQL-запитів у MySQL, і може бути корисною при аналізі продуктивності та оптимізації бази даних.

3.3. Визначення максимальної кількості запитів, які може витримати фізичний сервер проекту

Навантажувальне тестування бази даних MySQL проводиться з метою оцінки продуктивності та надійності сервера в умовах великого навантаження. Навантажувальне тестування допомагає встановити, скільки запитів на секунду сервер може обробляти в оптимальних умовах, це важливо для гарантування, що сервер може витримати очікувані навантаження. Тестування дозволяє виявити слабкі місця в архітектурі або конфігурації бази даних, це можуть бути питання з оптимізації запитів, конфліктів блокування, нестабільності під великим навантаженням і т. д. Навантажувальне тестування дозволяє визначити, чи вистачає ресурсів (пам'яті, CPU, дискового простору) для обробки потенційно великих обсягів даних і транзакцій. Якщо тестування показує, що сервер досягає своїх максимальних можливостей, це може бути сигналом для масштабування, тобто додавання додаткових ресурсів або використання розподілених систем.

Запропонуємо алгоритм для проведення навантажувального тестування та визначення максимальної кількості запитів до сервера MySQL (Рис.3.3):

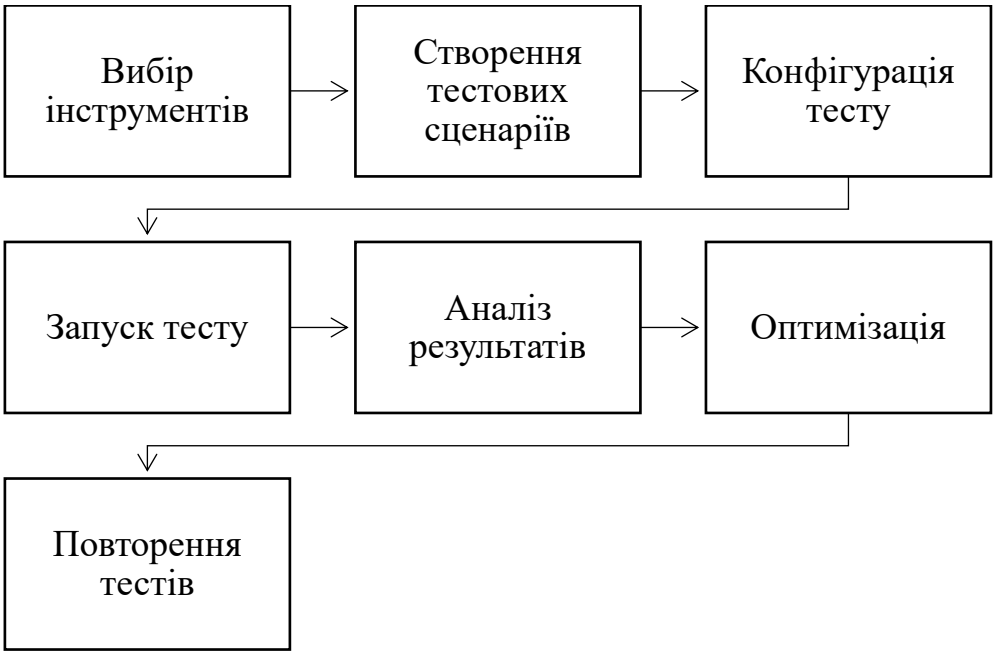

Рис. 3.3. Алгоритм для проведення навантажувального тестування та визначення максимальної кількості запитів до сервера MySQL

Один з найпопулярніших інструментів для проведення тестування навантаження - Apache JMeter. Він дозволяє створювати тестові сценарії та надсилати запити до сервера. У JMeter або вибраному інструменті можливо створити тестові сценарії, які імітуватимуть типові запити до MySQL сервера. Це можуть бути SELECT, INSERT, UPDATE та DELETE запити, а також інші операції. Потім необхідно встановити параметри тесту, такі як кількість одночасних користувачів (паралельних запитів), тривалість тесту та швидкість генерації запитів, починаючи з невеликих значень та поступово збільшувати навантаження. Наступний крок - запуск тесту та спостереження, як сервер обробляє запити, можливо моніторувати продуктивність сервера, такі метрики, як навантаження CPU, використання пам'яті та швидкість відповіді від бази даних. Після завершення тесту - аналіз результатів, оцінка як багато запитів сервер може обробити без втрати продуктивності. Це визначить максимальне навантаження на сервер. Якщо тести показують, що сервер досягає своєї межі, необхідно розглянути можливість оптимізації. Це може включати поліпшення індексації таблиць, кешування запитів у Redis, а також оптимізацію SQL-запитів. Після внесення змін - повтор тестів, щоб переконатися, що оптимізації збільшили максимальне навантаження на сервер.

При проведення навантажувального тестування на бойовому сервері має бути обережним, щоб уникнути негативного впливу на роботу реальних користувачів. Важливо також регулярно моніторити продуктивність сервера, особливо за умов зростання навантаження.

Максимальна кількість SQL-запитів, які 1 фізичний сервер може підтримувати в секунду, залежить від ряду факторів, таких як апаратне обладнання, оптимізація бази даних, структура таблиці, типи запитів і завантаження. У середньому добре налаштований сервер може обробляти від кількох сотень до кількох тисяч запитів у секунду. Однак це число може значно змінюватися в залежності від умов і конфігурації сервера. Точне значення можна визначити, провівши навантажувальне тестування на конкретній системі.

Так, обробка SQL-запитів може значно відрізнятися між MySQL і Oracle із-за відмінностей в апаратному обладнанні, оптимізації, налаштуваннях, масштабованості та ліцензуванні. Oracle, як більш потужна і масштабна система, часто забезпечує більш високу продуктивність, особливо в крупних корпораціях.

На Рис. 3.4 показано використання JMeter для проведення тестування навантаження.

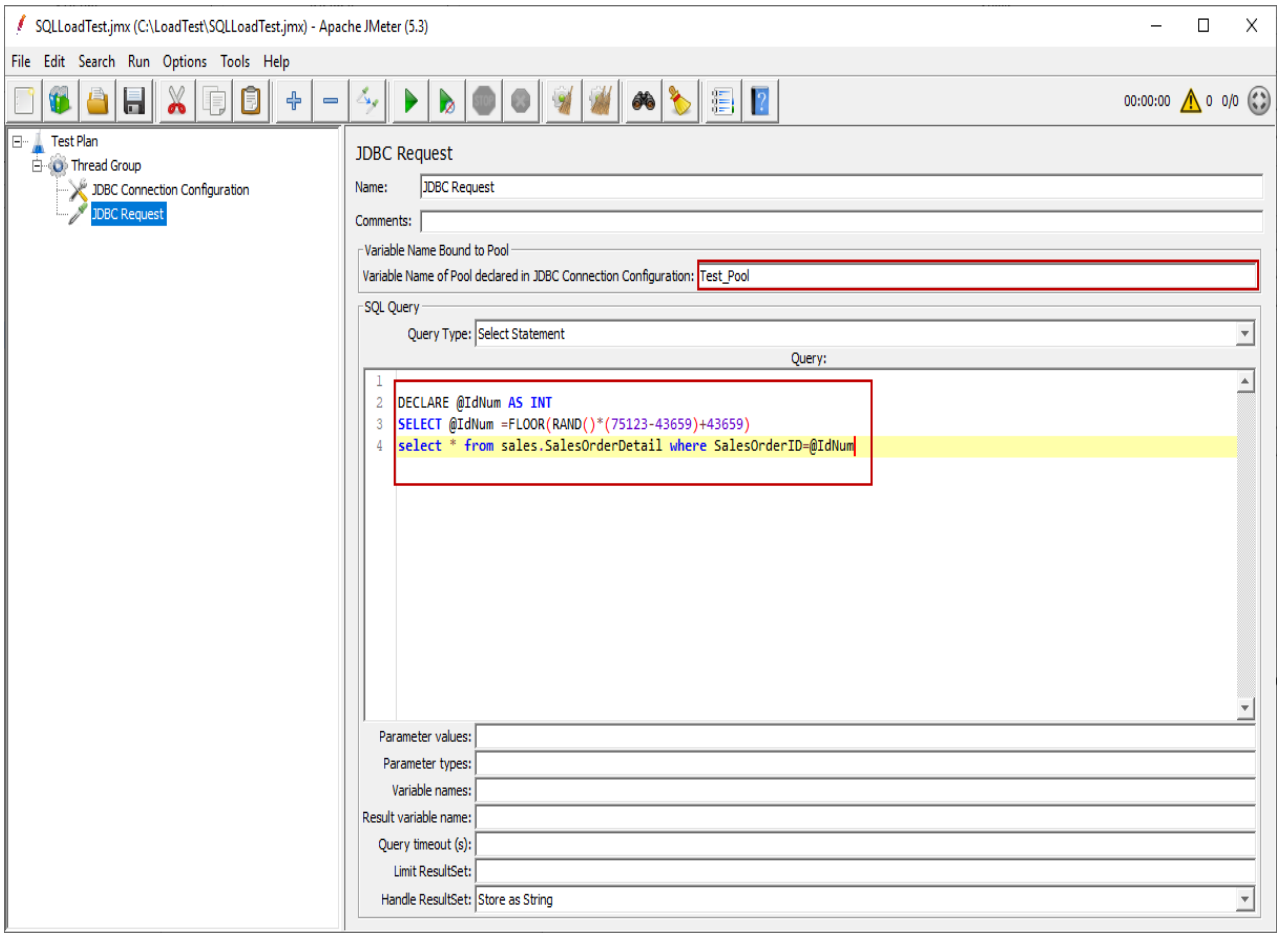

Рис. 3.4. JMeter load testing of SQL serve

Але, яка база даних SQL сама потужна, залежить від різних факторів і вимог. Різні СУБД SQL можуть бути потужними в різних аспектах.

Проаналізуємо відомі СУБД SQL та їх сильні сторони (Рис.3.5).

Отже, вибір СУБД SQL залежить від конкретної задачі, проєкту та вимог. Деякі можуть бути більш придатними для великих підприємств, а інші - для невеликих проєктів або стартапів.

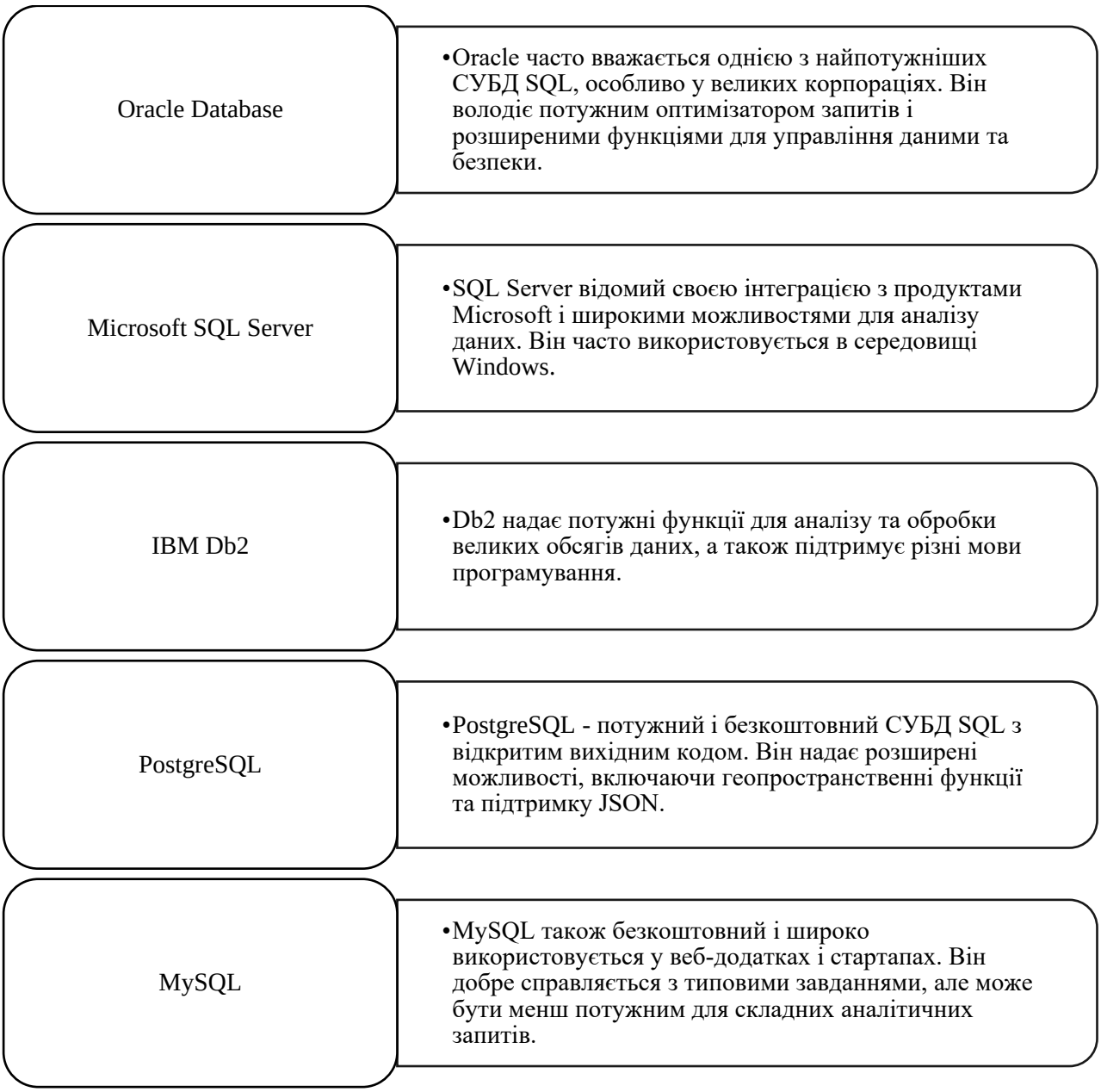

Рис. 3.5. Аналіз СУБД SQL

Для розрахунку середньої кількості запитів на секунду потрібно знати інтервал часу, протягом якого були виконані запити, та кількість запитів у цьому інтервалі. Наприклад, якщо за останні 10 хвилин було виконано 10000 запитів, середня кількість запитів на секунду можна розрахувати так:

$$\text{Середня кількість запитів на секунду} = \frac{\text{Кількість запитів}}{\text{Тривалість інтервалу за секунди}}$$

В даному випадку:

$$\text{Середня кількість запитів за секунду} = \frac{10000}{600} = 16{,}67 \text{ запитів за секунду}$$

Таким чином, протягом останніх 10 хвилин сервер MySQL обробляв у середньому близько 16.67 запитів на секунду. Можливо адаптувати цю формулу в залежності від інтервалу часу та кількості запитів, які цікавлять.

У MySQL всі налаштування зберігаються в конфігураційному файлі, який називається «my.cnf» на Unix-подібних системах (включаючи Linux) та «my.ini» на Windows. Цей файл містить параметри, які визначають поведінку сервера, включаючи параметри продуктивності, налаштування безпеки, розташування файлів баз даних, мережні налаштування та інші параметри, такі як кодування символів та логування.

Розташування конфігураційного файлу може відрізнятися залежно від операційної системи, але зазвичай він знаходиться в «/etc/mysql/my.cnf» або «/etc/my.cnf» на Unix-подібних системах і «C:\ProgramData\MySQL\MySQL Server X.X\my.ini» на Windows (де X.X - версія MySQL). Після внесення змін до цього файлу, необхідно перезапустити сервер MySQL, щоб зміни набули чинності.

3.4. Аналіз ризиків вибору реляційної бази даних

Майстер-майстер (Master-Master) реплікація в MySQL може бути корисною в певних сценаріях, але вона має свої обмеження та потенційні проблеми. На відміну від NoSQL баз даних, як MongoDB, які призначені для горизонтального масштабування і можуть бути більш підходящими в деяких випадках.

Проаналізуємо в Таблиці 3.3 причини, чому майстер-майстер реплікація може бути менш ефективною:

Таблиця 3.3

Причина малоефективності Master-Master реплікації в MySQL

Причина	Пояснення
1	2
Конфлікти даних	У майстер-майстер реплікації можуть виникати конфлікти даних, коли одне і те ж запис змінюється на обох серверах. Це може призвести до втрати даних або несподіваної поведінки.
Синхронізація даних	Синхронізація між майстерами вимагає додаткових зусиль і може призвести до затримок у відображенні змін.
Складність налаштування	Налаштування майстер-майстер реплікації вимагає більше уваги та зусиль в порівнянні з розподіленою NoSQL базою даних.
Оптимізація для читань та записів	Майстер-майстер реплікація важко оптимізована під читання і записи одночасно. Вона може призводити до конфліктів і погіршувати продуктивність.
Загальний обсяг даних	Якщо обсяг даних дуже великий, майстер-майстер реплікація може вимагати дорогої апаратної інфраструктури.

В порівнянні з цим, NoSQL бази даних, такі як MongoDB, зазвичай більш гнучкі для горизонтального масштабування і розподілення даних, що робить їх привабливими для сучасних проектів з великим обсягом даних та високими навантаженнями. Вони дозволяють легше додавати нові сервери для масштабування і зазвичай краще підходять для розподілених середовищ.

Наприклад, якщо є додаток зі значущою кількістю одночасних записів та читань. Причому, щоденний обсяг даних зростає в середньому на 10 ГБ та є план розширення на 5 років. У цьому випадку NoSQL база даних, як MongoDB, може бути більш підходящим варіантом, оскільки вона легше масштабується,

особливо при великому обсязі даних та потребі в забезпеченні високої доступності. Вона може включати можливості горизонтального масштабування, такі як реплікація та розділена обробка запитів, які допоможуть підтримувати високий рівень продуктивності та надійності.

Взагалі, вплив некоректно налаштованої майстер-майстер реплікації може бути драматичним і призвести до серйозних проблем для компаній. Наведемо приклад, коли майстер-майстер реплікація була використана невдало. У 2013 році, соціальна мережа «Twitter» використовувала майстер-майстер реплікацію між центральними даними у Сан-Франциско та Дубліні для забезпечення високої доступності та резервного копіювання даних [22]. Проте в один момент вони зіштовхнулися з проблемою некоректності даних через конфлікти між репліками. Причиною була затримка реплікації через Атлантичний океан, яка призводила до того, що одні дані оновлювалися на одному майстері, а інші на іншому. Це викликало конфлікти, інколи навіть втрату даних. В результаті, Twitter відключив мqастер-майстер реплікацію та перейшов на інші стратегії реплікації даних.

Отже, цей ей приклад підкреслює важливість правильної настройки та управління майстер-майстер реплікацією, а також необхідність ретельного тестування та моніторингу, оскільки некоректна робота може призвести до неконсистентності даних та втрати інформації.

3.5. Висновки до третього розділу

Використовуючи фактори впливу: питому вага SELECT запитів серед усіх запитів, максимальну кількість запитів, яка може витримати один фізичний сервер проекту, бажану кількість запитів, яку має тримати проект без лага, розроблена комплексна формулу прийняття рішення про вибір бази даних, що має числові та параметричні компоненти. Дана формула допомагає чітко визначити, який тип бази даних вибрати - реляційний чи не реляційний.

Для врахування різниці в навантаженні в різні часи доби, формулу для прийняття рішення про вибір бази даних, що має числові та параметричні компоненти, удосконалено, додано фактор, який враховує різницю в навантаженні в залежності від часу доби. Причому, цей фактор можливо визначати на основі аналізу навантаження в різні часові періоди. Наприклад, якщо навантаження зазвичай знижується вночі, то F може бути менше 1, щоб зменшити вагу навантаження в цей період. Таким чином, врахування F фактору допоможе при прийняття рішення про виборі між реляційними та NoSQL базами даних враховувати динаміку навантаження.

Для більш довгострокового моніторингу та збору інформації про запити MySQL, запропоновано використовувати інструменти, такі як Performance Schema, а саме таблиця «Events_statements_history_long», яка може бути використана для аналізу продуктивності виконаних SQL-запитів у системі. За допомогою якої можливо визначити, які запити займають найбільше часу або використовують найбільше ресурсів. Також таблиця корисна при профілюванні та оптимізації запитів у MySQL. Таким чином, таблиця «Events_statements_history_long» - потужний засіб моніторингу та налагодження SQL-запитів у MySQL, і може бути корисною при аналізі продуктивності та оптимізації бази даних.

Доведено, навантажувальне тестування бази даних MySQL проводиться з метою оцінки продуктивності та надійності сервера в умовах великого навантаження. Навантажувальне тестування допомагає встановити, скільки запитів на секунду сервер може обробляти в оптимальних умовах, це важливо для гарантування, що сервер може витримати очікувані навантаження. Тестування дозволяє виявити слабкі місця в архітектурі або конфігурації бази даних, це можуть бути питання з оптимізації запитів, конфліктів блокування, нестабільності під великим навантаженням. Навантажувальне тестування дозволяє визначити, чи вистачає ресурсів (пам'яті, CPU, дискового простору) для обробки потенційно великих обсягів даних і транзакцій. Для проведення навантажувального тестування та визначення максимальної кількості запитів до

сервера MySQL запропоновано алгоритм. Якщо тестування показує, що сервер досягає своїх максимальних можливостей, це може бути сигналом для масштабування, тобто додавання додаткових ресурсів або використання розподілених систем.

Зроблено аналіз ризиків вибору реляційної бази даних. Встановлено, що Master-Master реплікація в MySQL може бути корисною в певних сценаріях, але вона має свої обмеження та потенційні проблеми. На відміну від NoSQL баз даних, як MongoDB, які призначені для горизонтального масштабування і можуть бути більш підходящими в деяких випадках. Встановлені та проаналізовані причини, чому Master-Master реплікація може бути менш ефективною, наведені приклади, які підкреслюють важливість правильної настройки та управління Master-Master реплікацією, а також необхідність ретельного тестування та моніторингу, оскільки некоректна робота може призвести до неконсистентності даних та втрати інформації.

Розроблено програму, яка тестує підключення до бази даних MySQL та проводить оцінку відповідності цих баз даних критеріям проєкту для вибору між реляційними та нереляційними базами даних. Ця програма допомагає здійснити автоматизовану перевірку та вибір баз даних на основі критеріїв та їхньої відповідності проєктним вимогам.

ВИСНОВКИ

У процесі дослідження були отримані наступні результати. База даних є фундаментальним інструментом в сучасному інформаційному світі, який дозволяє зберігати, організовувати та отримувати доступ до великих обсягів даних. Реляційні бази даних є однією з технологій, що використовуються для зберігання структурованих даних, і вони можуть бути частиною більшої архітектури для роботи з різними типами даних. Встановлено хронологію виникнення та розвитку реляційних баз даних.

Зроблено огляд існуючих баз даних за їх типом та областю застосування. Визначені відмінності реляційних та нереляційних баз даних за такими класифікаційними ознаками як:

модель даних;

схема даних;

масштабованість;

транзакції;

застосування.

Досліджено підходи до збільшення продуктивності і доступності баз даних. Зроблено порівняння підходів до збільшення продуктивності і доступності баз даних, які мають свої переваги і недоліки і можуть бути використані в залежності від конкретних потреб і обмежень проекту. Виконано загальний огляд тенденцій щодо масштабування різних типів баз даних.

Прийняття рішення щодо вибору бази даних зазвичай ґрунтуються на вивченні конкретного проекту, його вимог і мети. Важливо провести аналіз, визначити пріоритети та обрати технологію, яка найкраще задовольнить потреби проекту. Встановлені фактори впливу на вибір типу бази даних, які вимагають аналізу конкретних вимог і обставин проєкту:

- структура даних;
- масштабування;
- швидкодія;

- гнучкість сховища даних;
- модель даних;
- досвід команди.

Виконано порівняння навантаження SELECT та UPDATE трафіку в реальному часі, що надає важливу інформацію про те, які типи операцій в базі даних домінують у конкретний момент часу. Порівняння навантаження SELECT та UPDATE трафіку надає контекст для ефективної роботи з базою даних, допомагає виявляти потреби в оптимізації та реагувати на можливі проблеми. Встановлено, що навантаження на базу даних MySQL може суттєво змінюватися в залежності від часу доби та дня тижня. Це залежить від безлічі факторів, включаючи активність користувачів, тип веб-додатку або сервісу та географічне розподілення користувачів.

Для більш довгострокового моніторингу та збору інформації про запити MySQL, запропоновано використовувати інструменти, такі як Performance Schema, а саме таблиця «Events_statements_history_long», яка може бути використана для аналізу продуктивності виконаних SQL-запитів у системі. За допомогою якої можливо визначити, які запити займають найбільше часу або використовують найбільше ресурсів.

Зроблено аналіз ризиків вибору реляційної бази даних. Встановлено, що Master-Master реплікація в MySQL може бути корисною в певних сценаріях, але вона має свої обмеження та потенційні проблеми. На відміну від NoSQL баз даних, як MongoDB, які призначені для горизонтального масштабування і можуть бути більш підходящими в деяких випадках. Встановлені та проаналізовані причини, чому Master-Master реплікація може бути менш ефективною, наведені приклади, які підкреслюють важливість правильної настройки та управління Master-Master реплікацією, а також необхідність ретельного тестування та моніторингу, оскільки некоректна робота може призвести до неконсистентності даних та втрати інформації.

Встановлені фактори впливу на вибір бази даних:

1. питому вага SELECT запитів серед усіх запитів;

2. максимальну кількість запитів, яка може витримати один фізичний сервер проекту;
3. бажану кількість запитів, яку має тримати проект без лага

Використовуючи фактори впливу, розроблена комплексна формулу прийняття рішення про вибір бази даних, що має числові та параметричні компоненти. Формула допомагає чітко визначити, який тип бази даних вибрати - реляційний чи не реляційний. Для врахування різниці в навантаженні в різні часи доби, формулу для прийняття рішення про вибір бази даних, що має числові та параметричні компоненти, удосконалено, додано фактор, який враховує різницю в навантаженні в залежності від часу доби. Причому, цей фактор можливо визначати на основі аналізу навантаження в різні часові періоди.

Розроблено програму, яка тестує підключення до бази даних MySQL та проводить оцінку відповідності цих баз даних критеріям проєкту для вибору між реляційними та нереляційними базами даних. Ця програма допомагає здійснити автоматизовану перевірку та вибір баз даних на основі критеріїв та їхньої відповідності проєктним вимогам.

Проведене дослідження буде корисно для науковців, розробників, аспірантів і студентів, що спеціалізуються або цікавляться проблематикою дослідження реляційних баз даних, як відмовостійкого компоненту високонавантаженого бізнесу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зінов'єва, І. С., Артемчук, В. О. Сучасні підходи до подальшої еволюції концепції баз даних. In: III International scientific and practical conference «Dynamics of the development of world science», Canada. 2019. p. 34-44.

2. Березький, О. М., et al. Розроблення реляційної бази даних інтелектуальної системи автоматизованої мікроскопії. Науковий вісник НЛТУ України, 2017, 27.5: 125-129.

3. Мулеса О.Ю., et al. Місце теми «Інструкція SELECT» в змістовому модулі «Реляційні бази даних» та методика її навчання. Физико-математическое образование, 2018, 1 (15): 260-263.

4. Брацький,В. О., М'якшило, О. М. Дослідження особливостей застосування реляційних і нереляційних баз даних на прикладі SQL Server та MongoDB. Наукові праці Національного університету харчових технологій, 2016, 22,№ 5: 15-24.

5. TAFT, Rebecca, et al. Cockroachdb: The resilient geo-distributed sql database. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. 2020. p. 1493-1509.

6. WANG, Lijie, et al. DuSQL: A large-scale and pragmatic Chinese text-to-SQL dataset. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2020. p. 6923-6935.

7. TANG, Peng, et al. Detection of SQL injection based on artificial neural network. Knowledge-Based Systems, 2020, 190: 105528.

8. CODD, Edgar F. A relational model of data for large shared data banks. Communications of the ACM, 1970, 13.6: 377-387.

9. GILBERT, Seth; LYNCH, Nancy. Perspectives on the CAP Theorem. Computer, 2012, 45.2: 30-36.

10. Карпенко А. С., Тарасюк О. М., Горбенко А. В. Дослідження узгодженості та продуктивності у нереляційних реплікованих базах даних. 2021. URL: http://repository.kpi.kharkov.ua/handle/KhPI-Press/56404

11. Белоус, Р., Нікітін, В. Варіанти забезпечення суворої узгодженості в NOSQL. Grail of Science, 2023, 24: 364-365.

12. Zawodny J, Balling D (2004) High performance MySQL: optimization, backups, replication, load-balancing, and more. O'Reilly & Associates, Inc., Sebastopol

13. Malahovskii, O. Y.; Ornatovska, V. V.; Brevus, V. M. Удосконалення у мові програмування C++ 11. Актуальні задачі сучасних технологій, 2014, 207.

14. Джулій, В. М.; Чешун, В. М.; Лєнков, О. С. Архітектура організації системи захисту баз даних з колонковим представленням даних. Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка, 2016, 51: 150-158.

15. Tretiak, V. F.; Pashnyeva, A. A. Оптимізація структури сховища даних у вузлах інфокомунікаційної мережі хмарного середовища. Системи управління, навігації та зв'язку. Збірник наукових праць, 2017, 4.44: 122-128.

16. LEE, Chao-Hsien; ZHENG, Yu-Lin. Automatic SQL-to-NoSQL schema transformation over the MySQL and HBase databases. In: 2015 IEEE International Conference on Consumer Electronics-Taiwan. IEEE, 2015. p. 426-427.

17. Статистика з трафіку Amazon. URL: https://dropkit.com/support/ru/article/amazon-traffic-statistics/

18. Інтернет – торгівля. URL: https://www.amazon.com/

19. Amazon Web Services . URL: https://aws.amazon.com/ru/

20. Трафік Amazon Prime Video. URL: https://www.similarweb.com/ru/website/primevideo.com/

21. SHAPIRO, Jeffrey. SQL Server 2000: the complete reference. McGraw-Hill, Inc., 2002.

22. Методи масштабування реляційних баз даних: переваги, недоліки та кейси використання. URL: https://dou.ua/forums/topic/45890/

23. Усатенко М.В. Поняття реляційної бази даних як відмовостійкого компонента високонавантаженого бізнесу. Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення (випуск 80): матеріали

Міжнародної наукової інтернет-конференції, (м. Тернопіль, Україна – м. Ополе, Польща, 19-20 вересня 2023 р.) / [редкол. : О. Патряк та ін.] ; ГО “Наукова спільнота”; WSZIA w Opolu. – Тернопіль : ФО-П Шпак В.Б. С.91-93

24. Усатенко М.В. Математичний інструментарій прийняття рішення про вибір бази даних.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Програма для автоматизованої перевірки та вибору баз даних на основі критеріїв та їхньої відповідності проектним вимогам.

```
using System;
using System.Collections.Generic;
using MySql.Data.MySqlClient;

class Program
{
    static void Main()
    {
        List<ConnectionInfo> connections = new List<ConnectionInfo>
        {
            new ConnectionInfo
            {
                Server = "your_server1",
                Database = "your_database1",
                Username = "your_username1",
                Password = "your_password1",
                TableName = "your_table_name1",
                ProjectId = "your_project_id1"
            },
            // Додати інші з’єднання за аналогією
        };

        try
        {
            foreach (var connectionInfo in connections)
            {
                using (MySqlConnection connection = new
MySqlConnection(GetConnectionString(connectionInfo)))
```

```
                {
                    connection.Open();

                    // Отримаємо метрики в режимі реального часу
                    double SEL = GetSelectCoefficient(connection);
                    double SMQPS = GetServerMaxQueriesPerSecond(connection);
                    double QPS = GetDesiredQueriesPerSecond(connection);

                    // Інші параметри для тестування
                    double F = 1.5;

                    // розраховуємо результат за формулою:
                    double R = (1 - SEL) * (SMQPS - QPS) + (QPS / (SMQPS * (1 - SEL))) * F;

                    // Визначаємо тип бази, реляційна чи нереляційна
                    string databaseType = (R > 0) ? "Реляційна" : ((R < 0) ? "NoSQL" :
"Можливі два варіанти");

                    // Вивід результату для кожного з’єднання
                    Console.WriteLine($"Результат для {connectionInfo.Database}:");
                    Console.WriteLine($"Значення R: {R}");
                    Console.WriteLine($"Тип бази даних: {databaseType}");

                    // Вивід запросів для бази даних для інформації
                    Console.WriteLine($"SELECT запрос для удельної вагиSELECT:
{GetSelectCoefficientQuery()}");
                    Console.WriteLine($"SELECT запрос для максимальної кількості запитів
на сервері: {GetServerMaxQueriesPerSecondQuery()}");
                    Console.WriteLine($"SELECT запрос для бажаної кількості запитів без
лага: {GetDesiredQueriesPerSecondQuery(connectionInfo.TableName,
connectionInfo.ProjectId)}");

                    Console.WriteLine();
                }
            }
```

```
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Ошибка: {ex.Message}");
            }
        }

        static string GetConnectionString(ConnectionInfo connectionInfo)
        {
            return $"Server={connectionInfo.Server};Database={connectionInfo.Database};User
ID={connectionInfo.Username};Password={connectionInfo.Password};";
        }

        static double GetSelectCoefficient(MySqlConnection connection)
        {
            // Приклад запиту для отримання удельної ваги SELECT запитів
            string query = GetSelectCoefficientQuery();
            using (MySqlCommand command = new MySqlCommand(query, connection))
            {
                using (MySqlDataReader reader = command.ExecuteReader())
                {
                    if (reader.Read())
                    {
                        int comSelect = Convert.ToInt32(reader["Value"]);
                        int comTotal = Convert.ToInt32(reader["Variable_value"]);
                        return (double)comSelect / comTotal;
                    }
                }
            }
            return 0;
        }

        static double GetServerMaxQueriesPerSecond(MySqlConnection connection)
        {
            // Приклад запиту для отримання максимальної кількості запитів на сервері
```

```
            string query = GetServerMaxQueriesPerSecondQuery();
            using (MySqlCommand command = new MySqlCommand(query, connection))
            {
                using (MySqlDataReader reader = command.ExecuteReader())
                {
                    if (reader.Read())
                    {
                        return Convert.ToInt32(reader["Value"]);
                    }
                }
            }
            return 0;
        }

        static double GetDesiredQueriesPerSecond(MySqlConnection connection)
        {
            // Приклад запиту для отримання бажаної кількості запитів без лага
            string query = GetDesiredQueriesPerSecondQuery("your_table_name",
"your_project_id");
            using (MySqlCommand command = new MySqlCommand(query, connection))
            {
                using (MySqlDataReader reader = command.ExecuteReader())
                {
                    if (reader.Read())
                    {
                        return Convert.ToInt32(reader["desired_qps"]);
                    }
                }
            }
            return 0;
        }

        static string GetSelectCoefficientQuery()
        {
            return "SHOW STATUS LIKE 'Com_select';";
```

```
        }

        static string GetServerMaxQueriesPerSecondQuery()
        {
            return "SHOW VARIABLES LIKE 'max_connections';";
        }

        static string GetDesiredQueriesPerSecondQuery(string tableName, string projectId)
        {
            return $"SELECT desired_qps FROM {tableName} WHERE project_id =
{projectId};";
        }

        // Класс для зберігання інформації про з’єднання
        class ConnectionInfo
        {
            public string Server { get; set; }
            public string Database { get; set; }
            public string Username { get; set; }
            public string Password { get; set; }
            public string TableName { get; set; }
            public string ProjectId { get; set; }
        }
    }
```

ДОДАТОК Б

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

Факультет інформаційних технологій

Кафедра програмного забезпечення комп'ютерних систем

ВІДГУК

Наукового керівника Приходченко С.Д., к.т.н., доц. каф. ПЗКС
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

на магістерську роботу

студента Усатенко Максима Володимировича
групи 121м-22-1
спеціальності 121 Інженерія програмного забезпечення
освітньої програми Інженерія програмного забезпечення
на тему Дослідження реляційної бази даних, як відмовостійкого компоненту високонавантаженого бізнесу

Актуальність теми Обумовлена тим, що за бізнеси та організації мають справлятися із надзвичайно великими обсягами інформації, яка включає в себе структуровані дані, такі як бази даних, а також напівструктуровані та неструктуровані дані, наприклад, тексти, фотографії, відео та сенсорні дані від пристроїв IoT. Це викликає потребу в ефективному зберіганні, обробці та аналізі цих даних.

Мета дослідження Розробка інструментарію та програми з автоматизації щодо прийняття рішень з вибору бази даних.

Коротка характеристика розділів роботи Перший розділ присвячено принципам роботи реляційних баз даних та дослідженню принципів до збільшення продуктивності і доступності баз даних. Другий розділ базується на технологіях вибору баз даних як відмовостійкого компоненту високонавантаженого бізнесу, а також дослідженню наявного математичного інструментарію прийняття рішень про вибір баз даних. У третьому розділі представлені результати роботи щодо комплексної формули, яка використовуючи фактори впливу допомагає приймати рішення щодо вибору бази даних. А також розроблено програму, яка тестує підключення до бази даних MySQL та проводить оцінку відповідності цих баз даних критеріям проєкту для вибору між реляційними та нереляційними базами даних. Ця програма допомагає здійснити автоматизовану перевірку та вибір баз даних на основі критеріїв та їхньої відповідності проєктним вимогам.

Практичне значення роботи Результати роботи, отриманні в ході дослідження, можуть застосовуватися як при роботі з базами даних, так і в практиці викладання дисциплін, пов'язаних з базами даних.

Зауваження та недоліки.

Висновки та оцінка Оформлення дипломної роботи магістра виконано на сучасному рівні і відповідає вимогам, що пред'являються до робіт даної кваліфікації. Ступінь самостійного виконання досить висока. Дипломна робота магістра в цілому заслуговує оцінки , а сам автор – присвоєння кваліфікації «інженерг-програміст»

Науковий керівник Приходченко С.Д., к.т.н., доц. каф. ПЗКС

«___»_____20__ р. _____

ДОДАТОК В

РЕЦЕНЗІЯ

на магістерську роботу

студента Усатенко Максима Володимировича
(прізвище, ім'я, по батькові)

групи 121м-22-1

кафедри програмного забезпечення комп'ютерних систем

спеціальності 121 Інженерія програмного забезпечення

освітньої програми Інженерія програмного забезпечення

Тема роботи Дослідження реляційної бази даних, як відмовостійкого компоненту високонавантаженого бізнесу

Стисла характеристика розділів роботи Розділ 1 присвячено основам функціонування реляційних баз даних і аналізу методів для підвищення продуктивності та доступності даних. Розділ 2 базується на виборі баз даних як надійного компонента високонавантажених бізнес-систем, включаючи вивчення математичних інструментів для прийняття рішень у цьому виборі. У Розділі 3 представлені результати роботи, включаючи комплексну формулу, що враховує різні впливові фактори для зручного вибору бази даних. Також створена програма, яка автоматизовано перевіряє з'єднання з базою даних MySQL та оцінює їх відповідність критеріям проєкту, допомагаючи при виборі між реляційними та нереляційними базами даних. Цей інструмент спрощує процес перевірки та вибору баз даних на основі проєктних вимог

Пропозиції, внесені студентом, рівень їх наукового обґрунтування Студентом проведений детальний аналіз баз даних. Дослідження ґрунтується на математичних та алгоритмічних методах обчислень

Практичне значення роботи Результати магістерської роботи дозволяють фахівцям різної кваліфікації використовувати систематизовані знання з даної тематики для написання більш продуктивних і ефективних програм з використанням баз даних.

Якість оформлення роботи Дана робота цілком відповідає вимогам, що пред'являються до кваліфікаційних робіт рівня магістра

Недоліки в роботі Суттєвих недоліків не було виявлено, але для верифікації результатів роботи з базами даних потрібно було провести більше експериментів

Загальний висновок Студент М.В.Усатенко досить добре розібрався в специфіці використання баз даних

Оцінка магістерської роботи В цілому автор заслуговує оцінки «відмінно

Рецензент_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання, посада, місце роботи)

«___»_____20__ р. _____

ДОДАТОК Г

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_Усатенко.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_Усатенко.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму.
Презентація	
Презентація_Усатенко.ppt	Презентація кваліфікаційної роботи.