Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)
Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження ефективності кешування даних
в реляційних та NoSQL базах даних
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-22-2
Семко Д.
(прізвище, ініціали)
Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми Освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Керівник доц. Мазурова О. О.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри, проф. _____ З.В. Дудар
(підпис) (посада, прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ
студента *Семка Дениса*
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження ефективності кешування даних в реляційних та NoSQL базах даних»

Затверджена наказом університету від "29" 03 20 24 р. № 250ст

2. Термін подання студентом роботи до екзаменаційної комісії "13" 06 2024 р.

3. Вихідні дані до роботи опис механізмів кешування, вимоги до розробки схеми баз даних для предметної області, електронні ресурси за обраною темою, мови програмування C# та Javascript, технології .NET 7 та React.js, СКБД MSSQL, MongoDb, Redis, середовища розробки Visual Studio та Visual Studio Code.

4. Перелік питань, що потрібно опрацювати в роботі аналіз та порівняння СКБД для дослідження, вибір критеріїв для оцінки, проектування логічних моделей для експериментальних досліджень, розробка програмних рішень, проведення експериментів та аналіз отриманих результатів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Позначка про виконання
1	Аналіз предметної галузі	22.01.2024 – 07.02.2024	*виконано*
2	Розробка постановки задачі	08.02.2024 – 15.02.2024	*виконано*
3	Аналіз існуючих СКБД для проведення дослідження	16.02.2024 – 22.02.2024	*виконано*
4	Планування експериментального дослідження	23.02.2023 – 06.03.2024	*виконано*
5	Проектування середовища для дослідження	07.03.2024 – 13.03.2024	*виконано*
6	Програмна реалізація дослідження	13.03.2024 – 30.03.2024	*виконано*
7	Експериментальні дослідження та їх аналіз	01.04.2024 – 15.04.2024	*виконано*
8	Оформлення статті або тез доповіді	16.04.2024 – 19.04.2024	*виконано*
9	Підготовка пояснювальної записки	19.04.2024 – 02.05.2024	*виконано*
10	Підготовка презентації та доповіді	03.05.2024 – 10.05.2024	*виконано*
11	Нормоконтроль	03.06.2024	*виконано*
12	Рецензування	05.06.2024	*виконано*
13	Занесення диплома в електронний архів	07.06.2024	*виконано*
14	Попередній захист	08.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	09.06.2024	*виконано*

Дата видачі завдання 22 січня 2024р.

Студент _____ Семко Д.

(підпис) (прізвище, ініціали)

Керівник роботи _____ доц.кафедри ПІ Мазурова О.О

(підпис) (прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка містить: 86 с., 21 рис., 6 табл., 25 джерел.

КЕШУВАННЯ ДАНИХ, БАЗА ДАНИХ, ВЕБ-СИСТЕМА, C#, REACT.JS, SQL, PLAN CACHE, BUFFER CACHE, INDEX CACHE.

Об’єктами дослідження є реляційні та нереляційні системи керування базами даних та методи кешування даних.

Метою дослідження є визначення ефективності кешування даних серед реляційних та нереляційних систем керування базами даних та надання можливості конфігурувати дані для різних систем задля дослідження.

Методи проведення дослідження – аналіз та моделювання предметної області, концептуальне моделювання, UML-моделювання, об’єктно-орієнтоване програмування, використання відповідних СКБД, побудова мікросервісів на основі Clean архітектури.

У результаті роботи були досліджені методи кешування даних кожної з обраних базах даних, проаналізовано процес кешування, визначено переваги та недоліки та запропоновано власний метод для вирішення існуючих проблем.

DATA CACHING, DATABASE, WEB-SYSTEM, C#, REACT.JS, SQL, PLAN CACHE, BUFFER CACHE, INDEX CACHE.

The objects of the research are relational and non-relational data management systems and data caching methods.

The purpose of the study is to determine the effectiveness of data caching among relational and non-relational data management systems and to provide the ability to configure data for different systems for research.

Research methods – analysis and modeling of the subject area, conceptual modeling, UML modeling, object-oriented programming, use of appropriate DBMS, construction of microservices based on the Clean architecture.

As a result, data caching methods of each of the selected databases were investigated, the caching process was analyzed, advantages and disadvantages were determined, and an own method was proposed for solving existing problems.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Семко Денис, студент групи ІПЗм-22-2, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження ефективності кешування даних в реляційних та нереляційних базах даних», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Будь-яка програмна система працює з різним об’ємом даних, починаючи від наявності декількох сутностей у системі керування базами даних (СКБД), закінчуючи величезним об’ємом, який слід швидко та детально опрацьовувати при кожному запиті від користувача. Уповільнення швидкості роботи системи, неможливість обробити дані та великі затрати призвели до появи такого явища як кешування.

Кешування - це процес тимчасового зберігання даних у кеш-пам’яті з метою збільшення продуктивності програм та систем у цілому. Кешування дозволяє ефективно використовувати раніше отримані дані безліч разів, тим самим скорочуючи час, який витрачається для доступу до цих самих даних.

Кешування може здійснюватись на різних рівнях, це і на рівні веб-браузеру, чи веб-серверу, CDN та на рівні вихідного серверу системи. Кожен з рівнів так чи інакше покращують спілкування користувача з системою задля пришвидшення відповіді на кожний з запитів. На рівні браузеру це відбувається за рахунок кешування HTML, зображень та коду, аби зменшити кількість запитів до веб-серверу. Веб-сервер кешує дані відповідей для зменшення навантаження на процесор та підвищення продуктивності системи. CDN кешують вміст та зменшують затримку між запитом-відповіддю. Не менш важливим є рівень вихідного серверу системи, який кешує дані під час роботи зі СКБД.

Будь-яке програмне забезпечення має власні сховища даних, використовуючи для цього реляційні або нереляційні бази даних. Відповідно у системах, які масштабуються, збільшується об’єм даних з якими вони можуть працювати. Не менш важливим є кількість одночасних запитів, які навантажують систему та дають окремий поштовх для використання такого функціоналу як кешування даних.

Багато існуючих СКБД реалізовують свій механізм кешування та передбачають власну вбудовану логіку, яка є доволі унікальною та прискорює швидкість взаємодії з базою даних. Отже постає питання, яка зі СКБД найкраще

забезпечує роботу з великим об’ємом даних за допомогою власних механік, в тому числі у рамках кешування.

Таким чином, метою кваліфікаційної роботи є проведення дослідження кешування реляційних та нереляційних баз даних з метою визначення найбільш ефективної імплементації. Отриманий результат буде дуже корисним у випадках етапу вибору СКБД для програмної системи або при міграції з однієї бази даних в іншу в рамках покращення продуктивності системи.

У ході кваліфікаційної роботи було проведено аналіз предметної галузі кешування баз даних на основі публікацій світових та вітчизняних фахівців в предметній галузі (див. додаток А). Розглянуті найбільш актуальні СКБД для проведення дослідження та методики кешування даних.

Робота пройшла успішну перевірку на академічну доброчесність (див. додаток Б).

На основі плану проведення дослідження та його результатів було розроблено презентацію (див. додаток В). За результатами роботи були створені тези доповіді на двадцять восьмий міжнародний молодіжний форум «РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ В XXI ст.», підготовлено матеріали доповіді «Research on the efficiency of data caching to solve the problem of building a heterogeneous database» для подачі до IEEE Міжнародної науково-практичної конференції «Проблеми інфокомунікацій. Наука і техніка» (PIC S&T-2024) (див. додаток Г).

Робота перевірена на відповідність вимогам оформлення (див. додаток Д).

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Аналіз предметної галузі дослідження

Ефективність програмного забезпечення – це одночасно складна та цікава задача, яку слід продумати на початковому етапі розробки системи. Доволі важко передбачити чи зможе система ефективно та швидко вирішувати задачі, оскільки це залежить як від декількох факторів, як від побудованої архітектури та коду, так від вибору додаткових інструментів. Тим не менш, ключовою складовою є вибір СКБД та максимальне використання її можливостей в рамках розробки програмного забезпечення.

Маючи за ідею створити програмну систему, яка буде добре масштабуватись та у кінцевому результаті оброблятиме великий об’єм даних, стає зрозуміло, що оброблювати такі дані слід доволі швидко та продуктивно. Тим самим, для цього слід ознайомитись з таким поняттям як кешування [1].

Кешування – це такий процес, який зберігає дані у кеш-пам’яті, яка є доволі швидко-доступною для користування. Для будь-якого розробника кешування є невід’ємним елементом роботи при розробці програмного забезпечення [2].

Існує дуже багато переваг у використанні кешування, серед яких прискорення вибірки даних та збільшення продуктивності. Завдяки ньому системи забезпечують швидкий доступ до часто використовуваних даних, знижуючи тим самим навантаження на базу даних та покращуючи загальну продуктивність. Кешування може проявлятись на багатьох рівнях, серед яких є і користування браузером або будь-якою СКБД (див.рис.1.1).

Рисунок 1.1 – Різні рівні кешування (за даними [2])

Та кожна система керування базами даних має свої реалізації та методи роботи з кешуванням. Виділяють декілька загальних стратегій, які СКБД зазвичай використовують в якості реалізації для своїх схем кешування даними. Наприклад, MySQL використовує Query Cache, для зберігання результатів запитів, які виконуються часто [3]. Гарним прикладом як саме СКБД взаємодіє зі стратегією Query Cache є наступною (див.рис.1.2).

Рисунок 1.2 – Взаємодія MySQL зі стратегією Query Cache (за даними [3])

З іншого боку, PostgreSQL використовує Page Cache на рівні операційної системи для кешування цілих сторінок даних [4]. А MongoDb взагалі використовує власний механізм роботи з кешуванням WiredTiger [5].

Таким чином, кожна з систем керування базами даними має власні методи реалізації кешування і тому власне постає питання, а який з цих методів є найефективнішим. Та навіть яка СКБД краще всього справляється з кешуванням за допомогою власних вбудованих реалізацій. Це важливо дослідити, оскільки процес

кешування починається з рівня браузеру та закінчується «найглибшим» рівнем, а саме на рівні СКБД [6].

Визначивши для яких цілей нам необхідно кешування, слід врахувати існуючі проблеми та сценарії, при яких нам важливо досліджувати кешування на рівні СКБД.

Перша з таких проблем – це наявність запитів, що вимагають багато ресурсів. Тобто такі запити, які потребують значних обчислювальних або мережевих ресурсів для виконання. Для таких випадків кешування може значно підвищити продуктивність, особливо при запитах з великим обсягом даних.

Наступною проблемою є висока навантаженість на базу даних. Це звичайна проблема для доволі популярної в користуванні системи. Велика кількість користувачів, одночасні запити – усе це є проблемою та кешування може забезпечити кращу та миттєву відповідь користувачам.

Не менш важливою проблемою є наявність системи з обмеженими ресурсами. Можливо система має обмежені обчислювальні ресурси на сервері бази даних і тому кешування може допомогти ефективно використовувати ці ресурси та запобігти перевантаженню сервера.

Зменшення часу відповіді, зменшення навантаження та дорогостоящих операцій – це все також наявні проблеми, які слід мати на увазі при розробці програмного забезпечення. І усі ці проблеми вирішує кешування. Результати запитів можуть бути збережені та використовуватись повторно, при цьому зменшуючи час відповіді та очікування для користувача.

Отже, існує доволі багато проблем з якими розробник програмного забезпечення може зіштовхнутись, особливо якщо система передбачає масштабованість та популярність серед користувачів. Усі ці проблемні сценарії кажуть про одне – яка зі СКБД найефективніше справляється з механізмами кешування та покращує роботу системи з даними.

1.2 Постановка задачі

Після аналізу предметної галузі була сформована задача дослідити ефективність реалізації кешування в різних системах керування базами даних. Передбачається перевірити обрані СКБД на можливість справитись з різними стрес ситуаціями, за якими формуватиметься висновок про ефективність роботи. При цьому слід визначити, які саме СКБД за розподілом на реляційні та NoSQL є найкращими для проведення дослідження.

Тому в ході дослідження слід виділити та вирішити наступні задачі:

- провести аналіз та обрати СКБД для подальшого дослідження ефективності кешування;
- провести аналіз реалізації механізму кешування в обраних СКБД;
- спланувати експериментальне дослідження, що включає в себе вибір, аналіз та моделювання предметної галузі для розробки БД для дослідження, розробку критеріїв оцінки ефективності та можливості конфігурації даних в певному форматі одночасно для усіх СКБД;
- спроектувати та розробити програмне забезпечення для дослідження;
- провести експериментальне дослідження ефективності кешування;
- оцінити якість результатів, сформулювати висновки стосовно використання методів кешування та відповідних СКБД.

2 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

2.1 Аналіз та вибір СКБД для проведення дослідження

Для вибору найпопулярніших СКБД скористаємось ресурсом DbEngines [7] та оберемо множину альтернатив згідно наявного рейтингу (див.рис.2.1)

416 systems in ranking, November 2023

Rank			DBMS	Database Model	Score		
Nov 2023	Oct 2023	Nov 2022			Nov 2023	Oct 2023	Nov 2022
1.	1.	1.	Oracle	Relational, Multi-model	1277.03	+15.61	+35.34
2.	2.	2.	MySQL	Relational, Multi-model	1115.24	-18.07	-90.30
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	911.42	+14.54	-1.09
4.	4.	4.	PostgreSQL	Relational, Multi-model	636.86	-1.96	+13.70
5.	5.	5.	MongoDB	Document, Multi-model	428.55	-2.87	-49.35
6.	6.	6.	Redis	Key-value, Multi-model	160.02	-2.95	-22.03
7.	7.	7.	Elasticsearch	Search engine, Multi-model	139.62	+2.48	-10.70
8.	8.	8.	IBM Db2	Relational, Multi-model	136.00	+1.13	-13.56
9.	9.	↑ 10.	SQLite	Relational	124.58	-0.56	-10.05
10.	10.	↓ 9.	Microsoft Access	Relational	124.49	+0.18	-10.53

Рисунок 2.1 – Фрагмент з рейтингу СКБД (за даними [7])

Надалі до множин альтернатив включимо найбільш поширені СКБД, враховуючи рейтинг, а також можливості кешування, які ми можемо дослідити у кожній з них.

Таким чином, вибір наступних СКБД буде найкращим рішенням для проведення необхідного дослідження.

Першою буде MySQL – це доволі популярна система керування реляційними базами даних, яка здобула значне місце у рейтингу завдяки своїй продуктивності, надійності та простоті у використанні. Найчастіше дану СКБД використовують для веб-додатків, включаючи різні електронні магазини та системи керування веб-сайтами. MySQL використовує різні механізми та стратегії кешування, включаючи Query Cache стратегію, яка може значно скоротити час відповіді на запити [3].

Другою буде Microsoft SQL Server – наступна популярна система керування реляційними базами даних, розроблена компанією Microsoft. SQL Server використовує буферний пул, який є важливою частиною його стратегії кешування [8]. Пул буферів кешує сторінки даних у пам’яті, зменшуючи потребу читати дані з диска та покращуючи продуктивність читання. MSSQL також використовує

функцію під назвою «columnstore indexes» для покращення продуктивності запитів для аналітичних навантажень [9].

Наступною буде PostgreSQL - це об’єктно-реляційна система керування базами даних із відкритим вихідним кодом, яка є популярною серед розробників завдяки підтримці розширених типів даних, методів індексування та інших можливостей [4]. PostgreSQL переважно використовує shared buffers як частину своєї стратегії кешування. За допомогою них кешуються часто доступні блоки даних у пам’яті, зменшуючи потребу в дисковому вводі-виводі. PostgreSQL також підтримує кешування результатів запитів, оскільки база даних зберігає плани виконання для конкретних запитів і повторно використовує їх, коли це необхідно.

Далі включимо до вибору MongoDb – це документно-орієнтована система керування базами даних, яка зберігає дані у гнучкому та схожому на JSON форматі під назвою BSON [10]. MongoDb є популярною завдяки своїй масштабованості та гнучкості, що робить її придатною для різних випадків користування. MongoDB використовує власний механізм зберігання в пам’яті WiredTiger [5], який використовує кеш-пам’ять для зберігання часто використовуваних даних у оперативній пам’яті, що значно покращує продуктивність читання. Механізм зберігання WiredTiger також підтримує стиснення, що дозволяє ефективно використовувати пам'ять і ресурси зберігання [11].

П’ятим на вибір для проведення дослідження буде Redis – це розподілене сховище пар ключ-значення, які зберігаються в оперативній пам’яті [12]. Оскільки Redis – це сховище даних у пам’яті, тому воно не покладається на зовнішні механізми кешування. Дані повністю зберігаються в пам’яті і використовуються різні структури даних для кешування, такі як сети, хеш-таблиці та інші.

Останньою системою, яку слід включити до списку – це Memcached. Даний варіант входить в рейтинг найпопулярніших СКБД серед key-value баз даних та є доволі цікавим варіантом для проведення дослідження. Memcached – це просте та ефективне рішення в рамках кешування, яке зазвичай використовується для підвищенні швидкості реагування додатків [13].

Після визначення множини альтернатив, слід відмітити множину критеріїв за якими будуть відібрані найкращі СКБД для проведення дослідження.

Популярність – наскільки кожна з систем є популярною за користуванням.

Використання СКБД для розподіленого кешування – даний тип кешування передбачає зберігання даних на кількох машинах або вузлах. Даний тип є необхідним для систем, які потребують масштабування на декількох серверах та гарантує наявність даних попри велике навантаження.

Інтеграція та сумісність з .NET – наскільки легко встановити зв’язок зі системою, виконувати складні запити та рівень підтримки Nuget пакетів.

Підтримка та розвиток СКБД – на якому рівні СКБД підтримуються, на даний критерій впливає належність систем до open-source категорії проектів.

Кількість стратегій (та механізмів) кешування, які СКБД використовує – наскільки кешування розвинуто у кожній з систем та яка приблизна кількість механізмів або стратегій використовується.

Таким чином, обравши критерії вибору, додамо їх до таблиці критеріїв з відповідними значеннями (див.табл.2.1).

Таблиця 2.1 – Опис альтернатив за обраними критеріями (таблиця створена самостійно)

	Популярність	Розподілене кешування	Сумісність з .NET (M)	Розвиток СКБД	К-сть механізмів кешування
MySQL	1115.24	Ні	72.3	2	2
MSSQL	911.42	Так	396.1	5.2	3
PostgreSQL	636.86	Ні	131.1	3	2
MongoDb	428.55	Ні	168.7	5.6	2
Redis	160.02	Так	392.7	14.4	1
Memcached	19.80	Так	4.1	4.8	1

Для заповнення значення критерію популярності, був використаний ресурс DbEngines.

Значення критерію підтримка СКБД для розподіленого кешування є так або ні.

Сумісність з .NET визначається не тільки наявністю відповідних Nuget пакетів або SDK, а за кількістю скачувань офіційного пакету.

Розвиток СКБД визначається за формулою 1.1:

$$R = c * p, \tag{1.1}$$

де c – кількість версій за останній рік (береться проміжок від листопаду 2022 до листопаду 2023),

p – коефіцієнт належності до open-source проекту, 0.4 якщо не належить, 0.6 якщо належить.

Така варіація значень у коефіцієнті належності передбачена тим, що open-source проекти частіше надають нові версії продукту, а також більш відкриті до загальних обговорень майбутніх версій.

Далі переведемо отримані шкали з якісних до кількісних.

Критерії популярності, розвитку СКБД та кількості механізмів кешування мають абсолютні значення і тому не має сенсу переводити ці значення.

Критерій розподіленого кешування має лише два значення Так і Ні, тому оцінимо їх по шкалі 0 – 1, де Так – це 1, Ні – це 0.

Сумісність з .NET був визначений за наявністю та кількістю скачувань офіційних пакетів для користування на платформі .NET. Оскільки всього на вибір 6 баз даних, тому оцінимо їх по шкалі від 1 до 6. Там де 1 – це найменша кількість скачувань, 6 – це найбільша.

Таким чином, показники мають наступний вигляд (див.табл.2.2).

Наступним кроком використаємо правило Парето та оптимізуємо кількість існуючих варіантів. Правило звучить наступним чином: «Варіант а краще варіанту

b, якщо а хоча б за одним критерієм краще за b, а за іншими критеріями не гірше, ніж b». Таким чином, СКБД MSSQL за всіма критеріями краща та не гірша ніж PostgreSQL. СКБД Redis за всіма критеріями краща та не гірша ніж Memcached. Тому PostgreSQL та Memcached можна викреслити з існуючих варіантів.

Таблиця 2.2 – Опис СКБД за кількісними шкалами (таблиця створена самостійно)

	Популярність	Розподілене кешування	Сумісність з .NET	Розвиток СКБД	К-сть механізмів кешування
MySQL	1115.24	0	2	2	2
MSSQL	911.42	1	6	5.2	3
PostgreSQL	636.86	0	3	3	2
MongoDb	428.55	0	4	5.6	2
Redis	160.02	1	5	14.4	1
Memcached	19.80	1	1	4.8	1

Надалі слід визначити вагові коефіцієнти для показників. Для цього згадаємо вимогу встановлення вагових коефіцієнтів за формулою 1.2:

$$\sum_{j=0}^{m} \beta_j = 1, \tag{1.2}$$

де β_j – ваговий коефіцієнт.

Оновлена таблиця з ваговими коефіцієнтами має наступний вигляд (див.табл.2.3).

Розберемо кожний ваговий коефіцієнт. Критерій кількості механізмів кешування є найважливішим, тому що ми повинні розуміти яку кількість сценаріїв необхідно враховувати для кожної СКБД, тому цей критерій має значення 0.4.

Таблиця 2.3 – Опис, що підготовлений для використання згортки (таблиця створена самостійно)

	Популярність	Розподілене кешування	Сумісність з .NET	Розвиток СКБД	К-сть механізмів кешування
MySQL	1115.24	0	2	2	2
MSSQL	911.42	1	6	5.2	3
MongoDb	428.55	0	4	5.6	2
Redis	160.02	1	5	14.4	1
β	0.05	0.3	0.2	0.05	0.4

Розберемо кожний ваговий коефіцієнт. Критерій кількості механізмів кешування є найважливішим, тому що ми повинні розуміти яку кількість сценаріїв необхідно враховувати для кожної СКБД, тому цей критерій має значення 0.4.

Використання СКБД для розподіленого кешування є наступним за важливістю критерієм, тому що ми повинні розуміти, яка СКБД має вбудовані можливості використання кешування для масштабованих систем з великим навантаженням. Тому цей критерій має значення 0.3.

Сумісність з .NET є наступним за важливістю, оскільки слід розуміти як часто оновлюються офіційні пакети згідно з останніми вимогами платформи .NET, тому цей критерій має значення 0.1.

Популярність та розвиток СКБД критерії однакові за важливістю, тому мають значення по 0.05.

Далі знайдемо ті СКБД, які найбільш задовольняють нашим вимогам, враховуючи, що остаточна кількість СКБД повинна бути як мінімум 2, по 1 на кожний тип СКБД (реляційна та NoSQL). Для цього розрахуємо лінійну адитивну згортку з ваговими коефіцієнтами (див.рис.2.2).

	Популярність	Розподілене кешування	Сумісність з .NET	Розвиток СУБД	К-сть механізмів кешування	Z*
MySQL	1115,24	0	2	2,00	2	0,137
MSSQL	911,42	1	6	5,2	3	0,362
MongoDb	428,55	0	4	5,6	2	0,142
Redis	160,02	1	5	14,4	1	0,259
β	0,05	0,3	0,1	0,05	0,4	
α	0,00038	0,5	0,05882	0,04	0,125	

Рисунок 2.2 – Результати розрахунку (рисунок створено самостійно)

Лінійна адитивна згортка з ваговими коефіцієнтами була розрахована за формулою 1.3:

$$Z* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij}, \tag{1.3}$$

де α_j – нормуючі множники,

β_j – вагові коефіцієнти.

Таким чином, аналізуючи розрахунки, серед реляційних систем найкращим варіантом є MSSQL, а серед NoSQL є Redis. Проте за результатами слід виділити систему MongoDb, яка має багато цікавих можливостей для дослідження в рамках кешування, тому даний варіант також буде гарним варіантом для проведення дослідження.

2.2 Аналіз реалізації механізмів кешування в обраних СКБД

MS SQL Server має складну структуру реалізації кешування, яка поєднує декілька механізмів одночасно для зменшення навантаження на диск та процесор, а також для прискорення доступу до даних [9].

Основним механізмом кешування є буферний кеш, який зберігає сторінки даних та індексів в оперативній пам'яті. Це дозволяє значно прискорити доступ до даних, оскільки читання з пам'яті відбувається набагато швидше, ніж з диска.

Другим механізмом є кеш виконуваних планів запитів. MSSQL зберігає виконувані плани для запитів, щоб уникнути повторної компіляції запитів, які часто виконуються. При цьому оптимізація використання планів запитів досягається за допомогою перевірки ефективності та актуальності плану [14].

MSSQL підтримує memory-optimized таблиці та індекси, які зберігаються повністю в оперативній пам'яті. Для підтримки memory-optimized таблиць використовується технологія In-Memory OLTP, яка використовує різновидні структури даних та алгоритми для зберігання таблиць повністю в пам’яті.

За своєю структурою Redis зберігає дані у вигляді пар ключ-значення в оперативній пам'яті. Це дозволяє забезпечити дуже швидкий доступ до даних, оскільки запити обробляються безпосередньо в пам'яті [12].

Redis використовує декілька механізмів для безпосередньої роботи кешування. Least Recently Used механізм, де Redis видаляє ключі, які не використовувалися протягом значного відрізку часу, є однією з таких. Тим самим СКБД дозволяє зберігати лише найбільш актуальні дані в пам'яті.

Least Frequently Used механізм кешування видаляє дані з кешу, які найрідше використовуються. Це може бути корисно для систем, які використовують невеликий набір даних. Також даних механізм кешування поєднується з механізмом розпаду, який зменшує вагу даних з часом. Тим самим, СКБД вирішує, які дані стають менш важливими на частоті їх використання.

Redis підтримує розподілений кеш шляхом реплікації і шардінгу. Реплікація дозволяє мати копії даних на декількох серверах, що забезпечує вищу доступність та безпеку даних. Шардінг дозволяє розподілити навантаження кешування між кількома Redis серверами.

MongoDb забезпечує інакші механізми кешування від Redis, які надають можливості роботи зі складними запитами та аналізом даних, в той час як Redis краще підходить для випадків, коли потрібен дуже швидкий доступ до даних з низькою затримкою.

Основним механізмом зберігання даних у MongoDb є WiredTiger, який використовує кешування в оперативній пам'яті для підвищення продуктивності.

Він підтримує компресію даних і використання кеша для швидкого доступу до часто використовуваних даних [15].

WiredTiger використовує два типи кешування:

- внутрішній кеш для зберігання сторінок даних, що недавно використовувалися;

- плани запитів, щоб уникнути повторної компіляції планів для однакових запитів.

Підтримка функцій реплікації даних та шардінгу дозволяє збільшити обсяг кешу, розподіляючи дані між різними фізичними машинами та зменшуючи навантаження на кожний окремий сервер.

2.3 Планування експериментального дослідження

Проведення експериментального дослідження полягає у тому, щоб ефективно дослідити поведінку різних баз даних при застосуванні власних механізмів кешування. При цьому забезпечити окреме тестове середовище, яке не матиме зовнішніх чинників впливу на швидкість роботи баз даних, а також спроектувати тестову систему, яка допомбоже якісно провести дослідження для усіх баз даних одночасно.

Тому першим кроком буде налаштувати середовище для кожної з баз даних, а це MSSQL, Redis та MongoDb. Середовище рівномірно розподілятиме обчислювальні можливості, оперативну пам’ять та дисковий простір для усіх баз даних. В якості середовища буде використана локальна машина, яка буде забезпечувати рівномірний розподіл та максимальну ефективність проведення дослідження [16].

Надалі буде створена тестова система, яка перевірятиме ефективність кешування кожної зі СКБД на різних об’ємах даних, які будуть залучені в рамках однієї системи предметної області.

Окремою частиною дослідження є вибір відповідної предметної галузі, яка зможе досконально проявити усі можливості кешування кожної з СКБД. Тому

також важливо проаналізувати та промоделювати предметну область, створивши відповідні логічні моделі кожної з СКБД.

2.3.1 Вибір критеріїв з оцінювання якості стратегій кешування

Для визначення якості стратегій кешування необхідно сформулювати такі показники, які зможуть оцінити кешування у повній мірі.

Час виконання експерименту – це показник, який вимірює середній час виконання експерименту та повернення даних з кеш-пам’яті. Оскільки об’єм даних на різних рівнях тестування та обчислювальні можливості машини будуть однакові, даний показник зможе максимально точно визначити яка з СКБД справляється з експериментами найкраще.

Коефіцієнт попадання у кеш-пам’ять – це перший показник, який символізує частку успішних звернень до кеш-пам’яті в залежності від кількості запитів до кешу. Чим вищий коефіцієнт, тим більш ефективним є механізм кешування у базі даних. Даний коефіцієнт визначається за формулою 1.4:

$$H = \left(\frac{s}{r}\right) * 100\%, \tag{1.4}$$

де s – це кількість успішних спроб звернутись до кеш-пам’яті,
r – це загальна кількість запитів до кешу.

Коефіцієнт промахів у кеш-пам’ять – це показник, який символізує частку невдалих спроб звернення до кеш-пам’яті. Даний коефіцієнт визначається за формулою 1.5:

$$M = \left(\frac{f}{r}\right) * 100\%, \tag{1.5}$$

де f – це кількість невдалих спроб звернутись до кеш-пам’яті,
r – це загальна кількість запитів до кешу.

Використання ресурсів – це показник, який відстежує використання системних ресурсів, включаючи використання процесора та дискового простору під час виконання запитів. Чим вище використання ресурсів, тим більше вузьких місць існує у продуктивності бази даних.

Розмір кешу – це показник, який визначає кількість виділеної пам’яті для кешу у кожній базі даних. Таким чином, ми можемо зрозуміти чи достатньо виділено пам’яті для зберігання даних, до яких часто звертаються.

2.3.2 Аналіз та моделювання предметної області для дослідження

Для проведення дослідження було обрано область із купівлею речей через інтернет-магазин, або іншими словами, електронна комерція. Дана предметна область підходить для проведення дослідження через наявність великого об’єму даних, високого навантаження та складної структури.

Ключовими компонентами ландшафту електронної комерції є:

- цифрові платежі: електронна комерція значною мірою залежить від безпечних і ефективних цифрових платіжних систем. Кредитні картки, цифрові гаманці та інші способи онлайн-платежів відіграють вирішальну роль у забезпеченні безперебійних транзакцій;
- логістика: ефективність електронної комерції часто залежить від ефективного управління ланцюгом поставок. Від контролю запасів до виконання замовлень і логістики доставки, спрощені процеси є важливими для задоволення очікувань клієнтів;
- платформа електронної комерції: важливо мати структуровану та технічно-налагоджену платформу, в якій пропонують інструменти для переліку продуктів, обробки замовлень і часто містять такі функції, як відгуки клієнтів і рейтинги;
- безпека та довіра: враховуючи цифрову природу електронної комерції, безпека є критичною проблемою. Побудова довіри споживачів передбачає впровадження надійних заходів кібербезпеки для захисту конфіденційної інформації, такої як особисті та фінансові дані.

Тому як предметна галузь, електронна комерція ідеально підходить для проведення дослідження, оскільки вона охоплює величезний об’єм різноманітних нюансів та потребує детально-побудованої системи для забезпечення усіх потреб клієнта. Архітектура такої системи зазвичай потребує гнучкості та масштабованості, тому дана предметна галузь буде розглядатися у вигляді мікросервісної архітектури, при якій кожен з сервісів існує собі окремо, є незалежним та має змогу працювати зі своєю базою даних [17].

Проаналізувавши обрану предметну галузь, необхідно виділити основні сутності системи. Тематикою електронної комерції буде продаж різноманітних гаджетів та пристроїв для персонального користування, такі як навушники, смартфони та ноутбуки.

Сутність «Product» представляє собою продукт, який користувач має змогу придбати за встановленою ціною та у необхідній кількості. Продукт має категорію, яка представляється сутністю «Category».

Сутність «User» представляє собою користувача системи з інформацією про себе та власний акаунт.

Сутність «Basket» являє собою кошик, до якої користувач системи додає та обирає дозволену кількість продукту для придбання.

Сутність «Order» відповідає за аналіз вмісту кошику, оформлення замовлення, аналіз інформації про доставку та оплату.

Після виявлення сутностей побудуємо загальну діаграму класів (див.рис.2.3).

З даної діаграми видно, що Користувач створює Кошик, Кошик містить обраний Продукт, Продукт має Категорію, Користувач створює Замовлення, Замовлення містить обраний Продукт.

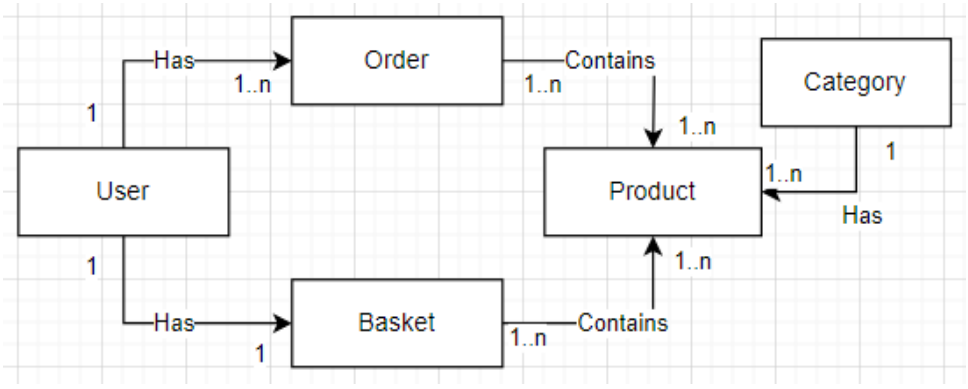

Рисунок 2.3 – Діаграма класів (рисунок створено самостійно)

2.3.3 Проектування баз даних для дослідження

2.3.3.1 Розробка інфологічної моделі бази даних

Провівши аналіз предметної галузі та визначивши основні класи системи, необхідно спроектувати схему бази даних, яка буде використовуватись для проведення дослідження.

Отже на основі предметної області слід виділити 7 сутностей, а саме: «User», «Category», «Product», «Basket», «Product_Basket», «Order» та «Product_Order».

Розглянемо сутності та створимо ER-діаграму, як попередній, перед логічним моделюванням, етап (див.рис.2.4).

Сутність «Basket» має наступні атрибути, такі як: унікальний ідентифікатор (Basket_Id), загальна кількість речей (Total_quantity) та загальна ціна (Total_price).

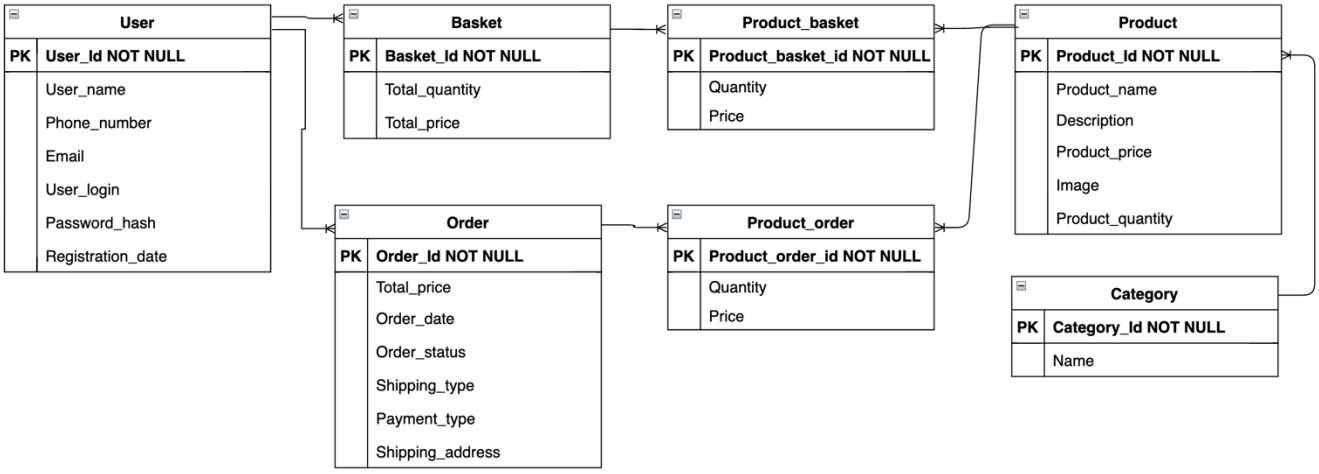

Рисунок 2.4 – ER-діаграма бази даних (рисунок створено самостійно)

Сутність «Category» має наступні атрибути, такі як: унікальний ідентифікатор (Category_Id), назва (Name).

Сутність «Product» має наступні атрибути, такі як: унікальний ідентифікатор (Product_Id), назва (Product_name), опис (Description), ціна (Product_price), зображення продукту (Image), наявна кількість (Product_quantity).

Сутність «User» має наступні атрибути, такі як: унікальний ідентифікатор (User_Id), ім’я (User_name), номер телефону (Phone_number), електронна поштова адреса (Email), нікнейм (User_login), пароль (Password_hash), та дата реєстрації (Registration_date).

Сутність «Product_basket» має наступні атрибути, такі як: унікальний ідентифікатор (Product_basket_Id), кількість речей (Quantity) та ціна (Price).

Сутність «Order» має наступні атрибути, такі як: унікальний ідентифікатор (Order_Id), загальна ціна (Total_price), дата замовлення (Order_date), статус замовлення (Order_status), метод доставки (Shipping_type), метод оплати (Payment_type) та адреса доставки (Shipping_address).

Сутність «Product_basket» має наступні атрибути, такі як: унікальний ідентифікатор (Product_basket_Id), кількість речей (Quantity) та ціна (Price).

2.3.3.2 Розробка схеми бази даних для MS SQL Server

На основі розробленої ER-діаграми створимо схему реляційної бази даних для подальшого дослідження MS SQL Server (див.рис.2.5).

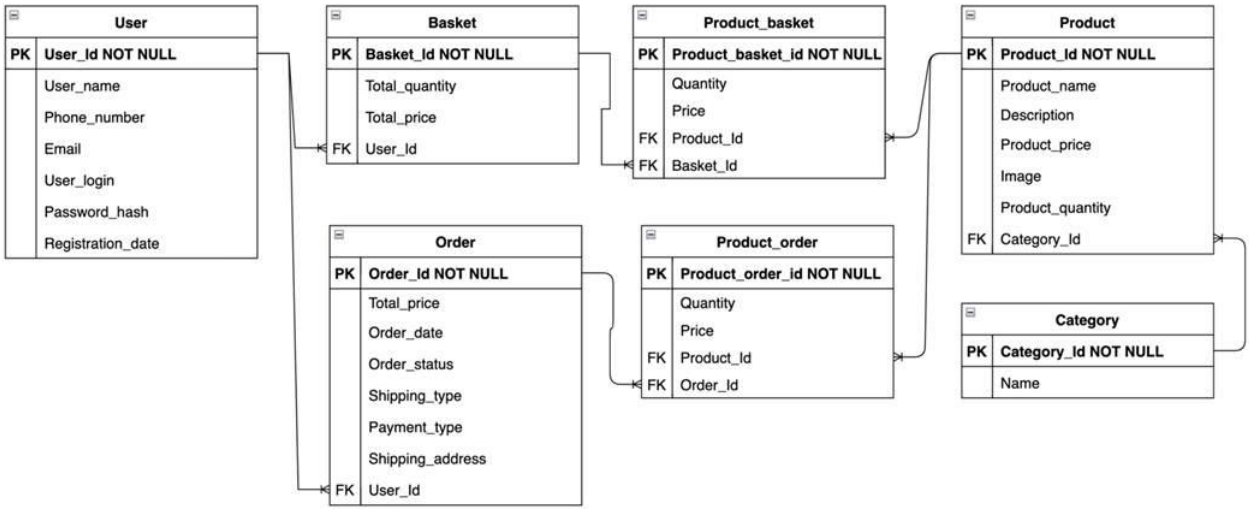

Рисунок 2.5 – Схема реляційної бази даних (рисунок створено самостійно)

База даних знаходиться у третій нормальній формі, так як вона знаходиться у другій та першій нормальній формі, неключові атрибути повністю функціонально залежать від первинних ключових атрибутів сутності та відсутні будь-які транзитивні залежності [18].

2.3.3.3 Розробка логічної моделі для Redis

Після створення схеми для реляційної бази даних, необхідно зазначити логічну модель для Redis та MongoDb, обидві з яких за своєю природою є NoSQL базами даних [19]. У Redis запис даних визначається за допомогою пари «ключ-значення», а у MongoDb за допомогою документів, які зберігаються у форматі подібному до JSON під назвою BSON [20].

Оскільки нам необхідно створити структуру даних подібну до таблиць в реляційній базі даних, то в випадку з Redis буде використовуватись структура даних хеш. Завдяки ній ми маємо змогу створити один ключ та декілька значень для кожного поля певної сутності (див.рис.2.6).

Хеш є найкращим вибором серед інших запропонованих структур даних у Redis, оскільки це надає можливість отримати прямий доступ до потрібного поля, просто використовуючи назву поля у відповідному ключі.

Рисунок 2.6 – Структура хешу (рисунок створено самостійно)

Тим самим зникає потреба у рендерінгу цілої колекції, що значно пришвидшує час роботи СКБД.

Таким чином, логічна модель для обох СКБД має наступний вигляд (див.рис.2.7).

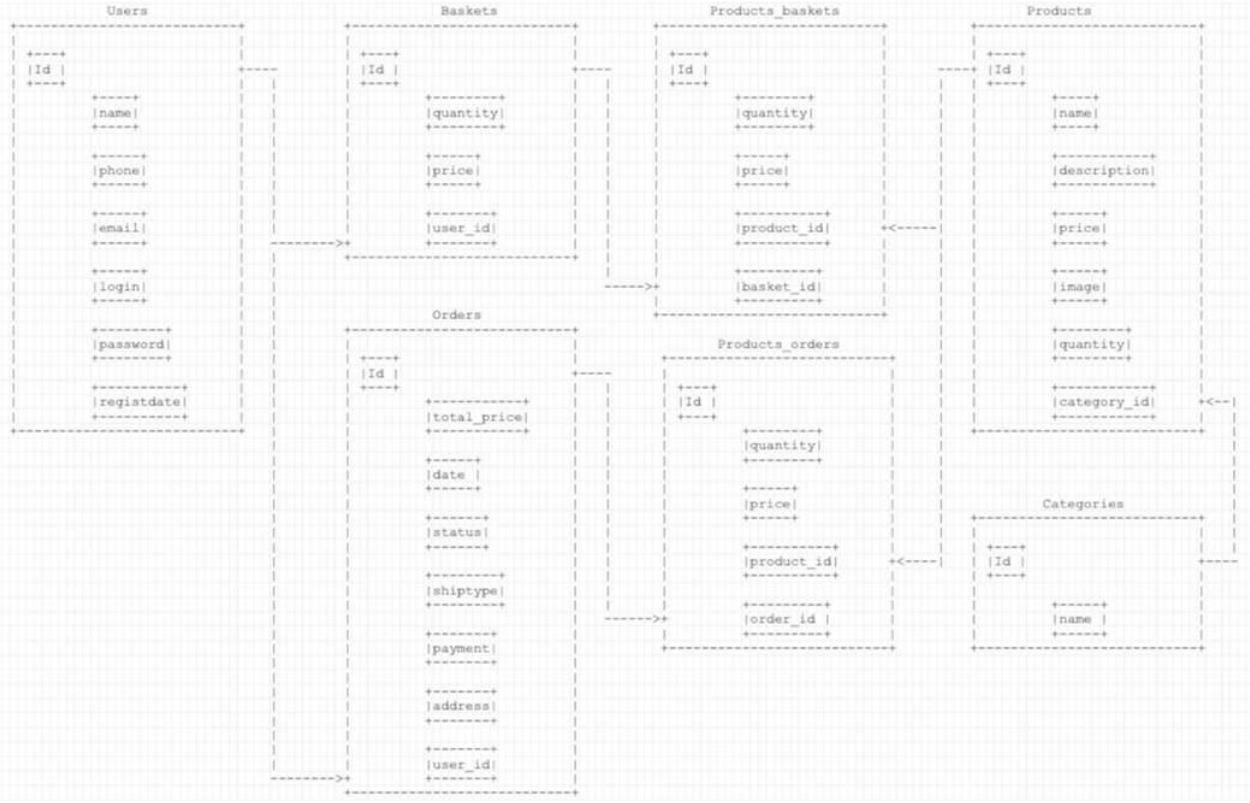

Рисунок 2.7 – Логічна модель бази даних Redis (рисунок створено самостійно)

Схема відображає усі існуючі ключі, які матимуть відповідні значення при використані цієї бази даних. Також за допомогою унікальних ідентифікаторів зображені зв’язки між сутностями.

2.3.3.3 Розробка логічної моделі для MongoDb

Наступним кроком створимо логічну модель для MongoDb (див.рис.2.8). Специфікою побудови колекцій у MongoDb є підтримка вбудованих колекцій, проте ми хочемо зберігати однаковий стиль побудови моделей у кожній СКБД.

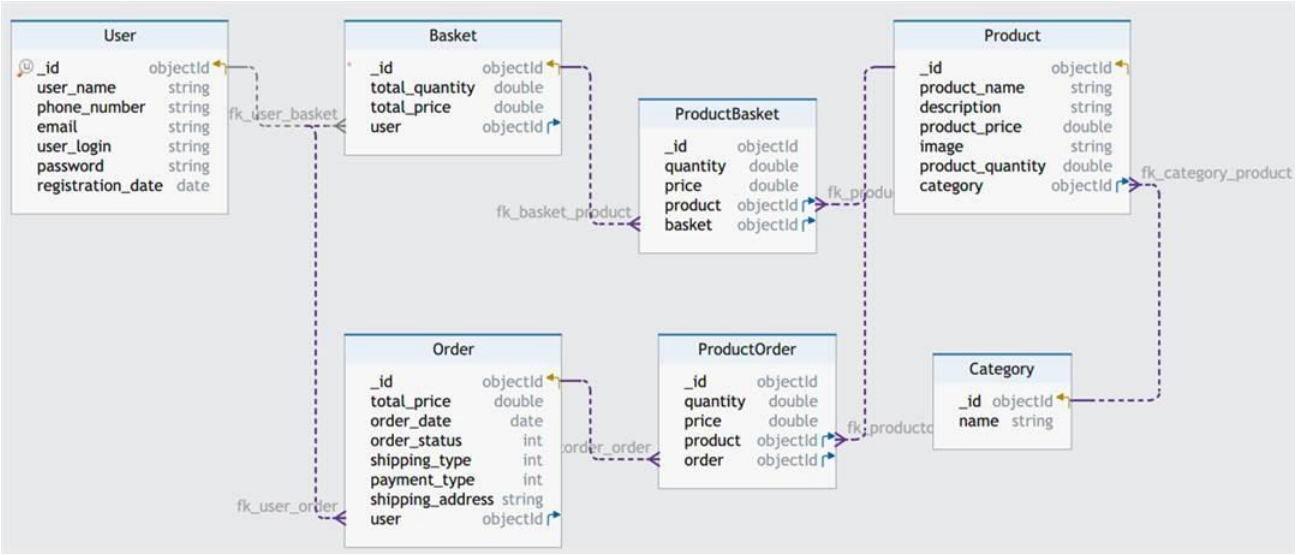

Рисунок 2.8 – Логічна модель MongoDb (рисунок створено самостійно)

Якщо у реляційних базах даних існують таблиці та поля, то MongoDb відображає дані у вигляді колекцій та документів у форматі BSON. Особливістю кожного документу є зберігання унікального ідентифікатора у форматі objectId розміру 12 байт, який складається з:

- 4-байтової позначки часу створення ідентифікатора;
- 5-байтового унікального значення;
- 3-байтового інкрементного лічильника.

Логічна модель, побудована вище відображає первинні ключі та посилання на атрибути типу objectId, а також зв’язки між колекціями [21].

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1 Опис фізичної моделі БД для MSSQL

Для проведення експерименту необхідно розробити фізичні моделі для кожної з СКБД. Тому треба створити відповідні SQL таблиці, які будуть використані для виконання запитів дослідження [8].

Запит для створення незалежних таблиць «User» та «Category» має наступний вигляд:

```
CREATE TABLE User (
     Id UNIQUEIDENTIFIER PRIMARY KEY,
     Name VARCHAR(255),
     MiddleName VARCHAR(255),
     Surname VARCHAR(255),
     BirthDate DATE,
     Address VARCHAR(255),
     PhoneNumber VARCHAR(255),
     Email VARCHAR(255),
     Username VARCHAR(255) NOT NULL,
     PasswordHash VARCHAR(255) NOT NULL,
     ProfileImage VARCHAR(255),
     RegistrationDate DATETIME DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE Category (
    Id UNIQUEIDENTIFIER PRIMARY KEY,
    Name VARCHAR(255)
);
```

Далі маємо запити для створення інших таблиць, кожна з яких має відношення з іншими у вигляді 1:M або 1:1.

```
CREATE TABLE Product (
  Id UNIQUEIDENTIFIER PRIMARY KEY,
  Name VARCHAR(255),
  Quantity INT,
  CategoryId UNIQUEIDENTIFIER,
  FOREIGN KEY (CategoryId) REFERENCES Category (Id)
);

CREATE TABLE Basket (
    Id UNIQUEIDENTIFIER PRIMARY KEY,
    TotalPrice INT,
    TotalItems INT,
    UserId UNIQUEIDENTIFIER,
    FOREIGN KEY (UserId) REFERENCES User (Id)
);
```

```
CREATE TABLE ProductBasket (
   Id UNIQUEIDENTIFIER PRIMARY KEY,
   Quantity INT,
   BasketId UNIQUEIDENTIFIER,
   ProductId UNIQUEIDENTIFIER,
   FOREIGN KEY (BasketId) REFERENCES Basket (Id),
   FOREIGN KEY (ProductId) REFERENCES Product (Id)

);


CREATE TABLE Order (
    Id UNIQUEIDENTIFIER PRIMARY KEY,
    TotalPrice INT,
    OrderDate DATETIME,
    OrderStatus INT,
    ShippingType INT,
    PaymentType INT,
    ShippingAddress VARCHAR(255),
    UserId UNIQUEIDENTIFIER,
    FOREIGN KEY (UserId) REFERENCES User (Id)
);


CREATE TABLE ProductOrder (
   Id UNIQUEIDENTIFIER PRIMARY KEY,
   Quantity INT,
   OrderId UNIQUEIDENTIFIER,
   ProductId UNIQUEIDENTIFIER,
   FOREIGN KEY (OrderId) REFERENCES Order (Id),
   FOREIGN KEY (ProductId) REFERENCES Product (Id)

);
```

Таким чином, всього створено 7 таблиць для бази даних MSSQL. Слід зазначити, що дані скрипти для створення таблиць є релевантними лише для реляційних баз даних.

3.2 Опис фізичної моделі БД для Redis

Оскільки Redis – це NoSQL база даних типу «ключ-значення», яка працює як структура даних в пам’яті, то неможливо створити заздалегідь підготовлені таблиці, відношення та ключі як це необхідно робити з реляційними базами даних [22].

Проте Redis має чимало структур даних, які можна використовувати для встановлення ключей та їх значення. Серед списку існують рядки, хеші, JSON, списки та сортовані множини [12]. Найбільш зручною структурою даних, беручи

до уваги обрану предметну галузь, є хеши. Завдяки їх використанню, фактична кількість полів, яка може бути поміщена у хеш, є необмеженою (обмеження існують лише за доступною пам’яттю) [23].

Наприклад, маємо SQL скрипт для додавання нового рядку даних для таблиці «ProductBasket», вказуючи унікальний ідентифікатор, кількість обраного товару, унікальний ідентифікатор обраного товару та ідентифікатор корзини.

```
INSERT INTO `ProductBasket` (`id`, `quantity`, `productId`, `basketId`)
VALUES('bea54e06-32d5-4be0-a179-cebe0511dcca', '1210, 'f56be677-4875-
4f6a-a35e-717100e18175', 'c67ad377-4875-4f6a-a35e-717112c19264');
```

Для Redis, використовуючи хеш як обрану структуру даних, це виглядатиме наступним чином:

```
HSET ProductBasket bea54e06-32d5-4be0-a179-cebe0511dcca quantity 1210
HSET ProductBasket bea54e06-32d5-4be0-a179-cebe0511dcca productId
f56be677-4875-4f6a-a35e-717100e18175
```

Ми вказуємо обрану структуру даних, назву хешу, унікальний ключ (в нашому випадку це поле Id), поле та його значення. Саме таким чином необхідно створювати скрипти для роботи з Redis.

3.3 Опис фізичної моделі БД для MongoDb

MongoDb – це документо-орієнтована база даних, яка зберігає документи у форматі BSON, які пізніше згруповані у колекції. Порівнюючи з реляційними базами даних, колекції – це таблиці, рядки яких представлені у вигляді документів формату BSON [15].

Слід зазначити, що MongoDb використовує мову запитів під назвою MongoDb Query Language (MQL). Однак важливо зазначити, що MQL не є окремою мовою, як SQL. Замість цього запити MongoDB виражаються за допомогою синтаксису, який нагадує JSON [24].

Розглянемо попередній приклад з додаванням нових даних до існуючої колекції. Скрипт для додавання нового значення матиме наступний вигляд:

```
db.ProductBasket.insertOne({
  "_id": "bea54e06-32d5-4be0-a179-cebe0511dcca",
  "quantity": 1210,
  "productId": "f56be677-4875-4f6a-a35e-717100e18175",
  "basketId": "c67ad377-4875-4f6a-a35e-717112c19264"
})
```

Ми використовуємо команду insertOne, попередньо вказуючи базу даних, до якої ми звертаємось та назву колекції до якої додаються нові дані. Після цього вказуємо значення для кожного унікального поля.

3.4 Розробка програмного застосунку для тестування

Для того, щоб ефективно проводити експериментальні дослідження, необхідно створити експериментальне середовище, яке матиме змогу відобразити роботу обраних баз даних, серверної та UI застосунків.

Для дослідження було розроблено чотири бази даних, event bus, для яких було створено відповідні конфігураційні файли Docker, docker-compose та були запущені на тестовому середовищі (див.рис.3.1).

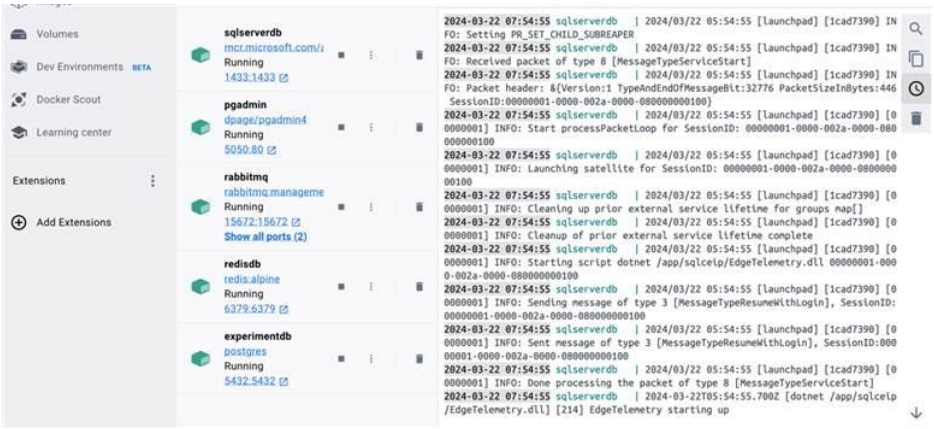

Рисунок 3.1 – Контейнери баз даних та супроводжуючих сервісів (рисунок створено самостійно)

Наступним кроком був розроблений серверний та UI додатки, кожен з яких використовується як незалежні сервіси, які поєднані через HTTP запити. Слід зазначити, що для захищеного спілкування між додатками були сформовані SSL сертифікати та доданий флаг HttpOnly для блокування можливих XSS атак [25].

Архітектура серверного додатку була обрана мікросервісна задля розробки гнучкої та швидкодійної системи [17]. Кожен з мікросервісів має власну базу даних для ефективного та захищеного зберігання даних.

Для зручного користування системою був створений Identity мікросервіс, який дозволяє аутентифікувати та авторизувати користувача системи (див.рис.3.2).

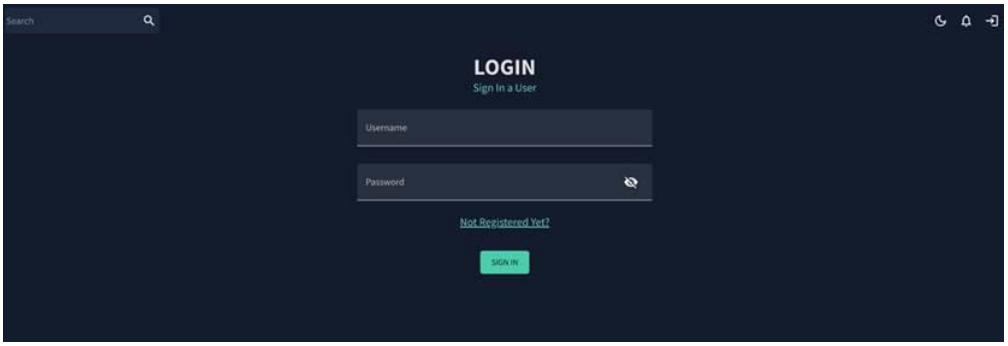

Рисунок 3.2 – Сторінка авторизації (рисунок створено самостійно)

Далі була створена сторінка конфігурації даних, на якій користувач має змогу додати файл типу json для підготовки системи до проведення досліджень. Як було зазначено, предметна галузь для проведення досліджень була обрана електронна комерція. Окрім додання даних, користувач має змогу побачити які дані були завантажені або вже існують у системі (див.рис.3.3).

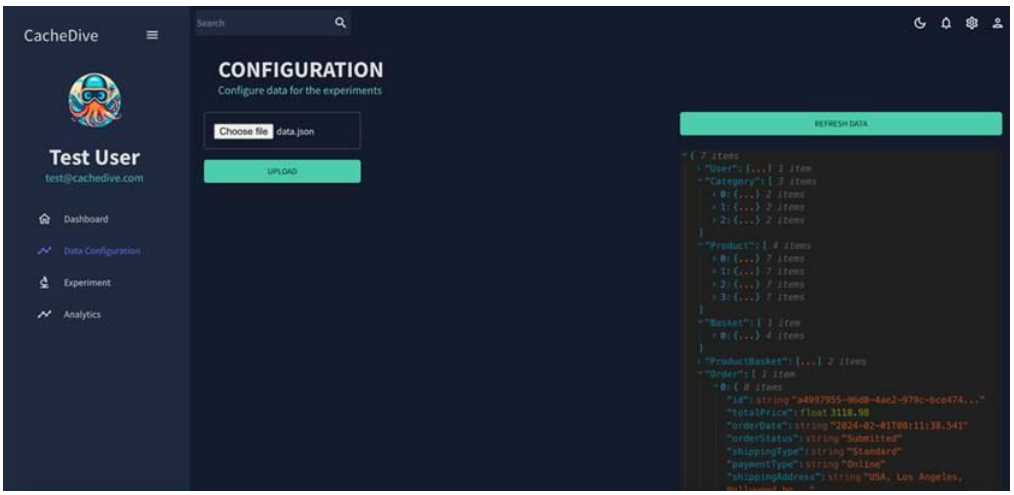

Рисунок 3.3 – Сторінка конфігурації даних для експериментів (рисунок створено самостійно)

Після конфігурації даних користувач має змогу провести експерименти, заповнивши відповідні поля та для конкретної бази даних. Необхідно зазначити,

що користувач може встановити кількість виконання запиту та чи необхідно очистити кеш перед виконанням (див.рис.3.4).

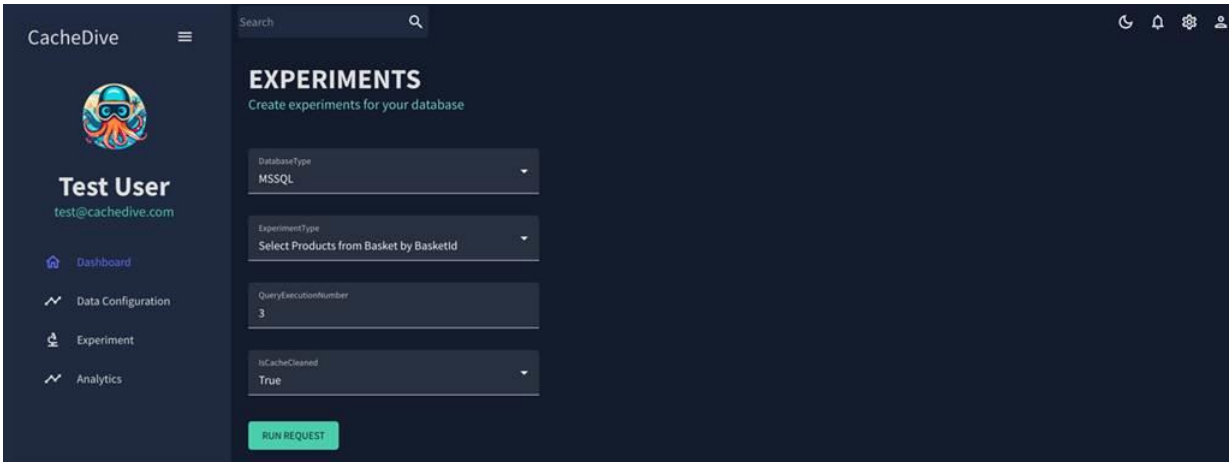

Рисунок 3.4 – Створення експерименту (рисунок створено самостійно)

Після виконання експерименту, користувач має змогу переглянути результати виконання, ознайомитися з більш конкретними деталями виконання, враховуючи усі критерії оцінювання якості стратегій кешування (див.рис.3.5).

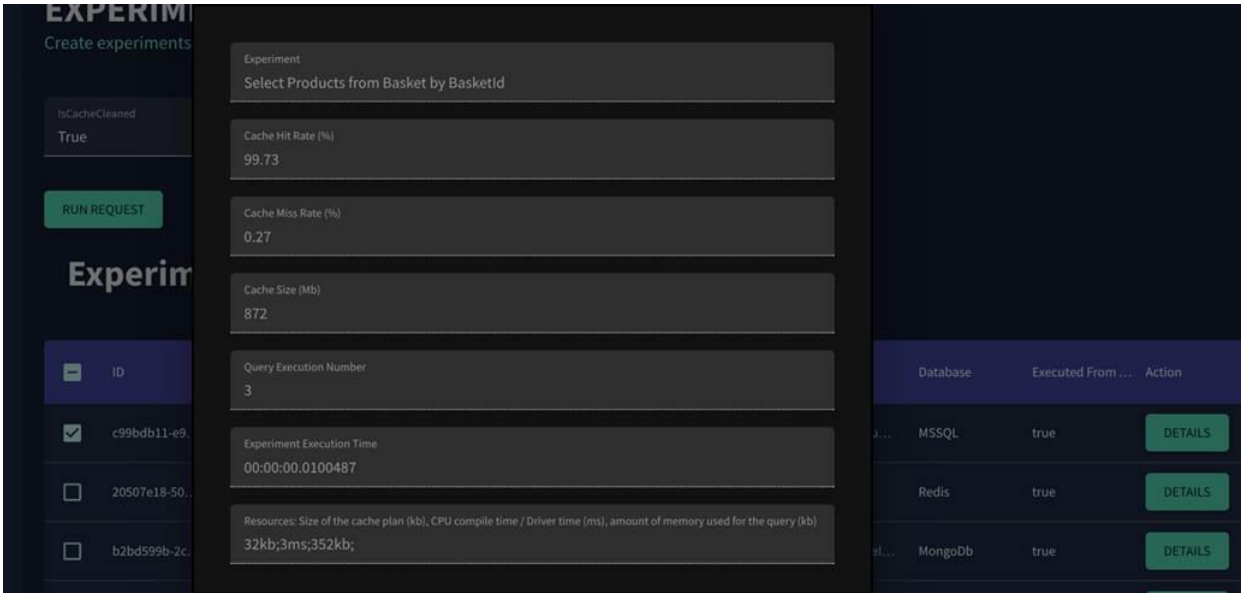

Рисунок 3.5 – Перегляд результатів проведення експерименту (рисунок створено самостійно)

Надалі користувач має змогу переглянути аналітику щодо проведення експериментів для кожної з баз даних, визначити яка база даних ефективніше справляється з механізмами кешування.

4 ОПИС ЕКСПЕРИМЕНТАЛЬНОГО ДОСЛІДЖЕННЯ

4.1 Проведення експериментального дослідження

4.1.1 Завантаження даних та визначення запитів

Для проведення експериментального дослідження необхідно завантажити набір даних відповідно до описаної схеми баз даних. Для цього використаємо формат JSON, який є текстовим форматом для зберігання структурованих даних [24].

Оскільки маємо 7 сутностей для заповнення даними, тому частина тестових даних має наступний вигляд:

```
{
  "entity": "Product",
  "data": {
    "id": "a5003afc-4005-4edc-9725-22aa87ecdeaf",
    "name": "Marshall Major IV",
    "description": "Major IV delivers the signature Marshall sound
that you have come to expect.",
    "price": 179.00,
    "image": "marshalls.png",
    "quantity": 250,
    "categoryId": "a5003afc-4005-4edc-9725-22aa87ecdeaf"
  }
},
{
  "entity": "Basket",
  "data": {
    "id": "6c780439-2b12-44d4-866f-bc1b500fab55",
    "totalQuantity": 3,
    "totalPrice": 3018.98,
    "userId": "2b9b47ba-958b-4406-a766-fcd25dfc5764"
  }
}
```

Слід зазначити, що для проведення експериментів необхідно зосередитися на декількох сутностях, таких як Product, Basket та Order. Як вже було зазначено, типовою архітектурою для створення системи електронної комерції є мікросервісна архітектура. Саме тому, кожну з обраних сутностей слід розглядати як незалежний сервіс зі своєю структурою та логікою реалізації. Додатковим аргументом вибору таких сутностей є те, що саме через них йде основне навантаження в електронній комерції.

Для тестування кожної бази даних варто визначити операції читання, які будуть використані для тестування ефективності кешування. Також для проведення тестування буде використано створений UI застосунок. Розглянемо найпопулярніші ситуації, за яких користувач або продавець прагне отримати швидку відповідь від застосунку.

Розглянемо сутність Product. Перша ситуація – це знаходження переліку продуктів за категорією «Смартфони» та відсортувавши за найвищою ціною. Дана ситуація описує можливість фільтрації та сортування продуктів у системі.

Наступна ситуація – це перелік продуктів за проміжком цін. Така ситуація також має право на тестування, оскільки є доволі простою у реалізації і кожна з баз даних може по-своєму оброблювати найпростіші запити.

Третя ситуація – це більш складна фільтрація продуктів, оскільки буде використано декілька атрибутів, а саме знаходження переліку продуктів бренду «Apple», ціна поміж 1000 і 2000 умовних одиниць та сортування за назвою продукту.

Розглянемо сутність Basket. Перша ситуація – це вибір продуктів, які вже додані до корзини за унікальним ідентифікатором сутності Basket. Для цієї ситуації буде використана сутність ProductBasket.

Друга ситуація – це вибір продуктів, доданих до корзини, кожного користувача. Така ситуація допоможе отримати колекцію користувачів та їх обраних продуктів.

Третя ситуація – це прорахунок загальної суми для продуктів, доданих до корзини, за унікальним ідентифікатором сутності Basket. Дана ситуація показує можливості використання агрегатних функцій для надання чіткої інформації користувачу про його майбутні витрати.

Розглянемо сутність Order. Оскільки попередні сутності врахували більшість атрибутів та функцій, які можна використати для виконання запитів, розглянемо лише одну типову ситуацію в рамках замовлення. Це прорахунок загальної суми витрат оплачених замовлень з групуванням кожного користувача системи. Така

ситуація підходить для аналітичних даних аби зрозуміти скільки усього коштів було витрачено на товари в магазині.

4.1.2 MSSQL

Після формування даних та експериментів сформуємо запити для бази даних MSSQL та протестуємо кожний з них завдяки UI застосунку. Слід зазначити, що для проведення експериментів, кожен запит буде виконуватися тричі, час виконання – це середня величина поміж кількості виконання. Для більш детальної перевірки ефективності кешування для усіх запитів, кеш бази даних буде очищено лише один раз, кожен наступний запит тільки збільшуватиме розмір кешу.

Перший запит до сутності Product, в якому ми знаходимо перелік продуктів категорії «Смартфони», має наступний вигляд:

```
SELECT p.Id, p.Name FROM Products p  JOIN Categories c ON p.CategoryId
= c.Id WHERE c.Name = 'Smartphones' ORDER BY p.Price;
```

В даному запиті використовуються дві таблиці: Product та Category.

Наступний запит, в якому необхідно знайти перелік продуктів за проміжком цін, має такий вигляд:

```
SELECT p.Id, p.NAME  FROM Products p WHERE p.Price > 119.00 AND Price
< 189.00;
```

Запит є доволі простим та розповсюдженим для користувачів електронної комерції.

Ще один запит до сутності Product, в якому необхідно знайти товари бренду «Apple» та встановити ціну поміж 1000 і 2000 умовних одиниць, є таким:

```
SELECT p.Id, p.Name FROM Products p WHERE p.NAME LIKE '%Apple%' AND
p.Price BETWEEN 1000.00 AND 2000.00 ORDER BY p.Name;
```

Окрім фільтрації, запит також має сортування за назвою товару. Такий запит створено для перевірки складної фільтрації та сортування в рамках кешування.

Наступний запит відноситься до сутностей Basket та ProductBasket, де необхідно встановити перелік продуктів, вже доданих до корзини, за вказаним унікальним ідентифікатором Basket. Побудований запит таким чином:

```
SELECT p.Id AS Product_Id, p.Name as Product_Name, pb.Quantity, b.Id AS Basket_Id FROM Products p JOIN ProductBaskets pb ON p.Id = pb.ProductId JOIN Baskets b ON pb.BasketId = b.Id WHERE b.Id = '6c780439-2b12-44d4-866f-bc1b500fab55';
```

Далі сформуємо запит для створеної ситуації, в якій необхідно вибрати продукти, доданих до корзини, кожного користувача. Запит має наступний вигляд:

```
SELECT p.Id, p.Name, pb.Quantity, u.Id FROM Products p JOIN ProductBaskets pb on p.Id = pb.ProductId JOIN Baskets b on pb.BasketId = b.Id JOIN Users u on b.UserId = u.Id;
```

Такий запит повертає набір даних, що допоможе зрозуміти, яким чином різні механізми кешування працюють з великим об’ємом даних.

Наступний запит – це використання функції SUM для знаходження загальної суми для обраних товарів за унікальним ідентифікатором сутності Basket. Запит має таку структуру:

```
SELECT SUM(pb.Price * pb.Quantity)  FROM ProductBaskets pb JOIN Baskets b ON pb.BasketId = b.Id WHERE b.Id = '6c780439-2b12-44d4-866f-bc1b500fab55';
```

Останній запит для проведення експериментів – це запит до сутностей Order та User для визначення загальної суми витрат замовлень, які вже були оплачені користувачами. Він має такий вигляд:

```
SELECT SUM(o.TotalPrice), u.Name, u.Email FROM Orders o JOIN Users u ON u.Id = o.UserId WHERE o.OrderStatus = 3 GROUP BY u.Name, u.Email;
```

З результатами виконання запитів для MS SQL Server можна ознайомитися нижче (див.табл.4.1).

Таблиця 4.1 – Результати виконання експериментів для MSSQL (таблиця створена самостійно)

№	CacheHit Rate %	CacheMiss Rate %	CacheSize Mb	Execution Number	Time ms	Resources Kb, ms, Kb
1	99.75	0.025	904	3	1.0255	32;2
2	0	100	-	-	1.750	-
3	99.93	0.07	7827	3	3.173	24;7
4	99.77	0.23	8168	3	1.6856	32;18
5	99.81	0.19	8488	3	2.2742	40;8
6	99.78	0.22	8744	3	2.3635	32;6
7	99.80	0.20	9088	3	4.7286	32;21

У даній таблиці відображені результати ефективності механізму кешування у базі даних MSSQL за визначеними метриками. Після проведення експериментів на кожній з баз даних буде проведений аналіз метрик та створені рекомендації.

4.1.3 Redis

Наступною базою даних для проведення експериментів є Redis. Оскільки дана СКБД є типу «ключ-значення», а кожен експеримент має доволі складну структуру, то для їх виконання будуть використанні вбудовані можливості бібліотеки StackExchange.Redis на платформі .NET.

Необхідно зазначити, що дана СКБД не підтримує функціонал очищення кешу, оскільки користувач може лише додати чи видалити пари ключ-значення.

Розглянемо перший запит до сутності Product, як приклад реалізації експерименту за допомогою можливостей бібліотеки StackExchange.Redis.

```
public async Task<List<dynamic>>
GetProductsFromBasketByBasketId(string basketId = "6c780439-2b12-44d4-
866f-bc1b500fab55")
{
    List<HashEntry[]> productBaskets = new();
    List<dynamic> result = new();
```

```
    List<RedisKey> keys = _server.Keys(pattern:
$"ProductBasket:*").ToList();

    foreach (var key in keys)
    {
        HashEntry[] hashEntries = await
_connectionMultiplexer.GetDatabase().HashGetAllAsync(key);

        if (hashEntries.Any(hashEntry => hashEntry.Value == basketId))
        {
            productBaskets.Add(hashEntries);
        }
    }

    foreach (var productId in productBaskets.Select(productBasket =>
productBasket.FirstOrDefault(property => property.Name ==
"ProductId")))
    {
        Product? product = await
_unitOfWork.Products.GetByKeyAsync($"{nameof(Product)}:{productId.Valu
e}");

        if (product is not null)
        {
            var resultObject = new
            {
                Product = product,
                BasketId = basketId
            };
            result.Add(resultObject);
        }
    }

    result.Add(resourcesResult);

    return result;
}
```

Метод знаходить товари, які вже були додані до корзини за унікальний ідентифікатором корзини.

З результатами виконання запитів для Redis можна ознайомитися нижче (див.табл.4.2).

Слід зазначити, що коефіцієнти попадання та промаху мають однакові результати серед усіх експериментів через структуру Redis, в якій коефіцієнт промаху буде більше нуля за умови відсутності відповідних ключів при виконанні експерименту.

Таблиця 4.2 – Результати виконання експериментів для Redis (таблиця створена самостійно)

№	CacheHit Rate %	CacheMiss Rate %	CacheSize Mb	Execution Number	Time ms	Resources Kb, ms
1	100	0	0.32	3	25.785	2.63;9.11
2	100	0	0.34	3	47.489	4.67;7.79
3	100	0	0.34	3	58.528	1.094;6.4
4	100	0	0.34	3	109.495	6.979;5.2
5	100	0	0.36	3	50.376	3.146;7.04
6	100	0	0.36	3	85.673	4.751;4.45
7	100	0	0.36	3	62.363	2.016;13.66

Отримані результати виконання експериментів будуть проаналізовані та допоможуть визначити ефективність механізму кешування.

4.1.4 MongoDb

Надалі протестуємо кешування у СКБД MongoDb, яка відзначається своєю структурою у вигляді документно-орієнтованої бази даних зі збереженням файлів у форматі BSON.

Для проведення експериментів використаємо вбудовані можливості бібліотеки MongoDb.Driver на платформі .NET та команди mongosh, середовище для роботи з MongoDb, для виконання запитів.

Перший експеримент який відноситься до Product, для знаходження переліку продуктів категорії «Смартфони», має наступний вигляд:

```
db.Product.aggregate([
  {
    $match: {
      "Name": "<categoryName>"
    }
  },
  {
```

```
    $lookup: {
      from: "Product",
      localField: "_id",
      foreignField: "CategoryId",
      as: "products"
    }
  },
  {
    $unwind: "$products"
  },
  {
    $project: {
      "Id": "$products.Id",
      "Name": "$products.Name",
      "_id": 0
    }
  },
  {
    $sort: {
      "Price": 1
    }
  }
])
```

Даний та наступні запити використовують метод aggregate, який обчислює сукупні значення для даних у колекції. Як аргумент, цей метод приймає послідовність етапів або операцій агрегації даних.

Другим запитом є знаходження переліку продуктів за проміжком цін:

```
db.Product.aggregate([
  {
    $match: {
      $and: [
        { "Price": { $gt: startPrice } },
        { "Price": { $lt: endPrice } }
      ]
    }
  },
  {
    $project: {
      "Id": "$Id",
      "Name": "$Name",
      "_id": 0
    }
  }
])
```

Останнім запитом до сутності Product є запит знаходження товарів бренду «Apple» за ціною поміж 1000 і 2000 умовних одиниць:

```
db.collection.aggregate([
  {
    $match: {
      $and: [
        { "Name": { $regex: productName, $options: "i" } },
        { "Price": { $gte: startPrice, $lte: endPrice } }
      ]
    }
  },
  {
    $project: {
      "Id": "$Id",
      "Name": "$Name",
      "_id": 0
    }
  },
  {
    $sort: {
      "Name": 1
    }
  }
])
```

Далі серія запитів відносяться до сутностей Basket та ProductBasket. Перший із запитів – знайти перелік товарів, доданих до корзини, за унікальним ідентифікатором корзини:

```
db.ProductBasket.aggregate([
  {
    $match: {
      "BasketId": "<basketId>"
    }
  },
  {
    $lookup: {
      from: "Product",
      localField: "ProductId",
      foreignField: "_id",
      as: "product"
    }
  },
  {
    $unwind: "$product"
  },
  {
    $project: {
      "Product_Id": "$product.Id",
      "Product_Name": "$product.Name",
      "Quantity": "$Quantity",
      "Basket_Id": "$BasketId"
    }
```

```
  }
])
```

Наступний запит є більш складнішим, хоч і подібний за структурою, це знайти продукти, додані до корзини, у кожного користувача. Цей запит повертає значно більшу колекцію даних за об’ємом.

```
db.ProductBasket.aggregate([
  {
    $lookup: {
      from: "Product",
      localField: "ProductId",
      foreignField: "_id",
      as: "product"
    }
  },
  {
    $unwind: "$product"
  },
  {
    $lookup: {
      from: "Basket",
      localField: "BasketId",
      foreignField: "_id",
      as: "basket"
    }
  },
  {
    $unwind: "$basket"
  },
  {
    $lookup: {
      from: "User",
      localField: "basket.UserId",
      foreignField: "_id",
      as: "user"
    }
  },
  {
    $project: {
      "ProductId": "$product.Id",
      "ProductName": "$product.Name",
      "Quantity": "$Quantity",
      "UserId": { $arrayElemAt: ["$user.Id", 0] }
    }
  }
])
```

Останній запит до сутності ProductBasket – це прорахунок загальної суми обраних товарів за унікальним ідентифікатором сутності Basket:

```
db.ProductBasket.aggregate([
  {
    $match: {
      "_id": basketId
    }
  },
  {
    $lookup: {
      from: "ProductBasket",
      localField: "_id",
      foreignField: "BasketId",
      as: "productBaskets"
    }
  },
  {
    $unwind: "$productBaskets"
  },
  {
    $project: {
      "TotalPrice":     {     $multiply:     ["$productBaskets.Price",
"$productBaskets.Quantity"] }
    }
  },
  {
    $group: {
      "_id": "$productBaskets._id",
      "Total": { $sum: "$TotalPrice" }
    }
  }
])
```

Останнім експериментом є запит до сутностей Order та User для визначення загальної суми витрат замовлень, які вже були оплачені користувачами:

```
db.Order.aggregate([
  {
    $match: {
      "Name": categoryName
    }
  },
  {
    $lookup: {
      from: "Product",
      localField: "_id",
      foreignField: "CategoryId",
      as: "products"
    }
  },
  {
    $unwind: "$products"
  },
  {
```

```
      $project: {
        "Id": "$products.Id",
        "Name": "$products.Name",
        "_id": 0
      }
    },
    {
      $sort: {
        "Price": 1
      }
    }
])
```

З результатами виконання запитів для MongoDb можна ознайомитися нижче (див.табл.4.3).

Таблиця 4.3 – Результати виконання експериментів для MongoDb (таблиця створена самостійно)

№	CacheHit Rate %	CacheMiss Rate %	CacheSize Mb	Execution Number	Time ms	Resources Kb, ms
1	98.72	1.28	0.98	3	2.748	0.29;6
2	98.73	1.27	0.99	3	0.8419	0.16;3
3	98.8	1.2	1	3	0.9769	0.21;11
4	98.68	1.32	1	3	1.1563	0.35;5
5	98.69	1.31	1.01	3	2.346	0.53;8
6	98.7	1.3	1.01	3	1.178	0.41;6
7	98.73	1.27	1.05	3	1.02	0.39;11

Дана таблиця відображає результати експериментів щодо виявлення ефективності механізму кешування у MongoDb.

4.2 Аналіз отриманих результатів

Спочатку розглянемо результати окремих метрик кожної з баз даних. Першою метрикою, яка є однією з найважливіших, це час виконання експерименту (див.рис.4.1).

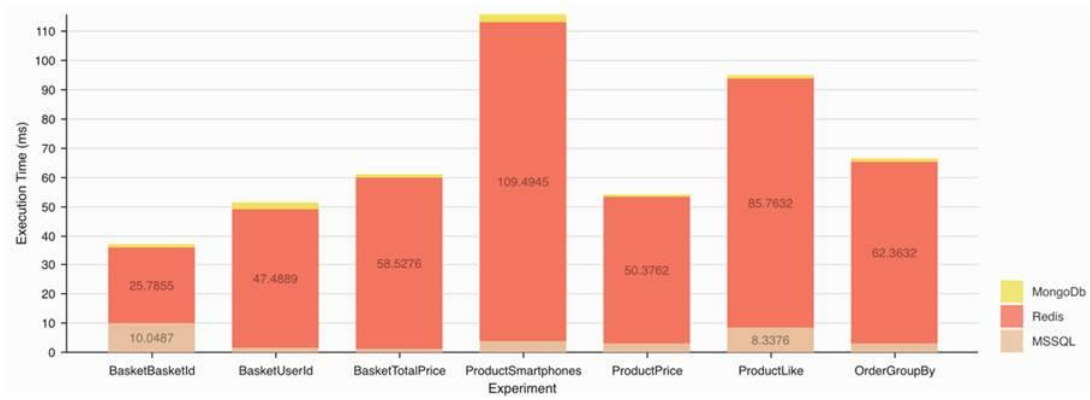

Рисунок 4.1 – Результати часу виконання експерименту (рисунок створено самостійно)

Даний графік відображає час виконання експерименту у мілісекундах для кожної бази даних. Слід зазначити, що Redis має доволі довгий час виконання на більшості експериментах, що є припустимим через складність експерименту. Кожен експеримент було втілено за допомогою вбудованих можливостей технології .NET та бібліотеки StackExchange.Redis. Відповідно СКБД Redis витрачала незначний час на знаходження відповідного ключа, проте далі вся логіка була виконана за допомогою функціоналу платформи .NET, що звісно займає більше часу. В той час як у СКБД MSSQL та MongoDb експерименти – це виконання прямих запитів або команд та фільтрація результатів у потрібній формі.

Також слід помітити, що MongoDb виявилась найшвидшою серед усіх СКБД, особливо швидкість виконання експерименту проявляється у запитах до сутності Product, яка повертає дані у вигляді колекції, що є значною перевагою перед реляційними базами даних. Такий результат підтверджує вибір цієї СКБД для мікросервісної архітектури електронної комерції, зокрема для мікросервісу пов’язаному з товарами, їх пошуком та фільтрацією.

Аналізуючи СКБД MSSQL, варто помітити, що вона справилася доволі непогано зі швидкістю проведених експериментів, часом навіть є перевага поміж усіх СКБД.

Далі слід проаналізувати метрики CacheHit та CacheMiss, які прораховують відсоток «попадання» у кеш-пам’ять для кожного експерименту. Розглянемо метрику CacheHitRate (див.рис.4.2).

Особливістю результатів за цією метрикою є відсоток попадання у кеш-пам’ять у СКБД Redis. Даний результат буде завжди 100%, тому що дані витягуються напряму з RAM.

Результат може бути 0 тільки у випадку, якщо одного з ключів не існує у пам’яті, а відповідно експеримент не може бути виконаний до кінця через брак даних у СКБД.

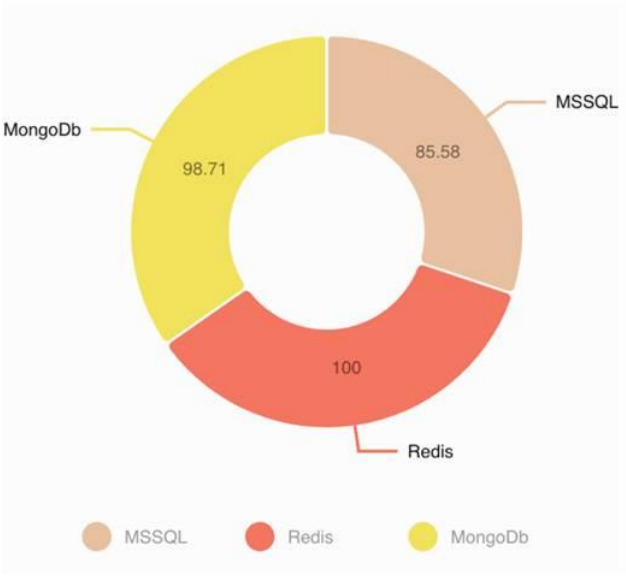

Рисунок 4.2 – Результати CacheHitRate метрики (рисунок створено самостійно)

Щодо результатів інших СКБД, кожна з них має свої особливості підрахунку даної метрики. Наприклад, у MSSQL CacheHitRate рахується за наявністю Query Execution Plan запиту, чи був він використаний для запиту, або ні. Проте на цю метрику може також впливати внутрішній вбудований механізм кешування, який перевіряє постійно чи необхідні старі плани, чи потрібно їх видалити, чи потрібно замінити існуючий, і тому дуже важко отримати 100% результат, навіть якщо запит вже був виконаний декілька разів попередньо [9]. Слід також зазначити, що MSSQL має найменший відсоток з поміж усіх через те, що для одного з експериментів, СКБД не створює Query Execution Plan та вважає його занадто простим для використання механізму кешування.

Щодо MongoDb, дана метрика прораховується за допомогою кількості створених планів у кеш-пам’яті та кількості прочитаних, тобто використаних, планів під час запиту. Оскільки для кожного експерименту, MongoDb використовує

функцію aggregate з об’єктом pipeline, який містить ще додаткові кроки для відтворення експерименту, кожен крок може потребувати використання кешування, або ні. Тому 100% результат також доволі важко отримати в даному випадку, лише один експеримент має такий результат.

Розглянемо метрику CacheMissRate (див.рис.4.3).

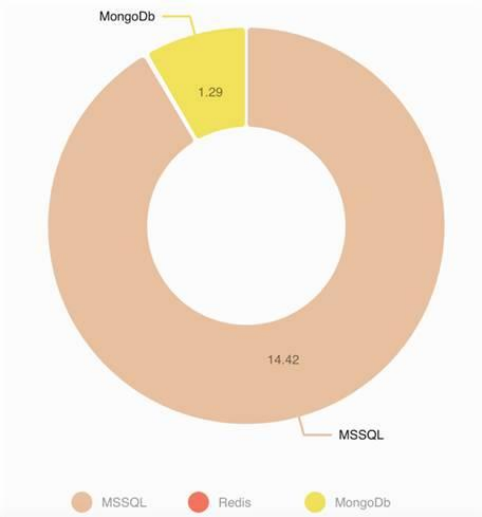

Рисунок 4.3 – Результати CacheMissRate метрики (рисунок створено самостійно)

Redis має нульовий результат, тому ця СКБД не попадає у відповідний графік. MSSQL має доволі значний результат через наявність експерименту, де не застосовуються механізми кешування. В той час MongoDb показує доволі малий відсоток промахів, що є гарним показником роботи кешування.

Розглянемо ще одну важливу метрику експериментів (див.рис.4.4).

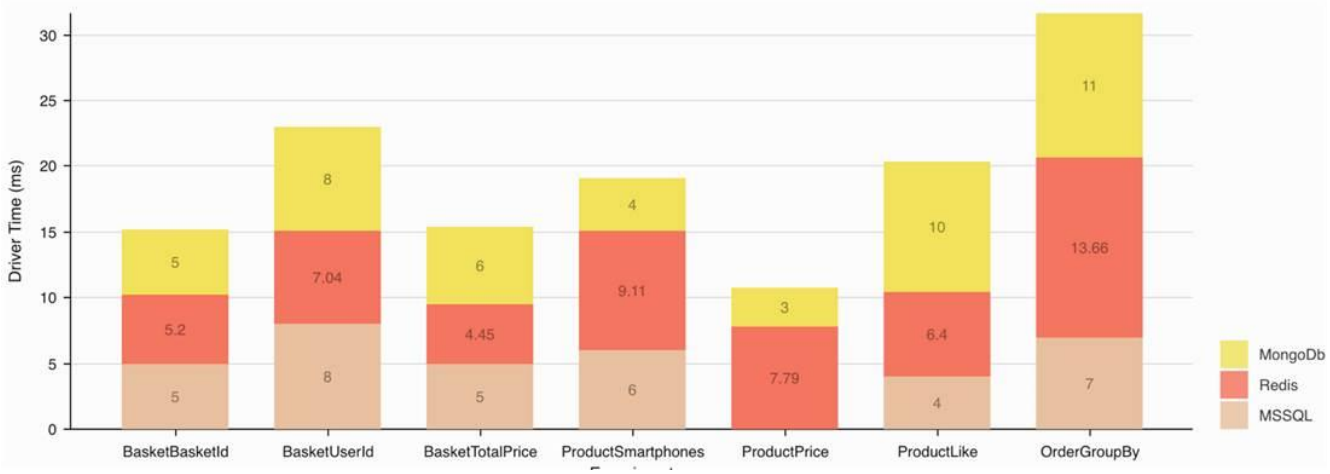

Рисунок 4.4 – Результати метрики часу обробки запиту бібліотеками (рисунок створено самостійно)

Метрика Resources показує кількість використаних системних ресурсів під час проведення експерименту. Для MSSQL – це розмір Query Execution Plan, час CPU та кількість зайнятої пам’яті при компіляції запиту. Для Redis дана метрика показує розмір ключів використаних для запиту та час драйверу бібліотеки на обробку усіх ключів. Для MongoDb – це розмір заданої команди та час драйверу бібліотеки на обробку запиту.

Серед цих складових, важливою є складова часу обробки запиту саме внутрішніми механізмами кожної зі СКБД. Ця складова показує скільки часу було витрачено в середині кожної з баз даних на відповідний запит викликаючого коду.

Доволі цікавою особливістю є результат СКБД Redis, яка витрачає невеликий час для обробки усіх ключів для двох експериментів до сутності Basket. Це підтверджує найчастіший вибір даної СКБД в мікросервісній архітектурі електронної комерції саме для мікросервісу кошик. Такий вибір передбачуваний тим, що дані у кошику зберігаються від моменту вибору товару до його замовлення, що є доволі малим об’ємом часу. І саме через те, що Redis розглядається як система дистрибутивного кешування, це пришвидшує роботу даних.

Розглянемо метрику розміру кеш-пам’яті, яку виділяє кожна зі СКБД (див.рис.4.5).

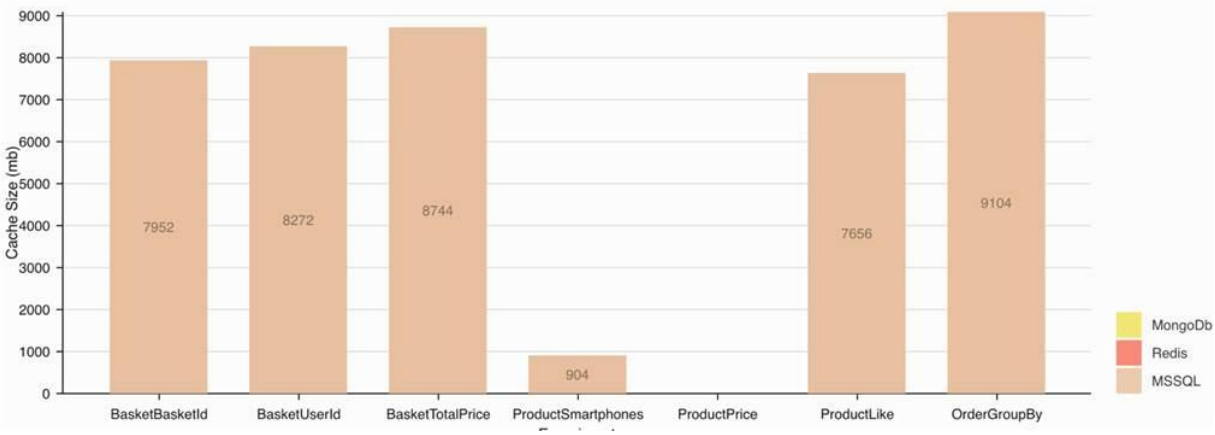

Рисунок 4.5 – Результати метрики розмір кеш-пам’яті (рисунок створено самостійно)

Як ми бачимо, розмір кеш-пам’яті СКБД MSSQL сягає величезних обсягів, що інші СКБД навіть не видно на графіку. Це пов’язано зі складністю механізмів

MSSQL, який керує декількома механізмами одночасно, зберігаючи дані для кешування у буферному кеші та у плани виконання запитів у кеші процедур.

Розглянемо результати цієї метрики проте для інших СКБД (див.рис.4.6).

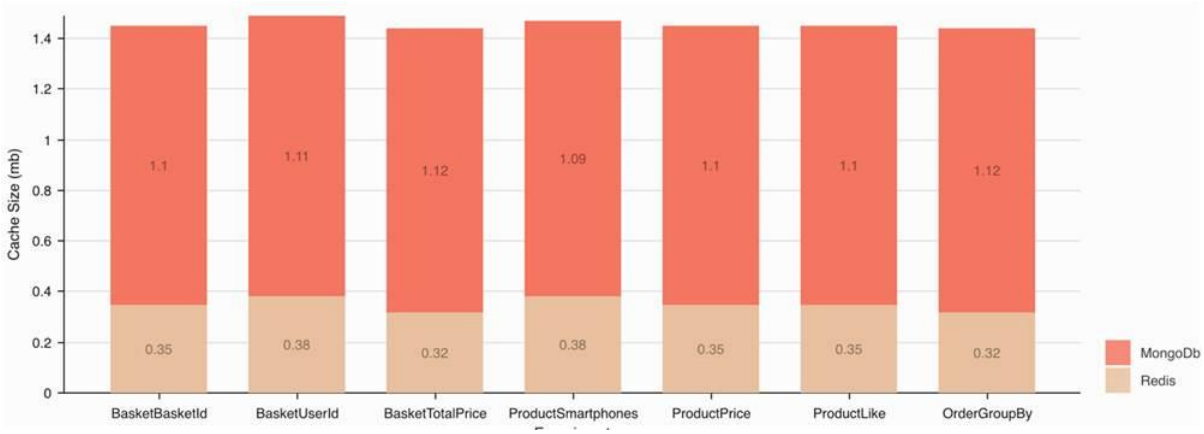

Рисунок 4.6 – Результати метрики розмір кеш-пам’яті (рисунок створено самостійно)

Зазначимо, що Redis використовує найменше пам’яті для зберігання сторінок кешування та ключів, що доводить про швидкісну роботу СКБД.

Враховуючи попередній аналіз окремих метрик, можна зробити висновок про вираження ефективності кешування Redis для експериментів сутності Basket. Тим не менш, слід зазначити, що Redis опрацьовувала експерименти доволі довго, через їх складність та обмежені можливості як і СКБД, так і бібліотеки для роботи з нею. Проте висновком щодо ефективності кешування даної СКБД є швидкість знаходження необхідних ключів та найкращий показник кількості зайнятих даних для кешування, що свідчить про вибір даної СКБД для зберігання даних будь-якого об’єму.

MongoDb проявила найкращі результати для експериментів сутності Product, яка повертає в якості результатів колекції даних. Це обумовлено структурою СКБД, яка зберігає дані у форматі BSON, тим самим гарантуючи швидкість та ефективність фільтрації та пошуку даних. Саме тому дана СКБД найкраще підходить для сервісів, пов’язаних з відображенням великого об’єму даних. Слід також зауважити, що MongoDb використовує доволі маленький об’єм пам’яті для

кешування, має значну швидкість виконання експериментів та показує гарну продуктивність.

Щодо MSSQL, то дана СКБД має найкращу структуру побудови кешування та можливостей, щодо отримання даних про цей процес. Попередні СКБД надають невеликий об’єм можливостей, аби отримати дані щодо механізмів кешування, чого не скажеш про MSSQL. Через те, що MSSQL розцінює один з експериментів доволі простим для кешування, відповідно там має найгірший показник серед усіх. Проте даний результат слід розглядати як і позитивний, оскільки це доводить те, що MSSQL має більш складний та досвідчений механізм кешування, який може визначати прості та складні запити. Швидкість роботи, мінімальне використання системних ресурсів та один з кращих показників «влучення» у кеш-пам’ять, ставить MSSQL як приклад надійної бази даних для використання кешування. Проте слід також бути обережним з кількістю даних для кешування, оскільки СКБД займає доволі великий об’єм пам’яті для кешування, який може сягати до 9 гб, в той час як інші СКБД витрачають не більше 2 МБ на кешування.

Важливо також помітити, що MSSQL доволі гарно справляється з експериментом до сутності Order. Загалом реляційні бази даних притаманно використовувати для мікросервісів, пов’язаних з обробкою персональної інформації користувача, в тому числі завдяки рівням ізольованості транзакцій.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено аналіз предметної галузі дослідження та визначено рівні на яких кешування може бути здійснене. Не менш важливим було визначення існуючих механізмів кешування в різних СКБД. Після чого був проведений аналіз існуючих проблем у кешуванні, які слід дослідити за допомогою експериментального дослідження.

Був проведений аналіз СКБД для дослідження механізмів кешування та було обрано найбільш підходящі СКБД шляхом вирішення багатокритеріальної задачі, а саме MSSQL, Redis та MongoDb.

Був створений план експериментального дослідження, в якому була обрана предметна галузь дослідження – інтернет-магазин або іншими словами електронна комерція. Також були проаналізовані та встановлені критерії перевірки якості механізмів кешування, серед яких є коефіцієнт попадання у кеш-пам’ять, коефіцієнт промахів, час виконання запиту, пропускна здатність, використання ресурсів та розмір кешу.

Була спроектована інфологічна модель бази даних, на основі якої були розроблені логічні моделі для MSSQL, Redis та MongoDb. Наступним етапом була описана програмна реалізація, серед якої були розроблені скрипти для створення таблиць в MSSQL, проявлені особливості роботи зі СКБД типу «ключ-значення» та документно-орієнтованою базою даних. Також була описана програмна реалізація системи для проведення дослідження, зокрема UI частина.

Були проведені експерименти, в ході яких визначена ефективність механізмів кешування для кожної з баз даних.

За результатами дослідження було опубліковано тези доповіді на двадцять восьмий міжнародний молодіжний форум «РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ В XXI ст.» , а також, підготовлено матеріали доповіді «Research on the efficiency of data caching to solve the problem of building a heterogeneous database» для подачі до IEEE Міжнародної науково-практичної конференції «Проблеми інфокомунікацій. Наука і техніка» (PIC S&T-2024) (див. додаток Г).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Blokdyk G. Caching A Complete Guide. The Art of Service - Caching Publishing, 2020 – 312 P.

2. Databases Caching [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/caching/database-caching/ (дата звернення 23.01.2024).

3. MySQL Query Cache – MySQL Docs [Електронний ресурс] – Режим доступу до ресурсу: https://dev.mysql.com/doc/refman/5.7/en/query-cache.html
(дата звернення 24.01.2024).

4. PostgreSQL Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/release/16.1/ (дата звернення 05.02.2024).

5. WiredTiger – MongoDb Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/docs/manual/core/wiredtiger/ (дата звернення 07.02.2024).

6. Mazurova, O. Research of ACID transaction implementation methods for distributed databases using replication technology / Mazurova, O., Naboka, A., Shirokopetleva, M. Innovative technologies and scientific solutions for industries, (2 (16), pp. 19-31. Doi: 10.30837/ITSSI.2021.16.019.

7. DbEngines [Електронний ресурс] – Режим доступу до ресурсу: https://db-engines.com/en/ (дата звернення 17.02.2024).

8. MS SQL Server Documentation – Microsoft [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16 (дата звернення 18.02.2024).

9. MS SQL Server Caching Mechanisms [Електронний ресурс] – Режим доступу до ресурсу: https://learning.oreilly.com/library/view/microsoft-r-sql-server-r/9780735634787/ch09s02.html#adhoc_query_caching (дата звернення 22.02.2024).

10. Mazurova O., Syvolovskyi I., Syvolovska O. NOSQL database logic design methods for MONGODB and NEO4J. Innovative technologies and scientific solutions for industries, (2 (20)), 2022. P. 52–63. DOI: 10.30837/ITSSI.2022.20.052.

11. Chodorow K. Scaling MongoDB: Sharding, Cluster Setup, and Administration. – O'Reilly Media, 2011. – 66 p.

12. Redis Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://redis.io/docs/data-types/hashes/ (дата звернення 22.02.2024).

13. Memcached Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://memcached.org/ (дата звернення 22.02.2024).

14. MS SQL Server Query Execution Plan – Microsoft Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/sql/relational-databases/performance/execution-plans?view=sql-server-ver16 (дата звернення 22.10.2023).

15. Chodorow K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage, 3rd Edition. O'Reilly Media, 2019 – 514 P.

16. Local Machines – Microsoft [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/uk-ua/windows-hardware/drivers/install/local-machine-and-current-user-certificate-stores (дата звернення 23.10.2023).

17. Williams T. Microservices Design Patterns in .NET. – Packt Publishing, 2023 – 300 с.

18. Teorey T., Lightstone S., Nadeau T. Database Modeling and Design. – Elsevier, 2006. – 296 P. – ISBN 978-0-12-685352-0.

19. Kuzochkina, A., Shirokopetleva, M., Dudar, Z. Analyzing and Comparison of NoSQL DBMS, International Scientific-Practical Conference on Problems of Infocommunications Science and Technology, PIC S and T 2018. Proceedings. 2019. P. 560–564. DOI 10.1109/INFOCOMMST.2018.8632133.

20. Redis and MongoDb Comparison [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/compare/mongodb-vs-redis (дата звернення 13.11.2023).

21. Halpin T. Entity Relationship modeling from an ORM perspective: Part 1 // [Електронний ресурс]: Object Role Modeling. Електрон. текст, дані. 1999. URL: http://www.orm.net/pdf/JCM11.pdf (дата звернення 25.03.2024).

22. Meier A., Kaufmann M. SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management. – Springer Vieweg, 2019. – 248 P. – ISBN 978-3658245481.

23. Barker T. Intelligent Caching. O’Reilly Media, Inc., 2017 – 215 P.

24. Bassett L., Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON. O'Reilly Media, 2020 – 124 P.

25. Gupta B., Cross-Site Scripting Attacks: Classification, Attack, and Countermeasures. – CRC Press, 2020 – 144 P.