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 продуктивності архітектурних рішень і СУБД для системи управління запасами
(тема)

Виконав:
студент 2 курсу, групи ІПЗм-22-3

Гужов В. О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

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

Керівник проф. Кирило СМЕЛЯКОВ
(посада, прізвище)

Допускається до захисту:

Зав. кафедри, проф. _____ Зоя ДУДАР
(підпис)

2024р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

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

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
зав. каф. ПІ, к.т.н., проф.
_____ Дудар З. В.
(підпис)
«___» _____ 2024р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Студентові Гужову Вячеславу Олександровичу

1. Тема роботи: «Дослідження продуктивності архітектурних рішень і СУБД для системи управління запасами»

Затверджена наказом університету від «29» березня 2024 р. № 250 Ст

2. Термін здачі студентом закінченої роботи «14» червня 2024 р.

3. Вхідні дані до проекту:

інформація щодо систем управління запасами, розробки програм на Spring Boot, інформація щодо СУБД, середовище розробки IntelliJ IDEA для розробки програм на Java.

4. Перелік питань, що потрібно опрацювати в роботі 1 постановка задачі

аналіз предметної галузі і постановка задачі, огляд та аналіз літературних джерел, розробка програмної системи управління запасами, проведення експерименту і аналіз результатів.

Календарний план

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	28.03.2024	*виконано*
2	Виявлення проблематики галузі	31.03.2024	*виконано*
3	Постановка задачі	01.04.2024	*виконано*
4	Написання програми і тестів до неї	03.04.2024 – 11.05.2024	*виконано*
5	Проведення порівняльного експерименту	12.05.2024	*виконано*
6	Підготовка пояснювальної записки	03.04.2024 – 25.05.2024	*виконано*
7	Підготовка презентації та доповіді	15.05.2024 – 25.05.2024	*виконано*
8	Нормоконтроль	04.06.2024	*виконано*
9	Рецензування	04.06.2024	*виконано*
10	Занесення диплома в електронний архів	11.06.2024	*виконано*
11	Попередній захист	11.05.2024	*виконано*
12	Допуск до захисту у зав. кафедри	12.06.2024	*виконано*
13	Захист магістерського дослідження	14.06.2024	*виконано*

Дата видачі «29» березня 2024р.

Керівник _____
(підпис)

Завдання отримав _____
(підпис)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить 67 стор., 31 рис., 2 табл., 11 джерел.

СИСТЕМА УПРАВЛІННЯ ЗАПАСАМИ, СУБД, SPRING BOOT, MONGODB, POSTGRESQL, MYSQL, SQLITE.

Об’єкт дослідження – системи управління запасами.

Мета роботи – Дослідження продуктивності різних СУБД в контексті систем управління запасами для електронної комерції, проведення експерименту і аналіз результатів.

Результат роботи – розроблена система управління запасами, проведений експеримент з дослідження продуктивності кожної з СУБД, зроблений аналіз результатів.

INVENTORY MANAGEMENT SYSTEM, DBMS, SPRING BOOT, MONGODB, POSTGRESQL, MYSQL, SQLITE.

Object of study - inventory management systems.

Purpose - to study the performance of various DBMS in the context of inventory management systems for e-commerce, conducting an experiment and analysing the results.

The result of the work is a developed inventory management system, an experiment to study the performance of each of the DBMSs, and an analysis of the results.

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

Я, *Гужов Вячеслав Олександрович*, студент гр.*ІПЗм-22-3*, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження продуктивності архітектурних рішень і СУБД для системи управління запасами» що буде представлена для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Усі 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 до допуску курсової роботи до захисту та застосування дисциплінарних заходів.

Робота виконана відповідно до вимог академічної доброчесності та пройшла перевірку на плагіат, який складає – 3.52%.

ЗМІСТ

ВСТУП

Управління запасами у сфері електронної комерції є критичним аспектом функціонування компаній. Сучасні системи управління запасами охоплюють різноманітні функції, починаючи від зберігання та моніторингу інформації про наявні товари до контролю за вантажами та їхнім статусом. Крім того, такі системи сприяють веденню обліку замовлень і їх доставці, надаючи компанії необхідну базу для оптимізації закупівель і управління логістичними процесами. Надійна система управління запасами дозволяє компаніям ефективно обробляти великі обсяги даних в межах єдиного сервісу, що, в свою чергу, сприяє підвищенню рентабельності [1]. Сучасні технології дозволяють навіть ідентифікувати товари за зображенням [2] або ідентифікувати співробітників за обличчям [3], що може бути інтегровано в сучасні системи управління запасами.

Ключовим чинником успішності бізнесу в електронній комерції є правильний вибір технологій для управління запасами. Цей вибір визначає ефективність операцій і конкурентоспроможність компанії на ринку. Сучасні технології значно полегшують процеси управління запасами, забезпечуючи автоматизацію та оптимізацію, що дозволяють уникати помилок та підвищувати продуктивність праці. Важливою є також можливість отримання актуальної інформації в режимі реального часу, що дозволяє приймати оперативні та обґрунтовані управлінські рішення.

Застосування системи аналітики дозволяє вдосконалювати прогнози попиту та ідентифікувати ринкові тенденції, що стає можливим завдяки систематичному збору та аналізу великих обсягів даних. Інтеграція з іншими бізнес-системами перетворює управління запасами на частину єдиної екосистеми, що сприяє їх злагодженій та ефективній взаємодії. При виборі технологій для управління запасами слід враховувати не лише їхню вартість, а й доцільність витрат. Існує можливість розробки власного програмного продукту, адаптованого під конкретні потреби, з використанням безкоштовних програмних ресурсів, що може значно знизити витрати на систему управління запасами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Проблематика що розглядається

Зростаюча значущість управління запасами для сучасних підприємств стає дедалі більш очевидною в умовах постійного технологічного розвитку та глобалізації ринків. В умовах збільшення обсягів операцій і різноманітності вимог, вибір оптимальної системи управління базами даних (СУБД) набуває важливого значення для розробки ефективної системи управління запасами. Різні СУБД мають свої переваги та обмеження, що можуть впливати на ефективність і швидкодію системи. У цьому дослідженні розглядаються основні вимоги до баз даних для систем управління товарними запасами та здійснюється порівняльний аналіз різних СУБД.

Для здійснення обґрунтованого вибору СУБД проводиться тестування продуктивності різних СУБД. Це передбачає визначення критеріїв, за якими буде оцінюватись кожна СУБД. У цьому розділі окреслені основні проблеми, які будуть розглянуті у цьому дослідженні, зокрема: вибір СУБД, тестування їх продуктивності для системи управління запасами.

1.2 Засоби проведення дослідження

Під час розробки системи управління товарними запасами було прийнято рішення скористатися мовою програмування Java, яка є однією з найпоширеніших у світі і має багаторічну історію успішного використання як у малому та середньому бізнесі, так і у корпоративних рішеннях [4].

Для спрощення процесу створення, налаштування та розгортання додатка було обрано фреймворк Spring Boot [5]. Цей фреймворк включає основні бібліотеки Spring, що дозволяє швидко створювати надійні та масштабовані додатки. Spring Boot автоматизує безліч рутинних завдань, що сприяє підвищенню продуктивності розробки та полегшує конфігурацію програм [6].

Для розробки та автоматизації процесів було вирішено використовувати

інтегроване середовище розробки IntelliJ IDEA. Це середовище забезпечує автодоповнення коду, підтримку бібліотек, вбудовану систему збірки, а також інструменти для зручного рефакторингу коду [7]. Додатковою перевагою є можливість розширення функціоналу за допомогою плагінів та наявність засобів для тестування.

Для автоматизації тестування Java-програм було обрано фреймворк JUnit. Використання JUnit дозволяє автоматизувати та спростити процес створення тестів, забезпечуючи ефективність, регулярність та зручність у виявленні та виправленні помилок [8].

У межах даного дослідження планується використовувати системи управління базами даних (СУБД) та драйвери для підключення СУБД до додатків, інтегровані з Spring Data. Spring Data являє собою проект в екосистемі Spring, що забезпечує спрощений та зручний доступ до різних джерел даних, таких як реляційні бази даних SQL та нереляційні NoSQL, через уніфіковані програмні інтерфейси. Головною метою Spring Data є полегшення розробки доступу до даних та зменшення обсягу повторюваного коду, пов'язаного з операціями з базами даних [9]. Використання Spring Data надає низку переваг:

- Спрощений доступ до даних. Spring Data забезпечує абстрактний рівень, який полегшує роботу з різноманітними типами сховищ даних. Розробникам не потрібно писати великий обсяг коду для взаємодії з базами даних чи іншими джерелами даних. Це робить Spring Data зручним інструментом для тестування різних СУБД.
- Операції CRUD. Spring Data автоматично генерує реалізації базових операцій створення, читання, оновлення та видалення (CRUD) для сутностей, дозволяючи розробникам зосередитися на бізнес-логіці, а не на операціях зі зберігання даних.
- Підтримка реляційних та нереляційних баз даних. Spring Data підтримує широкий спектр баз даних, включаючи реляційні (наприклад, MySQL, PostgreSQL) та нереляційні (NoSQL, наприклад, MongoDB, Redis) сховища даних. Це надає розробникам гнучкість у переході від одного

типу сховища до іншого.

- Інтеграція кешування. Завдяки Spring Data можна легко інтегрувати кешування для підвищення продуктивності додатків. Це особливо корисно для зменшення навантаження на базу даних та прискорення доступу до часто використовуваних даних.
- Вбудована підтримка транзакцій. Spring Data надає підтримку транзакцій, що дозволяє розробникам легко керувати транзакціями бази даних без необхідності написання великої кількості коду.
- Спрощення роботи з JPA. Для роботи з реляційними базами даних Spring Data JPA (Java Persistence API) дозволяє розробникам працювати з об'єктами Java, використовуючи анотації для мапування об'єктів на таблиці бази даних [10].
- Можливість створення власних запитів. Незважаючи на автоматично генеровані запити, розробники можуть використовувати власні SQL-запити або HQL-запити (Hibernate Query Language) для складніших випадків.

Таким чином, впровадження Spring Data дозволяє значно підвищити ефективність роботи з даними, мінімізуючи рутинні завдання та забезпечуючи високу гнучкість і продуктивність розробки. Spring Data значно полегшує взаємодію з різними сховищами даних, підвищуючи ефективність розробки та зменшуючи залежність від конкретних технологій зберігання даних.

1.3 Формування дослідницького завдання магістерського дослідження

Метою даного магістерського дослідження є проведення порівняльного аналізу різних систем управління базами даних (СУБД), використовуючи визначені критерії продуктивності. Для досягнення поставленої мети необхідно виконати наступні завдання:

- Сформулювати критерії відбору СУБД, які будуть застосовані для створення системи управління запасами.
- Встановити показники продуктивності, які використовуватимуться для

тестування систем управління запасами на основі різних СУБД.

– Розробити систему управління запасами та відповідні тести, спираючись на визначені показники продуктивності.
– Провести тестування системи управління запасами та визначити найбільш оптимальні СУБД.

2 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

2.1 Аналіз галузі систем управління запасами

Система управління товарними запасами є комплексним сервісом, розробленим для ефективного зберігання, планування, контролю та оптимізації запасів товарів чи сировини в компанії. Основною метою цієї системи є забезпечення належного рівня запасів для виробництва або продажу, ведення обліку товарів чи сировини, мінімізація витрат на утримання запасів та оптимізація процесів постачання.

Система управління запасами вирішує ряд ключових завдань для ефективного управління резервами товарів або матеріалів в організації. До основних завдань належать:

- Прогнозування потреби: Системи управління запасами дозволяють визначити очікуваний попит на товари або сировину. Це дає можливість підприємствам забезпечити наявність необхідних резервів і уникнути надлишкових запасів.
- Замовлення та постачання: На основі прогнозів попиту система автоматизує процес замовлення та постачання товарів. Це допомагає уникнути дефіциту або його подолання, забезпечуючи постійний доступ до необхідних ресурсів.
- Оптимізація рівня резервів: Система управління запасами підтримує оптимальний рівень запасів. Надмірні запаси можуть спричинити витрати на зберігання, втрати через застарівання товарів або знецінення, тоді як недостатні запаси можуть призвести до втрати продажів, затримок поставок або скасування замовлень.
- Моніторинг оборотності запасів: Система зберігає дані про надходження ресурсів, їх залишки, вантажі, кількість продажів, повернень та іншу супутню інформацію, яка слугує основою для аналізу оборотності запасів, оцінки ефективності їх використання та оптимізації бізнес-процесів у компанії.

– Мінімізація витрат: Правильний рівень запасів дозволяє підприємствам мінімізувати витрати на зберігання, управління та обробку наявних товарів.
– Управління життєвим циклом продукції: Системи управління запасами допомагають відстежувати життєвий цикл продукції, сприяючи прийняттю рішень щодо виробництва, реалізації або зміни рівня запасів товарів.

Загалом, система управління запасами сприяє оптимізації процесів управління товарами чи матеріалами, що дозволяє досягти ефективності та підтримувати бізнес-цілі організації. Впровадження системи управління запасами підвищує прибутковість бізнесу, забезпечуючи оптимальну роботу і управління ресурсами компанії.

2.2 Види систем управління запасами

Одним із найпростіших підходів до управління запасами є використання електронних таблиць. Ці рішення передбачають застосування програм, таких як Excel або Google Sheets, для ведення обліку запасів і можуть бути ефективними для базового управління резервами. Однак, системи обліку запасів, що ґрунтуються на електронних таблицях, мають низку значних недоліків. Наприклад, людські помилки можуть легко призвести до значних втрат при використанні електронних таблиць. Часто дані в таблицях погано структуровані, що призводить до додаткових витрат часу на їх формалізацію та аналіз. Крім того, існує ризик досягнення обмежень обсягу таблиць, що вимагатиме розподілу даних на кілька файлів. Ще однією проблемою є необхідність збирання та формалізації даних з різних таблиць для аналізу. Загалом, електронні таблиці є відносно недорогим рішенням, легко налаштовуються та підходять для компаній з простими потребами і невеликою кількістю продажів, що не перевищують обсяги таблиць.

На ринку також представлені спеціалізовані автоматизовані системи керування запасами, які можуть бути адаптовані під конкретні вимоги клієнтів. Автоматизоване програмне забезпечення для управління запасами розроблене

спеціально для відстеження та контролю резервів. Основною метою таких систем є зменшення адміністративних витрат, підвищення прозорості та точності, валідація даних, а також оптимізація операційних процесів. Це забезпечує менше витрат часу та коштів на управління запасами, дозволяючи більше часу приділяти продажам, розробці стратегії та прийняттю рішень. Недоліком таких систем може бути час, необхідний для налаштування, а також необхідність навчання персоналу для їх використання.

2.3 Огляд СУБД

Система управління базами даних (СУБД) являє собою програмний інструмент, призначений для створення, адміністрування та ефективного керування базами даних. База даних виступає структурованим сховищем даних, яке забезпечує зручний доступ, оновлення та аналіз інформації.

Використання СУБД має численні переваги. Перш за все, вони дозволяють ефективно оперувати великими обсягами даних, забезпечуючи швидкий доступ та зручний пошук. Крім того, СУБД гарантують цілісність даних, що допомагає уникнути помилок і суперечностей у базі даних.

Забезпечення безпеки є ключовим аспектом застосування СУБД. Вони надають механізми захисту даних від несанкціонованого доступу, включаючи контроль доступу та шифрування.

Одним з основних елементів взаємодії зі СУБД є мова запитів, яка дозволяє виконувати різноманітні операції, такі як вибірка, оновлення, вставка та видалення даних. Це спрощує роботу з інформацією і робить можливими різні автоматизації для роботи з даними. Крім того, багато СУБД підтримують одночасний доступ кількох користувачів до бази даних, що є особливо важливим для великих організацій із значною кількістю користувачів. Для кожного з користувачів зазвичай можна застосовувати правила доступу або робити це для груп користувачів.

Основні переваги використання СУБД для компаній, які займаються електронною комерцією, включають:

- Централізоване керування даними. СУБД забезпечують централізоване та консолідоване управління даними, що полегшує зберігання, оновлення та доступ до інформації.
- Структуроване зберігання інформації. Дані у СУБД зберігаються у структурованому вигляді, що дозволяє ефективно виконувати операції з пошуку, сортування та фільтрації.
- Можливість паралельного доступу. СУБД дозволяють багатьом користувачам одночасно працювати з базою даних, забезпечуючи конкурентний доступ і керування транзакціями.
- Забезпечення цілісності записів. СУБД надають механізми для забезпечення цілісності даних, запобігаючи некоректним або несанкціонованим змінам записів.
- Забезпечення безпеки інформації. Системи керування базами даних надають засоби для контролю доступу до даних і захисту конфіденційності записів.
- Ефективність та оптимізація запитів. Оптимізатор запитів у СУБД дозволяє виконувати запити до бази даних так, щоб вони використовували ресурси системи найефективнішим чином.
- Автоматичне відновлення та резервне копіювання. Багато СУБД надають засоби автоматичного відновлення та резервного копіювання даних для захисту від втрати інформації.
- Масштабованість. СУБД можуть масштабуватися від невеликих локальних систем до великих розподілених середовищ, що дозволяє пристосовувати їх до різних потреб користувачів.
- Спільне використання даних. СУБД дозволяють розділяти та спільно використовувати дані між різними додатками та користувачами, що сприяє інтеграції систем.
- Зручність для розробників. Використання СУБД спрощує роботу розробників, забезпечуючи високорівневі мови запитів та інші інструменти для роботи з даними.

Таким чином, використання СУБД сприяє збереженню, організації та ефективному управлінню великими обсягами даних, забезпечуючи при цьому їх цілісність та безпеку, а для розробників забезпечує зручність розробки.

2.4 Вибір СУБД для порівняння

Вибір системи управління базами даних (СУБД) відіграє важливу роль для компаній. Перехід на іншу СУБД може вимагати значних витрат часу і може спричинити фінансові збитки для організації. Щоб уникнути цього, слід здійснити ретельний аналіз доступних СУБД і вибрати ту, яка найкраще відповідає встановленим критеріям і вимогам. Важливими чинниками при виборі є популярність і присутність на ринку, оскільки це вказує на те, чи буде система підтримуватися та розвиватися в майбутньому. Для порівняння різних СУБД ми використаємо наступні критерії:

- ринкова частка - кількість компаній, які використовують дану СУБД;
- активність спільноти - активність на форумах підтримки, кількість відкритих питань і їх вирішення, кількість внесків у вихідний код;
- рейтинги та рейтингові списки - наприклад, рейтинг DB-Engines, який враховує багато різних факторів, включаючи кількість пошукових запитів у Google, Bing, GitHub тощо;
- ліцензія і вартість користування;
- підтримка ACID транзакцій для забезпечення надійного зберігання консистентних даних;
- можливості шифрування БД.

Перейдемо до кандидатів для порівняння у даному дослідженні.

MySQL - це система управління базами даних з відкритим кодом, яка використовує реляційну модель, що передбачає зберігання даних у таблицях, які можуть мати зв'язки між собою. Для виконання операцій створення, видалення, отримання та зміни даних використовується мова SQL. MySQL є безкоштовним програмним забезпеченням, яке розповсюджується за ліцензією GNU і підтримується Oracle. Ця система широко застосовується в різних додатках, від

малих проектів до великих корпоративних систем.

Oracle Database - це потужна система управління базами даних, яка надає надійне зберігання та ефективний доступ до великих обсягів даних. Вона використовує мову SQL, забезпечує цілісність транзакцій, високу доступність та розширені можливості безпеки. Oracle Database широко використовується в корпоративних середовищах для обробки різноманітних корпоративних даних.

Microsoft SQL Server - це високопродуктивна система управління базами даних (СУБД), розроблена Microsoft. Її основна функція - зберігання, структурування та надання швидкого доступу до даних. SQL Server використовує SQL для формування запитів, гарантує транзакційну цілісність, має високий рівень безпеки та інструменти для оптимізації запитів. Ця система широко застосовується для управління корпоративними базами даних, забезпечуючи надійність та високу продуктивність у різних галузях бізнесу.

PostgreSQL - це відкрита об'єктно-реляційна СУБД. Вона повністю підтримує стандарт SQL і має багато сучасних можливостей, таких як складні запити, зовнішні ключі, тригери та транзакційну цілісність. PostgreSQL здатна обробляти великі обсяги даних і є однією з найбільш використовуваних відкритих СУБД.

MongoDB - це NoSQL база даних, орієнтована на документи, яка використовується для зберігання великої кількості даних. Вона використовує колекції та документи замість традиційних таблиць і рядків, які використовуються в реляційних базах даних. Документи в MongoDB складаються з пар ключ-значення, які є основним елементом даних у MongoDB. Ця гнучкість структури даних робить MongoDB надзвичайно потужною. Наприклад, на відміну від SQL-баз даних, де потрібно визначити схему таблиці перед вставкою даних, колекції MongoDB не вимагають попереднього визначення структури документа.

SQLite - це компактна та легка вбудована система управління базами даних. Вона надає простий механізм для зберігання та доступу до даних в одному файлі без потреби в окремому сервері. SQLite використовує SQL для взаємодії з базою даних і є ідеальним вибором для малих проектів, мобільних застосунків та вбудованих систем.

Redis (REmote DIctionary Server) - це відкритий (за ліцензією BSD) інструмент для зберігання даних в пам'яті, який може використовуватися як база даних, кеш або брокер повідомлень. Redis пропонує структури даних, такі як рядки, хеші, списки, набори, відсортовані набори з діапазонними запитами, бітові мапи, гіперлоглоги, геопросторові індекси та потоки.

Cassandra - це розподілена NoSQL система управління базами даних, розроблена для масштабних та високодоступних застосунків. Вона розроблена для обробки великих обсягів даних на розподілених серверах, забезпечуючи високу доступність та стійкість до відмов. Cassandra використовує мережу вузлів для розподілення даних і не вимагає фіксованої схеми, що дозволяє легко масштабувати систему та працювати з різними типами даних. Вона часто використовується в великих веб-проектах та системах з великою кількістю записів.

IBM Db2 - це система управління базами даних (СУБД) від IBM, яка забезпечує надійне зберігання та обробку даних. Підтримує реляційну модель даних і використовує мову запитів SQL для взаємодії з базою даних. Включає різні версії для різних платформ, включаючи сервери, хмарні сервіси та мобільні пристрої. Пропонує ряд функцій, включаючи високу доступність, сховище даних у пам'яті для оптимізації продуктивності та інструменти для розробки та управління базою даних.

Зараз, обравши критерії вибору, додамо їх до таблиці критеріїв для кожної СУБД. Відповідна таблиця зображена на рисунку 2.1.

	Рейтинг DB-Engines	Доля на ринку	Stackoverflow QA	Ліцензія і вартість використання	Підтримка ACID транзакцій	Можливість шифрування даних
Oracle	1277.03	2.26%	9.80%	Proprietary commercial software, paid subscription	Так	Transparent Data Encryption (TDE), Advanced Security Option (ASO), Oracle Data Pump Encryption, Oracle Transparent Sensitive Data Protection, SecureFiles Encryption, DBMS_CRYPTO Package
MySQL	1115.240	38.90%	41.09%	GPL, free for commercial use	Так	Data at Rest Encryption, Connection Encryption (SSL/TLS), Authentication Encryption, Binary Log Encryption, Temporary File Encryption, Transparent Data Encryption (TDE), Audit Log Encryption
Microsoft SQL Server	911.420	31.24%	25.42%	Proprietary commercial software, paid subscription	Так	Transparent Data Encryption (TDE), Connection Encryption (SSL/TLS), Backup Encryption, Column-Level Encryption, Always Encrypted, Dynamic Data Masking
PostgreSQL	636.860	17.40%	45.55%	PostgreSQL License, free for commercial use	Так	Data at Rest Encryption, Connection Encryption (SSL/TLS), Transparent Data Encryption (TDE), Column-Level Encryption
MongoDB	428.550	24.60%	25.52%	Server Side Public License, free for commercial use	Так	Data at Rest Encryption, Connection Encryption (SSL/TLS), Field-Level Encryption
SQLite	124.580	3.12%	30.90%	MIT License, free for commercial use	Так	Encryption of Entire Database File
Cassandra	109.170	5.16%	2.51%	Apache License 2.0, free for commercial use	Частково	Data at Rest Encryption, Client-to-Node Encryption (SSL/TLS)
IBM Db2	136.000	5.42%	1.85%	Proprietary commercial software, paid subscription	Так	Data at Rest Encryption, Connection Encryption (SSL/TLS), Field Procedures for Column-Level Encryption
Redis	160.020	8.40%	20.41%	BSD, free for commercial use	Частково	Data at Rest Encryption, Connection Encryption (SSL/TLS)

Рисунок 2.1. – Таблиця з критеріями порівняння СУБД (рисунок створено самостійно)

Перші три критерії залишаються без змін за абсолютною шкалою. Вони оцінюються за такими параметрами, як “Рейтинг DB-Engines”, “Доля на ринку” і “Stackoverflow QA”, що відображає популярність кожної СУБД серед компаній, розробників та на ринку в цілому. Значення коефіцієнтів для цих критеріїв залишається 0.1.

Ліцензія та вартість використання є критичними критеріями, особливо для малого та середнього бізнесу. Чим менше коштів компанія витрачає на ліцензії, тим краще. Значення коефіцієнта для цього критерію залишається 0.3.

Підтримка ACID транзакцій має велике значення для систем управління запасами, забезпечуючи надійність та цілісність даних. ACID (Atomicity, Consistency, Isolation, Durability) гарантує, що транзакції виконуються атомарно (усе або нічого), забезпечують консистентність даних, ізолюються одна від одної та є стійкими до збоїв. Це означає, що такі операції, як замовлення, постачання та інші транзакції, будуть виконуватись надійно, зберігаючи цілісність даних, що є критичним для ефективного функціонування системи. Значення коефіцієнта для цього критерію також залишається 0.3.

Можливість шифрування даних є ще одним важливим аспектом для систем управління запасами, з точки зору безпеки та конфіденційності. Хоча цей критерій є важливим, шифрування на всіх рівнях може бути реалізовано за допомогою сторонніх програм. Наприклад, для шифрування даних на диску можна використовувати програми для шифрування вмісту диска, а для захищеної передачі даних - зашифровані канали передачі. Проте вбудовані методи шифрування забезпечують більшу зручність налаштування та підтримки. Значення коефіцієнта для цього критерію залишається 0.1.

Після опису шкал оцінок за критеріями та їх коефіцієнтів, ми можемо визначити, яка з моделей найбільш підходить для нас. Для цього використовується лінійна адитивна згортка з ваговими коефіцієнтами, оскільки деякі критерії є важливішими за інші. Проведемо розрахунки за формулою 2.1:

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij} \quad (2.1)$$

де Z^* – максимальне значення інтегрального показника, яке визначається серед усіх можливих моделей (і від 1 до m),

α_j – ваговий коефіцієнт j-го критерію,

β_j – коефіцієнт значимості j-го критерію,

a_{ij} – оцінка і-ї моделі за j-м критерієм,

n – кількість критеріїв,

m – кількість моделей.

Розрахунок даної формули було виконано у середовищі Excel. Результат показано на рисунку 2.2.

	Рейтинг DB-Engines	Доля на ринку	Stackoverflow QA	Ліцензія і вартість використання	Підтримка ACID транзакцій	Можливість шифрування даних	Z*
Oracle	1277.03	2.26%	9.80%	0	5	5	**0.110**
MySQL	1115.240	38.90%	41.09%	4	5	3	**0.186**
Microsoft SQL Server	911.420	31.24%	25.42%	0	5	5	**0.129**
PostgreSQL	636.860	17.40%	45.55%	5	5	5	**0.178**
MongoDB	428.550	24.60%	25.52%	4	5	4	**0.152**
SQLite	124.580	3.12%	30.90%	5	5	3	**0.138**
Cassandra	109.170	5.16%	2.51%	5	0	0	**0.064**
IBM Db2	136.000	5.42%	1.85%	0	5	4	**0.073**
Redis	160.020	8.40%	20.41%	4	0	4	**0.080**
β	0.1	0.1	0.1	0.3	0.3	0.1	
α	0.000276103	0.744934446	0.517464424	0.037037037	0.033333333	0.035714286	

Рисунок 2.2. – Результати розрахунку (рисунок створено самостійно)

У рамках магістерського дослідження було прийнято рішення включити чотири системи управління базами даних (СУБД), причому до їх числа мали входити як SQL, так і NoSQL СУБД. Згідно з отриманими результатами, провідні позиції займають MySQL, PostgreSQL, MongoDB та SQLite.

3 ПІДГОТОВКА ДО ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

3.1 Розробка схеми бази даних

Система управління запасами повинна мати базу даних, що включає інформацію про користувачів системи, з переліком ролей, які надають доступ до різних частин системи. На рисунку 3 представлені дві таблиці: одна з користувачами, інша з ролями доступу, а також допоміжна таблиця, яка зв’язує користувачів з їхніми ролями. У таблиці “admin_users” зберігаються дані про користувачів сервісу, а в таблиці “admin_roles” зберігаються ролі, які визначають доступ до різних частин сервісу.

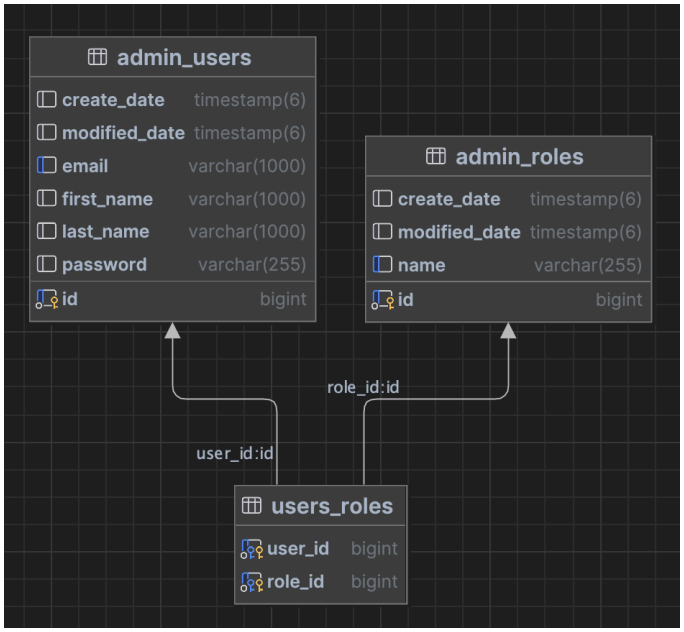

Рисунок 3.1 – Таблиці баз даних для користувачів і ролей (рисунок створено самостійно)

Для створення схеми бази даних, що зберігає інформацію про продукти, вантажі, замовлення, склади, відділи продажів та клієнтів, було розроблено структуру, представлену в додатку А. Усі таблиці, пов’язані з функціоналом системи управління запасами, мають префікс “company”. Розглянемо детальніше, яка інформація зберігається в кожній таблиці.

Таблиця “company_addresses” зберігає дані про адреси, що можуть бути адресами покупців або складів. Вона містить усі адреси, що використовуються в

системі.

Таблиця “company_cargoes” зберігає дані про вантажі, включаючи список продуктів, інформацію про склади, з яких і до яких транспортуються товари, та інші необхідні дані.

Таблиця “company_categories” містить дані про категорії товарів.

Таблиця “company_dimensions” зберігає інформацію про розміри товарів.

Таблиця “company_item_options” містить варіації товарів, такі як колір або інші характеристики.

Таблиця “company_marketplaces” зберігає інформацію про торговельні майданчики, на яких продаються товари, наприклад, Amazon, eBay або власний магазин.

Таблиця “company_meta_products” зберігає дані про мета-продукти, які можуть включати кілька різних товарів.

Таблиця “company_orders” містить інформацію про замовлення компанії, включаючи дати оплати, відправлення та платежу, дані про клієнтів, список товарів, склади, з яких відправляються замовлення, вагу, розміри, номери трекінгу, дані про магазини та інші необхідні дані.

Таблиця “company_order_items” зберігає інформацію про кожен товар у замовленні, включаючи кількість, ціну та інші характеристики.

Таблиця “company_prices” містить дані про ціни, включаючи валюту.

Таблиця “company_products” зберігає інформацію про продукти, їх розміри, вагу, місцезнаходження на складі, категорії та варіації.

Таблиця “company_sale_transactions” зберігає інформацію про те, який саме запис з таблиці “company_products” був відправлений як частина замовлення з таблиці “company_order_items”.

Таблиця “company_sizes” містить інформацію про всі розміри, що використовуються в системі.

Таблиця “company_stores” зберігає дані про облікові записи, на яких продаються товари, і список усіх замовлень з цих облікових записів.

Таблиця “company_warehouses” містить інформацію про склади компанії.

Вона пов’язана з вантажами, які транспортуються як до складу, так і з нього, а також зі списком усіх замовлень, відправлених зі складу, і списком усіх наявних товарів.

Таблиця “company_weights” зберігає дані про всі ваги, що використовуються в системі.

3.2Вибір критеріїв для оцінки продуктивності СУБД

Для оцінки ефективності різних систем управління базами даних (СУБД) було визначено такі ключові критерії:

- Час виконання запитів (CRUD): Важливо оцінювати тривалість операцій створення, читання, оновлення та видалення даних, оскільки вони є основними функціями будь-якої СУБД.
- Час виконання транзакцій: Для систем з високою інтенсивністю транзакцій цей параметр критично важливий для забезпечення ефективної роботи.
- Використання ресурсів: Ефективність використання CPU та RAM впливає на загальну продуктивність та витрати на апаратне забезпечення.

Застосування цих критеріїв дозволяє враховувати різноманітні потреби користувачів і визначити оптимальну СУБД, яка забезпечує баланс між швидкістю, надійністю та ефективністю використання ресурсів [11].

Варто відзначити що через те що SQLite запускається в тому ж процесі що й сама програма неможливо точно порахувати скільки CPU і RAM використовує ця СУБД. Тож порівняння з SQLite будуть відбуватися тільки з точки зору часу виконання транзакцій.

4 ПРОЦЕС РОЗРОБКИ СИСТЕМИ

1.1 Сутності системи

Процес створення системи управління запасами почався з визначення ключових сутностей, які формують основу для всієї системи. Розробка цих сутностей була необхідною для забезпечення повної функціональності системи та можливості її подальшого розширення. Нижче перераховані основні сутності системи з поясненням їхньої ролі та важливості:

- User. Сутність представляє користувачів системи. Кожен користувач має унікальні атрибути, такі як ім'я, прізвище, електронну пошту, пароль та набір ролей. Використання сутності User забезпечує контроль доступу до різних частин системи, що є критичним для безпеки та персоналізації інтерфейсу.
- Role. Сутність визначає ролі користувачів в системі, такі як адміністратор, менеджер складу тощо. Кожна роль має унікальну назву. Сутність Role дозволяє визначити різні рівні доступу для користувачів, спрощуючи управління правами доступу та забезпечуючи гнучкість у розподілі завдань.
- Product. Сутність представляє продукти, які зберігаються на складі. Включає інформацію про SKU (stock keeping unit), опис, кількість, вагу, ціну та категорії. Це дозволяє точно відстежувати наявність товарів, їх місцезнаходження та рух в системі,.
- Category. Сутність класифікує продукти в різні групи. Кожна категорія має унікальну назву та може містити багато продуктів. Використання категорій дозволяє легко знаходити потрібні продукти та здійснювати їх класифікацію, що спрощує управління та пошук.
- Warehouse. Сутність представляє склади, де зберігаються продукти. Містить інформацію про місце розташування та пов'язані продукти і замовлення. Це дозволяє ефективно контролювати запаси на різних етапах ланцюга постачання.

- Order. Сутність представляє замовлення на продукти. Містить інформацію про номер замовлення, дату створення, статус, загальну суму та пов'язані елементи замовлення. Це дозволяє відстежувати виконання замовлень та їх обробку на всіх етапах.
- OrderItem. Сутність представляє конкретний продукт у замовленні. Містить інформацію про продукт, кількість, ціну та пов'язані транзакції продажу. Це дозволяє точно обліковувати кожен товар у складі замовлення, забезпечуючи детальний облік та контроль.
- Cargo. Сутність представляє вантажі, які переміщуються між складами. Містить інформацію про продукти, дати відправлення та доставки, відправника та одержувача. Це дозволяє ефективно планувати та контролювати логістичні операції, забезпечуючи своєчасне переміщення товарів.
- Store. Сутність представляє магазини, які продають продукти. Містить інформацію про назву магазину, компанію-власника, вебсайт та валюту. Це дозволяє організувати продажі та забезпечити інтеграцію з торговими платформами, сприяючи ефективному управлінню продажами.
- Marketplace. Сутність представляє торгівельні платформи, на яких розміщуються магазини. Містить інформацію про назву та пов'язані магазини. Це дозволяє організувати взаємодію з різними торговими платформами та забезпечити масштабованість системи, що є важливим для розширення ринку збуту.
- SaleTransaction. Сутність представляє транзакції продажу продуктів. Містить інформацію про продукт, кількість, дату та суму транзакції. Це дозволяє відстежувати продажі та здійснювати аналіз комерційної діяльності, забезпечуючи прозорість та контроль за фінансовими операціями.

Кожна з цих сутностей є критично важливою для функціонування системи управління запасами. Їх взаємодія забезпечує цілісність даних, оптимізацію логістичних процесів та ефективне управління ресурсами. Обрані сутності мають

покрити той мінімальний функціонал який має бути присутнім у системі управління запасами.

4.2 Розробка шару repository

У Spring Boot, шар repository є ключовим компонентом, що забезпечує взаємодію між бізнес-логікою та базою даних. Repository шар використовує Spring Data JPA для автоматичного створення необхідних SQL-запитів, що значно спрощує процес роботи з даними. Цей шар абстрагує операції CRUD (Create, Read, Update, Delete), що дозволяє зосередитися на бізнес-логіці, а не на деталях реалізації доступу до даних.

У Spring Boot репозиторії визначаються як інтерфейси, що розширюють JpaRepository або CrudRepository. Ці інтерфейси надають базовий набір методів для роботи з базою даних. Ось детальний опис основних методів, які вони включають:

- Метод “save(S entity)” зберігає даний екземпляр сутності у базі даних. Якщо сутність вже існує, метод оновлює її, а якщо сутність нова, метод створює новий запис у базі даних. Це один з найважливіших методів, оскільки забезпечує можливість як створення нових записів, так і оновлення існуючих.
- Метод “findById(ID id)” повертає сутність за даним ідентифікатором. Якщо сутність з таким ідентифікатором не знайдена, метод повертає Optional.empty(). Це дозволяє безпечно обробляти ситуації, коли шуканий запис може бути відсутнім у базі даних.
- Метод “findAll()” повертає всі екземпляри сутностей. Це дуже зручно для отримання списку всіх записів певного типу з бази даних. Наприклад, якщо потрібно відобразити всі наявні продукти в системі, цей метод поверне повний список продуктів.
- Метод “deleteById(ID id)” видаляє сутність за даним ідентифікатором. Це дозволяє видаляти конкретні записи з бази даних, якщо відомий їх ідентифікатор. Наприклад, можна видалити продукт або замовлення за їх унікальним ідентифікатором.

- Метод “delete(S entity)” видаляє даний екземпляр сутності з бази даних. Це корисно, коли є об'єкт, який потрібно видалити, і не потрібно знати його ідентифікатор. Достатньо передати об'єкт, і він буде видалений з бази даних.
- Метод “count()” повертає кількість екземплярів сутностей у базі даних. Це дозволяє дізнатися загальну кількість записів певного типу. Наприклад, можна визначити кількість усіх продуктів або замовлень в системі.
- Метод “existsById(ID id)” перевіряє, чи існує сутність з даним ідентифікатором у базі даних. Це зручно для перевірки наявності запису перед виконанням якихось операцій, наприклад, перед видаленням або оновленням.

Окрім стандартних методів, Spring Data JPA дозволяє додавати власні методи у шар repository, просто визначаючи їх в інтерфейсі. Spring Data JPA аналізує назву методу і автоматично генерує необхідний SQL-запит. Це значно полегшує розширення функціональності репозиторію без необхідності написання складного SQL-коду.

У нашій системі були додані кілька специфічних методів для задоволення бізнес-вимог. Наприклад, у CargoRepository були додані методи, які дозволяють знаходити всі вантажі, що були відправлені з певного складу або прибули на певний склад. Ці методи використовують ідентифікатори складів для фільтрації результатів, що значно полегшує управління логістичними операціями.

У ProductRepository були додані методи для пошуку всіх продуктів, що належать до певної категорії, і для отримання сторінки продуктів, що належать до певного вантажу. Ці методи дозволяють більш ефективно управляти запасами, забезпечуючи можливість швидкого пошуку та фільтрації продуктів за різними критеріями.

У OrderRepository були додані методи, які дозволяють отримати замовлення порціями для конкретного магазину, а також знайти замовлення за його номером. Це значно полегшує управління замовленнями, забезпечуючи можливість швидкого доступу до необхідної інформації.

У WarehouseRepository був доданий метод, який повертає сторінку складів за їх ідентифікаторами. Це дозволяє легко отримувати інформацію про декілька складів одночасно, що є важливим для управління великими обсягами даних.

Таким чином, додавання кастомних методів у repository шар дозволяє розширити стандартну функціональність Spring Data JPA для задоволення специфічних потреб бізнесу. Це забезпечує гнучкість і ефективність у роботі з даними, дозволяючи швидко адаптувати систему до змінних вимог. Використання цих методів у поєднанні зі стандартними методами забезпечує повний набір інструментів для управління даними в системі управління запасами.

4.3 Розробка шару service

Service шар у Spring Boot є ключовим компонентом архітектури додатків, який забезпечує взаємодію між шаром repository (дані) та шаром сутностей (бізнес-логіка). Цей шар відповідає за обробку бізнес-логіки, виконання перевірок, трансформацію даних та забезпечення координації між різними частинами додатка. Service шар допомагає ізолювати бізнес-логіку від деталей реалізації доступу до даних, що робить код більш чистим, структурованим та легким для тестування.

Service шар отримує дані від repository шару, обробляє їх відповідно до бізнес-логіки, та повертає оброблені дані до верхніх шарів додатка, таких як контролери, що відповідають за взаємодію з користувачами. Завдяки цьому шарові можна уникнути дублювання коду та централізувати управління бізнес-логікою.

Всі класи Service шару зазвичай містять набір спільних методів, які забезпечують основні CRUD (Create, Read, Update, Delete) операції. Ось короткий опис кожного з них:

- Метод “create(Entity entity)” використовується для створення нового запису у базі даних. Цей метод приймає об'єкт сутності, який потрібно зберегти, виконує необхідні бізнес-логічні перевірки, та викликає відповідний метод репозиторію для збереження даних.
- Метод “findById(Long id)” дозволяє знайти запис у базі даних за його унікальним ідентифікатором. Цей метод спочатку викликає метод

репозиторію для отримання даних, а потім обробляє результат та повертає знайдений об'єкт або кидає виняток, якщо об'єкт не знайдено.

- Метод “findAll()” повертає всі записи з бази даних. Він викликає відповідний метод репозиторію для отримання списку всіх сутностей, а потім повертає цей список до верхнього шару додатка.
- Метод “update(Entity entity)” використовується для оновлення існуючого запису у базі даних. Цей метод приймає об'єкт сутності з оновленими даними, виконує необхідні перевірки, та викликає метод репозиторію для збереження змін.
- Метод “deleteById(Long id)” видаляє запис з бази даних за його унікальним ідентифікатором. Цей метод викликає відповідний метод репозиторію для видалення об'єкта та забезпечує обробку можливих помилок, пов'язаних з видаленням.
- Метод “existsById(Long id)” перевіряє, чи існує запис з даним ідентифікатором у базі даних. Він викликає метод репозиторію, який повертає логічне значення, що вказує на наявність або відсутність об'єкта.

Окрім стандартних методів, Service шар містить унікальні методи, які розроблені спеціально для задоволення потреб бізнес-логіки конкретного додатка. Нижче наведені деякі з них.

У CargoService були додані методи для знаходження вантажів за ідентифікаторами складів відправлення та призначення. Ці методи використовуються для отримання всіх вантажів, що були відправлені з певного складу або прибули на певний склад. Це дозволяє ефективно керувати логістичними операціями, забезпечуючи своєчасну доставку та відстеження переміщень товарів.

У ProductService були додані методи для пошуку продуктів за категоріями та вантажами. Ці методи дозволяють знаходити всі продукти, що належать до певної категорії або вантажу, а також підраховувати кількість таких продуктів. Це забезпечує більш ефективне управління запасами, дозволяючи швидко знаходити необхідні товари та проводити аналіз даних.

У OrderService були додані методи для обробки замовлень конкретних магазинів та пошуку замовлень за їх номерами. Ці методи дозволяють отримати всі замовлення, зроблені у конкретному магазині, та знайти замовлення за його унікальним номером. Це спрощує процес управління замовленнями, забезпечуючи швидкий доступ до необхідної інформації та підвищуючи ефективність обробки замовлень.

У WarehouseService був доданий метод для отримання складів за їх ідентифікаторами. Цей метод дозволяє отримати інформацію про декілька складів одночасно, що є важливим для управління великими обсягами даних та забезпечення ефективної роботи логістичної мережі.

Таким чином, Service шар у Spring Boot забезпечує важливу функціональність для обробки бізнес-логіки та взаємодії з даними. Використання стандартних методів CRUD дозволяє легко виконувати основні операції з даними, тоді як додаткові унікальні методи забезпечують вирішення специфічних бізнес-завдань, роблячи систему більш гнучкою та ефективною.

4.4 Розробка класів для вимірювання використаних ресурсів

Для вимірювання завантаженості процесора та використання оперативної пам'яті СУБД було створено класи MonitoringThread і PerformanceMonitor.

Клас MonitoringThread реалізує механізм моніторингу системних ресурсів, таких як завантаженість процесора та використання оперативної пам'яті. Клас запускається у окремому потоці та періодично зчитує поточні показники системи.

Основні методи:

- run() - основний метод, який запускає потік та здійснює зчитування системних показників у визначені інтервали часу;
- collectCpuUsage() - метод для збору даних про завантаженість процесора;
- collectMemoryUsage() - метод для збору даних про використання оперативної пам'яті.

Клас PerformanceMonitor відповідає за ініціалізацію та зупинку моніторингу ресурсів, а також за збереження зібраних даних для подальшого аналізу.

Основні методи:

- startMonitoring() - метод для запуску процесу моніторингу;
- stopMonitoring() - метод для зупинки процесу моніторингу та збереження зібраних даних;
- getCollectedData() - метод для отримання зібраних даних у вигляді структурованого формату для подальшого аналізу.

Використання класів MonitoringThread та PerformanceMonitor дозволило отримати детальні дані про використання системних ресурсів під час тестування різних СУБД.

4.5Вимірювання часу виконання тестів

Для вимірювання часу виконання тестів використовувалися стандартні інструменти Java, зокрема, метож “System.nanoTime()”, який забезпечує високу точність вимірювань. Час виконання вимірювався для кожного окремого тесту та записувався у лог-файли для подальшого аналізу. Основні етапи вимірювання часу:

- Початок тесту. Вимірювання початкового часу перед виконанням тесту.
- Виконання операцій. Виконання запланованих операцій з базою даних.
- Завершення тесту. Вимірювання кінцевого часу після завершення тесту.
- Обчислення часу виконання. Визначення різниці між кінцевим та початковим часом для отримання тривалості виконання тесту.

Для точнішого вимірювання часу виконання методів у процесі тестування використовувалось багаторазове повторення виконання одного і того ж тесту з подальшим усередненням результатів. Це дозволяє зменшити вплив випадкових коливань системної продуктивності та отримати більш стабільні показники часу виконання.

4.6 Створення тестових даних

Для симуляції реальних сценаріїв у системі управління запасами було розроблено класи кожен з яких генерував об’єкт з шару сутностей з заповненими полями і доданими пов’язаними з ним об’єктами. Генерація тестових даних, чисел

і дат здійснювалась за допомогою класу ValuesGenerator, який відповідає за створення випадкових значень для властивостей об'єктів базових типів. В таблиці 4.1 вказано скільки об’єктів кожного класу шару сутностей було створено для тестування відповідних шарів repository і service.

Таблиця 4.1 – Кількість згенерованих об’єктів для тестування шару сутностей (таблиця виконана самостійно)

Назва сутності	Кількість
Role	5000
User	5000
Address	20000
Cargo	300
Cartegory	5000
Dimensions	10000
ItemOption	20000
Marketplace	5
OrderItem	2000
Order	1000
Product	5000
SaleTransaction	10000
Store	10
Warehouse	5
Weight	20000

Для кожної зі створених сутностей заповнюються усі властивості. Робиться це за допомогою класів які створюють об’єкти класу шару сутностей з вже заповненими властивостями. Властивості базових типів заповнюються за допомогою ValuesGenerator. Властивості ж які є класами шару сутностей генеруються створеними. У таблиці 4.2 вказані ті класи в яких окрім властивостей базових типів генеруються властивості шару сутностей. У цій таблиці вказано який клас генерує об’єкт, назва властивості, тип сутності і кількість згенерованих сутностей цього типу. Для створення сутностей генерувалися сутності двох видів – ті що були головними і дочірні в яких не генерувалися об’єкти які присутні в головних об’єктах. Так, при створенні об’єкта Order дочірній об’єкт OrderItem не ствоював новий об’єкт Order, а отримував посилання на головний об’єкт.

Таблиця 4.2 – Кількість згенерованих об’єктів для заповнення властивостей кожного зненерованого класа (таблиця виконана самостійно)

Клас який генерує сутність	Назва властивості згенерованого об’єкта яка зповнюється	Клас пов’язаної сутності	Кількість згенерованих пов’язаних сутностей для головного об’єкта
UserTestEntity	roles	Role	10
RoleTestEntity	users	User	10
WarehouseTestEntity	incomingCargoes	Cargo	100
WarehouseTestEntity	outgoingCargoes	Cargo	100
WarehouseTestEntity	orders	Order	100
WarehouseTestEntity	products	Product	100
CargoTestEntity	shipper	Address	1
CargoTestEntity	receiver	Address	1
CargoTestEntity	products	Product	10
MarketplaceTestEntity	stores	Store	5
CategoryTestEntity	products	Product	30
OrderItemTestEntity	weight	Weight	1
OrderItemTestEntity	unitPrice	Price	1
OrderItemTestEntity	taxAmount	Price	1
OrderItemTestEntity	shippingAmount	Price	1
OrderItemTestEntity	itemOptions	ItemOption	5
StoreTestEntity	orders	Order	100
ProductTestEntity	price	Price	1
ProductTestEntity	unitCost	Price	1
ProductTestEntity	shippingCost	Price	1
ProductTestEntity	customs	Price	1
ProductTestEntity	itemWeight	Weight	1
ProductTestEntity	volumeWeight	Weight	1
ProductTestEntity	packageWeight	Weight	1
OrderTestEntity	billTo	Address	1
OrderTestEntity	shipTo	Address	1
OrderTestEntity	orderItems	OrderItem	3
OrderTestEntity	orderTotal	Price	1
OrderTestEntity	amountPaid	Price	1
OrderTestEntity	taxAmount	Price	1
OrderTestEntity	shippingAmount	Price	1
OrderTestEntity	weight	Weight	1
OrderTestEntity	dimensions	Dimensions	1

Створення тестових даних для порівняння продуктивності СУБД у системі

управління запасами є важливим етапом дослідження. Використання класів для генерації даних дозволяє створювати об’єкти для тестування які хоч і мають кожен раз різні властивості використовують приблизно ту ж кількість ресурсів.

4.7 Створення тестів шару repository

Для тестування шару repository був створений клас “JpaRepositoryTest” в якому було реалізовано тестування стандартних методів класу “JpaRepository” з використанням узагальнених типів (generics). Кожен інший клас написаний для тестування класу шару repository успадковує загальні налаштування та методи з "JpaRepositoryTest". Це дозволяє забезпечити уніфікованість підходу до тестування та зменшити дублювання коду.

Тестові методи у класі "JpaRepositoryTest" охоплюють всі основні операції CRUD. Кожен метод забезпечує тестування певного аспекту взаємодії з базою даних, включаючи створення нових записів, читання існуючих, оновлення і видалення даних. Наприклад, тестування створення запису включає перевірку того, що після збереження новий об'єкт має унікальний ідентифікатор. Тестування читання перевіряє, що об'єкт може бути успішно знайдений за його ідентифікатором. Тестування оновлення включає зміну атрибутів існуючого об'єкта та перевірку збереження цих змін у базі даних. Тестування видалення передбачає видалення об'єкта та перевірку, що після цього об'єкт більше не може бути знайдений у базі даних.

У класі "JpaRepositoryTest" тестуються такі методи шару repository:

- метод “save(S entity);
- метод “findById(ID id)”;
- метод “findAll()”;
- метод “deleteById(ID id)”;
- метод “delete(S entity)”;
- метод “count()”;
- метод “existsById(ID id)”.

Додаткові методи які були створені спеціально для окремих класів шару

repository тестуються у відповідних класах окремо від класу "JpaRepositoryTest" який наслідують.

Кожен тест повторюється по 10 разів. До початку виконання операції запускається додатковий потік моніторингу ресурсів за допомогою класів “PerformanceMonitor” і “MonitoringThread”, також після запуску додаткового потоку запускається відлік часу виконання операції. Після завершення операції відлік часу зупиняється і додатковий потік також зупиняється записуючи результати виміру у файл.

4.7 Створення тестів шару service

Тестування шару service є ключовим аспектом забезпечення надійності та продуктивності систем управління запасами. Тестування цього шару дозволяє оцінити не тільки коректність виконання логіки, але й продуктивність та ефективність доступу до даних у різних СУБД.

Шар service є посередником між шаром controller та шаром repository, і саме він відповідає за виконання бізнес-логіки застосунку. Тестування цього шару має кілька ключових цілей:

- Перевірка коректності бізнес-логіки. Тестування гарантує, що всі бізнес-правила і логіка виконуються правильно.
- Оцінка продуктивності. Важливо виміряти час виконання та ресурси, що використовуються при взаємодії з різними СУБД.
- Перевірка інтеграції. Тестування забезпечує коректну інтеграцію шару service з іншими компонентами системи.

Тестування щару service включає ті операції які є копіями операцій шару repository бо для складних об’єктів часто потребується описати взаємодію об’єктів для виконання бізнес-логіки. Окрім методів які вже описані в шарі repository і присутні в шарі service були додані ще унікальні методи до класів шару service які є специфічними для систем управління запасами.

Для тестування класу CargoService було написано клас CargoServiceTest в якому окрім методів описаних в шарі repository були написані ще такі методи для

тестування:

- Метод setDestinationWarehouse тестує метод setDestinationWarehouse в cargoService. Перевіряє різні сценарії, включаючи нульові значення, неіснуючі об'єкти та заміну існуючого складу.
- Метод setSourceWarehouse тестує метод setSourceWarehouse в cargoService. Перевіряє різні сценарії, включаючи нульові значення, неіснуючі об'єкти та заміну існуючого складу.
- Метод addProduct тестує метод addProduct в cargoService. Перевіряє різні сценарії, включаючи нульові значення, неіснуючі об'єкти та додавання продуктів до вантажу.
- Метод addProductUsingId тестує метод addProduct в CargoService, використовуючи ідентифікатор продукту. Перевіряє різні сценарії, включаючи нульові значення, неіснуючі об'єкти та додавання продуктів до вантажу.
- Метод addProducts тестує метод addProducts в cargoService. Перевіряє різні сценарії, включаючи нульові значення, неіснуючі об'єкти та додавання списку продуктів до вантажу.
- Метод addProductsUsingId тестує метод addProducts в cargoService, використовуючи ідентифікатори продуктів. Перевіряє різні сценарії, включаючи нульові значення, неіснуючі об'єкти та додавання списку продуктів до вантажу.
- Метод removeProduct тестує метод removeProduct в cargoService. Перевіряє різні сценарії, включаючи нульові значення, неіснуючі об'єкти та видалення продукту з вантажу.

Для тестування класу CategoryService було написано клас CategoryServiceTest в якому окрім методів описаних в шарі repository були написані ще такі методи для тестування:

- Метод addProduct тестує додавання продукту до категорії.
- Метод addProductUsingProductId тестує додавання продукту до категорії за ID продукту.

– Метод removeProduct тестує видалення продукту з категорії.

Для тестування класу MarketplaceService було написано клас MarketplaceServiceTest в якому окрім методів описаних в шарі repository були написані ще такі методи для тестування:

– Метод getStores тестує отримання магазинів для конкретного майданчика.
– Метод addStore тестує додавання магазину до майданчика.
– Метод addStoreUsingStoreId тестує додавання магазину до майданчика за його ідентифікатором.

Для тестування класу OrderItemService було написано клас OrderItemServiceTest в якому окрім методів описаних в шарі repository був написаний ще додатково метод getTransactions. Він перевіряє поведінку при наданні недійсного orderItemId, створенні OrderItem з транзакціями та отриманні транзакцій для orderItemId.

Для тестування класу OrderService було написано клас OrderServiceTest в якому окрім методів описаних в шарі repository були написані ще такі методи для тестування:

– Метод addOrderItemUsingOrderItemId, цей метод використовується для тестування додавання елемента замовлення до замовлення за його ідентифікатором.
– Метод removeOrderItem, цей метод використовується для тестування видалення елемента замовлення з замовлення за його ідентифікатором.
– Метод findAllByStoreId, цей метод використовується для тестування пошуку всіх замовлень за ідентифікатором магазину.
– Метод findAllByIdWithPageable, цей метод використовується для тестування пошуку всіх замовлень за їх ідентифікаторами з використанням об'єкта Pageable для отримання результатів пошуку порціями.
– Метод findByOrderNumber, цей метод використовується для тестування пошуку замовлення за номером замовлення.

Для тестування класу ProductService було написано клас ProductServiceTest в

якому окрім методів описаних в шарі repository був написаний ще додатково метод getTransactions. Тест перевіряє різні сценарії, включаючи додавання, видалення транзакцій та виконує перевірку зв’язків між об’єктами.

Для тестування класу SaleTransactionService було написано клас SaleTransactionServiceTest в якому окрім методів описаних в шарі repository був написаний ще додатково метод revertTransaction. Тест перевіряє чи правильно відбувається відкат транзакції та чи відновлюється кількість продуктів після операції.

Для тестування класу StoreService було написано клас StoreServiceTest в якому окрім методів описаних в шарі repository були написані ще такі методи для тестування:

- Метод addOrderUsingOrderObject тестує додавання замовлення до магазину за допомогою об'єкта замовлення.
- Метод addOrderUsingOrderId - тестує додавання замовлення до магазину за допомогою ідентифікатора замовлення.
- Метод removeOrder - тестує видалення замовлення з магазину.
- Метод findByName - тестує пошук магазину за назвою.

Для тестування класу WarehouseService було написано клас WarehouseServiceTest в якому окрім методів описаних в шарі repository були написані ще такі методи для тестування:

- Метод addIncomingCargo тестує додавання вхідного вантажу до складу.
- Метод addIncomingCargoUsingCargoId тестує додавання вхідного вантажу до складу за ID вантажу.
- Метод removeIncomingCargo тестує видалення вхідного вантажу зі складу.
- Метод addOutgoingCargoUsingCargoId тестує додавання вихідного вантажу до складу за ID вантажу.
- Метод addOutgoingCargo тестує додавання вихідного вантажу до складу.
- Метод removeIncomingCargoUsingCargoId тестує видалення вихідного вантажу зі складу за ID вантажу.
- Метод addOrder тестує додавання замовлення до складу.

- Метод addOrderUsingOrderId тестує додавання замовлення до складу за ID замовлення.
- Метод removeOrder тестує видалення замовлення зі складу.
- Метод addProduct тестує додавання продукту до складу.
- Метод addProductUsingProductId тестує додавання продукту до складу за ID продукту.
- Метод removeProduct тестує видалення продукту зі складу.

Тестування шару service є важливим етапом забезпечення надійності та продуктивності систем управління запасами. Основними цілями тестування цього шару є перевірка коректності бізнес-логіки, оцінка продуктивності, та забезпечення інтеграції з іншими компонентами системи.

4.8 Підготовка додатку до тестування з різними СУБД

Для тестування кожної СУБД було зроблена своя копія програми в якій були внесені мінімальні зміни специфічні для кожної СУБД. Так, наприклад, для MongoDB був змінений тип ID сутності з Long на String. Також моніторинг процесу СУБД у кожній версії програми різний. Завдяки використанню Spring Data зміни для кожної версії програми були мінімальними.

4.9 Підготовка середовищ для запуску тестів

Тести проводилися на двох платформах - Linux Ubuntu 22.04.3 LTS та Windows Server 2019. В обох операційних системах на момент тестування було встановлено по 4 СУБД останньої версії. Для тестування використовувалися такі СУБД - PostgreSQL 16.2, MySQL 8.3, MongoDB 7.0, SQLite 3.45.02.

Віртуальна машина Windows мала процесор Intel Core i5-13500, оперативної пам’яті було 64 Гб, тип пам’яті – DDR4, дані і система зберігалися на двох дисках 512 GB NVMe SSD (Gen4) об’єднаних у RAID 1.

Віртуальна машина Linux мала процесор AMD Ryzen 5 3600, оперативної пам’яті було 64 Гб, тип пам’яті – DDR4, дані і система зберігалися на двох дисках 512 GB NVMe SSD (Gen4) об’єднаних у RAID 1.

5 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

5.1Збір результатів для аналізу

Всі результати тестів були зібрані та систематизовані в Excel-файлі. Для кожного з 10 повторів кожного тесту були зібрані дані щодо наступних метрик:

– використання оперативної пам’яті;

– використання процесорних потужностей;

– час виконання операцій.

Для класу з тестами шару repository і шару service було отримано середнє значення по кожній метриці і ці дані були записані у таблиці, кожна метрика має свою таблицю з даними. Далі були отримані середні значення по метрикам з усіх проведених тестів. Таблиця з даними додана до репозиторію з кодом програми.

5.2 Аналіз результатів на платформі Linux

На рисунку 5.1 зображений графік середнього використання оперативної пам’яті на платформі Linux трьома СУБД. Варто відзначити що через те що SQLite не має окремого процесу виконання виміряти використання пам’яті було неможливо.

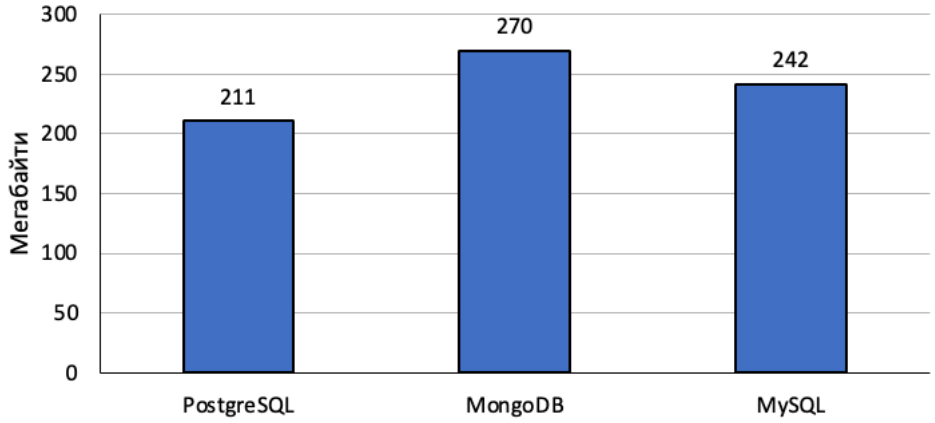

Рисунок 5.1 – Середнє використання оперативної пам’яті під час тестування на операційній системі Linux (рисунок створено самостійно)

Згідно з отриманими даними, PostgreSQL використовувала в середньому 211 МБ оперативної пам’яті, що є найменшим показником серед трьох розглянутих

систем. MongoDB продемонструвала найвище середнє споживання оперативної пам’яті — 270 МБ, тоді як MySQL використовувала 242 МБ, займаючи проміжне положення між PostgreSQL та MongoDB. З отриманих результатів можна зробити висновок, що PostgreSQL є найбільш економною щодо споживання оперативної пам’яті серед розглянутих СУБД. Високе споживання пам’яті MongoDB можна пояснити її архітектурними особливостями, зокрема орієнтацією на зберігання документів у форматі BSON, що може потребувати більше пам’яті для забезпечення швидкого доступу до даних. MySQL, яка поєднує реляційні та документно-орієнтовані підходи до зберігання даних, демонструє помірне споживання пам’яті.

На рисунку 5.2 зображений графік середнього використання CPU на платформі Linux трьома СУБД – PostgreSQL, MongoDB, MySQL.

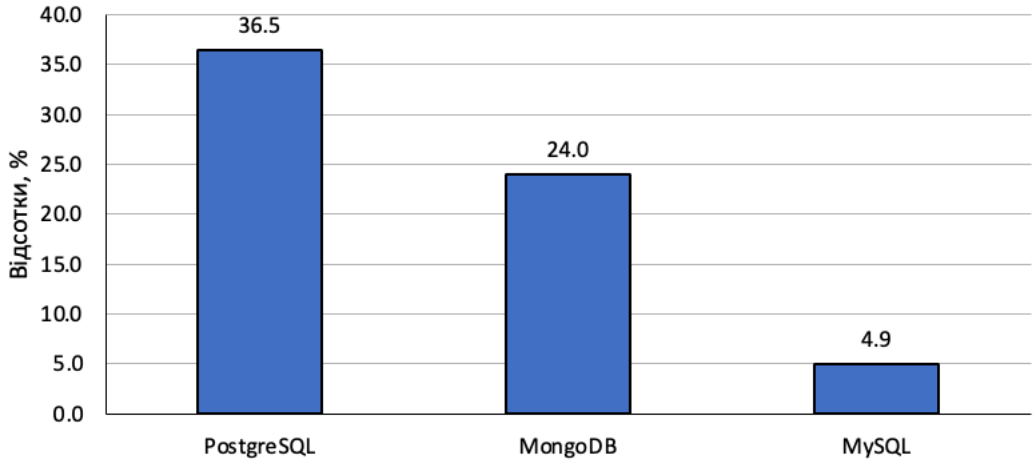

Рисунок 5.2 – Середнє значення використання CPU під час тестування на операційній системі Linux (рисунок створено самостійно)

Високий відсоток використання процесора PostgreSQL може бути обумовлений її ефективним виконанням складних запитів та підтримкою транзакційного режиму роботи. MongoDB, маючи середнє значення використання процесора 24%, також демонструє значне навантаження на процесор, що можна пояснити її документно-орієнтованою архітектурою, яка потребує додаткових обчислювальних ресурсів для обробки та індексації документів. MySQL показує найнижче використання процесорних ресурсів серед розглянутих СУБД — 4.9%.

Це може свідчити про її оптимізацію для роботи з меншими запитами або менш складними операціями, що дозволяє досягти ефективного використання процесора.

На рисунку 5.3 зображений графік середнього часу виконання транзакцій під час тестування на платформі Linux.

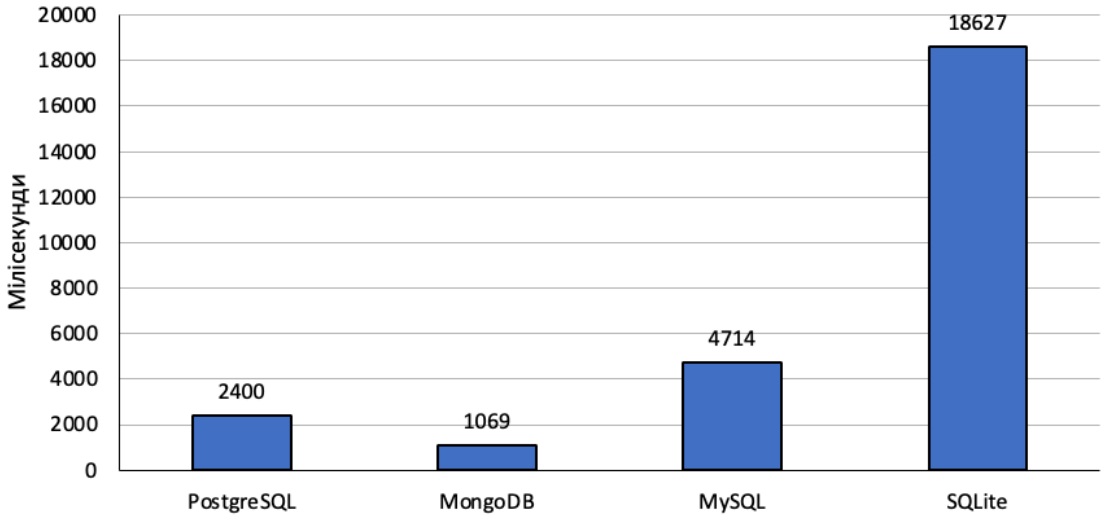

Рисунок 5.3 – Середній час виконання операцій під час тестування на операційній системі Linux (рисунок створено самостійно)

Результати тестування вказують на значні відмінності у продуктивності розглянутих СУБД щодо часу виконання транзакцій. MongoDB виявилася найшвидшою у виконанні транзакцій, що можна пояснити її оптимізацією для обробки великих обсягів даних та швидким доступом до документів. PostgreSQL також показала доволі високий рівень продуктивності, хоча і поступається MongoDB за швидкістю. MySQL демонструє вдвічі довший час виконання транзакцій порівняно з PostgreSQL, що може свідчити про її відносно нижчу продуктивність у виконанні складних транзакцій або ж необхідність додаткової оптимізації. SQLite показала найнижчу продуктивність серед усіх розглянутих СУБД. SQLite зберігає всю базу даних у єдиному файлі на диску. Це спрощує використання та інтеграцію, але може спричиняти значні затримки під час виконання транзакцій, особливо при великій кількості одночасних запитів, оскільки доступ до файлу блокується під час запису даних. Також для забезпечення цілісності даних, SQLite використовує файл-журнал, в якому зберігаються всі зміни

перед записом у базу даних. Цей додатковий крок додає час до виконання кожної транзакції.

5.3 Аналіз результатів на платформі Windows

На рисунку 5.4 зображений графік середнього використання оперативної пам’яті на платформі Windows трьома СУБД - PostgreSQL, MongoDB, MySQL.

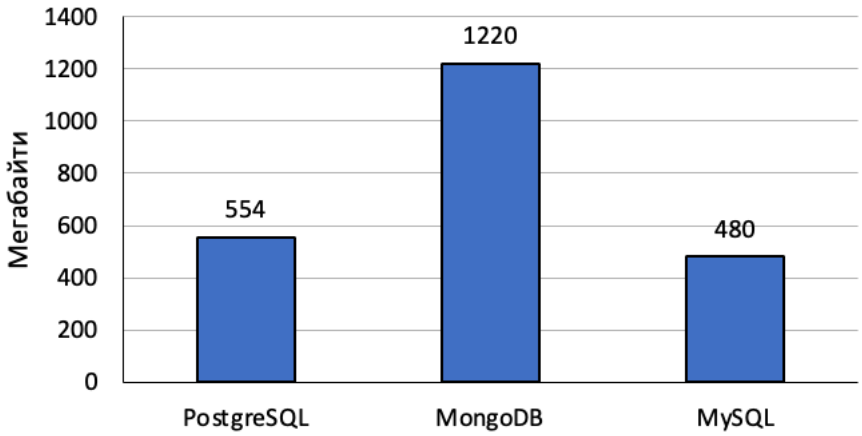

Рисунок 5.4 – Середнє використання оперативної пам’яті під час тестування на операційній системі Windows (рисунок створено самостійно)

Результати дослідження показують суттєві відмінності у використанні оперативної пам’яті різними СУБД під час виконання аналогічних навантажень. MongoDB, як документно-орієнтована СУБД, використовує найбільше ресурсів пам’яті, що може бути критичним фактором при виборі СУБД для проектів з обмеженими ресурсами. MySQL, навпаки, показує найбільш економічне використання оперативної пам’яті, що може бути перевагою для систем, де оптимізація ресурсів є пріоритетом. PostgreSQL забезпечує компромісний варіант з достатнім рівнем продуктивності при помірному використанні пам’яті.

На рисунку 5.5 зображений графік середнього використання CPU на платформі Windows.

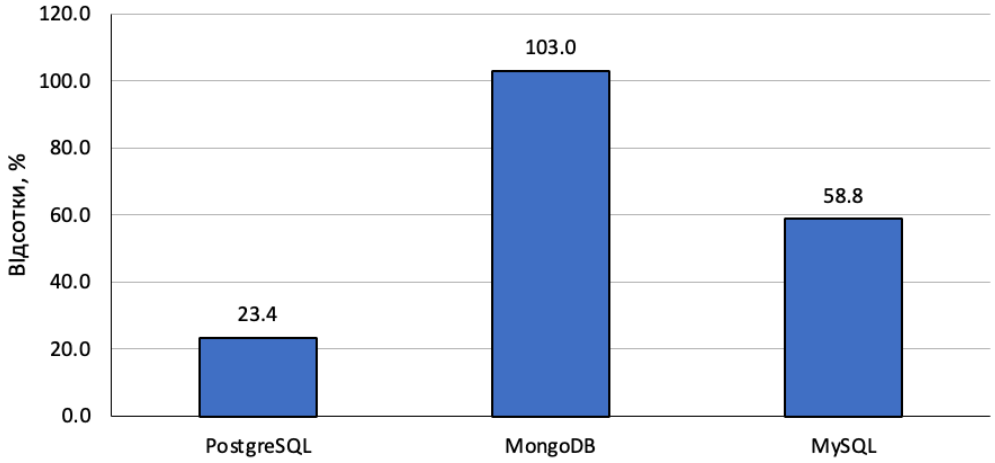

Рисунок 5.5 – Середнє використання CPU під час тестування на операційній системі Windows (рисунок створено самостійно)

Результати дослідження вказують на значні відмінності у використанні процесора різними СУБД під час виконання аналогічних навантажень. MongoDB, яка є документно-орієнтованою СУБД, використовує найбільше ресурсів процесора, що може бути критичним фактором при виборі СУБД для проектів з обмеженими обчислювальними ресурсами. MySQL демонструє помірне навантаження на процесор, що може бути вигідним для систем з середнім рівнем вимог до обчислювальних потужностей. PostgreSQL забезпечує найнижче використання процесора, що може бути перевагою для систем, де ефективність використання обчислювальних ресурсів є пріоритетом.

На рисунку 5.6 зображений графік середнього часу виконання транзакцій на платформі Windows. Результати дослідження вказують на значні відмінності у часу виконання транзакцій різними СУБД під час виконання аналогічних навантажень. MongoDB демонструє найшвидше виконання транзакцій, що робить її привабливим варіантом для застосунків, де важлива швидкість обробки даних. PostgreSQL забезпечує збалансовану продуктивність, що може бути вигідним для багатьох корпоративних застосунків. MySQL показує значно вищий час виконання транзакцій, що може бути обмежуючим фактором у системах з високими вимогами до швидкості транзакцій. SQLite має найвищий час виконання транзакцій, що робить її менш придатною для середовищ з високим навантаженням транзакцій,

але вона залишається корисною для менш вимогливих або вбудованих застосунків.

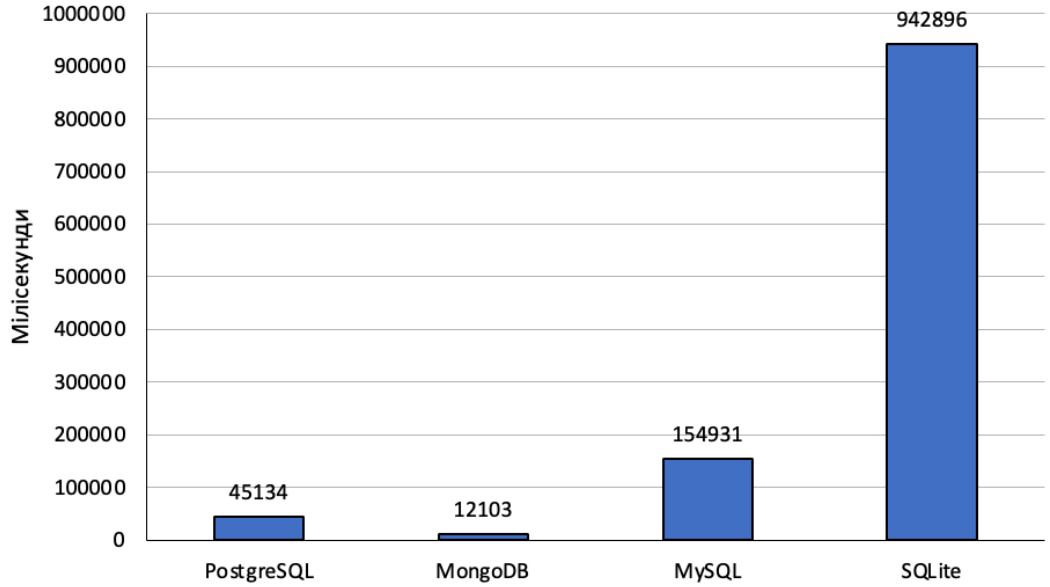

Рисунок 5.6 – Середній час виконання транзакцій під час тестування на операційній системі Windows (рисунок створено самостійно)

З результатів тестування на платформі Windows можна зробити такі висновки. MongoDB може підійти для систем, де важливий швидкий доступ до великих обсягів неструктурованих даних, але при цьому вона потребує значних ресурсів оперативної пам’яті та CPU. PostgreSQL забезпечує збалансовану продуктивність та ефективність використання ресурсів, підходить для багатьох корпоративних середовищ із середніми та високими вимогами до обробки даних. MySQL демонструє економне використання пам’яті, але значно поступається у швидкості виконання транзакцій, що обмежує її використання у високонавантажених системах. SQLite показує найвищий час виконання транзакцій, що робить її менш придатною для високонавантажених систем, але вона залишається корисною для менш вимогливих або вбудованих застосунків.

ВИСНОВКИ

У результаті виконання даного магістерського дослідження було досліджена продуктивність чотирьох СУБД з точки зору використання оперативної пам’яті, CPU і часу виконання транзакцій.

Підсумовуючи, продуктивність цих чотирьох СУБД на платформі Linux, ми бачимо, що SQLite є дуже повільною СУБД і навряд чи може вважатися СУБД з хорошою продуктивністю для використання у системах управління запасами. MongoDB була швидшою за інших кандидатів, але вона споживає більше процесорних ресурсів, ніж MySQL. Якщо брати оптимальний вибір серед тих СУБД які були розглянуті з точки зору швидкості виконання транзакцій, то для NoSQL СУБД це MongoDB, для SQL - PostgreSQL. Якщо критично важлива завантаженість процесора, то хорошим вибором може стати MySQL.

Для платформи Windows найкращим вибором є PostgreSQL з точки зору баланса між використанням ресурсів і часу виконання транзакцій. Якщо ж необхідно виконувати транзакції якомога швидше, а використання ресурсів є менш важливим, то для систем управління запасами можна обрати MongoDB. Інші кандидати оглянутих СУБД мають гірші результати на платформі Windows.

Вибір СУБД залежить від вимог компанії і це дослідження може допомогти зробити вибір в залежності від обраних критеріїв.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Benefits of Inventory Management and Inventory Management Systems. URL: https://www.netsuite.com/portal/resource/articles/inventory-management/inventory-management-benefits.shtml (дата звернення: 05.04.2024).

2. K. Smelyakov, A. Chupryna, D. Sandrkin and M. Kolisnyk, "Search by Image Engine for Big Data Warehouse," 2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2020, pp. 1-4, DOI: 10.1109/eStream50540.2020.9108782.

3. K. Smelyakov, A. Chupryna, O. Bohomolov and I. Ruban, "The Neural Network Technologies Effectiveness for Face Detection," 2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP), 2020, pp. 201-205, DOI: 10.1109/DSMP47368.2020.9204049.

4. Why Java is the Best Choice for Enterprise Development. URL: https://broscorp.net/java-for-enterprise-development/ (дата звернення: 05.04.2024)

5. Spring Boot. URL: https://spring.io/projects/spring-boot (дата звернення: 07.04.2024)

6. Why Spring? URL: https://spring.io/why-spring (дата звернення: 07.04.2024)

7. IntelliJ IDEA overview. URL: https://www.jetbrains.com/help/idea/discover-intellij-idea.html (дата звернення: 07.04.2024)

8. What is Junit and How it works? An Overview and Its Use Cases. URL: https://www.devopsschool.com/blog/what-is-junit-and-how-it-works-an-overview-and-its-use-cases/ (дата звернення: 10.04.2024)

9. Spring Data. URL: https://spring.io/projects/spring-data (дата звернення: 10.04.2024)

10. Spring Data JPA. URL: https://spring.io/projects/spring-data-jpa (дата звернення: 10.04.2024)

11. Guzhov, Viacheslav. “Free DBMSs Performance for an Inventory Management System based on Spring Boot.” In 2024 IEEE Open Conference of

Electrical, Electronic and Information Sciences (eStream). Conference Paper 697613, 2024.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

2. K. Smelyakov, A. Chupryna, D. Sandrkin and M. Kolisnyk, "Search by Image Engine for Big Data Warehouse," 2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), Vilnius, Lithuania, 2020, pp. 1-4, DOI: 10.1109/eStream50540.2020.9108782.

3. K. Smelyakov, A. Chupryna, O. Bohomolov and I. Ruban, "The Neural Network Technologies Effectiveness for Face Detection," 2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP), 2020, pp. 201-205, DOI: 10.1109/DSMP47368.2020.9204049.