Міністерство освіти і наукии України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка сервісу профілювання реляційних баз даних

Виконав: студент IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Бронецький Т.І.
(підпис) (прізвище та ініціали)

Керівник _____ Литвиненко Я.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Яцишин В.В.
(підпис) (прізвище та ініціали)

Тернопіль - 2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«_08_» ____06_____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Бронецькому Тарасу Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка сервісу профілювання реляційних баз даних

Керівник роботи Литвиненко Ярослав Володимирович, д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 14 » 05 2026 року № 4/9-239

2. Термін подання студентом завершеної роботи 21.06.2026 р.

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1. Постановка задачі та проектування архітектури.
2. Розробка допоміжного сервісу для імітації трафіку запитів до бази даних
3. Реалізація та тестування профілювальника
4. Безпека життєдіяльності, основи охорони праці
Висновки
Додаток А. Фрагмент коду з логікою проставлення параметрів запиту для подальшої побудови плану запиту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титулка. 2. Актуальність роботи , мета, задачі дослідження. 3. Функціональні вимоги до сервісу. 4. Пропонована архітектура рішення. 5. Використовувані технології 6. Таблиці БД допоміжного сервісу. 7. Список запитів, що підтримуються 8, 9. Відповіді сервера на окремі запити. 10. Приклад налаштування профілювальника та конфігурації даних. 11. Скріншоти записів журналів. 12. Аналіз отриманих результатів 13. Висновки. Основні результати проведеного дослідження.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 26.01.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01.26	*Виконано*
2.	Підбір джерел про профілювання реляційних баз даних	27.01 – 10.02.26	*Виконано*
3.	Опрацювання джерел про профілювання реляційних		
	баз даних	11.02 – 16.02.26	*Виконано*
4.	Виконання дослідження щодо розробки програмного		
	забезпечення для профілювання баз даних	17.02 – 10.05.26	*Виконано*
5	Розроблення програмного коду	11.05 – 20.05.26	*Виконано*
6.	Оформлення розділу «Постановка задачі та		
	проектування архітектури»	21.05 – 02.06.26	*Виконано*
7.	Оформлення розділу «Розробка допоміжного сервісу для		
	імітації трафіку запитів до бази даних»	02.06 – 04.06.26	*Виконано*
8.	Оформлення розділу «Реалізація та тестування		
	профілювальника»	05.06 – 07.06.26	*Виконано*
9.	Виконання завдання до підрозділу «Безпека		
	життєдіяльності, основи хорони праці»	31.05 – 03.06.26	*Виконано*
10.	Оформлення кваліфікаційної роботи	30.05 – 07.06.26	*Виконано*
11.	Нормоконтроль	03.06 – 08.06.26	*Виконано*
12.	Перевірка на плагіат	04.06 – 10.06.26	*Виконано*
13.	Попередній захист кваліфікаційної роботи	10.06 – 18.06.26	*Виконано*
14.	Захист кваліфікаційної роботи	22.06.26	

Студент _____ Бронецький Т.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Литвиненко Я.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка сервісу профілювання реляційних баз даних // Бронецький Тарас Ігорович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем та програмної інженерії, кафедра комп'ютерних наук, група СН–41 // Тернопіль, 2026 // С. – 53, рис. – 29, табл. – 8, слайдів – 13, бібліогр. – 17.

Ключові слова: база даних, запит, профілювання, сховище даних, PostgreSQL, Spring, Hibernate

Кваліфікаційна робота присвячена розробці спеціалізованого серверного інструменту для профілювання реляційних баз даних.

У розділі 1 здійснено постановку задачі на проектування. Проаналізована статистична інформація щодо зростання цифрових даних упродовж останніх 15 років. Проведено огляд окремих програмних продуктів для профілювання баз даних. Сформульовані вимоги до розроблюваного сервісу. Запропонована програмна архітектура розробки.

У розділі 2 наведено особливості розробки допоміжного сервісу, який використовується для імітації трафіку запитів до бази даних. Здійснено вибір технологій та засобів розробки для бази даних і серверної частини профілювальника. Побудована його архітектура, сформовано список підтримуваних запитів до бази даних. Наведено приклад роботи допоміжного сервісу.

У розділі 3 описано реалізацію та тестування основного профілювальника. Програмно втілено усі необхідні компоненти сервісу. Проведено тестування його функціональності та проаналізовано отримані результати.

У розділі 4 описані важливі питання безпеки життєдіяльності та основ охорони праці.

ANNOTATION

Development of relational database profiling service // Bronetskyi Taras // Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science // Ternopil, 2026 // P. - 53, Fig. - 29, Table - 8, Slide - 13, References - 17.

Keywords: database, query, profiling, data warehouse, PostgreSQL, Spring, Hibernate

The thesis deals with the development of a specialized server tool for profiling relational databases.

In chapter 1, the design task is formulated. Statistical information on the growth of digital data over the past 15 years is analyzed. An overview of individual software products for profiling databases is conducted. Requirements for the developed service are formulated. The software architecture of the development is proposed.

In chapter 2, the features of the development of an auxiliary service are presented, which is used to simulate the traffic of database requests. The technologies and development tools for the database and the server part of the profiler are selected. Its architecture is built, a list of supported database requests is formed. An example of the operation of the auxiliary service is given.

In chapter 3, the implementation and testing of the main profiler is described. All the necessary components of the service are implemented in software. Its functionality is tested and the results obtained are analyzed.

In chapter 4, important issues of life safety and the basics of labor protection are described.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ

БД (англ. database) – база даних.

ПЗ (англ. software) – програмне забезпечення.

Профілювальник БД (англ. data profiler) — це інструмент для аналізу структури, якості та вмісту даних у БД, який допомагає виявляти помилки, оптимізувати запити та покращувати керування інформацією.

ЗМІСТ

ВСТУП

Актуальність теми. Зі зростанням обсягів даних та складності додатків продуктивність БД стає дедалі критичнішою. Множина дослідницьких організацій, аналітичних агентств та технологічних компаній випускають регулярні звіти про обсяги та тенденції даних.

Необхідність у сервісах профілювання БД безпосередньо випливає з вимог сучасного світу у більш ефективному управлінні великими обсягами даних, їх аналізу та оптимізації роботи з ними. Від ефективної роботи з даними залежать прибутки компаній. Оптимізація продуктивності БД прямо впливає на досвід користувача. Швидкі та чуйні запити забезпечують більш плавне функціонування програм, покращуючи задоволеність користувачів. Профілювальники допомагають виявити вузькі місця в роботі БД, такі як повільні запити, неефективні індекси або неправильна конфігурація, що дозволяє вчасно вживати заходів щодо їх усунення. Також вони допомагають розробникам швидше та ефективніше знаходити та виправляти проблеми продуктивності, що зрештою скорочує час розробки та покращує якість коду.

Таким чином, профілювальники БД є актуальним та затребуваним інструментом для фахівців з БД та їх розробників.

Мета роботи – реалізація серверної програми для профілювання реляційних БД.

Для досягнення мети виділено ряд завдань:

- аналіз існуючих аналогів;
- формулювання вимог до функціоналу;
- проектування архітектури;
- реалізація функціоналу програми;
- розгортання програми;
- тестування створеного ПЗ.

Практичне значення одержаних результатів. Розробка може бути успішно використана для невеликих і середніх компаній або для невеликих проектів.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ ТА ПРОЕКТУВАННЯ АРХІТЕКТУРИ

1.1 Статистика зростання цифрових даних

У таблиці 1.1 наведені статистичні показники збільшення цифрових даних за оцінкою незалежного аналітичного агента [2].

Таблиця 1.1 – Статистика зростання даних

Рік	Отримано даних	Зміна порівняно із попереднім роком	Зміна порівняно із попереднім роком (%)
2011	5 зетабайт	3 зетабайт	150%
2012	6.5 зетабайт	1.5 зетабайт	30%
2013	9 зетабайт	2.5 зетабайт	38.46%
2014	12.5 зетабайт	3.5 зетабайт	38.89%
2015	15.5 зетабайт	3 зетабайт	24%
2016	18 зетабайт	2.5 зетабайт	16.13%
2017	26 зетабайт	8 зетабайт	44.44%
2018	33 зетабайт	7 зетабайт	26.92%
2019	41 зетабайт	8 зетабайт	24.24%
2020	64.2 зетабайт	23.2 Зетабайт	56.59%
2021	79 зетабайт	14.8 зетабайт	23.05%
2022	97 зетабайт	18 зетабайт	22.78%
2023	120 зетабайт	23 зетабайт	23.71%
2024	147 зетабайт	27 зетабайт	22.5%
2025	181 зетабайт	34 зетабайт	23.13%

Лише у період із 2019 по 2020 рік кількість даних збільшилася на 56,59% [2]. На початок 2025 року у світі згенеровано 181 зетабайт даних. Наприклад, для зберігання одного зетабайта потрібно приблизно 83 мільйони жорстких дисків, ємністю 12 терабайт.

1.2 Огляд окремих наявних рішень

У ході аналізу було розглянуто кілька популярних профілювальників, таких як HibernateProfiler, Dynatrace. Розглянемо докладніше Hibernate Profiler [3], на рисунку 1.1 зображено фрагмент його інтерфейсу користувача, який зображує швидкість виконання запиту і його текст.

Рисунок 1.1 – Дані про запит Hibernate Profile

Hibernate Profiler дозволяє відстежувати всі запити, що виконуються фреймворком Hibernate [4], в режимі реального часу. Це дозволяє виявляти повільні запити та проблеми продуктивності.

Також інструмент надає можливість аналізувати продуктивність виконання запитів, включаючи час виконання, кількість дзвінків та використання ресурсів. Відмінно сумісний із Hibernate, підтримує велику кількість його версій. Крім цього, він надає інтерфейс користувача.

Однак, інструмент сумісний тільки з Hibernate фреймворком для Java. Крім цього, до недоліків можна віднести складність налаштування цього профілювальника, тому що потрібне гарне знання Hibernate. З вузькоспрямованості даного профілювальника слід вказати погану інтеграцію із різними середовищами розробки, та іншими технологіями.

Розглянемо докладніше Dynatrace, на рисунку 1.2 представлений інтерфейс цієї платформи, де зазначено тимчасове розподіл для запитів, і навіть середні значення.

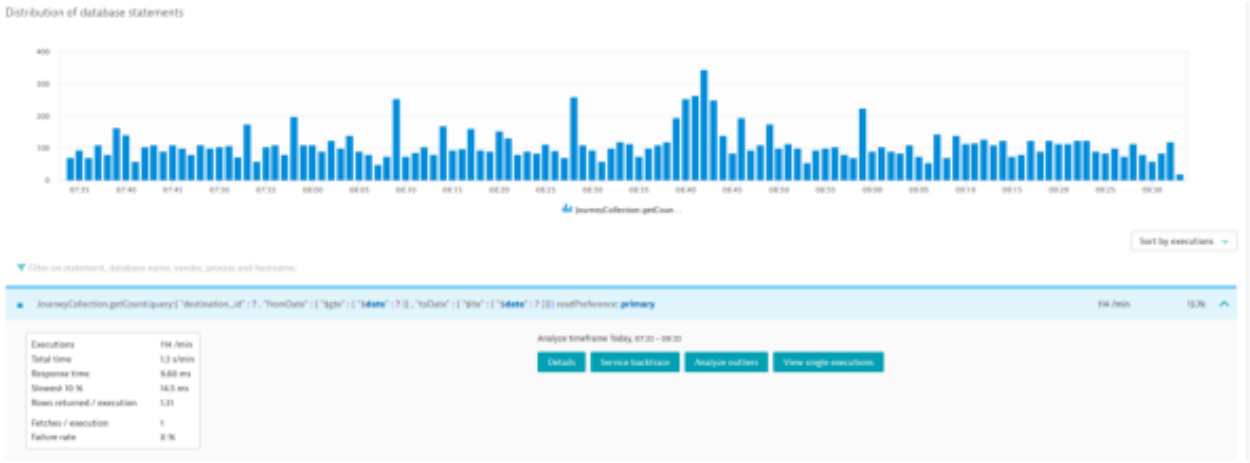

Рисунок 1.2 – Дані про трафік запитів у Dyntrace

Dynatrace - це платформа моніторингу продуктивності та управління продуктивністю додатків, яка надає широкий набір інструментів для аналізу та оптимізації продуктивності як хмарних, так локальних застосунків [5]. Вона відстежує стан мережі та інфраструктури, включаючи сервери, віртуальні машини, контейнери та хмарні сервіси, та подає деталізовану інформацію щодо продуктивності і доступності. Сама платформа забезпечує інструменти для аналізування продуктивності коду та запитів, виявлення вузьких місць, оптимізації ресурсів та покращення продуктивність застосунків.

До недоліків можна віднести те, що Dynatrace є комерційним продуктом, його використання може бути достатньо коштовним, особливо для невеликих і середніх компаній або для невеликих проектів. Також цю платформу складно налаштовувати, для цього потрібні навчені співробітники. Крім того, Dynatrace вимагає великої кількості ресурсів на розгорнутому сервері.

Отже, до переваг готових рішень можна віднести те, що вони добре інтегруються з різними реляційними БД, надають візуалізацію та деталізовані звіти. А до недоліків варто віднести обмеження таких рішень для роботи безпосередньо з фреймворками певних мов програмування та додаткове

навантаження на продуктивність, здебільшого ліцензії таких програмних продуктів досить дорогі.

1.3 Вимоги до сервісу

Відповідно до поставлених завдань, має бути розроблений сервіс для профілювання реляційних БД, що дозволяє відстежувати роботу БД, а також надавати поради щодо оптимізації запитів. Графічний інтерфейс не передбачається, оскільки він ускладнюватиме архітектуру та вимагатиме додаткові ресурси для власної роботи. Крім цього, профілювальник орієнтований на розробників та скриптову роботу, тому він повинен просто надавати зручний API. Виділено такі функціональні вимоги:

– профілювальник повинен мати можливість перехоплювати та аналізувати SQL -запити, що надсилаються додатком до БД;

– профілювальник повинен автоматично збирати статистичні дані про виконання SQL -запитів, включаючи час виконання, кількість дзвінків та інші метрики продуктивності;

– профілювальник повинен проводити аналіз продуктивності SQL -запитів та виявляти вузькі місця, причини уповільнення та інші проблеми, що впливають на продуктивність;

– на основі аналізу продуктивності профілювальник повинен надавати рекомендації щодо оптимізації SQL -запитів для покращення їх продуктивності;

– профілювальник повинен бути ефективним і мати мінімальний вплив на продуктивність програми, навіть при активному використанні.

До функціоналу системи мають бути пред'явлені такі вимоги:

– сумісність із найпопулярнішими реляційними БД (PostgreSQL, Oracle, MySQL);

– сумісність із будь-якими технологіями розробки.

1.4 Пропонована архітектура рішення

У цій роботі використана архітектура проміжного шару між основним додатком та базою даних, так званий проксі-шар (рисунок 1.3). Він представляє абстракцію для взаємодії між застосунком та БД, що дозволяє контролювати та аналізувати запити, що виконуються застосунком. Таким чином, можна контролювати доступ до БД, застосовувати різні політики безпеки та обмеження доступу, а також фільтрувати та обробляти запити перед їх виконанням. Можна додавати потрібний функціонал без необхідності зміни основної програми.

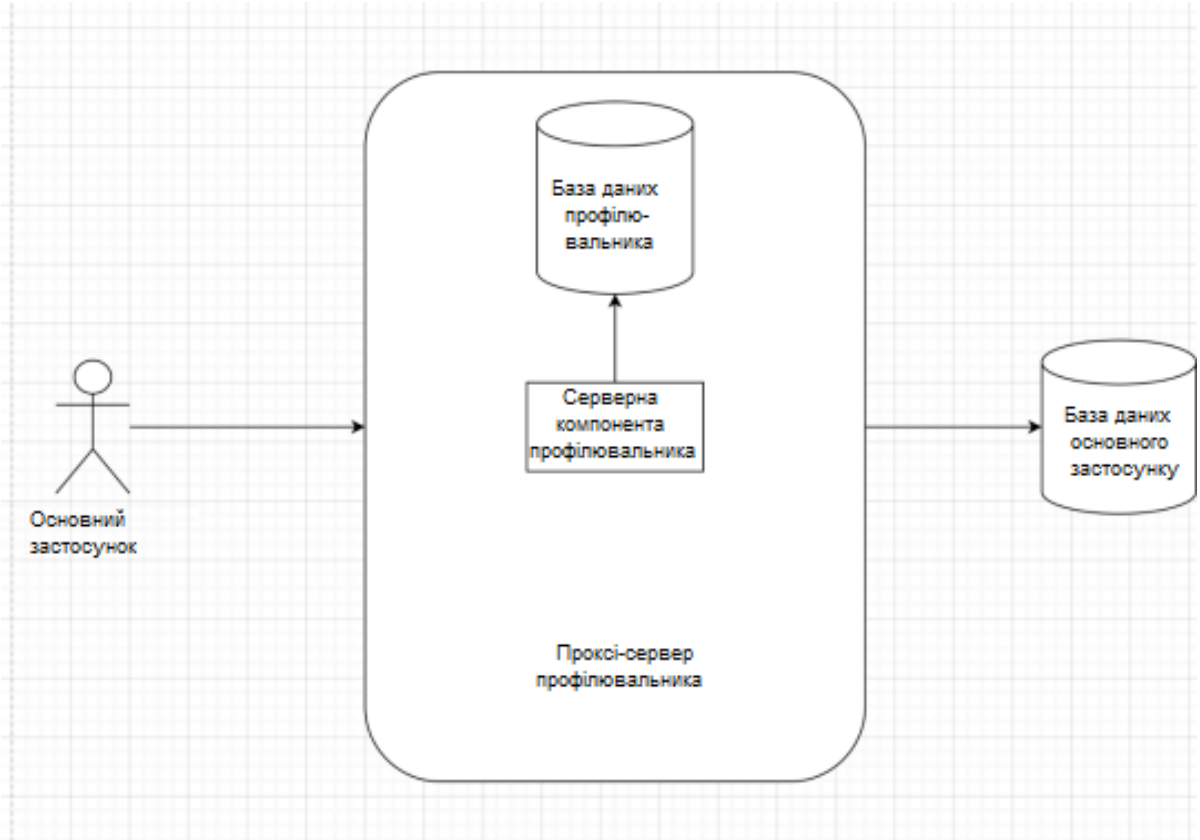

Рисунок 1.3 – Схема архітектури профілювальника, проксі шару

Даний сервіс організує клієнт-серверну взаємодію. У цьому випадку профілювальник виступає сервером для основної програми, і клієнтом для БД. У той же час основна програма, як і раніше, виступає клієнтом для БД, але тепер вона стає клієнтом і для проміжного шару – профілювальника. Крім цього, для

профілювальника спроектовано власну БД для допоміжних цілей. Основними компонентами логічної моделі є:

– серверний компонент профілювальника - відповідає за обробку запитів від основної програми;

– основний застосунок – є клієнтом для профілювальника;

– БД - зберігає дані основної програми, і навіть історію виконання запитів через вбудований аудит;

– БД профілювальника – допоміжне сховище для швидкого доступу до допоміжних даних.

Профілювання БД у будь-якому випадку пов'язане з додатковим навантаженням на неї, тому найефективнішим способом використовувати вбудовані можливості аудіювання сховища з метою їх подальшої обробки. Тому для налаштування роботи профілювальника потрібно включити логування запитів на рівні БД, це простіше та ефективніше, ніж перенаправляти історію запитів до іншого сховища.

Відповідно до функціональних вимог була розроблена наступна високорівнева архітектура сервісу.

1. Шар клієнтського інтерфейсу:

– компонент сервісу, через який клієнти зможуть взаємодіяти з профілювальником та отримувати необхідну інформацію;

– клієнт може надсилати запити до профілювальника через мережу Інтернет та отримувати відповіді у певному форматі. Звернення до сервера формуються за правилами GET, POST, UPDATE та DELETE запитів, а відповіді від сервера повертаються у форматі JSON.

2. Контролерний шар:

– обробляє вхідні запити від клієнтського шару та перенаправляє їх у сервісний шар;

– обробляє помилки при зверненні, перевіряє вхідні запити на коректність.

3. Сервісний шар:

– містить компоненти, що відповідають за обробку запитів та виконання

бізнес-логіки програми;

– включає сервіси для збору статистики, аналізу запитів, надання порад з оптимізації та інші прикладні компоненти;

4. Шар доступу до даних:

– взаємодіє з БД для отримання та оновлення даних профілювальника;

– містить компоненти, що відповідають за з'єднання програмного коду та мови SQL;

5. БД профілювальника:

– зберігає інформацію про користувачів програми;

– зберігає тимчасову інформацію, необхідну для швидкого доступу;

– може бути використане будь-яке сховище даних, як реляційне, так і не реляційне;

6. Компонент збору статистики:

– відповідає за збір статистики із під’єднаної до моніторингу БД;

– може інтегруватися з різними механізмами, вбудованими в реляційні БД, наприклад, для PostgreSQL [6] це буде подання pg_stat_statements, яке спрощує збереження інформації про запити;

7. Компонент аналізу запитів:

– здійснює аналіз статистики запитів, отримує побудовані БД плани виконання запитів;

– включає основні алгоритми для аналізу запитів, таких як: обробка статистики на кількість запитів читання або запису до таблиці, автоматична побудова планів запитів та їх аналіз, інші прикладні алгоритми, наприклад зіставлення та визначення пов'язаних відносин в БД;

8. Компонент надання порад щодо оптимізації:

– відповідає за генерацію оптимізаційних порад для запитів користувача;

– на основі аналізу запитів та виявлених дефектів вибирає відповідні рекомендації щодо оптимізації запитів;

9. Шар безпеки:

– забезпечує безпеку даних і доступ до функцій програми , а також

безпечне підключення до БД;

– включає вбудовані механізми для автентифікації, авторизації, шифрування даних.

Клієнтська частина для профілювальника безпосередньо з інтерфейсом користувача не передбачена в рамках поточної роботи.

РОЗДІЛ 2. РОЗРОБКА ДОПОМІЖНОГО СЕРВІСу ДЛЯ ІМІТАЦІЇ ТРАФІКУ ЗАПИТІВ ДО БАЗИ ДАНИХ

2.1 Використовувані технології

2.1.1 База даних профілювальника

Для допоміжних цілей створено БД провайдера PostgreSQL. Застосовується вона для збереження інформації щодо користувачів профілювальника та налаштування під'єднання до БД (таблиця 2.1).

Таблиця 2.1 – Відношення user_mfo у базі профілювальника

Відношення	Атрибут	Тип даних	Опис
user_info	user_id	varchar	Ідентифікатор користувача
	login	varchar	Логін користувача
	password	varchar	Пароль користувача
	url	varchar	Рядок під'єднання до БД
	size_threshol	varchar	Ліміт розміру таблиць

Поле user_id зберігає ідентифікатор користувача, який користується БД, це потрібно, щоб розуміти, запити якого користувача потрібно профілювати. Поля url, login, password використовуються для під'єднання до БД, що профілюється. Поле size_threshold використовується для налаштування ліміту розміру таблиць, при досягненні якого користувач отримуватиме пораду про необхідність оптимізувати таблицю.

2.1.2 Серверна частина профілювальника

Як мову програмування для розробки сервісу профілювальника була обрана мова Java, а також фреймворки Spring [1] та Hibernate.

Java була обрана як основна мова програмування через її популярність,

масштабованість і широке співтовариство розробників, також вона має платформну незалежність, що дозволяє використовувати профілювальник на будь-якій платформі [7]. Також важливою перевагою є наявність величезної кількості бібліотек для цієї мови. Крім цього, Java є ефективною та продуктивною мовою, що важливо для профілювальника. Важливим аспектом, що впливає на вибір мови, є її широка поширеність, тому при виникненні проблем у розробці можна легко знайти рішення, оскільки спільнота людей, які пишуть цією мовою та супутні технології є великою. Таким чином, вибір мови ґрунтується на широті її поширеності, швидкості роботи та зручності використання.

Spring Framework дуже зручний для розробки сервісних програм, завдяки широким можливостям і наявності великого інструментарію роботи з компонентами web і серверних програм [1]. Spring дозволяє управляти залежностями, впроваджувати їх та управління життєвим циклом бінів, так званих сервісних компонентів програми, що спрощує розробку та підтримку програми. Це надзвичайно важливо для масштабних програм, щоб винести з розробника необхідність управління класами, їх залежностями, впровадженнями та життєвим циклом вручну.

У зв'язку з тим, що профілювальнику потрібно постійно взаємодіяти з БД, то для роботи з даними був обраний Hibernate як ORM (Object-Relational Mapping), так як він пропонує зручний спосіб опрацювання даних в об'єктно орієнтованому стилі.

2.2 Розробка допоміжного сервісу

Оскільки для повноцінної перевірки роботи профілювальника потрібна основна програма, яку слід профілювати, то в рамках поточної роботи було розроблено допоміжний сервіс, покликаний виконувати роль програми, що профілюється. Застосунок, що профілюється, може бути абсолютно будь-яким, як мобільним, так і web. Головне, щоб він взаємодіяв із реляційною БД, і був трафік, який профілювальник зможе відстежувати.

Виходячи з цього було прийнято рішення створити додатковий web застосунок, який генеруватиме різні запити до БД з метою подальшої обробки трафіку та надання оптимізаційних порад.

Таким чином, головною вимогою для такої програми є можливість надати максимально різноманітний і великий трафік до БД. У ньому мають бути представлені різні види запитів, навмисно неефективні звернення, важкі з погляду процесорного використання запити та тривалі транзакції. Ця програма повинна забезпечувати інтенсивне тестування продуктивності БД, імітуючи реальні умови експлуатації та виявляючи потенційні вузькі місця та проблеми масштабованості.

Детальні моменти його архітектури та реалізації знаходяться поза рамками поточної роботи, оскільки цей застосунок виконує прикладну функцію, і його архітектура може бути будь-якою. Проте далі розглянемо кілька можливих підходів до його реалізації.

2.2.1 Архітектура допоміжного сервісу

При проектуванні програми використовувався шаблон RESTful застосунків (рисунок 2.1), таким чином, представлений сервіс реалізує концепцію клієнт-серверної взаємодії за протоколом HTTP. Цей підхід був обраний, оскільки він максимально простий та зручний у використанні

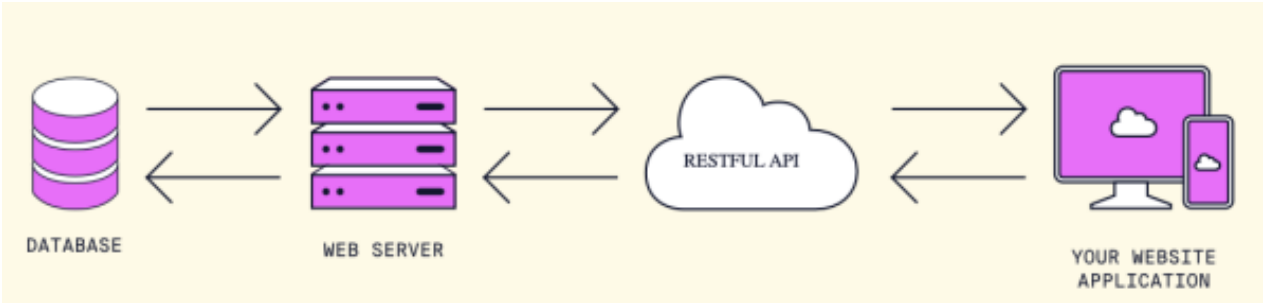

Рисунок 2.1 – Схема архітектури допоміжної програми RESTful сервісу

Таким чином, сервіс надає набір методів з бізнес-логікою, яка взаємодіє з БД, та інтерфейс, яким ця логіка може викликатися користувачем. Все це реалізовано через контракт HTTP- запитів.

Ця програма розроблена за спрощеною схемою, оскільки це допоміжний компонент. Вона не немає графічного інтерфейсу, лише підтримує інтерфейс для запитів, які одночасно викликають SQL запити БД.

Принциповим моментом є вибір сховища, адже воно має бути реляційним. Був обраний підтримуваний профілювальником провайдер PostgreSql, а також створені там дані були мігровані на Oracle [8] та MySql [9] для перевірки роботи взаємодії з цими провайдерами.

2.2.2 Схема бази даних

Для цього сервісу створено кілька таблиць (таблиці 2.2 – 2.5).

Таблиця 2.2 – Таблиця пісень у допоміжному сервісі

Таблиця	Атрибут	Тип даних	Опис
songs	id	bigint	Ідентифікатор пісні
	title	varchar	Назва пісні
	duration_in seconds	integer	Тривалість пісні
	lyrics	varchar	Текст пісні
	album_id	bigint	Посилання на альбом
	band_id	bigint	Посилання на групу

Таблиця 2.3 – Таблиця груп

Таблиця	Атрибут	Тип даних	Опис
bands	id	bigint	Ідентифікатор групи
	genre	varchar	Жанр групи
	name	varchar	Назва групи
	lyrics	varchar	Текст пісні
	band_id	bigint	Посилання на групу

Таблиця 2.4 – Таблиця альбомів

Таблиця	Атрибут	Тип даних	Опис
albums	id	bigint	Ідентифікатор альбому
	name	varchar	Назва альбому
	producer_i	bigint	Посилання на продюсера
	year	bigint	Рік випуску

Таблиця 2.5 – Таблиця продюсерів

Таблиця	Атрибут	Тип даних	Опис
producers	id	bigint	Ідентифікатор продюсера
	name	varchar	Назва продюсера

2.2.3 Список запитів, що підтримуються

Для імітації трафіку до БД було розроблено інтерфейс взаємодії із створеними сутностями, що включає різні запити на вставку даних, і навіть їх читання.

Сервіс підтримує запит на додавання бажаної кількості записів із випадково згенерованими полями, всілякі запити на читання записів, як читання всіх записів, читання записів по всіх полях сутності, а також підтримується запит на видалення всіх записів.

На рисунку 2.2 представлений список запитів, що підтримуються для таблиці пісень, для інших сутностей список запитів такий же.

Таким чином, підтримуються запити на створення випадкових пісень записів у потрібній кількості з прив'язкою до зв'язаних відносин, а також запити на очищення таблиці та на читання за різними параметрами.

Рисунок 2.2 – Список запитів, що підтримуються для сутності пісень

2.2.4 Приклад роботи сервісу

На рисунках 2.3, 2.4 представлені результати запуску допоміжного сервісу - деякі створені взаємозалежні таблиці, які автоматично додаються до БД під час першого запуску сервісу:

id	duration_in...	lyrics	title	band_id
1	300	Random lyrics f...	Random title fo...	1
2	230	Random lyrics f...	Random title fo...	3
3	240	Random lyrics f...	Random title fo...	4
4	410	Random lyrics f...	Random title fo...	2
5	370	Random lyrics f...	Random title fo...	7
6	220	Random lyrics f...	Random title fo...	2
7	300	Random lyrics f...	Random title fo...	8

Рисунок 2.3 – Додані записи до таблиці songs

id	name	genre
1	Random band 1	Heavy metal
2	Random band 2	Heavy metal
3	Random band 3	Trash metal
4	Random band 4	Rock
5	Random band 5	Death metal
6	Random band 6	Power metal
7	Random band 7	Black metal

Рисунок 2.4 – Додані записи до таблиці bands

Також написані методи, що дозволяють заповнювати таблиці випадковими даними, зокрема з реляційно пов'язаними між собою сутностями. На рисунку 2.5 наведено результат запиту на додавання 1000 нових випадкових пісень з посиланнями на випадкові музичні колективи.

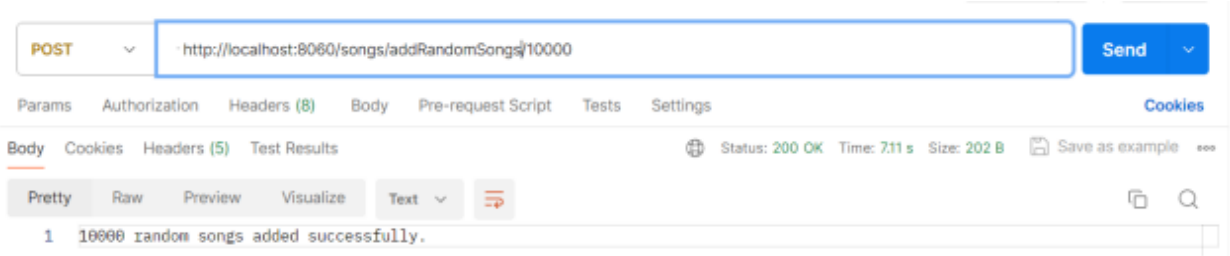

Рисунок 2.5 – Відповідь сервера на запит додавання 10000 випадкових пісень

Також в даному сервісі розроблені різні методи для зчитування з БД, в тому числі зчитування за необхідними полями. На рисунку 2.6 представлено результат виконання запиту отримання музичних композицій.

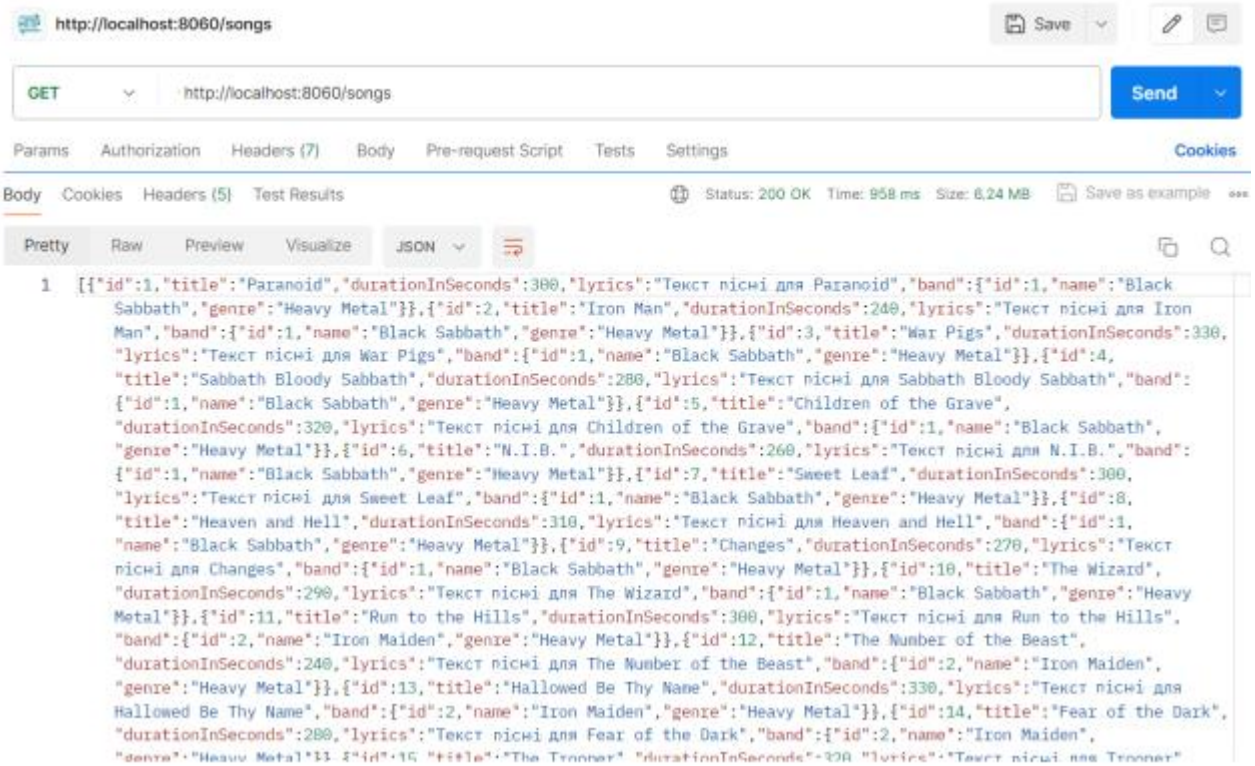

Рисунок 2.6 – Відповідь сервера на запит отримання всіх пісень

Таким чином, створений сервіс дозволяє проводити різні операції, наприклад запис та зчитування різних сутностей, що дозволить профілювальнику відстежувати та аналізувати різні запити до БД.

Наприклад, серед запитів буде повне послідовне сканування, індексне сканування, послідовний запис даних у базу, а також пакетна вставка.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОФІЛЮВАЛЬНИКА

3.1 Компоненти сервісу

Виходячи із спроектованої архітектури, були виділені основні компоненти сервісу:

– компонент під’єднання до аудиту сховища та отримання статистики. Цей компонент відповідає за під’єднання до системи аудиту БД та збирання даних про її використання. Аудит може включати інформацію про виконані запити, час їх виконання, використання індексів, частоту оновлень і видалень даних, а також будь-яку іншу статистику, яка може допомогти в аналізі продуктивності та ефективності роботи БД;

– компонент надання метрик та аналітичних даних на основі зібраної статистики. Цей компонент відповідає за аналіз зібраних даних та надання метрик та аналітичних звітів, які допомагають оцінити продуктивність БД. Він включає інструменти для візуалізації даних, такі як графіки, діаграми і звіти, які спрощують розуміння поточного стану і виявлення проблемних областей;

– компонент надання порад щодо оптимізації запитів. Цей компонент використовує дані, зібрані із системи аудиту та аналізу продуктивності, для генерації рекомендацій щодо покращення запитів та структури БД. Рекомендації можуть включати пропозиції щодо додавання індексів, переписування неефективних запитів, зміни структури таблиць тощо;

– компонент налаштування профілювальника. Цей компонент дозволяє користувачам налаштовувати профіль БД для точного збору даних про продуктивність та поведінку системи. Установки можуть включати вибір метрик для моніторингу, частоту збору даних, рівні деталізації та інші параметри, які допоможуть сконцентруватися на потрібних аспектах роботи БД.

3.2 Реалізація компонентів

3.2.1 Реалізація компоненту зв'язку з базою даних та отримання статистики

Профілювальник підключається до вбудованого аудиту реляційної БД, щоб обробляти статистичні дані звідти. Це найшвидший шлях аудіювання запитів БД. Однак у провайдерів незначно відрізняються способи зберігання даних. Наприклад, для PostgreSQL використовується подання pg_stat_statements. у MySQL використовується audit_log таблиця, а Oracle - AUD$.

На рисунках 3.1 – 3.3 наведено відповідні інструменти для перерахованих популярних провайдерів.

pg_stat_statements

select query, calls, round(total_time) as total_time, round(i Enter a SQL expression to filter results (use Ctrl+Space)

Grid Text

	query	calls	total_time	ms_per_call	shared_blks_hit	shared_blks_read
1	SELECT COALES	264,407,328	405,955	651	300,289,723	278
2	SELECT COALES	264,407,328	814,931	324	794,323,332	4
3	SELECT thisYear	264,407,328	16,285,849	16	2,224,608,068	18,840
4	SELECT COUNT	264,407,328	745,949	354	331,796,153	66
5	SELECT	261,491,364	552,850	473	785,574,762	1
6	SELECT thisYear	147,111,140	9,725,535	15	1,031,073,687	570
7	SELECT COALES	147,111,140	504,043	292	441,962,270	486
8	SELECT COALES	147,111,140	129,320	1,138	0	0
9	SELECT	147,111,140	319,186	461	441,962,756	0
10	SELECT COUNT	147,111,140	537,315	274	147,110,655	0

Рисунок 3.1 – Подання pg_stat_statements у PostgreSQL

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	6.6	2.0	0.01	0.05
DB CPU(s):	1.3	0.4	0.00	0.01
Redo size:	2,567,433.2	769,179.8		
Logical reads:	46,431.8	13,910.6		
Block changes:	1,790.9	536.5		
Physical reads:	11,888.1	3,561.6		
Physical writes:	336.0	100.7		
User calls:	119.3	35.8		
Parses:	30.3	9.1		
Hard parses:	3.6	1.1		
W/A MB processed:	25.6	7.7		
Logons:	9.1	2.7		
Executes:	676.9	202.8		

Рисунок 3.2 – Таблиця audit_log у MySQL

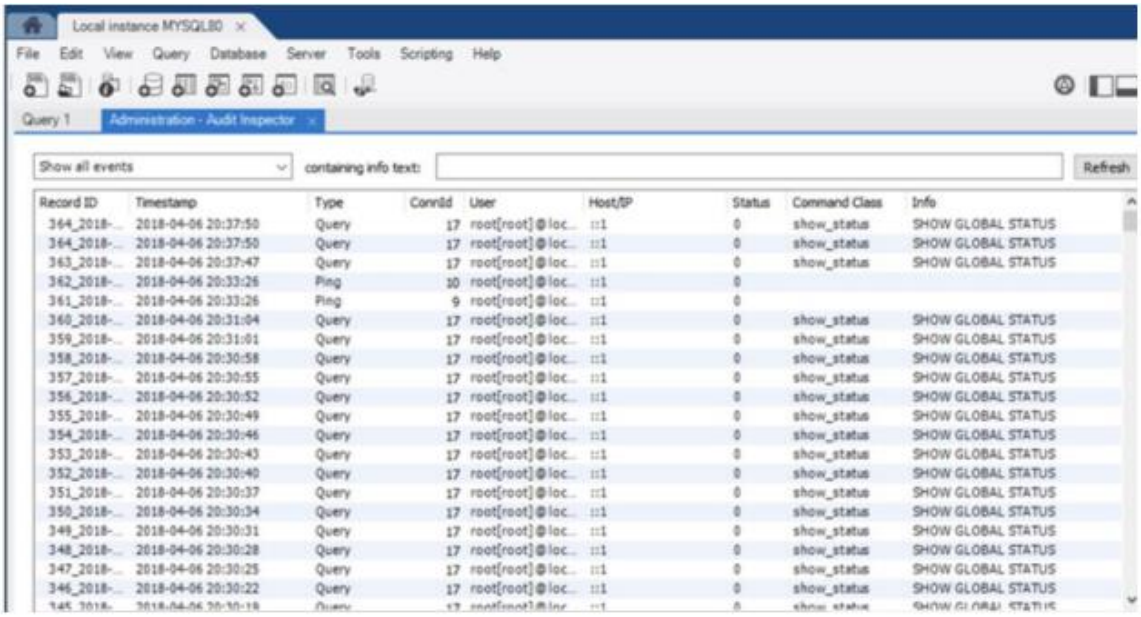

Рисунок 3.3 – Таблиця AUD$ в Oracle

Оскільки для різних сховищ спосіб зберігання журналів виконаних запитів дещо відрізняється, тут застосований шаблон стратегії (рисунок 3.4). З його допомогою визначається сімейство алгоритмів (способів взаємодії з різними провайдерами) та інкапсулюємо кожен із них у єдиний інтерфейс взаємодії з аудіованими даними. Таким чином, можна легко додавати підтримку нових провайдерів і замінювати наявні алгоритми.

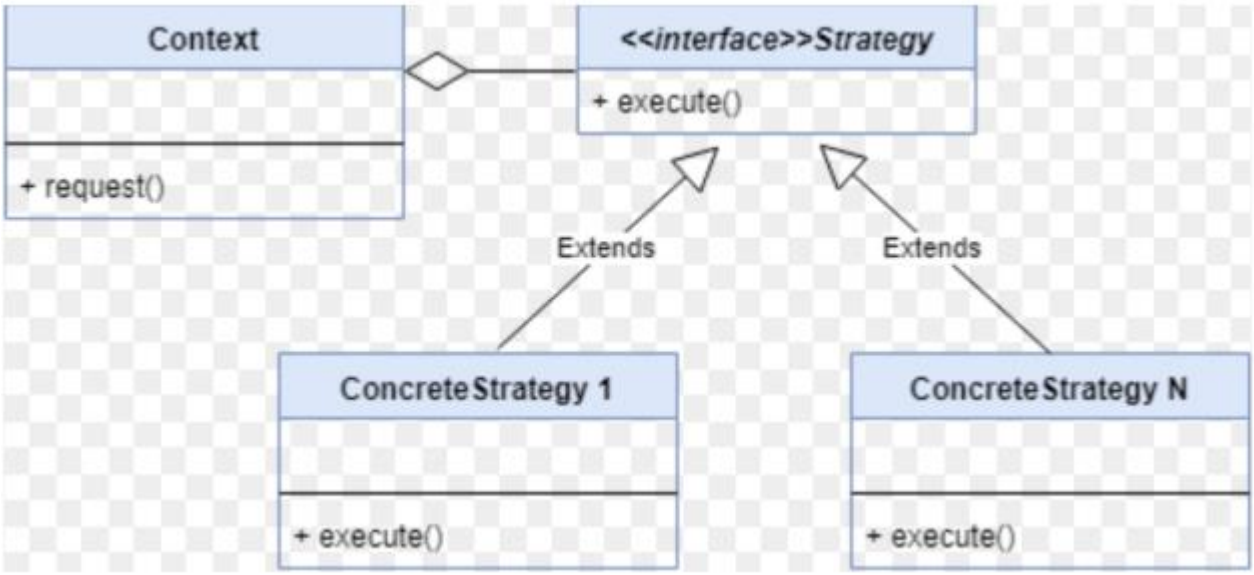

Рисунок 3.4 – Схема шаблону «Стратегія»

Тут виділяється сутності кожної з БД, але згодом переводимо все до єдиної моделі, з якою взаємодіятимемо. Сімейство алгоритмів для під'єднання до конкретних провайдерів і є тими стратегіями, які підмінюються. Так виглядає загальна модель для аудійованого запису (таблиця 3.1).

Таблиця 3.1 – Модель для аудійованих даних

Модель	Поле	Тип даних	Опис
Query	id	Long	Ідентифікатор запиту
	text	String	Текст запиту
	userId	String	Ідентифікатор користувача
	maxExecTime	Long	Максимальний час виконання такого запиту
	minExecTime	Long	Мінімальний час виконання такого запиту
	meanExecTime	Long	Середній час виконання запитів такого типу
	totalExecTime	Long	Загальний час виконання таких запитів
	sessionId	String	Ідентифактор сесії

3.2.2 Реалізація компонента надання метрик продуктивності та аналітичних даних

Далі було реалізовано можливість надання аналітичної інформації за запитами до БД на основі отриманих доступів до аудитів сховищ. На рисунку 3.5 продемонстровано список запитів, що підтримуються.

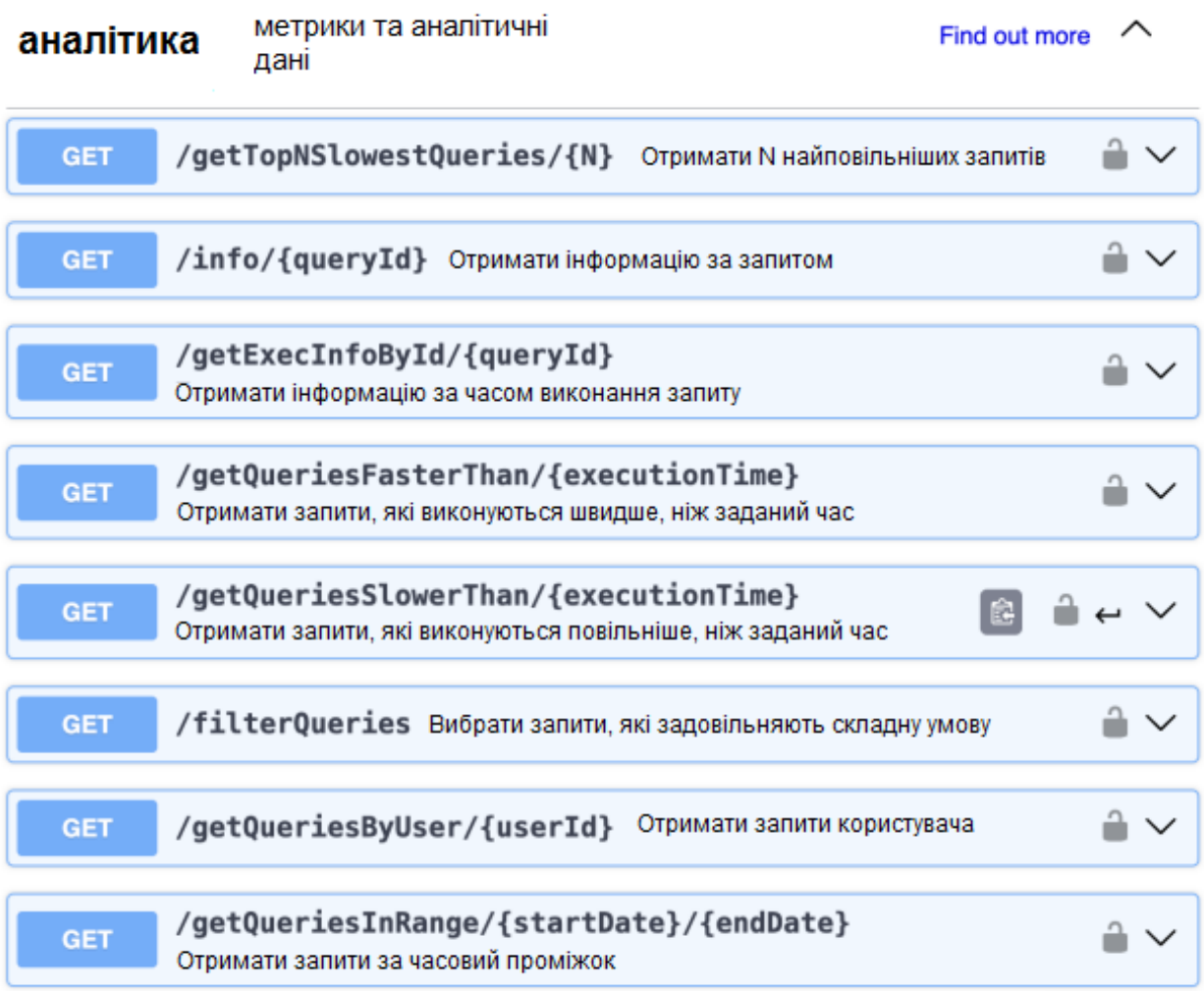

Рисунок 3.5 – Список запитів, що підтримуються для отримання аналітичних даних

Так, наприклад, профілювальник дозволяє отримати список з необхідною кількістю повільних запитів. На рисунку 3.6 показаний приклад запиту та відповіді для отримання трьох найповільніших запитів до БД, у відповіді отримую ідентифікатори запиту та користувача, а також текст самого запиту та час його виконання.

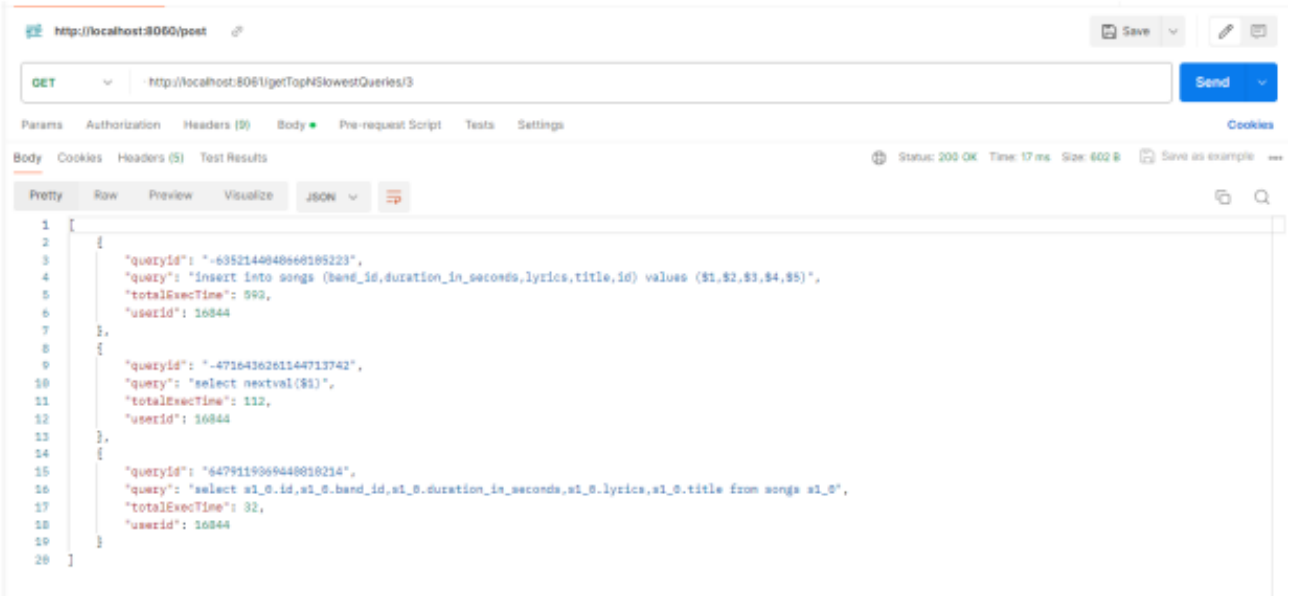

Рисунок 3.6 – Відповідь сервера на запит отримання трьох найповільніших запитів

Рисунок 3.7 демонструє відповідь сервера на запит отримання інформації про виконання запитів з ідентифікатором 13754. У відповіді повертається середній час виконання одного такого запиту, загальний час, витрачений на виконання цих запитів, а також максимальний і мінімальний час для запитів.

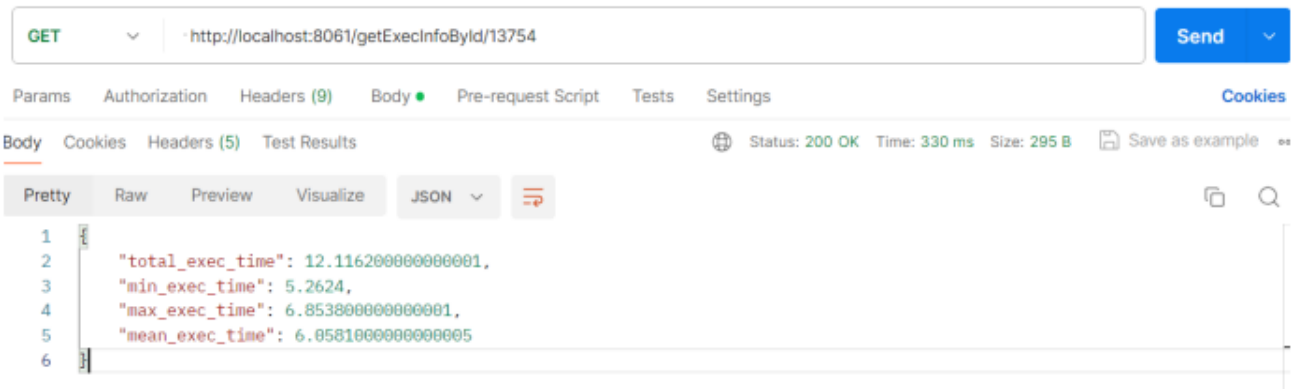

Рисунок 3.7 – Відповідь сервера на запит отримання тимчасових метрик конкретного запиту

На рисунках 3.8, 3.9 наведено результати запитів для отримання запитів, повільніше, ніж певний час, а також отримання запитів певного користувача.

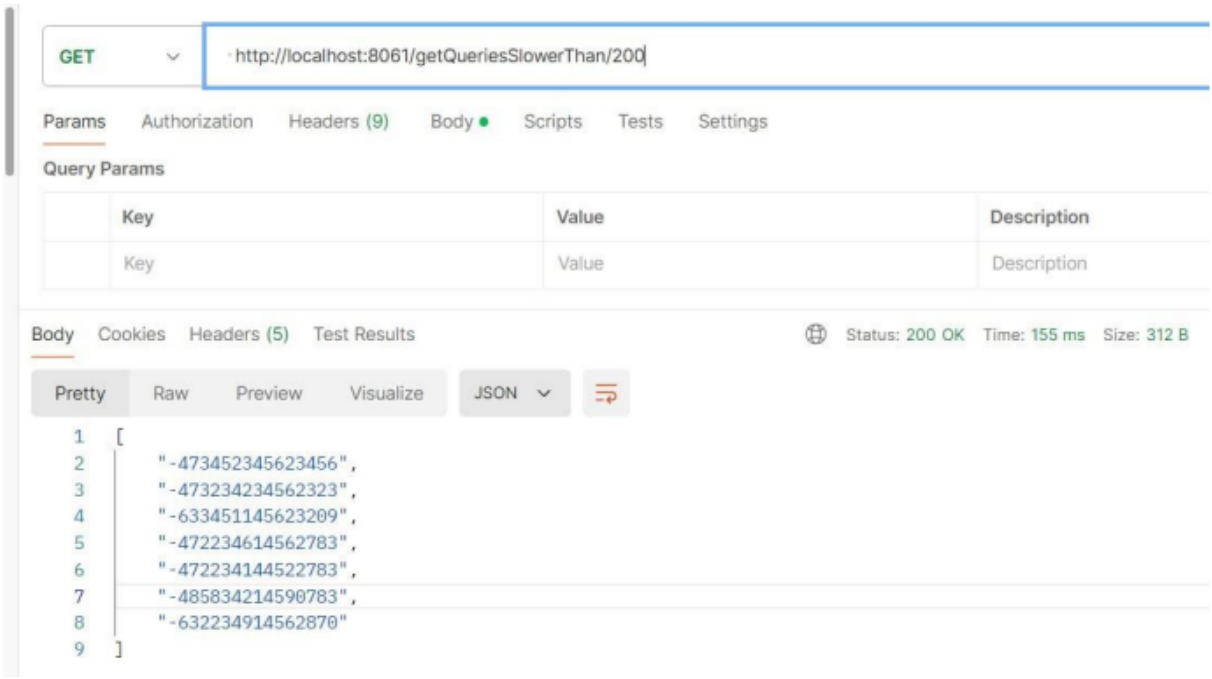

Рисунок 3.8 – Відповідь сервера на запит отримання всіх запитів, повільніших, ніж 200мс

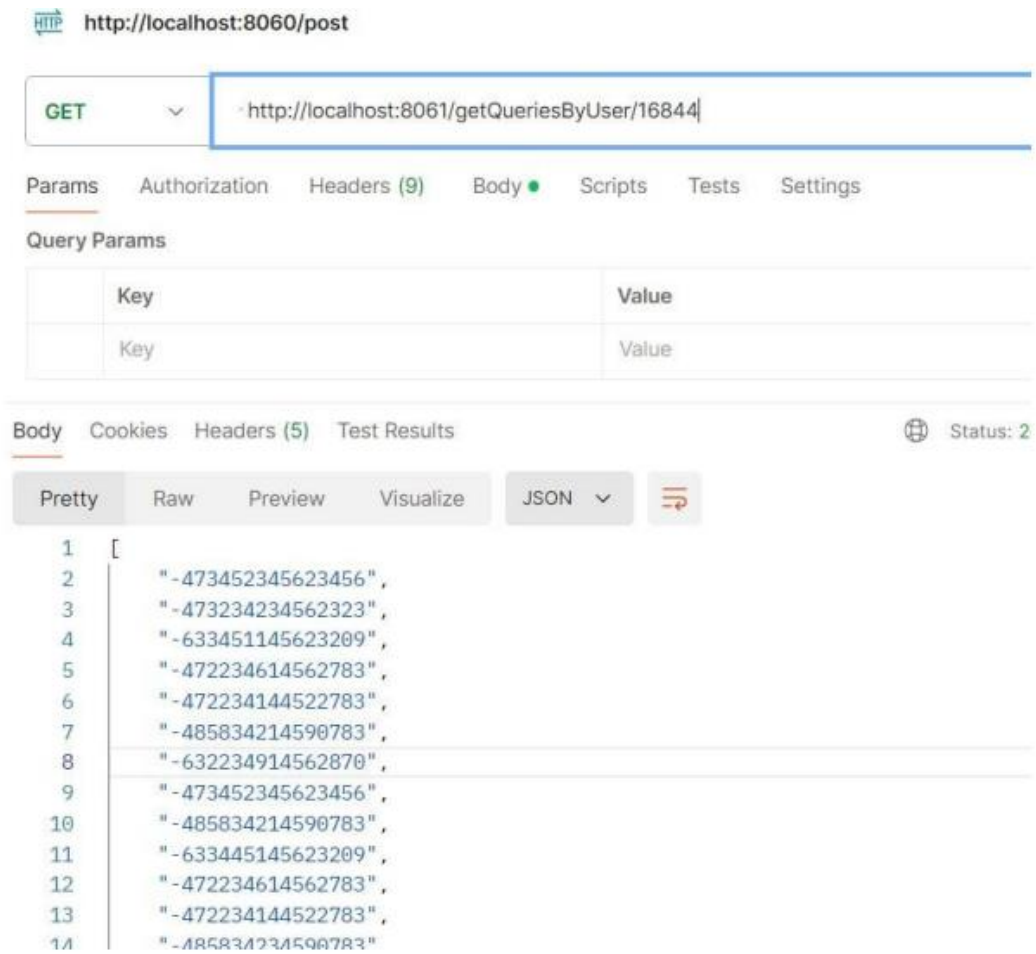

Рисунок 3.9 – Відповідь сервера на запит отримання всіх запитів певного користувача

На рисунку 3.10 показано результат запиту для отримання запитів за складними критеріями, одразу за декількома параметрами. Для цього використовується ResponseBody у запиті, куди передаються необхідні параметри для фільтрації.

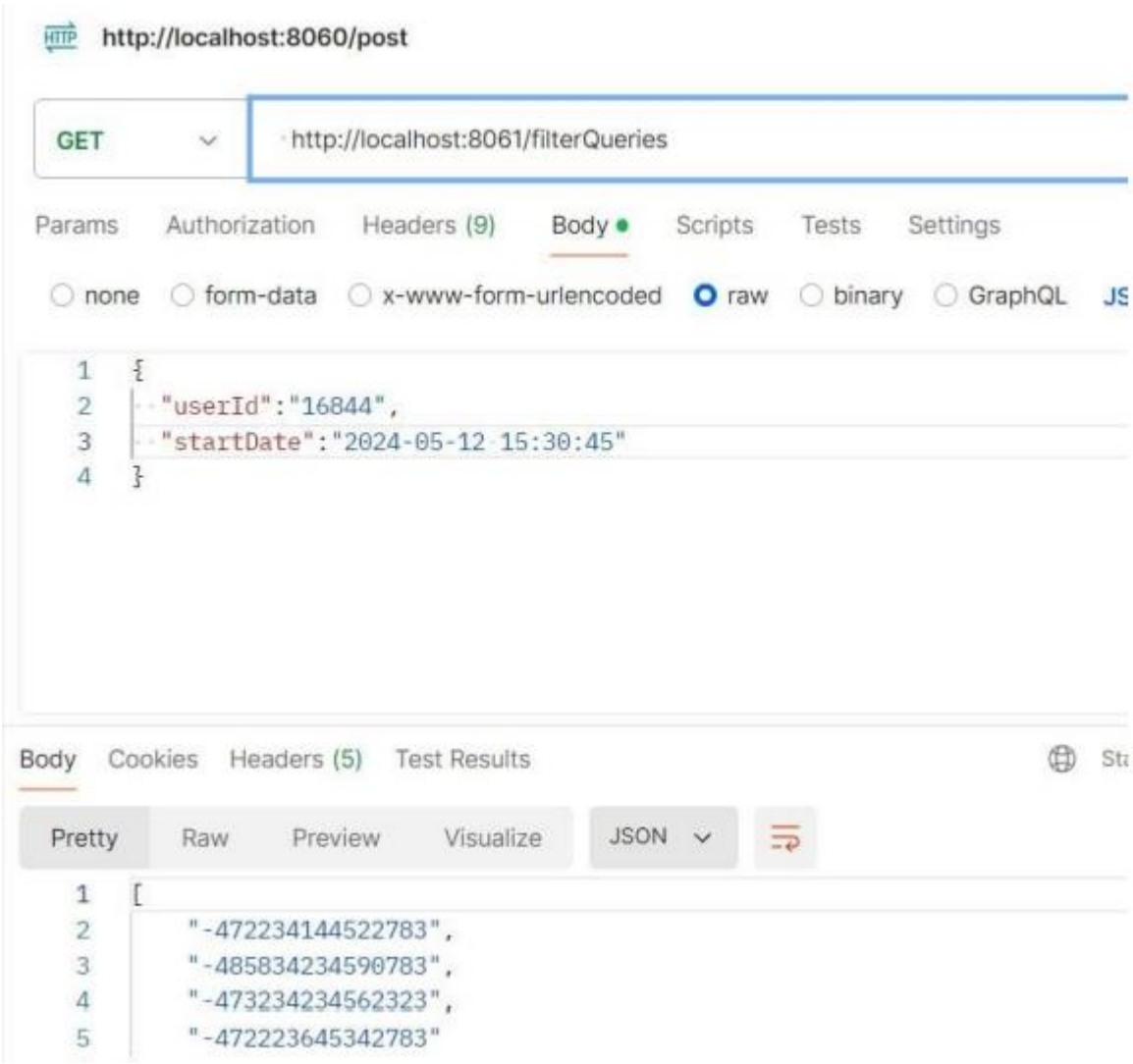

Рисунок 3.10 – Відповідь сервера на складний запит отримання запитів за декількома параметрами

3.2.3 Реалізація компоненту надання рекомендацій щодо покращення роботи бази даних

У профілювальнику реалізовано сервісний шар, який відповідає за аналіз запитів до БД та надання порад щодо оптимізації. Для реалізації подібної функціональності використовується шаблон «ланцюжок обов'язків». Він дозволяє створити ланцюг об'єктів-обробників, де кожен об'єкт може спробувати

опрацювати запит самостійно або передати його наступному об'єкту в ланцюзі. Така організація дозволяє розділити обробку запиту на окремі етапи та забезпечити гнучкість додавання нових перевірок чи перетворень без зміни клієнтського коду. На рисунку 3.11 представлена схема ланцюжка, що містить необхідні оптимізаційні перевірки запитів.

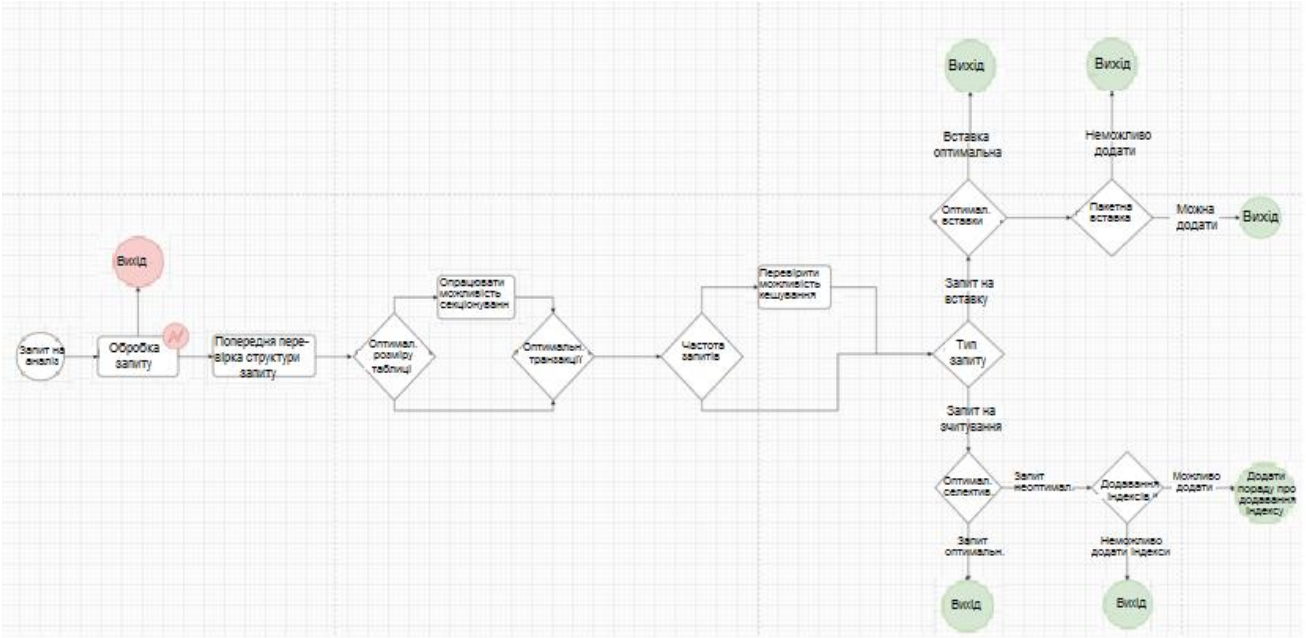

Рисунок 3.11 – Загальна схема ланцюжка обов'язків для алгоритму оптимізації

Інформація з оптимізації надається шляхом звернення на єдину адресу, яка повертає зведення за запитами користувача у форматі запит - порада. Сервер надасть відповідь на запит про отримання порад для запитів поточного користувача.

3.2.4 Реалізація алгоритму перевірки оптимальності запиту

Розглянемо ланцюжок докладніше. На вході отримую запит із БД. Перша проблема обробки полягає в тому, що нам не слід пропускати різні системні запити, наприклад select version(),COMMIT, SET ROLE 'spring', set search_path = 'public', CREATE index idx і т. д. За допомогою регулярних виразів відсіваються службові запити, які не є цінними і не можуть бути оптимізовані.

Друга проблема полягає в тому, що запити надходять із заглушеними

параметрами у вигляді таких символів: $1. Тому алгоритм проставляє певні значення залежно від того, яке це поле, і що там має бути, щоб надалі можна було скласти план запиту (рисунок 3.13).

У разі виникнення будь-яких помилок на цьому етапі, а також при надходженні службового запиту завершується оптимізаційна перевірка. На наступному етапі проводиться низка первинних перевірок структури запиту, які дозволяють виявити проблеми з продуктивністю. Наприклад, під час перевірки запиту на використання операторів «*» та на неефективне використання різних конструкцій, таких як «OUTER JOIN», або вбудованих функцій, що викликаються на кожній ітерації.

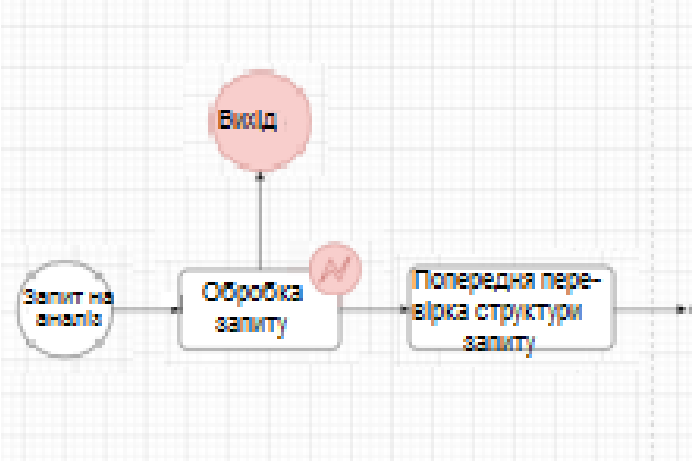

Рисунок 3.13 – Елементи ланцюжка, що відповідають за отримання та обробку запиту

На наступних етапах проводяться перевірки роботи з транзакціями, перевірки на частоту запитів, а також розмір таблиці. Так, якщо запити виконуються часто, проводиться перевірка можливості додавання кешування. Якщо дані оновлюються не часто і немає динамічних параметрів, а також у запиті немає конфіденційних даних (наприклад, полів password, login, token), буде запропоновано додавання кешування.

Якщо розміри таблиці перевищують налаштовані ліміти, перевіряється можливість додавання секціонування даних. Якщо немає запитів на всі поля, великих вибірок даних, а також немає складних умов, то буде виведена порада

про розгляд можливості поділу таблиці на дрібніші та керованіші частини за певним ключем, наприклад, за датою створення записів, якщо такий існує (рисунок 3.14).

Крім того, перевіряється оптимальність роботи транзакцій. Перевіряється правильність використання рівнів ізоляцій, наприклад, якщо використовується serializable рівень і при цьому немає паралельного доступу до даних, виводиться порада, що такий рівень ізоляції зайвий.

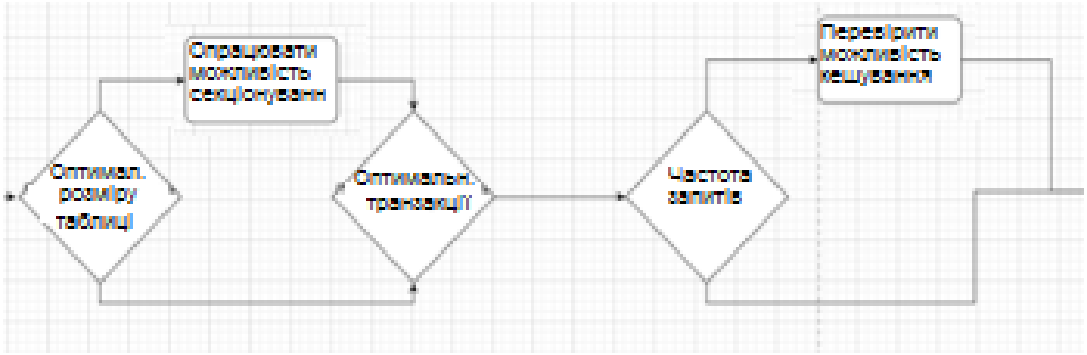

Рисунок 3.14 – Елементи ланцюжка, що відповідають за обробку секціонування та кешування

Наступним етапом перевіряється тип запиту. У тому випадку, якщо отримали запит на читання даних, то перевіряється ефективність вибірки, перевіряється, чи використовується індексне сканування, який розмір вибірки.

Якщо вибірка неефективна, перевіряється можливість додавання індексу. Якщо до таблиці здійснюється більше звернень для зчитування, аніж для запису, то пропонується додати індекс, залежно від запитів (рисунок 3.15). Наприклад, якщо проводяться лише вибірки з двох полів, буде запропоновано створити складовий індекс для цих полів.

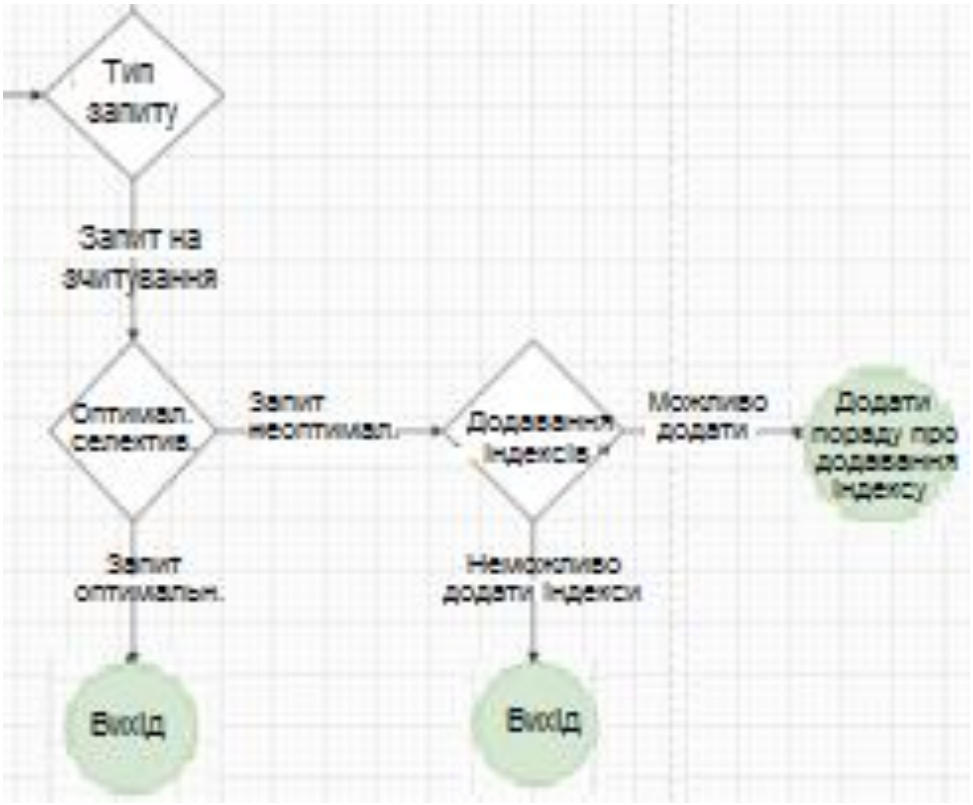

Рисунок 3.15 – Елементи ланцюжка, що відповідають за обробку запитів зчитування

Якщо надходить запит на запис даних, необхідно перевірити можливість використання пакетної вставки даних відповідно до поточної стратегії додавання даних. Якщо для таблиці застосовуються послідовності для генерації ключів і відсутні динамічні дані, рекомендується розглянути варіант пакетної вставки (рисунок 3.16).

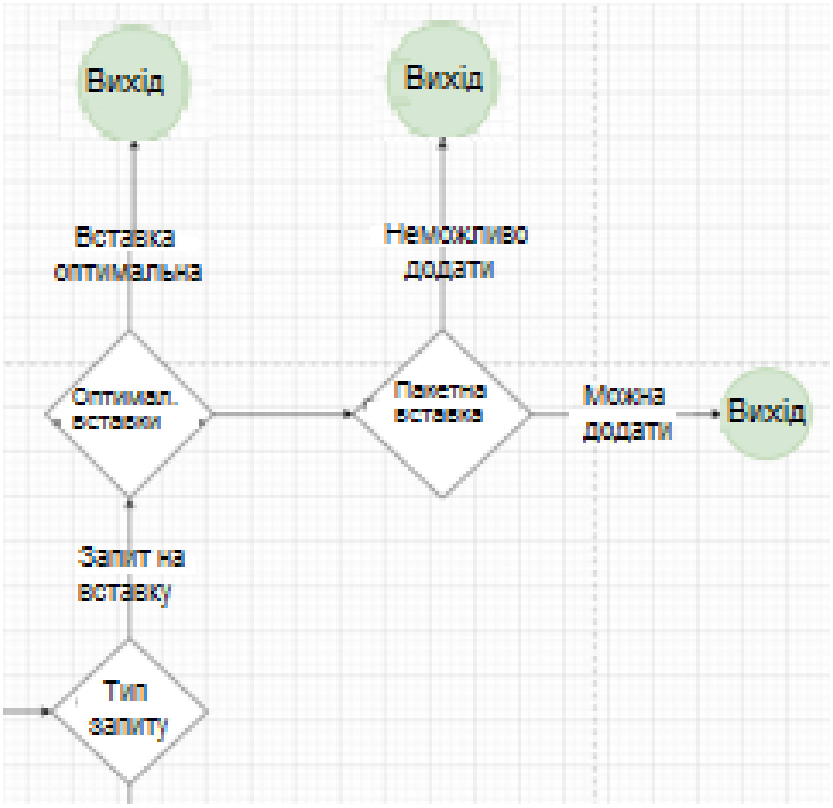

Рисунок 3.16 – Елементи ланцюжка, що відповідають за обробку запитів вставки

3.2.5 Реалізація компонента надання налаштувань роботи профілювальника

Профілювальник виконаний у вигляді повноцінного сервісу з використанням фреймворку Spring, в результаті можна налаштувати обсяг пам'яті, що виділяється, параметри під’єднання до реляційних БД, налаштувати таймаути, також профілювальник підтримує налаштування для лімітів, перевищення яких спричинить за собою сповіщення користувача про необхідність оптимізувати роботу сховища. Приклад налаштування профілювальника наведено у лістингу 3.1. Деяка частина параметрів виймається з проставлених користувача налаштувань у БД профілювальника.

Приклад конфігурації даних для під’єднання користувача БД, а також ліміт, що налаштовується для таблиць за займаною пам'яттю також показаний у лістингу 3.1.

Лістинг 3.1 – Приклад налаштування профілювальника та конфігурації даних

```
spring.profiles.active=@spring.profiles.active@
spring.application.name=database-profiler
server.port=${DATABASE_PROFILER_PORT:8061}
spring.jpa.open-in-view=false

log.directory=${PG_LOG_DIR:/pg_log/}
log.reader.interval=${LOG_UPDATING_RATE:360000}

spring.config.activate.on-profile=dev
spring.profiles.group=pg

# PostgreSQL
spring.datasource.url=${PDB_CONNECTION_STRING:jdbc:postgresq
l://localhost:5432/postgres}
spring.datasource.username=${PDB_USERNAME:spring}
spring.datasource.password=${PDB_PASSWORD:admin}

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=false

spring.flyway.enabled=false
spring.flyway.locations=classpath:db/migration/postgres
spring.flyway.baseline-on-migrate=false
```

Таким чином, профілювальник надає зручний інтерфейс для налаштування необхідних параметрів.

3.3 Тестування функціональності

Обидва створені сервіси були протестовані локально. Рисунок 3.17 демонструє записи журналів запущеного сервісу, що інформують про успішну роботу програми.

Рисунок 3.17 – Записи журналів про успішний запуск програми

Створений сервіс для імітацій трафіку до БД відразу при запуску застосовує скрипти міграцій.

Також послуги були розгорнуті хмарно за допомогою засобів GitHub [10]. Це відображає рисунок 3.18, на ньому показані логи програми, розгорнутої на віддаленій машині, продемонстрований етап із монтуванням.

Також там проводяться й інші дії щодо забезпечення подальшої інтеграції та доставки, такі як запуск тестів та зв'язок з віддаленою БД.

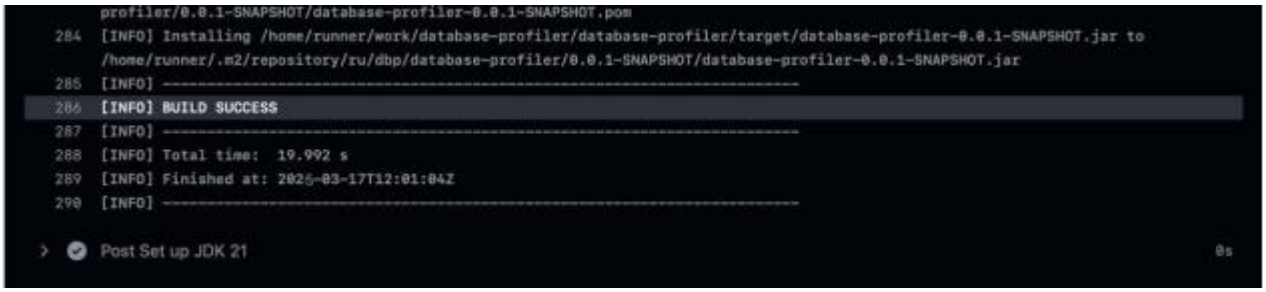

Рисунок 3.18 – Записи журналів для розгорнутої у хмарі програми

На рисунку 3.19 показані налаштовані етапи для розгортання тестового стенду програми на віддаленій машині разом з БД, всередині контейнера Docker, а також конфігурація налаштувань Java.

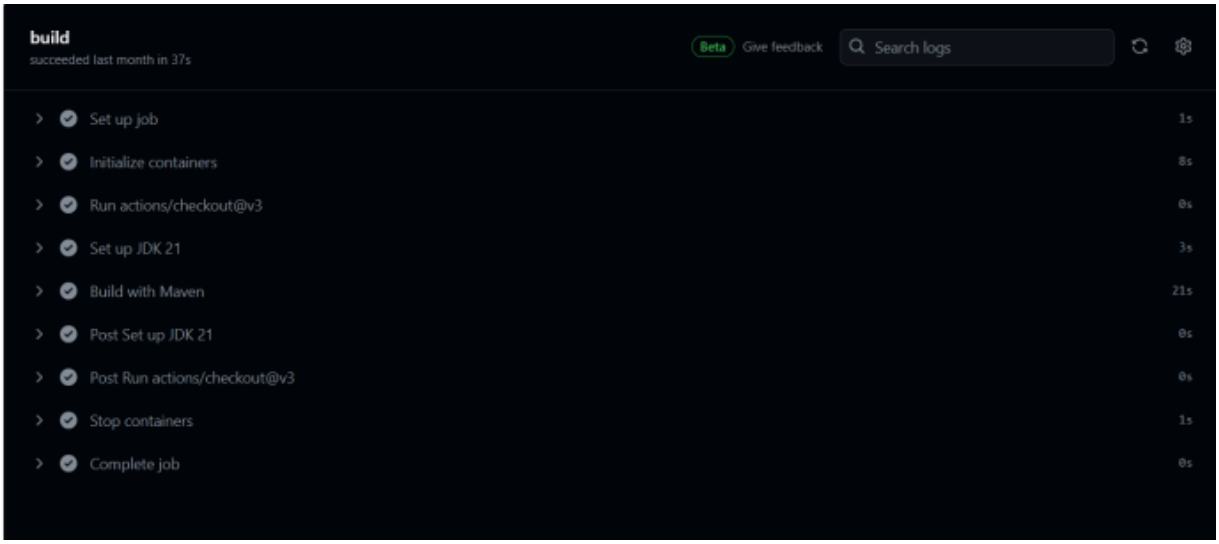

Рисунок 3.19 – Інформація про успішні процеси доставки програми на віддалений сервер

3.4 Аналіз отриманих результатів

Розроблений допоміжний сервіс пропонує зручний спосіб виконання запитів до БД, заповнення таблиць даними та встановлення потрібного трафіку запитів. Можливість створення різних сутностей, а також заповнення їх випадковими даними корисна для створення різних тестових сценаріїв.

Аналітична інформація, що надається профілювальником, дозволяє виявити вузькі місця та покращити продуктивність БД. Наприклад, аналіз найповільніших запитів може допомогти ідентифікувати запити, що потребують оптимізації, а інформація про тимчасові метрики дозволяє оцінити ефективність запитів та їх вплив на загальну продуктивність системи, більш того, для всіх запитів користувача можна отримати оптимізаційні поради, що також дозволить прискорити роботу БД.

Для наступного запиту оптимізатор виявив застосування повного сканування з огляду на відсутність індексу на атрибуті «title»: «select s1_0.id,s1_0.band_id,s1_0.duration_in_seconds,s1_0.lyrics,s1_0.title from songs s1_0 where s1_0.title=$1»: «Для даного запиту застосовується повне сканування без індексів, розгляньте варіант додавання індексу для атрибутів: s1_0.title». Було виміряно час виконання неоптимального запиту, як показано на рисунку 3.20.

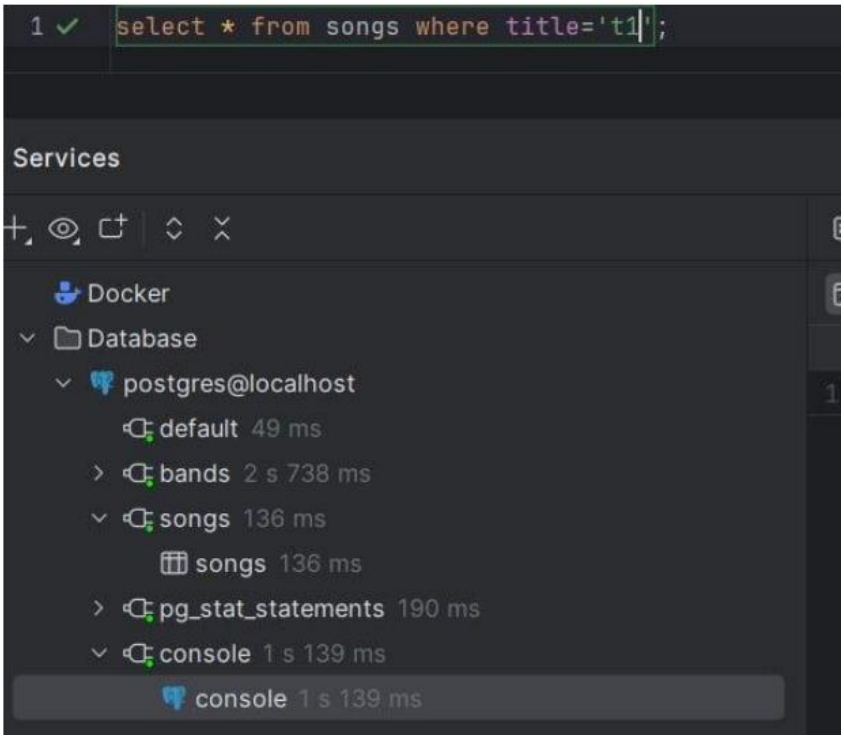

Рисунок 3.20 – Результат виконання неоптимального запиту

А потім виміряно час виконання після додавання індексу, як показує рисунок 3.21.

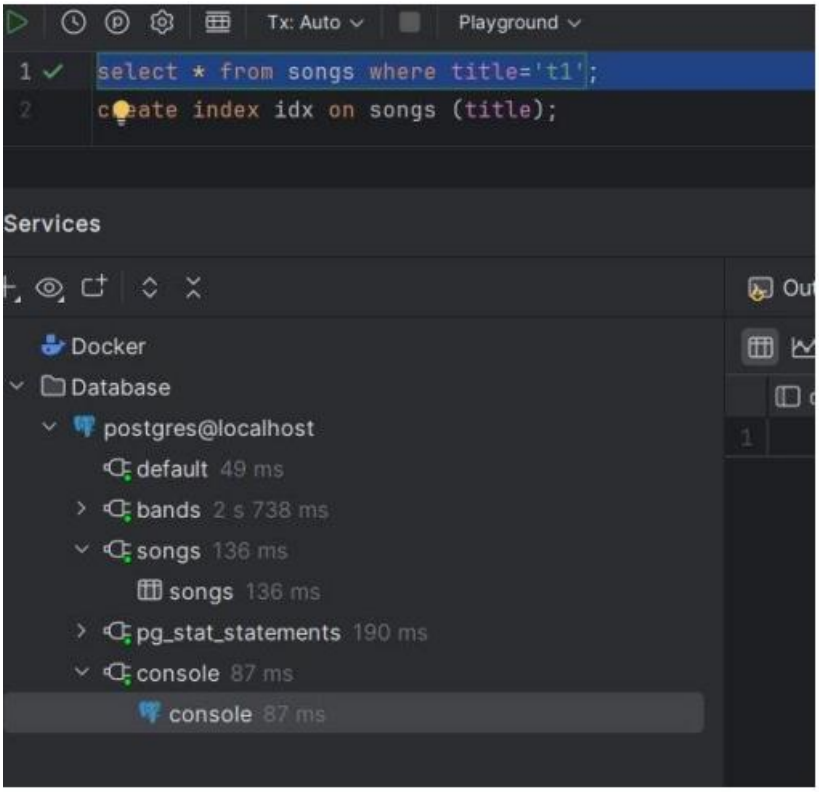

Рисунок 3.21 – Результат виконання оптимізованого запиту

Тут варто відзначити, що по-справжньому величезну різницю помітити не вдасться через технічні обмеження. Справа в тому, що для відчутної різниці у часі виконання запитів потрібно, щоб у БД були мільярди записів, тільки в такому разі буде помітна повільна робота повного сканування, і на противагу їй - швидка індексна робота. Але навіть так різниця у 12 разів є суттєвою оптимізацією.

Загалом, навіть у межах обмежених технічних можливостей можна помітити оптимізацію роботи БД при дотриманні порад профілювальника.

Таким чином, допоміжний сервіс готує хороший ґрунт для подальшого тестування, наповнюючи статистику запитами різного штибу, по багатьох з яких профілювальник пропонує оптимізаційні поради, застосування яких веде до виграшу у швидкості роботи БД.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Класифікація шкідливих та небезпечних виробничих факторів

Шкідливий виробничий фактор – небажане явище, яке супроводжує виробничий процес і вплив якого на працюючого може призвести до погіршення самопочуття, зниження працездатності, захворювання, виробничо зумовленого чи професійного, і навіть смерті, як результату захворювання. Небезпечний виробничий фактор – небажане явище, яке супроводжує виробничий процес і дія якого за певних умов може призвести до травми або іншого раптового погіршення здоров'я працівника (гострого отруєння, гострого захворювання) і навіть до раптової смерті [16].

Поділ несприятливих чинників виробничого середовища на шкідливі та небезпечні зумовлене різним характером їх дії на людський організм, тим, що вони потребують різних заходів та засобів для боротьби з ними та профілактики викликаних ними ушкоджень, а також рядом причин організаційного характеру. В той же час між шкідливими та небезпечними виробничими факторами інколи важко провести чітку межу. Один і той же чинник може викликати травму і профзахворювання (наприклад, високий рівень іонізуючого або теплового випромінювання може викликати опік або навіть призвести до миттєвої смерті, а довготривала дія порівняно невисокого рівня цих же факторів – до хвороби; пилинка, що потрапила в око, спричиняє травму, а пил, що осідає в легенях, – захворювання, що зветься пневмоконіоз). Через це всі несприятливі виробничі чинники часто розглядаються як єдине поняття – небезпечний та шкідливий виробничий фактор (НШВФ) [16]. За своїм походженням та природою дії всі НШВФ можна поділити на 5 груп: фізичні, хімічні, біологічні, психофізіологічні та соціальні До фізичних НШВФ відносяться машини та механізми або їх елементи, а також вироби, матеріали, заготовки тощо, які рухаються або обертаються; конструкції, які руйнуються; системи, устаткування або елементи обладнання, які знаходяться під підвищеним тиском; підвищена запиленість та

загазованість повітря; підвищена або понижена температура повітря, поверхонь приміщення, обладнання, матеріалів; підвищені рівні шуму, вібрації, ультразвуку, інфразвуку; підвищений або понижений барометричний тиск та його різкі коливання; підвищена та понижена вологість; підвищена швидкість руху та підвищена іонізація повітря; підвищений рівень іонізуючих випромінювань; підвищене значення напруги в електричній мережі; підвищені рівні статичної електрики, електромагнітних випромінювань; підвищена напруженість електричного, магнітного полів; відсутність або нестача світла; недостатня освітленість робочої зони; підвищена яскравість світла; понижена контрастність; прямий та віддзеркалений блиск; підвищена пульсація світлового потоку; підвищені рівні ультрафіолетової та інфрачервоної радіації; гострі крайки, зачипки, шершавість на поверхні заготовок, інструментів та обладнання; розташування робочого місця на значній висоті відносно землі (підлоги); слизька підлога; невагомість.

Хімічні НШВФ:

- за характером дії на організм людини поділяються на токсичні, задушливі, наркотичні, подразнюючі, сенсибілізуючі, канцерогенні, мутагенні та такі, що впливають на репродуктивну функцію;

- за шляхами проникнення в організм людини поділяються на такі, що потрапляють через: 1) органи дихання; 2) шлунково-кишковий тракт; 3) шкіряні покриви та слизова оболонка;

- які перебувають у різному агрегатному стані: 1)твердому 2)газоподібному 3)рідкому.

Біологічні НШВФ – це: - патогенні мікроорганізми (бактерії, віруси, рикетсії, спірохети, грибки, найпростіші) та продукти їхньої життєдіяльності; - макроорганізми (тварини та рослини) та продукти їхньої життєдіяльності. До психофізіологічних НШВФ відносяться фізичні (статичні та динамічні) перевантаження і нервово-психічні перевантаження (розумове перенапруження, перенапруження аналізаторів, монотонність праці, емоційні перевантаження). 6 Соціальні НШВФ – це неякісна організація роботи, понаднормова робота, змушеність праці в колективі з поганими відносинами між його членами,

соціальна ізольованість з відривом від сім_ї, зміна біоритмів, незадоволеність роботою, фізична та/або словесна образа та її ризик, насильство та його ризик. Один і той же НШВФ за природою своєї дії може належати водночас до різних груп.

4.2 Вплив вібрації на людину

Вібрація - це механічні коливання пружних тіл або коливальні рухи механічних систем. Для людини вібрація є видом механічного впливу, який має негативні наслідки для організму [17].

Причиною появи вібрації є неврівноважені сили та ударні процеси в діючих механізмах. Створення високопродуктивних потужних машин і швидкісних транспортних засобів при одночасному зниженні їх матеріалоємності неминуче призводить до збільшення інтенсивності і розширення спектру вібраційних та віброакустичних полів. Цьому сприяє також широке використання в промисловості і будівництві високоефективних механізмів вібраційної та віброударної дії.

Дія вібрації може приводити до трансформування внутрішньої структури і поверхневих шарів матеріалів, зміни умов тертя і зносу на контактних поверхнях деталей машин, нагрівання конструкцій. Через вібрацію збільшуються динамічні навантаження в елементах конструкцій, стиках і сполученнях, знижується несуча здатність деталей, ініціюються тріщини, виникає руйнування обладнання. Усе це приводить до зниження строку служби устаткування, зростання імовірності аварійних ситуацій і зростання економічних витрат. Вважають, що 80% аварій в машинах і механізмах здійснюється внаслідок вібрації. Крім того, коливання конструкцій часто є джерелом небажаного шуму. Захист від вібрації є складною і багатоплановою в науково-технічному та важливою у соціальноекономічному відношеннях проблемою нашого суспільства [17].

Вплив вібрації на людину залежить від її спектрального складу, напрямку дії, прикладення, тривалості впливу, а також від індивідуальних особливостей людини. При оцінці вібраційного впливу потрібно враховувати, що коливальні

процеси притаманні живому організму. В основі серцевої діяльності і кровообігу та біострумів мозку лежать ритмічні коливання. Внутрішні органи людини можна розглядати як коливальні системи з пружними зв'язками. Частоти їх власних коливань лежать у діапазоні 3..6 Гц. Частоти власних коливань плечового пояса, стегон і голови щодо опорної поверхні (положення стоячи) складають 4...6 Гц, голови щодо плечей (положення сидячи) 25...30 Гц.

При впливі на людину зовнішніх коливань (хитавиці, струсів, вібрації) відбувається їхня взаємодія з внутрішніми хвильовими процесами, виникнення резонансних явищ. Так, зовнішні коливання частотою менш 0,7 Гц утворюють хитавицю і порушують у людини нормальну діяльність вестибулярного апарата. Інфразвукові коливання (менш 16 Гц), впливаючи на людину, пригнічують центральну нервову систему, викликаючи почуття тривоги, страху. При певній інтенсивності на частоті 6..7 Гц інфразвукові коливання, втягуючи у резонанс внутрішні органи і систему кровообігу, здатні викликати травми, розриви артерій, тощо [17].

Вібрація, що діє на людину, має широкий діапазон – від десятих часток одного до декількох тисяч Гц. Характерними ознаками шкідливого впливу вібрації на людину є можливі зміни у функціональному стані: підвищена втома, збільшення часу моторної реакції, порушення вестибулярної реакції. Медичними дослідженнями встановлено, що вібрація є подразником периферичних нервових закінчень, розташованих на ділянках тіла людини, що сприймають зовнішні коливання. Адекватним фізичним критерієм оцінки її впливу на організм людини є коливальна енергія, що виникає на поверхні контакту, а також енергія, поглинена тканинами і передана опорно-руховому апарату та іншим органам. У результаті впливу вібрації виникають нервовосудинні розлади, ураження кістково-суглобної та інших систем організму. Відзначаються, наприклад, зміни функції щитовидної залози, сечостатевої системи, шлунково-кишкового тракту. Так, медичні дослідження показали , що у працюючих в умовах вібрації відбуваються значні зміни кістковосуглобної системи, які виражаються у функціональній перебудові кісткової тканини, регіональному остеопорозі, кістовидних утвореннях у кістках, асептичному некрозі кісток, хронічних

переломах. Відзначається, що терміни виникнення змін у кістках у працівників вібраційних професій коливається в межах від 6-8 місяців до 2-5 років.

Шкідливість вібрації збільшується при одночасному впливі на людину таких факторів, як знижена температура, підвищений шум, запиленість повітря, тривала статична напруга тощо. Сучасна медицина розглядає виробничу вібрацію як могутній стрес-фактор, що має негативний вплив на психомоторну працездатність, емоційну сферу і розумову діяльність, підвищує ймовірність виникнення різних захворювань і нещасних випадків. Особливо небезпечний тривалий вплив вібрації для жіночого організму. Широкий комплекс патологічних відхилень, викликаний впливом вібрації на організм людини, кваліфікується як віброзахворювання [17].

Вібрація як фізичний чинник виробничого середовища спостерігається в металообробній, гірничодобувній, металобудівній, машинобудівній, авіаційній та інших галузях народного господарства. Джерелом вібрації можуть бути різні механізми, вібраційне устаткування, віброінструменти, акустичні системи, транспортні та сільськогосподарські машини.

Загальна вібрація поділяється на транспортну вібрацію, яка діє на людину на робочих місцях в транспортних засобах (трактори сільськогосподарські та промислові, самохідні сільськогосподарські машини (комбайни), тягачі, грейдери ті інші); транспортно-технологічну вібрацію, яка діє на людину на робочих місцях машин з обмеженою рухливістю (екскаватори, крани промислові та будівельні, гірничі комбайни, транспорт виробничих приміщень та інші) та технологічну вібрацію, яка діє на людину на робочих місцях стаціонарних машин чи передається на робочі, де немає джерел вібрації (верстати та метало-деревообробне, пресувально-ковальське обладнання, ливарні машини, електричні машини, насосні агрегати та вентилятори, обладнання для буріння свердловин, бурові верстати, машини для тваринництва, очищення та сортування зерна (у тому числі сушарні), обладнання промисловості будматеріалів (крім бетоноукладачів), установки хімічної та нафтохімічної промисловості та інші.

Оператори машин, які зазнають у процесі трудової діяльності впливу вібрації, підлягають попереднім та періодичним медичним оглядам відповідно

до Порядку проведення медичних оглядів працівників певних категорій, затвердженого Наказом МОЗ України від 21.05.2007 р. №246. Обов'язкові попередні (під час прийняття на роботу) та періодичні (протягом трудової діяльності) медичні огляди дозволять визначити стан здоров'я працівника та можливість виконання без погіршення стану здоров'я професійних обов'язків, своєчасно виявити ранні ознаки хронічного професійного захворювання, забезпечує динамічне спостереження за станом здоров`я в умовах дії шкідливих та небезпечних факторів і трудового процесу, вирішує питання щодо можливості продовжувати роботу в умовах дії шкідливих та небезпечних факторів і трудового процесу [17].

За результатами періодичних медичних оглядів роботодавець забезпечує проведення відповідних оздоровчих заходів Заключного акта у повному обсязі та усуває причини, що призводять до професійних захворювань. Організовує проведення лабораторних досліджень умов праці на робочих місцях та вживає заходів до усунення небезпечних і шкідливих для здоров'я виробничих факторів.

До роботи операторами машин допускаються особи не молодші 18 років, які пройшли попередній медичний огляд, мають відповідну кваліфікацію та ознайомлені з характером впливу вібрації на організм.

ВИСНОВКИ

У процесі роботи було реалізовано сервіс профілювальника БД, котрий володіє наступним функціоналом:

– робота з реляційними сховищами: PostgreSql, Oracle, MysSql;

– надання аудиту сховища;

– надання даних про швидкість виконання запитів;

– надання даних про запити користувача;

– надання даних за точковими запитами;

– надання можливості ранжування запитів за тимчасовими даними;

– надання можливості налаштовувати профілювальник під сценарії користувача використання;

– надання можливості виявляти неефективний синтаксис запитів sql ;

– надання можливості виявляти неефективні транзакції;

– надання можливості виявляти неефективне зчитування даних;

– надання перевірок можливості додавання індексів;

– надання можливості виявляти неефективні вставки даних;

– надання можливості виявляти неефективне розподіл даних у таблицях;

– надання перевірок на можливість секціонування БД.

У процесі роботи було виконано наступне:

– проведено аналіз ринку та існуючих аналогів додатка для виявлення їх основних функцій, особливостей та переваг;

– складено список функціональних вимог на основі зібраних даних та аналізу існуючих аналогів;

– визначено основні можливості та особливості, які мають бути включені до застосунку;

– спроектована та розроблена на основі зібраних вимог архітектура застосунку, обрані технології та інструменти для його реалізації;

– складено схему для реляційної БД профілювальника;

– розроблений застосунок був успішно розгорнутий на хмарному

сервері з урахуванням вимог до масштабованості та безпеки;

– застосунок був підданий ретельному тестуванню, під час якого виявлено та виправлено можливі помилки та недоліки.

Незважаючи на те, що створений застосунок вже володіє широким функціоналом і успішно пройшов процес розробки та тестування, можливості його покращення та розширення залишаються значними.

Надалі для розвитку проекту можна додати підтримку інших реляційних сховищ. Також можна буде реалізувати зручний інтерфейс користувача і розробити можливість інтеграції з різними системами відтворення та надання візуальних даних.

ПЕРЕЛІК ДЖЕРЕЛ

1. Spring [Електронний ресурс] – Режим доступу: https://spring.io/projects/spring-framework (дата звернення 11.05.2026).

2. Volume of data or information created, captured, copied, and consumed worldwide from 2010 to 2029 [Електронний ресурс]. - Режим доступу: https://www.statista.com/statistics/871513/worldwide-data-created/ (дата звернення 14.05.2026).

3. Hibernate [Електронний ресурс] – Режим доступу: https://hibernate.org (Дата звернення 16.05.2026).

4. Hibernate Profiler [Електронний ресурс]. – Режим доступу: https://hibernatingrhinos.com/ (дата звернення 16.05.2026).

5. Dynatrace [Електронний ресурс]. – Режим доступу: https://www.dynatrace.com/ (дата звернення 16.05.2026).

6. PostgreSQL [Електронний ресурс]. – Режим доступу: https://www.postgresql.org (Дата звернення 18.05.2026).

7. Java [Електронний ресурс]. – Режим доступу: https://www.java.com/ru/ (Дата звернення 18.05.2026).

8. Oracle [Електронний ресурс]. – Режим доступу: https://www.oracle.com/cis/database/ (дата звернення 19.05.2026).

9. MySql [Електронний ресурс]. – Режим доступу: https://www.mysql.com/ (Дата звернення 24.05.2026).

10. GitHub CI/CD [Електронний ресурс]. – Режим доступу: https://github.com/solutions/ci-cd (дата звернення 24.05.2026).

11. Методичні вказівки до виконання кваліфікаційної роботи ОР Бакалавр для студентів спеціальності 122 – Комп’ютерні науки, всіх форм навчання / укладачі: Готович В.А., Дуда О.М. Никитюк В.В. Тернопіль: ТНТУ імені Івана Пулюя, 2024. 43 с.

12. Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021). Mathematical model of gas consumption process in the form of

cyclic random process. 2021 IEEE 16th International Conference on Computer Sciences and Information.

13. Lytvynenko I. V. Method of segmentation of determined cyclic signals for the problems related to their processing and modeling. Scientific journal of the Ternopil National Technical University. No. 4 (88). 2017. ISSN: 2522-4433. P. 153–169. https://doi.org/10.33108/visnyk_tntu2017.04.153

14. Lupenko, S. A., Lytvynenko, I. V., Sverstiuk, A., Shelestovskyi, B., & Horkunenko, A. (2021). Software for Statistical Processing and Modeling of a Set of Synchronously Registered Cardio Signals of Different Physical Nature. CMIS, 194-205.

15. Bodnarchuk, I., Skorenkyy, Y., Kramar, T., Duda, O., & Nykytyuk, V. (2022). Use of Analytical Hierarchy Process in Scenarios Design for a Digital Museum with XR components. ITTAP, 414–425

16. Заікіна Д., Глива В. Основи охорони праці та безпека життєдіяльності. 2019. [Електронний ресурс]. – Режим доступу: https://doi.org/10.31435/rsglobal/001 (дата звернення: 27.05.2025).

17. Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання / укл.: Стручок В. С. Тернопіль: ФОП Паляниця В. А., 2022. 156 с.

ДОДАТКИ

Додаток А

Фрагмент коду з логікою проставлення параметрів запиту для подальшої побудови плану запиту

```
private String linkParamsToQueryIfRequired(String query) {
    if (!query.contains(«$1»))
        return query;

    StringBuilder queryWithParams = new StringBuilder(query);
    HashMap<String,
HiddenValueAndTableNamePair>columnHiddenValueAndTableNameMap =
new HashMap<>();
    HashMap<String, String>hiddenValueRealValueMap = new
HashMap<>();

fillColumnHiddenValueMapWithHiddenValues(query,
columnHiddenValueAndTableNameMap);
fillColumnHiddenValueAndTableNameMapWithTableNames(query,
columnHiddenValueAndTableNameMap);

    for (Map.Entry<String, HiddenValueAndTableNamePair> e :
columnHiddenValueAndTableNameMap.entrySet()) {
        String key = e.getKey();
        String hiddenValue = e.getValue().getPair()[0];
        String tableName = e.getValue().getPair()[1];
        String columnName =
StringHelper.getSubstringAfterFirstDot(key);
        String paramType = getColumnType(tableName,
columnName).orElseThrow(NoSuchElementException::new);

hiddenValueRealValueMap.put(hiddenValue, paramType);
    }

    int paramIndex = 1;
    int shiftIndex=0;
    for (Map.Entry<String, String> e :
hiddenValueRealValueMap.entrySet()) {
        String hiddenValue = «$» + paramIndex;
        int hiddenValueIndex = query.indexOf(hiddenValue)-
shiftIndex;
        String paramToReplace = switch (e.getValue()) {
            case «bigint», «integer» -> «1»;
            case «varchar» -> «''«;
            default -> throw new
IllegalArgumentException(«Unsupported data type»);
        };
queryWithParams.replace(hiddenValueIndex, hiddenValueIndex +
hiddenValue.length(), paramToReplace);
paramIndex++;
shiftIndex+=hiddenValue.length()-1;
    }
```