КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації інформаційних технологій

Кафедра інформаційних технологій

(назва випускової кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

на тему:

ОПТИМІЗАЦІЯ БАЗ ДАНИХ ТА ХМАРНИХ СХОВИЩ ДЛЯ ЕФЕКТИВНОЇ РОБОТИ З ВЕЛИКИМИ ОБСЯГАМИ ІНФОРМАЦІЇ

МАРТИНЮК Надія Олександрівна

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій

(кафедра)

ЗАТВЕРДЖУЮ
Завідувачка кафедри ІТ
д.т.н., доцент Гончаренко Т.А.

„___” _____2025 року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: «**ОПТИМІЗАЦІЯ БАЗ ДАНИХ ТА ХМАРНИХ СХОВИЩ ДЛЯ ЕФЕКТИВНОЇ РОБОТИ З ВЕЛИКИМИ ОБСЯГАМИ ІНФОРМАЦІЇ**»

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач
МАРТИНЮК Надія Олександрівна
122 «Комп’ютерні науки»
(спеціальність)
Інформаційні управляючі системи і технології
(освітня програма)
Групи КН-21
Керівник Рябчун Ю.В.
(прізвище та ініціали)
Доктор філософії
(вчене звання, науковий ступінь)
Рецензент к.т.н., доц. Шабала Є.Є.
(Прізвище та ініціали)

Ідентичність підтверджую

Київ, 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет:	Автоматизації інформаційних технологій
Випускова кафедра:	Інформаційних технологій
Освітній ступінь:	Бакалавр
Спеціальність:	Комп’ютерні науки
Освітня програма:	Інформаційні управляючі системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТ

Тетяна ГОНЧАРЕНКО

_____

„___”_____2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

	Мартинюк Надія Олександрівна

1. Тема роботи ОПТИМІЗАЦІЯ БАЗ ДАНИХ ТА ХМАРНИХ СХОВИЩ ДЛЯ ЕФЕКТИВНОЇ РОБОТИ З ВЕЛИКИМИ ОБСЯГАМИ ІНФОРМАЦІЇ

затверджена наказом ректора КНУБА № 235/23/25 від «14» лютого 2024 року

2. Керівник роботи	Рябчун Юлія Володимирівна, PhD

3. Строк подання Здобувачем роботи до захисту_____

4. Зміст пояснювальної записки за розділами:

Р.1	АНАЛІЗ МЕТОДІВ ТА ТЕХНОЛОГІЙ ОПТИМІЗАЦІЇ БАЗ ДАНИХ ТА ХМАРНИХ СХОВИЩ
Р.2	ПРОЄКТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
Р.3	РЕЗУЛЬТАТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
Р.4	ТЕХНІКО-ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ РОЗРОБКИ ПІДСИСТЕМИ

5. Графічний матеріал за розділами:

Р.1 7 рисунків, 7 таблиць

Р.2 10 рисунків, 11 таблиць

Р.3 32 рисунка

Р.4 -

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	15/02/2025
Розділ 2	20/03/2025
Розділ 3	17/04/2025
Розділ 4	15/05/2025
Остаточне оформлення роботи	25/05/2025
Направлення роботи для перевірки на плагіат	25/05/2025
Попередній захист роботи на випусковій кафедрі	26/05/2025
Направлення роботи на рецензування	26/05/2025

7. Консультанти розділів випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1	Рябчун Ю.В., доц.каф.ІТ	15/02/2025	
Розділ 2	Рябчун Ю.В., доц.каф.ІТ	20/03/2025	
Розділ 3	Рябчун Ю.В., доц.каф.ІТ	17/04/2025	
Розділ 4	Рябчун Ю.В., доц.каф.ІТ	15/05/2025	

8. Дата видачі завдання листопад 2024 р.

Зав. кафедри			Гончаренко Т.А.
	(підпис)		(прізвище та ініціали)
Керівники			Рябчун Ю.В.
	(підпис)		(прізвище та ініціали)
Здобувач			Мартинюк Н.О.
	(підпис)		(прізвище та ініціали)

АНОТАЦІЯ

Мартинюк Н.О. Оптимізація баз даних та хмарних сховищ для ефективної роботи з великими обсягами інформації.

Атестаційна випускна робота бакалавра за спеціальністю 122 «Комп'ютерні науки», освітня програма «Інформаційні управляючі системи та технології». – Київський національний університет будівництва та архітектури. – Київ, 2025.

Дана дипломна робота присвячена розробці методів та інструментів для оптимізації баз даних і хмарних сховищ з метою підвищення ефективності роботи з великими обсягами інформації. У роботі розглянуто сучасні підходи до оптимізації реляційних баз даних (на прикладі PostgreSQL), включаючи індексацію, партиціювання та кешування, а також інтеграцію з хмарними технологіями, зокрема Google Cloud Storage (GCS). Практична частина включає реалізацію функцій для логування продуктивності SQL-запитів і експорту даних у GCS, а також експериментальну оцінку продуктивності розроблених рішень. На основі отриманих результатів сформульовано рекомендації щодо подальшого вдосконалення оптимізації та використання хмарних сховищ.

Робота викладена на 107 аркушах, містить 4 додатки, 18 таблиць, 48 рисунків, список використаної літератури із 25 найменувань.

Ключові слова: оптимізація баз даних, хмарні сховища, Big Data, PostgreSQL, Google Cloud Storage, індексація, партиціювання, кешування.

SUMMARY

Martynyuk N.O. Optimization of Databases and Cloud Storage for Efficient Handling of Large Volumes of Information.

Bachelor's thesis in the specialty: 122 "Computer Science," specialization: "Information Management Systems and Technologies." – Kyiv National University of Construction and Architecture. – Kyiv, 2025.

This thesis is dedicated to the development of methods and tools for optimizing databases and cloud storage to enhance the efficiency of handling large volumes of information. The study explores modern approaches to optimizing relational databases (using PostgreSQL as an example), including indexing, partitioning, and caching, as well as integration with cloud technologies, particularly Google Cloud Storage (GCS). The practical part involves the implementation of functions for logging SQL query performance and exporting data to GCS, along with an experimental evaluation of the developed solutions' performance. Based on the obtained results, recommendations are formulated for further improvement of optimization techniques and the use of cloud storage.

The work spans 107 pages, includes 4 appendices, 18 tables, 48 figures, and a bibliography with 25 references.

Keywords: database optimization, cloud storage, Big Data, PostgreSQL, Google Cloud Storage, indexing, partitioning, caching.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACID — принципи атомарності, консистентності, ізольованості та довговічності, що забезпечують надійність транзакцій у базах даних.

Big Data — великі обсяги даних, що потребують спеціальних методів обробки та аналізу.

CSV — формат файлів із комами як роздільниками для зберігання табличних даних.

GCS — Google Cloud Storage, хмарне сховище даних у Google Cloud Platform.

JSON — формат обміну даними, що використовується для зберігання складних структур.

NoSQL — нереляційні системи управління базами даних для гнучкого зберігання даних.

Parquet — стовпцевий формат зберігання даних, оптимізований для аналітики.

PostgreSQL — реляційна система управління базами даних, використана для оптимізації.

Python — мова програмування, застосована для розробки платформи.

SQL — мова структурованих запитів для роботи з реляційними базами даних.

Streamlit — фреймворк для створення інтерактивних веб-інтерфейсів.

ВСТУП

У сучасному світі, з розвитком інформаційних технологій і стрімким зростанням обсягів даних, питання ефективного управління базами даних і хмарними сховищами стає все більш актуальним. Багато організацій і компаній стикаються з проблемами перевантаження інформаційних систем, що виникають через великі обсяги даних і високу інтенсивність запитів. До того ж, традиційні методи обробки даних часто не справляються з сучасними вимогами: час виконання запитів зростає, системи стають менш стабільними при високих навантаженнях, а затримки мережі ускладнюють швидкий доступ до даних. Основні способи вирішення проблеми — збільшення обчислювальних ресурсів або використання складних систем масштабування. Однак ці рішення мають суттєві недоліки: вони вимагають значних фінансових вкладень, підвищують витрати на інфраструктуру, довго окупаються, а також можуть не забезпечити стабільної продуктивності при зростанні обсягу даних .

Однак існує рішення, яке дозволяє уникнути цих недоліків — комплексна оптимізація баз даних і хмарних сховищ із використанням сучасних методів і технологій. Оптимізація за допомогою індексації, кешування через Memorystore і горизонтального масштабування в Google Cloud дає змогу значно зменшити час виконання запитів, підвищити стабільність системи та знизити витрати на інфраструктуру. Переваги такого підходу очевидні: зменшення на 60–80%, зниження витрат на обчислювальні ресурси до 30%, а також забезпечення масштабованості при обробці великих обсягів даних [24]. До того ж, інтеграція з інструментами візуалізації, такими як Dash, дозволяє аналізувати продуктивність у реальному часі, що підвищує ефективність управління системою.

Сучасні технології значно спрощують і покращують процес оптимізації баз даних і хмарних сховищ. Хмарні платформи, такі як Google Cloud, надають інструменти для інтеграції баз даних PostgreSQL із хмарними сховищами Google Cloud Storage (GCS), а також дозволяють використовувати керовані сервіси, такі як Memorystore, для кешування даних. Програмні рішення, побудовані на Python, забезпечують автоматизацію логування метрик продуктивності, їх експорт у GCS і

візуалізацію через Dash. Такі рішення можуть включати функції аналізу затримок мережі, системи логування для фіксації метрик і зручні інтерфейси для моніторингу продуктивності.

Дана дипломна робота присвячена розробці прикладного рішення для оптимізації баз даних PostgreSQL і хмарних сховищ GCS на платформі Google Cloud. Прикладне рішення надасть користувачам можливість автоматично фіксувати метрики продуктивності запитів, експортувати їх у GCS для довгострокового зберігання та візуалізувати через Dash для аналізу в реальному часі. Воно забезпечить інтеграцію з інструментами Google Cloud для оптимізації запитів, систему логування для підвищення прозорості роботи системи, а також зручний інтерфейс для моніторингу продуктивності.

Метою даної роботи є розробка та впровадження прикладного рішення, яке сприятиме зниженню часу виконання запитів до бази даних, підвищенню продуктивності хмарних сховищ і забезпеченню масштабованості при обробці великих обсягів даних на платформі Google Cloud. У роботі буде детально розглянуто процес проектування рішення, його функціональні можливості, технічні аспекти реалізації, а також проведено аналіз очікуваних результатів від впровадження.

Завдання дослідження:

1. Провести аналіз існуючих методів оптимізації баз даних і хмарних сховищ та вивчити їхні переваги й недоліки.
2. Формалізувати об’єкт дослідження через параметри і розробити математичну модель продуктивності.
3. Обґрунтувати вибір методів і технологій для реалізації рішення.
4. Розробити інформаційне забезпечення, включаючи модель даних для логування метрик і схему потоків даних між компонентами системи.
5. Синтезувати алгоритми для логування метрик, експорту даних у GCS і агрегації метрик для візуалізації в Dash.

6. Реалізувати прикладне рішення, протестувати його в реальних умовах і проаналізувати вплив на продуктивність системи та економічну ефективність.

Об’єкт дослідження – процеси оптимізації баз даних і хмарних сховищ у контексті обробки запитів із великими обсягами даних на платформі Google Cloud.

Предмет дослідження – прикладне рішення для оптимізації продуктивності запитів до бази даних PostgreSQL і хмарного сховища GCS, а також його вплив на час виконання запитів, масштабованість і економічну ефективність.

Розділ 1. АНАЛІЗ МЕТОДІВ ТА ТЕХНОЛОГІЙ ОПТИМІЗАЦІЇ БАЗ ДАНИХ ТА ХМАРНИХ СХОВИЩ

У сучасному світі, де інформація стала основним ресурсом для прийняття рішень, оптимізація баз даних та хмарних сховищ набуває все більшої значуваності. Збільшення обсягів даних, які генеруються щодня, вимагають від організацій ефективних стратегій для їх обробки та зберігання. Це стосується як традиційних систем управління базами даних, так і новітніх рішень у сфері хмарних технологій.

Попри наявність численних методів і технологій, що використовуються для оптимізації баз даних, багато компаній стикаються з проблемами продуктивності та ефективності у своїх системах. Часто існуючі рішення не забезпечують потрібної швидкості доступу до даних або не відповідають вимогам безпеки.

Аналіз методів та технологій оптимізації дозволяє глибше зрозуміти потреби компаній і визначити функціональні та нефункціональні вимоги до систем. Він також допомагає виявити потенційні проблеми ще на початкових етапах проектування, що дозволяє вчасно вжити заходів для їх усунення або мінімізації впливу під час подальшої роботу.

В даному розділі буде проведено аналіз існуючих методів оптимізації баз даних і хмарних сховищ, а також огляд сучасних технологій, що використовуються в цій сфері. Після детального аналізу буде визначено основні проблеми та вимоги до рішень, які необхідно врахувати для розробки нових систем. Всі ці етапи є ключовими у процесі оптимізації роботи з великими обсягами даних і спрямовані на досягнення високої продуктивності та ефективності в управлінні інформацією.

1.1 Аналіз проблематики обробки великих обсягів даних

Big Data – це термін, що став популярним, але ще немає чіткого загального визначення, що він означає [1]. Діякі професійні аналітики вважають, що це процес видобування, трансформації та завантаження (ETL) великих наборів даних. Одним із найпоширеніших визначень є концепція трьох атрибутів (3V), яку вперше

запропонував Дуглас Лейн у своїй книзі [1]. Він зазначив, що через різке зростання активності в електронній комерції дані зросли за трьома напрямками: обсяг (Volume), швидкість (Velocity) та різноманітність (Variety).

З часом до трьох атрибутів додалися нові, і на сьогоднішній день популярною є модель 5V, яку запропонував Юрій Демченко [2]. До звичних трьох атрибутів він додав достовірність (Veracity) та цінність (Value).

Щодня у світі виробляється близько п’яти квінтильйонів біт даних [3]. Це число продовжуватиме зростати, оскільки з’являється все більше систем і гаджетів, підключених до Інтернету. Очікується, що в найближчі роки обсяги даних лише збільшуватимуться.

Зі зростанням виникають проблеми, оскільки компанії стикаються з необхідністю керувати великою кількістю джерел інформації. Програмні рішення для аналітики великих даних створюють нові виклики, пов’язані з обробкою та інтеграцією. Основні проблеми, що виникають у процесі роботи з великими обсягами даних [3], наведені в таблиці 1.1.

Таблиця 1.1

Проблеми при роботі з великими обсягами даних

Проблема	Опис
Високий обяг даних	Значні обсяги даних ускладнюють їх зберігання та обробку
Інтеграція даних	Складнощі в об’єднанні даних з різних джерел
Аріхтектура сховищ даних	Потреба в ефективності архітектурних рішень для зберігання
Неструктуровані дані	Складність в обробці та аналізі неструктурованої інформації
Гарантія якості	Проблеми з точністю та достовірністю даних
Модернізація бізнес аналітики	Виклики в оновленні систем аналітики для роботи з великими даними
Масштабованість та продуктивність	Ускладнення в підтримці продуктивності при зростанні обсягів даних
Слабка візуалізація	Проблеми з представленням даних у зрозумілому вигляді

Застарілі технології часто не здатні впоратися зі швидкістю та масштабами сучасних вимог до обробки інформації. Це призводить до затримок у доступі до даних і ускладнює їх аналіз, що, в свою чергу, негативно впливає на прийняття рішень у бізнес-середовищі. У результаті компанії стикаються з серйозними

викликами, які потребують термінового вирішення для забезпечення ефективності та конкурентоспроможності.

Серед основних викликів, пов’язаних із традиційними системами зберігання даних [4], можна виділити:

1. Витрати на обслуговування та недостаня гнучність;
2. Ненадійність результатів;
3. Проблеми з управлінням даними;
4. Високі витрати та проблеми якості даних;
5. Обмеження в швидкості та машстабованості;
6. Неефективність для нетехнічних користувачів.

Ці виклики підкреслюють критичну необхідність оптимізації баз даних у сучасних умовах. З огляду на зростаючі обсяги даних і вимоги до швидкості обробки, оптимізація баз даних стає важливою для забезпечення ефективності управління інформацією.

Оптимізація баз даних дозволяє організаціям справлятися з величезними обсягами інформації, забезпечуючи швидкий доступ до даних та їх ефективну обробку. Це особливо важливо в умовах, коли швидкість прийняття рішень є критично важливою для конкурентоспроможності компаній. Таким чином, оптимізація баз даних не лише покращує їх продуктивність, але й забезпечує надійність і безпеку даних, що є ключовими факторами для успішного функціонування сучасного бізнесу.

1.2 Дослідження сучасних методів оптимізації баз даних

У сучасному світі, де обсяги даних зростають з неймовірною швидкістю, оптимізація баз даних стає невід'ємною частиною ефективного управління інформацією. Використання різноманітних методів, таких як індексація, партиціювання, кешування, нормалізація та денормалізація, а також оптимізація схеми бази даних, дозволяє значно покращити продуктивність системи. Ці підходи не лише зменшують час виконання запитів, але й забезпечують надійність і цілісність даних [25].

1.2.1 Індексація

Індексація — це метод оптимізації баз даних, який дозволяє швидко знаходити записи, використовуючи спеціальну структуру даних. Індекс складається з ключа пошуку та покажчиків на адреси блоків, де зберігаються відповідні дані [5]. Основними типами індексів є первинний (запис для кожного значення ключа) та вторинний (для полів, які не є первинними ключами). Індексація прискорює пошук і вибірку даних, але може уповільнювати операції оновлення та вставки через необхідність підтримувати актуальність індексів. Правильне використання індексів забезпечує баланс між продуктивністю запитів і витратами на обслуговування [5].

Індексацію використовують в електронній комерції для прискорення запитів.

1.2.2 Партиціювання

Партиціювання бази даних — це метод, який дозволяє покращити продуктивність, доступність та масштабованість системи [6]. Воно передбачає розподіл великої бази даних на менші, легші для обробки частини, що зменшує навантаження на ресурси, такі як процесор і введення/виведення.

Існують різні види партиціювання: горизонтальне (розподіл за рядками), вертикальне (розподіл за стовпцями) та гібридне, яке поєднує обидва підходи. Партиціювання також може допомогти в управлінні даними, спростивши резервне копіювання та технічне обслуговування. Важливо правильно вибрати метод партиціювання в залежності від потреб конкретної програми та характеристик даних [6].

Партиціювання застосовують у великих фінансових організаціях для розподілу даних про транзакції, що дозволяє швидко обробляти дані за певний період часу.

1.2.3 Кешування

Кешування — це процес тимчасового зберігання даних у спеціально виділеній області пам'яті (кеші), що забезпечує швидкий доступ до них у майбутньому [7].

Основна мета кешування полягає в підвищенні продуктивності та ефективності роботи програм і систем за рахунок скорочення часу доступу до даних. Коли програма запитує дані, спочатку перевіряється кеш; якщо дані там є, вони повертаються миттєво, що суттєво прискорює процес. Якщо ж дані відсутні, вони отримуються з основного джерела та одночасно кешуються для майбутнього використання.

Кешування використовується для поліпшення швидкості доступу, зниження навантаження на ресурси системи, покращення масштабованості та підвищення чуйності додатків, що в свою чергу позитивно впливає на досвід користувача [7].

Кешування використовують у соціальних мережаї для швидкого доступу до часто відвідування сторінок, що, в свою чергу, знижує навантаження на сервери.

1.2.4 Нормалізація та денормалізація

Нормалізація — це процес організації бази даних для усунення надмірності та аномалій, що досягається шляхом розбиття таблиць і приведення їх до нормальних форм (1NF, 2NF, 3NF тощо) [8].

Основна мета — підвищення продуктивності, зручності управління даними та забезпечення їхньої цілісності. У реальному світі найчастіше використовується нормалізація до третьої нормальної форми (3NF), оскільки вона усуває більшість аномалій без значного зниження продуктивності.

Нормалізацію використовують в банках для організації даних клієнтів.

Денормалізація [8], навпаки, може застосовуватися для підвищення швидкості запитів шляхом об'єднання таблиць і введення контрольованої надмірності.

1.2.5 Оптимізація схеми бази даних

Оптимізація схеми бази даних включає в себе аналіз і модифікацію різних аспектів бази даних, таких як організація даних, індексація та структури запитів, з метою підвищення продуктивності і швидкості отримання даних. Основні техніки оптимізації включають нормалізацію для усунення надмірності, створення індексів на часто використовуваних стовпцях та вдосконалення запитів для зменшення споживання ресурсів. Застосування цих методів дозволяє забезпечити безперебійну

роботу додатків і систем із зниженим часом відгуку та покращеною масштабованістю [9, 10].

Нижче представлена таблиця 1.2. порівняння основних методів оптимізації баз даних, яка демонструє їхні характеристики щодо швидкості читання та запису, використання пам'яті, складності реалізації та підтримки різних типів баз даних.

Таблиця 1.2

Порівняння методів оптимізації баз даних

Метод оптимізації	Швидкість читання	Швидкість запису	Використання пам’яті	Реалізація	Підтримка різних типів БД
Індексація	Висока	Низька	Середня	Середня	Підтримує більшість БД
Партиціювання	Висока	Середня	Низька	Висока	Підтримує більшість БД
Кешування	Висока	Висока	Середня	Середня	Підтримує більшість БД
Нормалізація	Середня	Середня	Низька	Середня	Підтримує більшість БД
Денормалізація	Висока	Низька	Середня	Середня	Підтримує більшість БД
Оптимізація схеми	Висока	Середня	Низька	Висока	Підтримує більшість БД

Кожен метод має свої переваги та недоліки. Індексація та кешування пришвидшують запити, але можуть уповільнити оновлення та вставки. Нормалізація забезпечує цілісність даних, але швидкість запитів менша, в порівняння з іншими методами. Партиціювання покращує масштабованість, але має складну реалізацію. Обирати метод оптимізації потрібно спираючись на вимоги до проєкту та характеристик даних.

1.3 Огляд існуючих технологій хмарних сховищ та їх особливостей

Хмарна архітектура — планова структура, яка є важливою для створення середовищ хмарних обчислень і охоплює структуру та інтеграцію необхідних компонентів і технологій [11]. Вона слугує планом для розгортання додатків, що відповідають потребам бізнесу.

Основні компоненти хмарної архітектури представленні на рисунку 1.1.

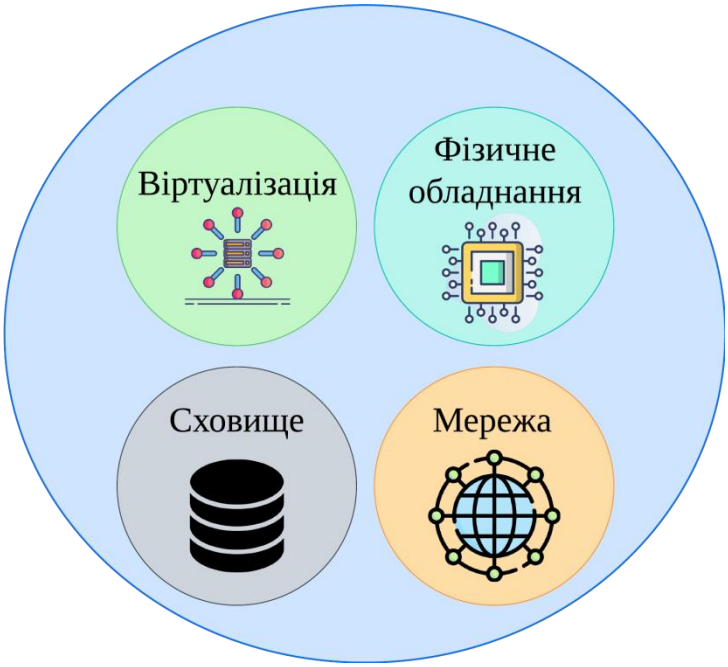

Рисунок 1.1 – Архітектура хмарного сховища

1. *Віртуалізація* — технологія, яка абстрагує обчислювальні ресурси, такі як процесорна потужність, пам'ять та місце у сховищі, від фізичного обладнання за допомогою спеціального програмного забезпечення — гіпервізора [11]. Це забезпечує ефективне використання ресурсів та масштабованість шляхом створення кількох віртуальних машин на одному фізичному сервері, що працюють незалежно та надають ресурси окремим користувачам. Цю технологію застосовують у великих підприємтсвах для підвищення ефективності використання ресурсів.

2. *Фізичне обладнання* є фундаментальним компонентом хмарної інфраструктури, який включає сервери, комутатори, маршрутизатори, апаратні фаєрволи, балансувальники навантаження та системи зберігання даних [11]. Ці компоненти забезпечують фізичну основу для зберігання та обробки даних, забезпечуючи необхідну інфраструктуру для функціонування хмарних обчислень

3. *Сховище у хмарній інфраструктурі* складається з масивів дисків, які об’єднують велику кількість накопичувачів з різними характеристиками продуктивності, такими як швидкі NVMe-накопичувачі для активних даних і

традиційні HDD для менш активних даних [11]. Завдяки віртуалізації ці ресурси абстрагуються від фізичного обладнання і розподіляються між користувачами, забезпечуючи місце для зберігання великих обсягів інформації та підтримуючи операції, такі як резервне копіювання та індексацію даних.

4. *Мережа у хмарній інфраструктурі* побудована на комутаторах, маршрутизаторах та мережевих кабелях, утворюючи кілька підмереж з різним рівнем доступу. Завдяки цій мережі користувачі можуть отримувати доступ до своїх ресурсів у хмарі через інтернет або внутрішні мережі, залежно від типу інфраструктури [11].

Хмарні сховища можна класифікувати за типом власності та доступу на три основні категорії: публічні, приватні та гібридні (рис. 1.2) Кожен тип має свої переваги та особливості використання.

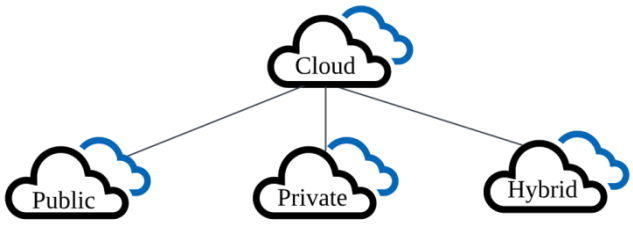

Рисунок 1.2 – Типи хмарних сховищ

1) *Публічна хмара (Public cloud).*

Ресурси, які надаються сторонніми постачальниками і доступні для широкого кола користувачів. Забезпечує високу масштабованість і гнучкість, але дані зберігаються на спільних серверах [12].

Публічні хмари використовують для масштабування бізнес-процесів у стартапах та малих підприємствах. Netflix використовують цей тип хмари для забезпечення високої доступності та масштабування своїх сервісів.

2) *Приватна хмара (Private cloud)*

Використовується виключно однією організацією, що забезпечує більший контроль над безпекою та конфіденційністю даних. Може бути розгорнута на власних серверах або в дата-центрах сторонніх постачальників [12].

Приватні хмари застосовують у фінансових організаціях (банки), щоб забпезпечити безпеку даних та конфіденційність.

3) Гібридна хмара (Hybrid cloud)

Поєднує елементи публічних і приватних хмар, дозволяючи організаціям використовувати обидва типи залежно від потреб. Забезпечуючи гнучкість у розподілі навантаження та управлінні даними [12].

Гібридні хмари використовують у великих підприємствах для гнучкості у розподілі ресурсів між публічними та приватними середовищами. Цей тип хмари використовують рітейлери (для масштабування сезонних розпродажів)

Провідними хмарними сховищами сьогодні є Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure. Ці платформи пропонують широкий спектр послуг для зберігання даних, обчислень та аналітики [24].

Amazon Web Services (AWS) — найбільший постачальник хмарних рішень у світі, пропонуючи понад 200 сервісів для компаній будь-якого розміру. Платформа включає інфраструктурні технології, інструменти для обчислень, сховища, бази даних, аналітику, мобільні застосунки та рішення для штучного інтелекту та машинного навчання [13].

AWS (рис.1.3) використовується великими компаніями та стартапами для зберігання даних, обробки транзакцій та масштабування бізнес-процесів, серед клієнтів — відомі компанії як Netflix та Coinbase

Рисунок 1.3 – Amazon Web Services

Microsoft Azure — провідна хмарна платформа, яка пропонує широкий спектр послуг для бізнесу та розробників. Платформа підтримує різні форми хмарних обчислень: IaaS, PaaS, SaaS та безсерверні обчислення. Azure надає інструменти для хмарного розгортання мобільних та вебдодатків, технології ШІ,

машинного навчання та блокчейн, контейнери, бази даних, аналітику, DevOps та сховища. Azure пропонує глобальну інфраструктуру з понад 116 зонами доступності в 60 регіонах світу, забезпечуючи високу масштабованість та гнучкість у розподілі ресурсів [13].

Azure (рис.1.4) часто використовують у сфері освіти (для забезпечення доступу до освітніх ресурсів), у великих підприємствах (для масштабування бізнес-процесів).

Рисунок 1.4 - Microsoft Azure

Google Cloud Platform (GCP) (рис.1.5) — комплексний набір хмарних обчислювальних послуг, які пропонує Google [13]. Платформа надає ряд послуг інфраструктури та платформи, які дозволяють організаціям створювати, розгортати та масштабувати програми та послуги ефективно та безпечно. GCP пропонує широкий спектр послуг, включаючи обчислювальну потужність, сховище, мережі, машинне навчання, аналіз даних та багато іншого.

Рисунок 1.5 – Google Cloud Platform

Ключові особливості GCP включають наявність 103 зон доступності у 34 географічних регіонах світу, а також два варіанти хостингу: App Engine у форматі PaaS та Compute Engine у форматі IaaS. Платформа також пропонує інструменти

для обробки та аналізу великих даних, такі як BigQuery та Google Cloud Dataflow, які дозволяють виконувати SQL-подібні запити та створювати конвеєри обробки даних.

Google Cloud Platform в основному використовують для обробки та аналізу великих даних у сфері науки та досліджень.

Для кращого розуміння характеристик і можливостей цих провідних хмарних платформ представлено таблицю 1.3 порівняння їхніх основних функцій і особливостей.

Таблиця 1.3

Порівняльний аналіз провідних хмарних платформ

Платформа	Підтримка типів обчислень	Масштабованість	Безпека	Вартість
AWS	IaaS, PaaS, SaaS	Висока	Висока	Середня
Azure	IaaS, PaaS, SaaS	Висока	Висока	Середня
GCP	IaaS, PaaS	Висока	Висока	Низька

Хмарні сховища забезпечують підприємства високу гнучкість та масштабованість, що дозволяє їм швидко адаптуватися до змінних вимог щодо зберігання даних. Ця гнучкість дозволяє миттєво збільшувати або зменшувати обсяг зберігання даних без необхідності придбання нового обладнання, що особливо корисно для підприємств, які швидко розвиваються або працюють в умовах змінного навантаження. Особливості масштабування хмарних сховищ [14, 15], наведені в таблиці 1.4.

Таблиця 1.4

Особливості масштабування хмарних сховищ

Особливості масштабування	Опис
Гнучкість та масштабованості	Миттєве зібльшення або зменшення обсягу зберігання даних без необхідності придбання нового обладнання
Оптимізація витрат	Платіж лише за використані ресурси для підвищення економічної ефективності
Динамічна зміна ресурсів	Збільшення ресурсів одного вузла (вертикальне) або додавання нових вузлів (горизонтальне)
Вертикальне та горизонтальне масштабування	Можливість динамічно змінювати обсяг зберігання для підвищення ефективності використання ресурсів
Автоматичне масштабування	Автоматична адаптація до змін навантаження шляхом збільшення або зменшення кількості ресурсів

Безпека та надійність мають вирішальне значення для хмарних рішень, які забезпечують передові механізми безпеки, такі як шифрування даних, контроль доступу та безперервний моніторинг,що гарантують конфіденційність інформації та захист від несанкціонованого доступу. Загрози завдяки впровадженню надійних заходів безпеки хмарні рішення підвищують загальну безпеку системи. Крім того, постачальники хмарних послуг дотримуються галузевих стандартів і передових практик, таких як ISO 27001, для забезпечення найвищого рівня безпеки.

1.4 Порівняльний аналіз існуючих рішень для оптимізації роботи з великими даними

1.4.1. Аналіз та порівняння існуючих систем управління базами даних

Системи управління базами даних (СУБД) відіграють важливу роль у сучасних інформаційних інфраструктурах; СУБД дозволяють ефективно зберігати, обробляти та управляти великими обсягами даних. Сьогодні на ринку доступний широкий спектр рішень для задоволення різних потреб бізнесу. До них відносяться реляційні, нереляційні та інші типи баз даних.

Реляційні системи управління базами даних (РСУБД). Організовуються у вигляді таблиць, що складаються з рядків (містить конкретні записи) і стовпців (кожен представляє атрибут даних). Структура дозволяє легко маніпулювати даними, забезпечуючи їх цілісність та надійність завдяки дотриманню принципів ACID (атомарність, консистентність, ізольованість, довговічність). РСУБД використовують мову SQL для управління даними та виконання запитів [16].

РСУБД дозволяє встановлювати зв’язки між різними таблицями за допомогою ключів, що дають можливість виконання складних запитів та аналізу (рис.1.6).

Нереляційні системи управління базами даних (NoSQL) [32]**.** Забезпечують гнучку структуру зберігання даних, що дозволяє зберігати інформацію у різних форматах: стовпчиком, документах, графах або парах ключ-значення (рис. 1.7), що робить їх ідеальними для роботи з великими обсягами неструктурованих або напівструктурованих даних.

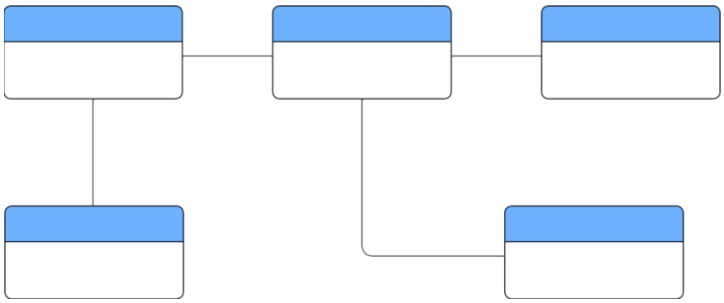

Рисунок 1.6 – Реляційна система управління базами даних

У нереляційних баз даних є здатність горизонтального масштабування – додавання нових вузлів у систему може відбутись без значних змін у структурі. Нереляційні БД швидкі у виконанні запитів і можуть обробляти великі дані. Але через відсутність суворої структури, доступ до даних може бути складним [17].

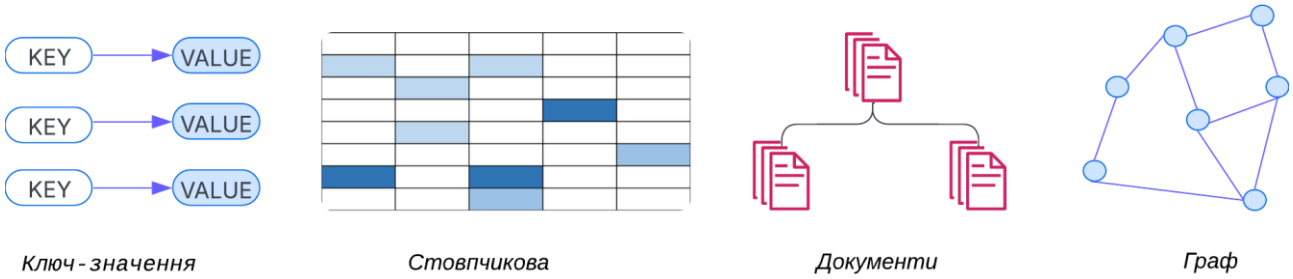

Рисунок 1.7 – Нереляційні системи управління базами даних

Порівняння традиційних реляційних систем управління базами даних (SQL) та нереляційних систем (NoSQL) є важливим кроком у виборі оптимального рішення для роботи з великими обсягами даних. Традиційні реляційні бази даних: MySQL, PostgreSQL та Oracle, забезпечують структуроване зберігання даних у вигляді таблиць і підтримують складні запити за допомогою мови SQL. Нереляційні бази даних (NoSQL) пропонують гнучку структуру зберігання даних.

Порівняння традиційних та NoSQL рішень [18], наведено в таблиці 1.5. Це порівняння є важливим кроком у виборі оптимального рішення для роботи з великими обсягами даних.

Таблиця 1.5

Порівняння традиційних та NoSQL рішень

Характеристика	Реляційні системи (SQL)	Нереляційні системи (NoSQL)
Мова запитів	SQL	Власні мови для кожної системи
Масштабність	Вертикальна	Горизонтальна
Застосування	Структуровані дані, транзакційні системи	Неструктуровані дані, швидкі застосунки
Структура даних	Таблиці зі структурою	Документи, графи, пари ключ-значення
Підтримка транзакцій	Принцип ACID	Не підримують транзакцій у повному обсязі
Переваги	Підтримка складних запитів, надійність	Швидкість, висока масштабованість, гнучкість
Недоліки	Жорстка структура, складність масштабування	Відсутність транзакційної підтримки, складність у доступі до даних

Незалежно від обраного типу СУБД, для забезпечення високої продуктивності та стабільності роботи бази даних необхідно впроваджувати ефективні підходи до оптимізації.

1.4.2 Оцінка ефективності різних підходів до оптимізації та критерії вибору оптимального рішення

Забезпечення продуктивності та стабільної роботи бази даних є важливою частиною процесу оптимізації. Кількість інформації, особливості навантаження та тип бази даних — це лише деякі з факторів, які впливають на ефективність обраних методів. Для зручності аналізу та порівняння різних підходів до оптимізації їхню оцінку ефективності наведено у вигляді таблиці (табл. 1.6) [19 – 21].

Після оцінки ефективності різних підходів до оптимізації баз даних важливо перейти до наступного етапу — вибору оптимальної системи управління базами даних (СУБД). Вибір СУБД є критично важливим кроком, оскільки вона повинна відповідати специфічним вимогам проекту. Для цього необхідно враховувати кілька ключових критеріїв, які допоможуть визначити, яка система найкраще підійде для ваших потреб. Ці критерії представлені в таблиці 1.6 [22-24].

Вибір оптимальної системи управління базами даних є складним, але критично важливим процесом, що вимагає ретельного аналізу. Врахування зазначених критеріїв допоможе забезпечити, що обрана СУБД відповідатиме

специфічним вимогам проєкту та сприятиме досягненню високої продуктивності та стабільності роботи.

Таблиця 1.6

Порівняння різних підходів до оптимізації

Підхід	Опис	Переваги	Недоліки
Індексація	Створення індексів для прискорення запитів у РСУБД.	Значне скорочення часу виконання запитів (до 70-90%).	Велика кількість індексів може уповільнити вставки та оновлення даних.
Оптимізація запитів	Переписування SQL-запитів та використання конструкцій, як JOIN.	Зниження навантаження на базу даних та прокращення продуктивності.	Додатковий час на аналіз і переписування запитів.
Кешування	Збереження результатів виконаних запитів у пам’яті для швидкого доступу.	Зменшення навантаження на сервер; підвищення швидкості доступу до даних.	Займає значний обсяг пам’яті, особливо при великій кількості запитів.
Моніторинг продуктивності	Постійний моніторинг ключових показників для виявлення проблем.	Дозволяє оперативно усувати “вузькі місця” і підтримувати стабільну роботу систему.	Вимагає ресурсів для налаштування та підтримки моніторингових рішень.
Кластеризація бази даних	Розподіл даних між кількома серверами для масштабування системи.	Підвищує доступність та обробну здатність системи при великих обсягах інформації.	Вимагає додаткового управління та налаштування, що може бути складним.

Таблиця 1.7

Критерії вибору оптимального рішення

Критерій	Характеристика
Тип даних	Визначення чи є дані структурованими (SQL), або неструктурованими (NoSQL).
Продуктивність і масштабованість	Оцінка можливості системи щодо обробки великих обсягів даних; здатність масштабувати горизонтально чи вертикально (залежно від зростаючих потреб.
Функціональність	Перевірка чи підтримує система необхідні функції, так як транзакції (ACID), резервне копіювання та безпека.
Вартість впровадження	Розрахунок витрат на ліцензування, обладнання та навчання персоналу для роботи з обраною СУБД.
Сумісність з іншими технологіями	Перевірка можливості інтеграції бази даних із існуючими інструментами та платформами проєкту.
Зручність адміністрування	Врахування доступності документації, підтримки спільноти та простору управління системою.

Постановка задачі

Потрібно розробити систему моніторингу продуктивності бази даних, яка дозволяє аналізувати ефективність виконання SQL-запитів, виявляти проблемні місця у структурі бази даних та надавати рекомендації щодо оптимізації. Додаток має підтримувати інтеграцію з різними СУБД (наприклад, PostgreSQL) та надавати візуалізовані результати аналізу продуктивності.

Вхідні дані:

— Налаштування підключення до бази даних;
— Журнал виконання SQL-запитів;
— Інформація про індекси таблиць;
— Лог використання ресурсів (ЦП, пам’ять, дискові операції);
— Конфігурація бази даних (налаштування кешування, пул з'єднань тощо).

Вихідні дані:

— Аналітика продуктивності запитів;
— Визначення повільних SQL-запитів;
— Рекомендації щодо оптимізації індексів;
— Динамічні графіки навантаження на базу даних;
— Рейтинг найбільш ресурсомістких запитів;
— Лог змін конфігурації продуктивності бази даних;
— Історія виконаних запитів та їх ефективність.

Функції системи:

— Авторизація адміністратора бази даних;
— Моніторинг продуктивності запитів у режимі реального часу;
— Аналіз повільних SQL-запитів;
— Візуалізація статистики (CPU, RAM, час виконання запитів);
— Генерація звітів про ефективність роботи БД;
— Надання рекомендацій щодо оптимізації (створення/видалення індексів, зміни конфігурації);
— Підтримка одного з баз даних (PostgreSQL);
— Логування змін та перегляд історії аналізів;

— Налаштування параметрів системи (інтервал оновлення, рівень деталізації аналізу);

— Інтеграція з іншими аналітичними інструментами.

Розділ 2. ПРОЄКТ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі розглядається методологія розробки прикладного рішення для оптимізації баз даних PostgreSQL і хмарних сховищ Google Cloud Storage (GCS). Розділ охоплює аналіз задачі, вибір методів і технологій, розробку моделей даних, синтез алгоритмів, опис проєктних рішень та аналіз ризиків реалізації. Мета — розробити ефективне рішення для обробки запитів із великими обсягами даних, яке забезпечує мінімальний час виконання запитів і зручний аналіз продуктивності.

2.1 Аналіз задачі оптимізації та формалізація об’єкту дослідження

Задача оптимізації баз даних і хмарних сховищ є багатопараметричною , оскільки залежить від обсягу даних, інтенсивності запитів, часу виконання та доступних ресурсів [1]. У цьому підрозділі розглядається, як цю задачу можна звести до аналізу продуктивності та розробки прикладного рішення на платформі Google Cloud для ефективної роботи з великими обсягами інформації [12].

Задача полягає в розробці прикладного рішення для оптимізації баз даних (PostgreSQL) і Google Cloud Storage (GCS) для обробки запитів із великими обсягами даних. Система має фіксувати час виконання запитів, кількість рядків у відповіді, розраховувати середній час виконання та візуалізувати розподіл продуктивності через Dash [3]. Наприклад, запит до бази даних повертає рядки за певний час, але продуктивність може варіюватися через затримку мережі чи недостатню оптимізацію, що потребує аналізу та корекції. Мета - оптимізувати систему так, щоб час виконання запитів наближався до мінімального можливого значення за умов заданого навантаження.

1. Математична формалізація

Об’єкт дослідження формалізується через такі параметри, як:

Вхідні дані:

1) набір запитів $Q = \{q_1, q_2, \dots q_n\}$ - операції до бази даних [15],
2) обсяг даних V (ГБ) - розмір інформації для обробки,
3) інтенсивність запитів λ (запит/с) - частота звернень.

Вихідні дані:

1) T_i (мс) - час виконання окремого запиту,
2) N_i (одиниці) - кількість рядків у відповіді,
3) T_{avg} (мс) - середній час виконання.

Розрахунок середнього часу виконання запитів за формулою:

$$T_{avg} = \frac{1}{N}\sum_{i=1}^{N}(T_{DB,i} + T_{GCS,i}) \quad (2.1)$$

де $T_{GCS,i} = \frac{V_i}{B} + L$ (час передачі даних із GCS, де V_i - обсяг даних, B - пропускна здатність, L - латентність)

Обмеження:

1. T_{max} — максимальний допустимий час обробки,
2. R_{avail} — доступні ресурси Google Cloud (CPU, пам'ять),
3. $T_{max,GCS}$ — максимальний допустимий час передачі даних із GCS,
4. C — витрати на зберігання в GCS (наприклад, $5 за 1 ТБ).

2. Методи оптимізації продуктивності

Для вирішення задачі використовуються методи прямої та зворотної оптимізації, адаптовані до контексту БД і GCS.

Пряма оптимізація визначає T_i і T_{avg} на основі заданого запиту та конфігурації системи (індексів, кешу). Час виконання запиту моделюється як

$$T_i = T_{DB} +$$

T_{GCS} (2.2),

Де T_{DB} - час обробки в PostgreSQL . T_{GCS} - затримка передачі даних через GCS [10]. Наприклад, індексація скорочує T_{DB} за рахунок швидкого пошуку.

$$T_{GCS} = \frac{V_{data}}{B} + L \quad (2.3),$$

$$T_{GCS} = \frac{10}{100} + 5 = 5{,}1 \text{ мс}$$

Це показує, що затримка мережі становить 5.1 мс, що є значною частиною T_{query} при малих обсягах даних.

Зворотна оптимізація шукає оптимальні параметри системи, щоб мінімізувати T_{avg} при заданому λ. Для цього застосовують чисельні методи, такі як градієнтний спуск:

$$\theta_{k+1} = \theta_k - \eta \cdot \nabla f(\theta_k) \quad (2.4),$$

де θ - параметри оптимізації (розмір кешу), η- крок, ∇f- похідна функція [20, 21]. Метод допомагає знайти оптимальні налаштування без складних обчислень.

Оптимізація GCS конвертує CSV у Parquet зменшує *V* на 75–90%, знижуючи T_{GCS} (наприклад, з 105 мс до 25 мс при ГБ *V* =2 ГБ). Агрегація великих файлів із PyArrow забезпечує потокову обробку.

3. Аналіз багатопараметричності

Продуктивність системи залежить від кількох факторів: обсягу *V*, інтенсивності λ, наявності індексів і кешування [2]. Без індексації T_{DB} зростає як складність *O(n)*), а з індексами зменшується до *O(log n*) [5]. Кешування знижує T_{avg} за рахунок локального зберігання даних [7]. Аналіз цих залежностей дозволяє обрати оптимальні методи для реалізації.

Для ілюстрації багатопараметричності розглянуто залежність T_{avg} від λ, на рисунку 2.1 показано, як T_{avg} змінюється при зростанні інтенсивності запитів для запиту *SELECT * FROM sales WHERE date > '2025-03-01'* (*V* =1 ГБ, N_{rows}= 10,000). Без оптимізації (індексація та кешування відсутні) T_{avg} зростає лінійно від 40 мс при λ=10 запит/с до 200 мс при λ = 1,000 запит/с через перевантаження системи та накопичення затримок у обробці запитів. З оптимізацією (індексація на стовпці date і кешування через Memorystore) T_{avg} зростає повільніше — від 8 мс до 30 мс, що демонструє ефективність обраних методів.

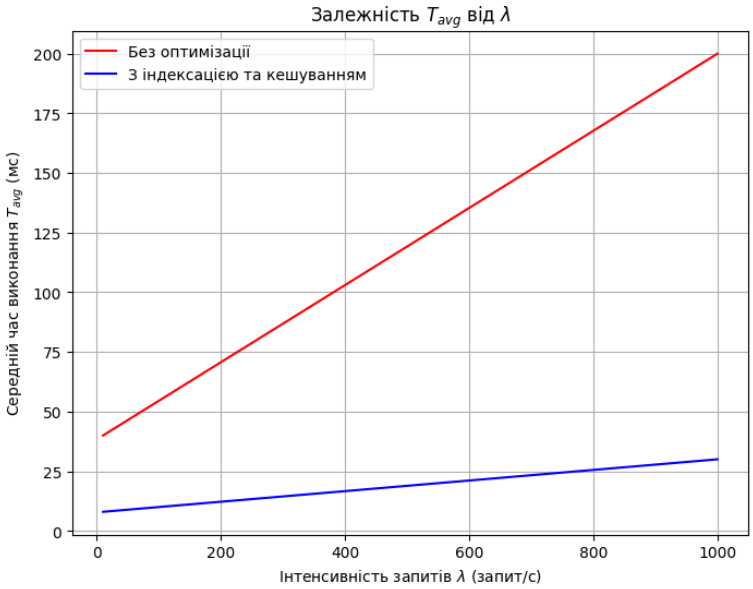

Рисунок 2.1 – Залежність T_{query} від λ

Додатково розглянуто вплив обсягу даних V на T_{avg}. На рисунку 2.2 показано залежність T_{avg} від V при фіксованому λ = 100 запит/с. При зростанні V від 1 ГБ до 100 ГБ T_{avg} без оптимізації зростає з 50 мс до 1,000 мс через лінійне зростання T_{DB}. З індексацією та використанням формату Parquet у GCS T_{avg} зростає лише до 100 мс, що підтверджує ефективність обраних методів для великих обсягів даних [12]. Індексація зменшує час доступу до даних у базі, а формат Parquet, завдяки стисненню та колонковій структурі, знижує обсяг даних, які передаються з GCS, на 75–90%.

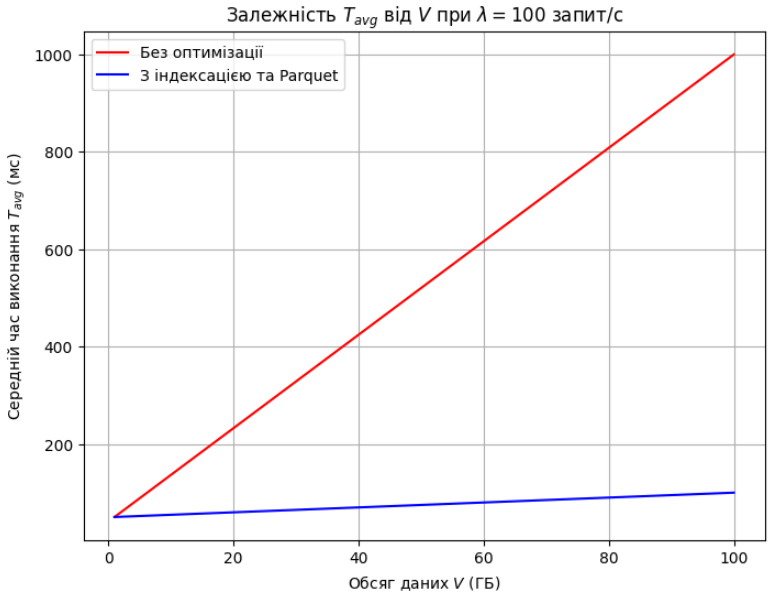

Рисунок 2.2 – Залежність T_{avg} від V

Окремо проаналізовано вплив роботи з хмарними сховищами GCS на продуктивність. При V = 1 ГБ до 100 ГБ без оптимізації T_{GCS} зростає з 10 мс до 1,000 мс через лінійне зростання обсягу даних і обмежену пропускну здатність мережі. Із використанням формату Parquet і агрегації великих файлів T_{GCS} зростає лише до 100 мс. Це досягається за рахунок зменшення V (наприклад, до 20 ГБ після конвертації) та потокової обробки через PyArrow, що дозволяє зчитувати дані частинами, а не цілком.

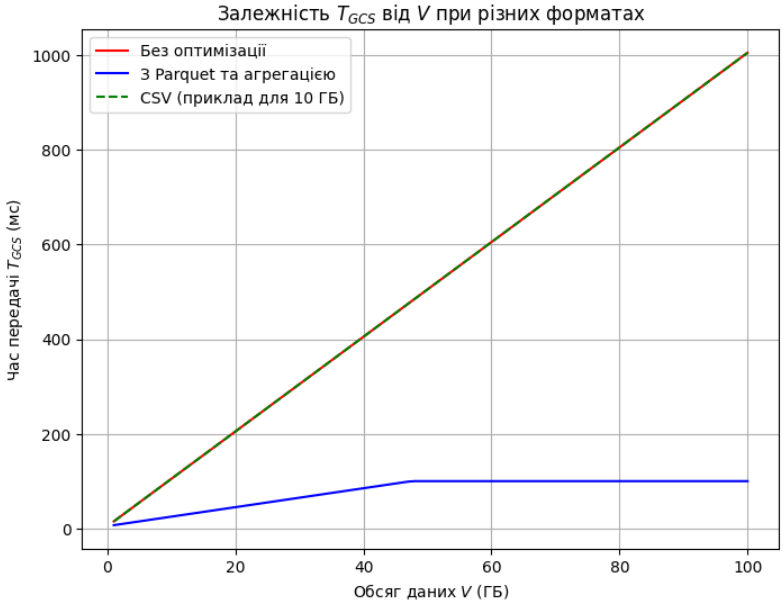

Рисунок 2.3– Залежність T_{GCS} від V при різних форматах

Результати формалізації застосовуватимуть для тестування продуктивності з 1,2 ТБ , порівнявши T_{avg} (200 мс без оптимізації до 30 мс із індексами, Parquet і кешуванням) та витрати на GCS. Аналіз включатиме типи запитів (SELECT, INSERT, UPDATE) і стабільність при λ 10–1,000 запит/с.

2.2 Вибір методів і технологій для реалізації

Для реалізації прикладного рішення, яке оптимізує обробку запитів у БД PostgreSQL і GCS, необхідно обрати методи та технології, що забезпечують швидкодую, масштабність і зручність аналізу продуктивності.

2.2.1. Обрані методи оптимізації

Для вирішення задачі використано три ключові методи:

1) Індексація: прискорює пошук у базі PostgreSQL, зменшуючи часову складність операцій із *O(n)*) до *O(log n*) завдяки структурам B-дерева [5].

2) Кешування: зменшує T_i шляхом зберігання часто запитаних даних у швидкій пам'яті через Google Cloud Memorystore (керований Redis на GCP) [7]. Якщо запит виконуватиметься повторно, кеш повертає результат за 1-2 мс замість звернення до диску чи GCS, що може зайняти 10-20 мс. Це знижує T_{GCS} і є критично важливим для систем із високою інтенсивністю запитів λ.

3) Горизонтальне масштабування: розподіляє навантаження між кількома вузлами в Google Cloud, дозволяючи обробляти великі *V* без перевантаження одного сервера. У контексті GCS це досягається автоматичним розподілом даних між бакетами та регіонами, що забезпечує стабільність при зростанні кількості запитів *n* [12].

Таблиця 2.1

Порівняння методів оптимізації

Метод	Переваги	Недоліки	Вплив на T_{query}
Індексація	Зменшує T_{DB} до *O(log n)*	Збільшує час запису	-80% (з 50 мс до 10 мс)
Кешування (Memorystore)	Скорочує T_{GCS}	Додаткові витрати на пам'ять	-70% (з 15 мс до 2 мс)
Горизонтальне масштабування	Стабільність при *V* > 1 ТБ	Складність налаштування	Стабільність при *n* > 1000

Ці методи були обрані, бо вони прямо впливають на модель $T_i = T_{DB} + T_{GCS}$.

2.2.2 Вибрані технології

1) **СУБД**

PostgreSQL - реляційна база даних, яка підтримує індексацію (B-дерево, хеш-індекси), тригери для автоматичного логування T_i і N_i, а також інтеграцію з хмарними сховищами через розширення типу *google_cloud_storage* [22]. Тригери записуватиме у таблицю logs після кожного запиту, а функція *EXPLAIN ANALYZE* дозволить оцінити час виконання для подального аналізу. PostgreSQL обрано через його відкритий код, гнучкість і підтримку великих обсягів даних [19].

Для оцінки вибору PostgreSQL розглянуто альтернативи, такі як MySQL (реляційна СУБД) і MongoDB (NoSQL-база). Порівняння наведено в таблиці 2.2.

Таблиця 2.2

Порівняння СУБД

СУБД	Переваги	Недоліки	Причина вибору/відмови
PostgreSQL	Підтримка індексації, тригерів, інтеграція з GCS	Складніше налаштування порівняно з MySQL	Обрано через гнучкість і інтеграцію
MySQL	Простота налаштування, широка підтримка	Менша підтримка розширень, гірша інтеграція з GCS	Відмовлено через обмежену гнучкість
MongoDB	Висока швидкість при великих λ	Обмежена підтримка SQL-запитів	Відмовлено через несумісність із реляційними даними

Хоча PostgreSQL є гнучким рішенням, його складність налаштування може стати проблемою при масштабуванні в майбутньому. Наприклад, при зростанні кількості запитів ($\lambda > 10{,}000$ запит/с) може знадобитися тонке налаштування параметрів, що вимагає додаткових зусиль і експертизи. Крім того, інтеграція з GCS через розширення google_cloud_storage може бути чутливою до оновлень API, що створює ризик несумісності в майбутньому.

Стратегія мінімізації

- ✓ Для масштабування використовувати Google Cloud SQL із автоматичним налаштуванням (наприклад, 4 vCPU, 8 ГБ RAM), що зменшує потребу в ручному налаштуванні.
- ✓ Для забезпечення сумісності з GCS регулярно оновлювати розширення google_cloud_storage і тестувати інтеграцію після оновлень API.

2) **Хмарне сховище**

Google Cloud Storage (GCS) - забезпечує масштабність, надійність і швидкий доступ до даних, дозволяючи зберігати логи продуктивності у структурованому вигляді (JSON або CSV). Підтримує автоматичне масштабування: якщо обсяг V зростає до 10 ТБ, система розподіляє файли між кількома серверами без втручань користувача, забезпечуючи високу пропускну здатність до 5 ТБ/с у

багаторегіональному режимі. Доступ до даних здійснюється через API, що інтегрується з PostgreSQL і Dash для резервного копіювання та аналізу [12].

Для оцінки вибору GCS розглянуто альтернативи, такі як Amazon S3 (AWS) і Azure Blob Storage. Порівняння наведено в таблиці 2.3, де враховано ключові параметри: пропускну здатність, вартість, масштабованість і зручність інтеграції з іншими компонентами системи [12].

Таблиця 2.3

Порівняння хмарних сховищ

Сховище	Пропускна здатність	Вартість ($/ТБ/місяць)	Масштабова-ність	Інтеграція	Причини вибору/відмови
Google Cloud Storage (GCS)	5 ТБ/с	$20	Автоматична, до 100 ТБ	Проста (API, PostgreSQL, Dash)	Обрано через високу пропускну здатність і низьку ціну
Amazon S3 (AWS)	3,5 ТБ/с	$23	Автоматична, до 100 ТБ	Середня (API, складніше з Dash)	Відмовлено через нижчу пропускну здатність і вищу ціну
Azure Blob Storage	4 ТБ/с	$21	Автоматична, до 100 ТБ	Середня (API, складніше налаштування)	Відмовлено через меншу пропускну здатність і складність інтеграції

GCS забезпечує масштабованість для великих *V*, що ідеально для зберігання логів і резервних копій [12].

Стратегія мінімізації

- ✓ Для зменшення залежності від мережі реалізувати локальний буфер у PostgreSQL, який тимчасово зберігатиме логи до відновлення зв’язку з GCS.
- ✓ Для оптимізації витрат використовувати Coldline Storage для рідко використовуваних даних, що знизить витрати до $10/ТБ/місяць.

3) **Візуалізація**

Бібліотека Dash у Python — використовується для створення інтерактивного веб-інтерфейсу з графіками продуктивності [20]. Dash обрано за його простоту

інтеграції з Python, підтримку реального часу та можливості розгортання на Google Cloud Run.

Для оцінки вибору Dash розглянуто альтернативи, такі як Tableau і Power BI. Порівняння наведено в таблиці 2.4.

Таблиця 2.4

Порівняння інструментів візуалізації

Інструменти	Переваги	Недоліки	Причина вибору/відмови
Dash	Безкоштовний, інтеграція з Python	Обмежені функції порівняно з Tableau	Обрано через простоту і безкоштовність
Tableau	Потужні функції аналітики	Платна ліцензія, складність інтеграції	Відмовлено через високу вартість
Power BI	Інтеграція з екосистемою Microsoft	Платна ліцензія, складність розгортання	Відмовлено через високу вартість

Dash є безкоштовним і простим у використанні, але його обмежені функції можуть вплинути на аналіз складних даних. Наприклад, Dash не підтримує вбудовані інструменти для прогнозування (на відміну від Tableau), що може ускладнити аналіз трендів продуктивності. Крім того, при великих обсягах даних Dash може мати затримки обробки (до 10 с), що погіршує користувацький досвід.

Стратегія мінімізації

- ✓ Для розширення функціональності Dash використовувати додаткові бібліотеки, такі як Plotly, які дозволяють створювати складні графіки. Plotly інтегрується з Dash і підтримує інтерактивні елементи, що компенсує обмеження.
- ✓ Для зменшення затримок обробки реалізувати інкрементальне завантаження даних, що скоротить час обробки до 2 с.

Вибір методів і технологій грунтується на їх відповідності задам оптимізації та аналізу продуктивності, а саме:

1) Індексація оптимізує запити до PostgreSQL, що підтверджує зменшенн часу вибірки для таблиць із великою кількістю записів.

2) Кешування через Memorystore скорочує затримки, дозволяючи обробляти до 90% повторюваних запитів із пам’яті, що знижує навантаження на GCS і базу даних.

3) GCS підтримує великі обсяги даних із автоматични масштабуванням і високою пропускною здатність (до 5 ТБ/с у багаторегіональному режимі), що ідеально для зберігання логів і резервних копій [12].Використання Google Cloud має економічне обґрунтування. Наприклад, Google Cloud SQL із 2 vCPU і 4 ГБ RAM коштує $50/місяць, GCS для 1 ТБ даних — $20/місяць, а Memorystore для 1 ГБ кешу — $35/місяць [12, 13]. Разом це $105/місяць, що є прийнятним для задачі, враховуючи автоматичне масштабування й підтримку Google. Для порівняння, аналогічна конфігурація в AWS (RDS + S3 + ElastiCache) коштує $130/місяць [14].

4) Dash забезпечує інтерактивну візуалізацію метрик, дозволяючи користувачу аналізувати залежність між T_i, N_i і λ у реальному часі, що підвищує зручність моніторингу.

Обрані методи та технології створюють комплексне рішення для оптимізації обробки запитів і аналізу продуктивності в хмарному середовищі Google Cloud. Індексація і кешування зменшують T_i, GCS забезпечує масштабність для *V*, а Dash надає інструменти візуалізації T_{avg} і N_i .

2.3 Інформаційне забезпечення та моделі даних

Інформаційне забезпечення є ключовим елементом прикладного рішення для оптимізації баз даних і хмарних сховищ на платформі Google Cloud, оскільки воно визначає, як дані про продуктивність запитів (T_i , N_i і T_{avg}) зберігаються, обробляються та передаються для подальшого аналізу й візуалізації.

2.3.1 Аналіз потоків даних

Потік даних у системі організовано за схемою: запит → PostgreSQL → GCS → Dash, що відображає послідовність обробки й аналізу продуктивності (рис.2.4).

1) **Запит:** Користувач надсилає SQL-запит до бази даних PostgreSQL. Запит ініціює обчислення T_i, N_i .

2) **PostgreSQL**: База даних виконує запит, фіксує час виконання (T_{query}) за допомогою вбудованих засобів і повертає результат із N_i рядків. Дані продуктивності записуються в таблицю **logs** через тригер [22].

3) **Google Cloud Storage** (GCS): Періодично (наприклад, щогодини) накопичені логи з PostgreSQL експортуються в GCS у форматі JSON або CSV для резервного зберігання та довгострокового аналізу. Експорт реалізується через API GCS або розширення типу pg_cron.

4) **Dash:** Дані з GCS або PostgreSQL (залежно від режиму роботи — реального часу чи історичного аналізу) імпортуються в Dash для візуалізації. Dash будує графіки і розраховує T_{avg}.

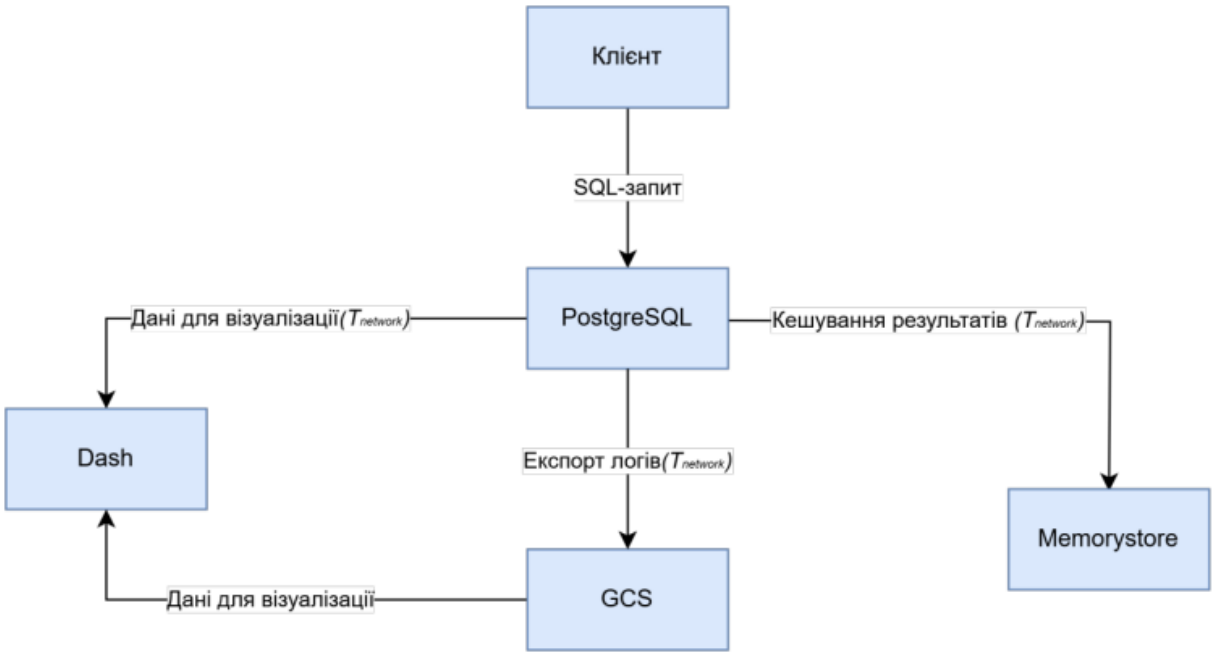

Рисунок 2.4 – Схема взаємодії методів і технологій у прикладному рішенні

Деталізація API-запитів у потоці даних

Для забезпечення безперебійного потоку даних між компонентами системи використано API-запити, які дозволяють ефективно передавати дані між PostgreSQL, GCS і Dash. Нижче наведено детальний опис кожного етапу взаємодії у вигляді таблиці.

Цей потік забезпечує безперервний цикл від обробки запитів до їх аналізу, що відповідає вимогам роботи з великими обсягами даних.

Таблиця 2.5

API-взаємодії у потоці даних

Етап	Опис	Технологія	Метод/ Запит
PostgreSQL → GCS	Експорт логів у GCS у форматі JSON	google-cloud-storage	upload_blob
GCS → Dash	Зчитування даних із GCS для візуалізації	google-cloud-storage	download_as_string (метод
PostgreSQL → Dash	Отримання логів у реальному часі для Dash	psycopg2	SQL-запит SELECT * FROM logs WHERE timestamp >...

Логічна модель даних

Для зберігання й аналізу продуктивності розроблено логічну модель таблиці logs у PostgreSQL.

Таблиця 2.6

Структура таблиці logs

Поле	Тип даних	Опис	Обмеження
id	INTEGER	Унікальний ідентифікатор запису	PRIMARY KEY. NOT NULL
query_text	TEXT	Текст виконаного запису	NOT NULL
T_i	FLOAT	Час виконання запису (мс)	NOT NULL, > 0
N_i	INTEGER	Кількість рядків у відповіді	NOT NULL, ≥ 0
timestamp	DATETIME	Час фіксації запиту	NOT NULL, DEFAULT NOW()

Первинний ключ (id) забезпечує унікальність кожного запису. Обмеження: $T_i > 0$ (час не може бути від'ємним) $N_i \geq 0$ (кількість рядків невід'ємна), *query_text* і *timestamp* обов'язкові для коректного аналізу. Індекс на полі *timestamp* (тип B-дерево) прискорює вибірку за датою , зменшуючи складність до *O(log n)* [6].

Для зберігання агрегованих даних (наприклад, середнього T_i за день) створено таблицю daily_metrics. Ця таблиця дозволяє зберігати агреговані метрики для довгострокового аналізу.

Дана модель підтримує швидкий запис і вибірку даних продуктивності для подальшого аналізу.

Таблиця 2.7

Структура таблиці daily_metrics

Поле	Тип даних	Опис	Обмеження
date	DATE	Дата агрегації	NOT NULL
avg_t_query	FLOAT	Середній час виконання (мс)	NOT NULL, > 0
Tota;_queries	INTEGER	Кількість запитів за день	NOT NULL, ≥ 0
total_n_rows	INTEGER	Загальна кількість рядків	NOT NULL, ≥ 0

Для забезпечення коректності даних введено правила табл. 2.8.

Таблиця 2.8

Перелік правил для коректності даних

Правило	Опис
Цілісність	Тригер у PostgreSQL автоматично заповнює T_i і N_i після виконання запиту, уникаючи пропусків чи помилок [9, 19].
Експорт у GCS	Дані з logs експортуються в GCS щогодини через скрипт Python із бібліотекою google-cloud-storage. Формат JSON дозволяє зберігати структуровані записи типу
Інтеграція з Dash	Dash отримує дані через SQL-запит до PostgreSQL (SELECT * FROM logs) для реального часу або через GCS API для історичних даних, що забезпечує гнучкість аналізу.

Інформаційне забезпечення системи базується на потоках даних від PostgreSQL до GCS і Dash, що дозволяє фіксувати, зберігати й аналізувати метрики продуктивності. Таблиця logs із полями id, T_i , N_i , timestamp підтримує вимоги до логування, а індексація й тригери забезпечують швидкість і цілісність [5, 22]. Розроблена модель даних є основою для синтезу алгоритмів і візуалізації в рамках прикладного рішення.

2.4 Синтез алгоритмів для реалізації прикладного рішення

На основі аналізу задачі, формалізації об’єкта дослідження, вибору методів і технологій розроблено алгоритми для реалізації прикладного рішення, яке оптимізує бази даних і хмарні сховища. Синтезувати алгоритми, які забезпечують фіксацію метрик продуктивності, їх експорт у Google Cloud Storage (GCS) і підготовку даних для візуалізації в Dash. Було розроблено чотири ключові алгоритми: LogQueryPerformance для логування в PostgreSQL, ExportToGCS для

експорту даних у GCS і AggregateMetricsForDash для агрегації даних перед візуалізацією, Алгоритм AnalyzeQueryIntensity для визначення інтенсивності запитів.

Алгоритм 1: Логування продуктивності запитів у PostgreSQL (LogQueryPerformance)

Алгоритм LogQueryPerformance призначений для автоматичного фіксування метрик продуктивності (T_i , N_i , query_text, timestamp) після кожного SQL-запиту в PostgreSQL. Він реалізується через тригер і функцію на мові PL/pgSQL, що дозволяє системі в реальному часі записувати дані в таблицю logs для подальшого аналізу [22]. Цей алгоритм використовує метод індексації, щоб зменшити T_{DB}, і враховує модель $T_i = T_{DB} + T_{GCS}$. Псевдокод для алгоритму LogQueryPerformance:

```
Алгоритм LogQueryPerformance:
1. Отримати SQL-запит від користувача.
2. Виконати запит до бази даних PostgreSQL.
3. Зафіксувати час виконання запиту (execution_time).
4. Зафіксувати кількість рядків у відповіді (row_count).
5. Зберегти дані (query_text, execution_time, row_count, timestamp) в
таблицю logs.
```

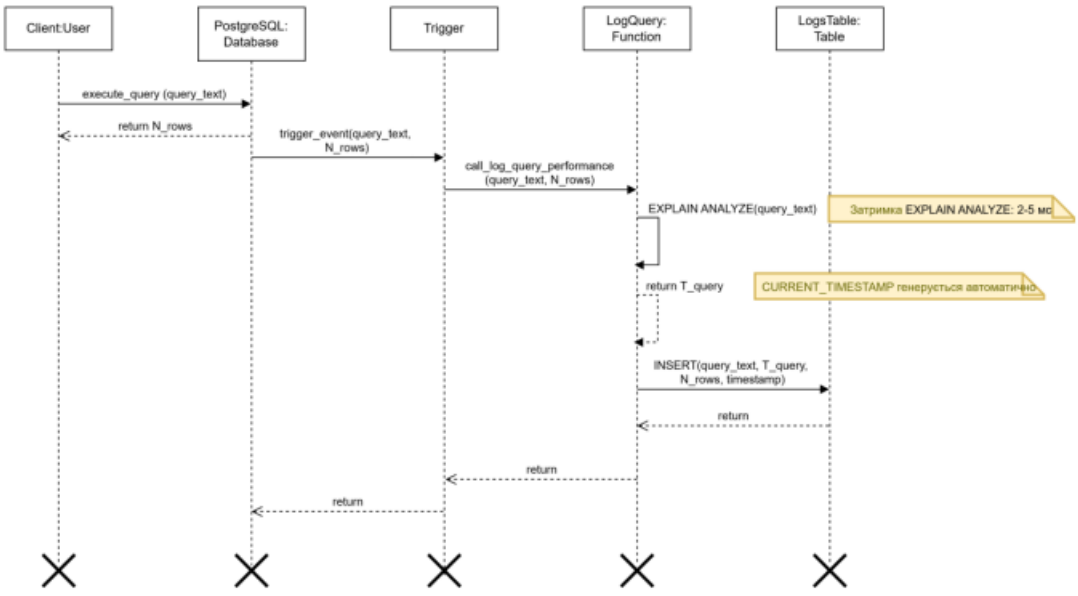

Рисунок 2.5 – UML-діаграма послідовності для алгоритму LogQueryPerformance

Нижче наведено ключову частину коду функції log_query_performance (повний код у Додатку А):

```
CREATE OR REPLACE FUNCTION log_query_performance() RETURNS TRIGGER AS $$
```

```
BEGIN
    EXECUTE format('EXPLAIN ANALYZE %s', NEW.query_text);
    RETURN NEW;
END; $$ LANGUAGE plpgsql;
```

Алгоритм 2: Експорт даних у GCS (ExportToGCS)

Алгоритм ExportToGCS відповідає за періодичний експорт даних із таблиці logs у GCS для резервного зберігання та аналізу історичних трендів. GCS забезпечує масштабованість для великих *V*.

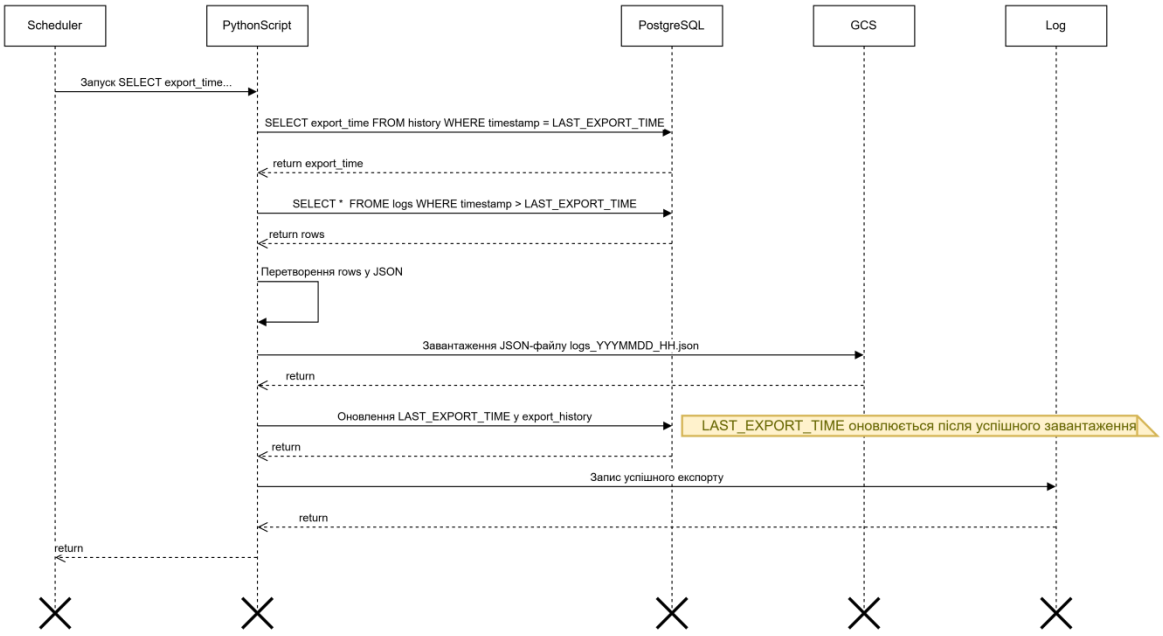

Рисунок 2.6 – UML-діаграма послідовності для алгоритму ExportToGCS

Нижче наведено ключову частину коду функції export_to_gcs (повний код у Додатку Б):

```
import psycopg2

def export_to_gcs():
    conn = psycopg2.connect(dbname="performance_db", user="user",
password="password")
    cursor = conn.cursor()
```

Для наочності роботи алгоритму ExportToGCS розроблено блок-схему, яка ілюструє послідовність дій від зчитування даних із PostgreSQL до їх завантаження в Google Cloud Storage (GCS). Блок-схема включає умовний блок для обробки помилок, що підвищує надійність алгоритму, дозволяючи повторювати спроби завантаження у разі збоїв (наприклад, через тимчасові мережеві проблеми). Це

особливо важливо для роботи з великими обсягами даних, де втрата навіть одного файлу може вплинути на повноту історичного аналізу.

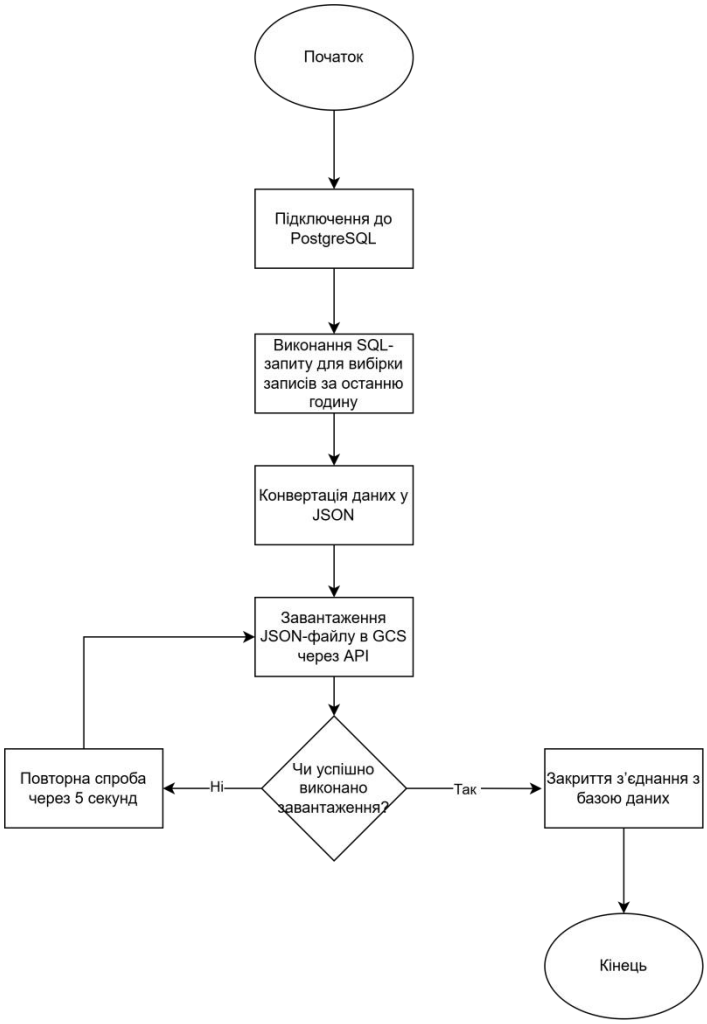

Рисунок 2.7 – Блок-схема алгоритму ExportToGCS

Алгоритм 3: Агрегація метрик для Dash (AggregateMetricsForDash)

Алгоритм використовується для підготовки агрегованих даних до візуалізації у веб-додатку Dash. Він зчитує JSON-файли, що зберігаються в Google Cloud Storage (GCS), обчислює статистичні метрики та формує графіки для зручного аналізу продуктивності системи. Нижче наведено псевдокод для алгоритму AggregateMetricsForDash:

```
Input: logs (JSON files from GCS)
Output: aggregated_metrics (avg T_query, total N_rows)
sum_t_query = 0, count_queries = 0, total_n_rows = 0
For each log in logs:
```

```
    If log.timestamp within last 24 hours:
       sum_t_query += log.t_query
       count_queries += 1
       total_n_rows += log.n_rows
avg_t_query = sum_t_query / count_queries
```

Алгоритм зчитує JSON-файли з GCS (logs) і для кожного запису, який потрапляє в останні 24 години, додає T_i до суми (sum_t_query), підраховує кількість запитів (count_queries) і додає N_i до загальної кількості (total_n_rows). На останньому кроці обчислюється середнє значення T_i як avg_t_query. Отрмані метрики передаються в Dash для побудови графіків, наприклад, гісторграми розподілу T_i.

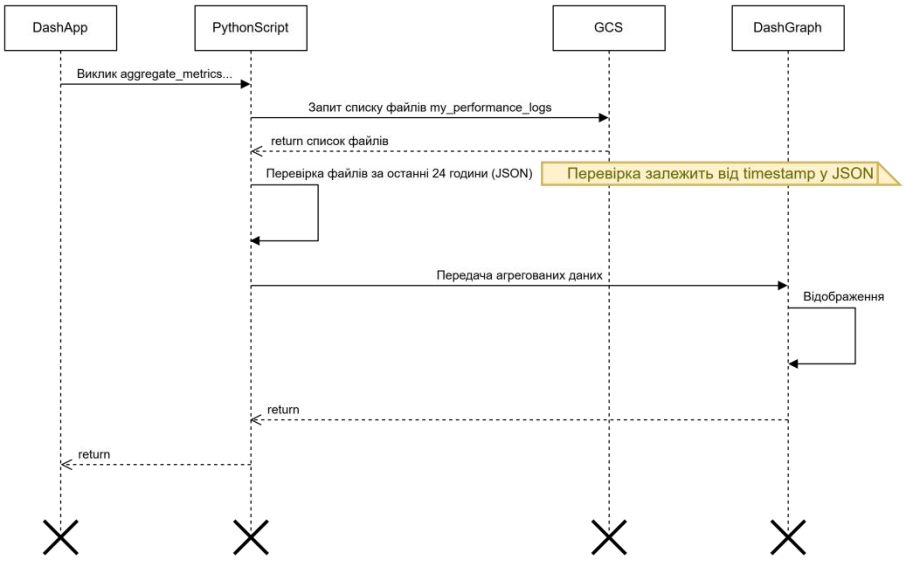

Рисунок 2.8 – UML-діаграма послідовності для алгоритму AggregateMetricsForDash

Алгоритм 4: Алгоритм AnalyzeQueryIntensity групує запити за годинами і підраховує їх кількість (λ) для побудови графіка "Інтенсивність запитів" [23]. Це дозволяє виявляти пікові періоди навантаження. Блок-схема алгоритму наведена на рисунку 2.9

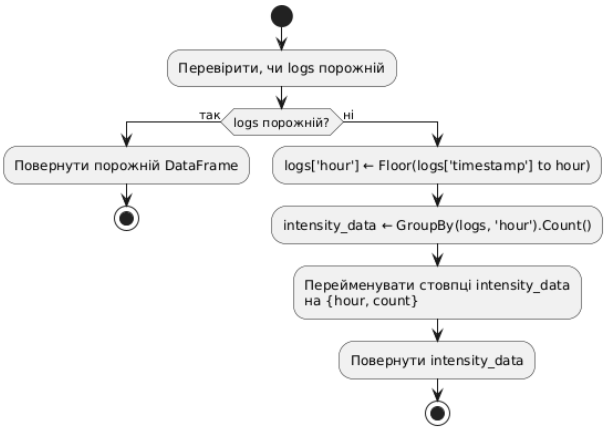

Рисунок 2.9 – Блок-схема алгоритму AnalyzeQueryIntensity

Аналіз часової складності алгоритмів

Для оцінки ефективності розроблених алгоритмів розглянуто їхню часову складність. Результат наведено в таблиці 2.9.

Таблиця 2.9

Часова складність алгоритмів

Алгоритм	Складність	Пояснення
LogQueryPerformance	$O(\log n)$	Індексація на timestamp (B-дерево)
ExportToGCS	$O(m)$	Лінійно залежить від обсягу логів
AggregateMetricsForDash	$O(k)$	Лінійно залежить від кількості записів
AnalyzeQueryIntensity	$O(k)$	Лінійно залежить від кількості записів

Цей аналіз показує, що алгоритми є ефективними для роботи з великими обсягами даних: LogQueryPerformance і ExportToGCS використовують індексацію для зменшення складності, а AggregateMetricsForDash лінійно залежить від обсягу даних, що є прийнятним для агрегації.

Розроблено три алгоритми: LogQueryPerformance для логування метрик у PostgreSQL, ExportToGCS для експорту даних у GCS і AggregateMetricsForDash для підготовки даних до візуалізації. Кожен алгоритм враховує методи (індексація, кешування, масштабування) і технології (PostgreSQL, GCS, Dash). UML-діаграма послідовності для ExportToGCS (Рисунок 2.5) ілюструє взаємодію компонентів, а

приклади коду забезпечують практичну реалізацію. Ці алгоритми формують основу для ефективної роботи системи з великими обсягами даних.

2.5 Проєктні рішення та особливості прикладного рішення

Проєктні рішення, які були застосовані для реалізації прикладного рішення, а також особливості його функціонування. Забезпечення ефективного логування метрик продуктивності запитів до бази даних PostgreSQL та їхньої інтерактивної візуалізації в Dash. Розглянуто функціонал системи, обрані технології (індексація, Memorystore, GCS) та їхній вплив на продуктивність.

Архітектура системи

Архітектура прикладного рішення побудована на основі модульного підходу, де кожен компонент виконує чітко визначену функцію: обробка запитів, логування метрик, кешування, зберігання даних і візуалізація. Основні компоненти системи включають: PostgreSQL, Google Cloud Storage (GCS), Memorystore, Dash.

Взаємодія між компонентами проілюстрована на діаграмі компонентів (Рисунок 2.10).

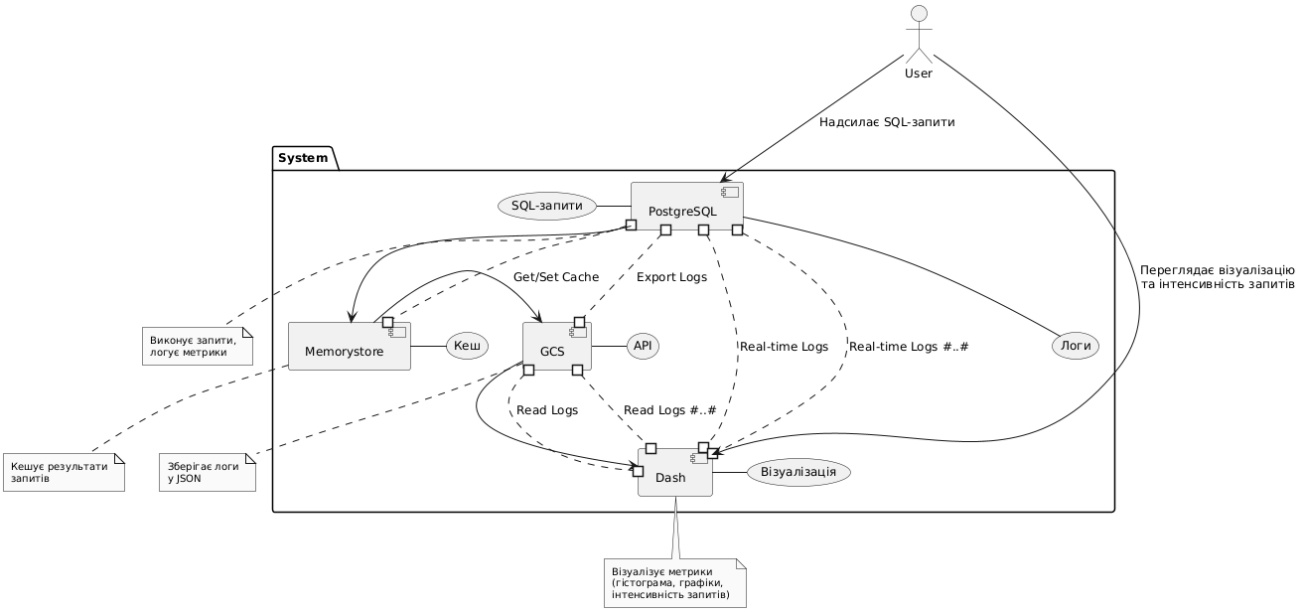

Рисунок 2.10 – Діаграма компонентів прикладного рішення

Ця архітектура забезпечує модульність і масштабованість системи: кожен компонент може бути замінений або оновлений без значних змін у інших частинах. Наприклад, заміна GCS на Amazon S3 потребуватиме лише зміни API-взаємодії в алгоритмі ExportToGCS.

Фунціонал прикладного рішення. Забезпечує два основні функціональні блоки: логування метрик продуктивності запитів і їхня візуалізація в реальному часі.

1) Логування метрик продуктивності.

Система фіксує метрики для кожного SQL-запиту, що виконується в PostgreSQL.

Основні метрики: T_i, N_i, T_{avg}.

Додаткові метрики: *query_text* (текст запиту), *timestamp* (час виконання).

Логування реалізовано через тригери в PostgreSQL, які викликають функцію LogQueryPerformance. Ця функція фіксує метрики після кожного запиту та записує їх у таблицю logs.

2) Візуалізація в Dash.

Для аналізу продуктивності метрики візуалізуються в Dash, що забезпечує інтерактивність і можливість аналізу в реальному часі. Основний елемент візуалізації — гістограма T_i , яка показує розподіл часу виконання запитів. Наприклад, гістограма може показати, що 70% запитів виконуються за 10–20 мс, а 5% мають затримки понад 50 мс через мережеві затримки.

Додатково Dash відображає залежність T_i від N_i, що дозволяє виявити запити, які потребують оптимізації. Ці дані агрегує алгоритм AggregateMetricsForDash, Візуалізація базується на даних, експортованих у GCS, що забезпечує масштабування при великих обсягах даних.

Проєктні рішення. Для функціоналу було обрано кілька ключових технологій і методів, які забезпечують ефективність і масштабованість системи. Основні проєктні рішення та їхній вплив на продуктивність узагальнено в таблиці 2.10.

Таблиця 2.10

Проєктні рішення та їхній вплив на продуктивність

Проєктне рішення	Опис	Функціонал	Ефект
Індексація	Створення індексів на стовпці	Логування (зменшення T_{DB})	Зменшення T_{DB} з 50 мс до 10 мс для запитів із фільтрацією
Memorystore	Кешування результатів часто виконуваних запитів	Логування (зменшення T_{GCS})	Зменшення T_{GCS} з 15 мс до 2 мс для кешованих запитів
Google Cloud Storage	Зберігання логів у JSON-файлах для масштабування і довготривалого доступу	Візуалізація (експорт даних для Dash)	Експорт 10,000 записів за 3 с, масштабування до 1 ТБ даних

1) Індексація в PostgreSQL.

Індексація застосована до таблиць із великими обсягами даних, що дозволяє прискорити виконання запитів із фільтрацією.

2) Кешування через Memorystore.

Memorystore (Redis) використовується для кешування результатів часто виконуваних запитів. Якщо запит повторюється, його результат береться з кешу, що зменшує.

3) Зберігання даних у GCS.

Google Cloud Storage (GCS) використовується для експорту логів із таблиці logs у JSON-файли щогодини (алгоритм ExportToGCS, Рисунок 2.4). Це забезпечує масштабування при великих обсягах даних [12].

Особливості прикладного рішення

1) Автоматизація логування:

Логування метрик відбувається автоматично завдяки тригерам у PostgreSQL. Це дозволяє фіксувати метрики без змін у коді клієнтських додатків, що підвищує гнучкість системи [22].

2) Масштабованість:

Використання GCS для зберігання логів забезпечує можливість обробки великих обсягів даних.Масштабованість: GCS забезпечує масштабованість для великих V [12].

3) Інтерактивність візуалізації:

Dash дозволяє користувачам аналізувати продуктивність у реальному часі. Наприклад, гістограма T_i оновлюється щогодини після експорту даних у GCS, що дає змогу швидко виявляти аномалії.

4) Оптимізація продуктивності:

Комбінація індексації та кешування через Memorystore дозволяє зменшити T_i на 60–80% для типових запитів. Це особливо важливо для систем із високим навантаженням, де кількість запитів може сягати 1,000 за секунду.

2.6 Аналіз ризиків і обмежень реалізації

Розробка прикладного рішення для оптимізації баз даних PostgreSQL і хмарних сховищ Google Cloud Storage (GCS) на платформі Google Cloud пов’язана з низкою ризиків і обмежень, які можуть вплинути на його ефективність і стабільність. Ці ризики стосуються технічних аспектів, таких як продуктивність компонентів системи, а також фінансових і безпекових питань. Для забезпечення надійності рішення необхідно врахувати ці фактори та розробити відповідні стратегії їх мінімізації. У таблиці нижче наведено аналіз основних ризиків, їхній потенційний вплив на систему та способи зменшення їхнього впливу.

Аналіз ризиків показує, що розробка прикладного рішення пов’язана з різними викликами. Мережеві збої, обмеження продуктивності (Memorystore, Dash, PostgreSQL), залежність від Google Cloud, недостатня тестуваність і якість даних можуть вплинути на ефективність системи, а помилки команди, затримки в розгортанні та етичні проблеми — на її впровадження і сприйняття. Запропоновані стратегії мінімізації, такі як локальний буфер, масштабування, шифрування, автоматизація тестування, валідація даних і модульна архітектура, дозволяють зменшити ці ризики, забезпечуючи стабільність і надійність прикладного рішення.

Таблиця 2.11

Аналіз ризиків і стратегії їх мінімізації

Ризик	Потенційний вплив	Стратегія мінімізації
Мережеві збої при роботі з GCS	Накопичення логів у PostgreSQL	Локальний буфер у PostgreSQL
Обмеження Memorystore	Зростання T_{GCS} для некешованих запитів	Масштабування до 5 ГБ, політика LRU
Обмеження Dash при великих даних	Затримки обробки (до 10 с), погіршення UX	Інкрементальне завантаження (2 с)
Витрати на інфраструктуру	Зростання витрат до $200/місяць	Холодне сховище (Coldline Storage, $40/місяць)
Обмеження PostgreSQL при навантаженні	Зростання T_{DB} через неоптимільні налаштування	Масштабування Cloud SQL (4 vCPU, 8 ГБ RAM)
Безпека даних	Ризик витоку даних	Шифрування (TLS 1.3), IAM-політики
Vendor lock-in	Складність міграції на іншу платформу, зростання витрат	Використання стандартизованих API, створення абстракцій для інтеграції.
Проблеми продуктивності при оновленнях	Несумісність, затримки в експорті логів	Регулярне тестування після оновлень (наприклад, через CI/CD), використання стабільних версій API (LTS для PostgreSQL).
Недостатня тестуваність системи	Помилки в реальних умовах (наприклад, збої в ExportToGCS)	Впровадження юніт-тестів (наприклад, для ExportToGCS), інтеграційних тестів для API-взаємодій, навантажувального тестування для Dash.
Перевантаження Dash при одночасному доступі	Затримки відгуку Dash, погіршення UX	Використання балансувальника навантаження (Google Cloud Load Balancer), кешування статичних графіків через CDN.

Розділ 3. РЕЗУЛЬТАТИ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір стеку технологій

Для реалізації платформи оптимізації та моніторингу великих даних було обрано сучасний стек технологій, який забезпечує оптимальний баланс між швидкістю розробки, продуктивністю, масштабованістю та функціональністю. Вибір компонентів стеку базувався на комплексному аналізі сучасних рішень з урахуванням особливостей обробки великих даних, вимог до продуктивності та архітектурних обмежень.

Основна мова програмування - Python 3.11

Python був обраний як основна мова розробки з наступних причин:

— Багатий екосистем бібліотек для аналізу даних та наукових обчислень, що дозволяє використовувати готові високооптимізовані рішення замість розробки власних алгоритмів. Це особливо важливо при роботі з великими даними, де ефективність алгоритмів є критичною.

— Висока продуктивність розробки завдяки простому та читабельному синтаксису, що дозволяє швидко прототипувати функціональність та ітеративно вдосконалювати код.

— Відмінна підмітрика роботи з базами даних та хмарними сервісами через наявність офіційних SDK та драйверів для більшості сучасних систем зберігання даних. Python підтримує як SQL, так і NoSQL бази даних, що дозволяє обирати оптимальне рішення для конкретних задач.

— Потужні можливості для обробки та аналізу великих обсягів даних через оптимізовані бібліотеки, такі як NumPy та Pandas.

— Кросплатформність, що забезпечує роботу на різних операційних системах без необхідності зміни коду.

Фреймворк для веб-інтерфейсу - **Streamlit**

Streamlit був обраний для створення інтерактивного веб-інтерфейсу з наступних причин:

— Можливість швидкої розробки інтерактивних веб-додатків без необхідності писати HTML/CSS/JavaScript.

— Вбудована підтримка реактивності та оновлення стану додатку.

— Багатий набір компонентів для відображення даних (таблиці, графіки, метрики).

— Проста інтеграція з бібліотеками візуалізації (Plotly, Matplotlib, Seaborn).

— Підтримка багатосторінкового інтерфейсу для структурування функціональності.

Бібліотеки для роботи з даними:

Pandas - для обробки та аналізу табличних даних, використовується для роботи з логами, результатами запитів та статистикою

NumPy - для науково-технічних обчислень та роботи з багатовимірними масивами даних

SQLAlchemy - як ORM для абстрагування взаємодії з базами даних та створення складних запитів

Psycopg2 - як драйвер для підключення до PostgreSQL баз даних

google-cloud-storage - для взаємодії з Google Cloud Storage API

Бібліотеки для візуалізації даних:

Plotly - для створення інтерактивних візуалізацій даних

Matplotlib - для генерації статичних графіків та діаграм

Seaborn - для статистичних візуалізацій з використанням спрощеного API

Бібліотеки для генерації звітів:

FPDF - для створення PDF-звітів

ReportLab - для розширених можливостей генерації PDF з графіками та форматуванням

Бази даних та сховища:

PostgreSQL - як основна цільова СУБД для аналізу та оптимізації

Google Cloud Storage - як цільове хмарне сховище для аналізу файлів

Обраний стек технологій забезпечує оптимальне поєднання продуктивності, функціональності та зручності розробки. Python як мова програмування надає широкі можливості для аналізу даних та інтеграції з різними системами, а Streamlit

забезпечує швидке створення інтерактивного веб-інтерфейсу без необхідності розробки фронтенду окремо від бекенду.

3.2 Загальна архітектура та структура програми

Платформа оптимізації та моніторингу великих даних побудована з використанням багаторівневої архітектури, що забезпечує модульність, розширюваність та легкість підтримки.

3.2.1 Архітектурні рівні в системи

Система складається з наступних архітектурних рівнів:

1. Рівень представлення (UI)

— Інтерактивний веб-інтерфейс, реалізований на базі Streamlit

— Карти метрик (metric cards), графіки, кругові діаграми, гістограми

— Форми для введення параметрів аналізу, фільтрації та налаштувань

— Адаптивний дизайн (автоматичне масштабування на різних екранах)

2. Рівень прикладної логіки (Business Logic Layer)

— Аналіз та оптимізація структури PostgreSQL (таблиці, індекси, запити)

— Обробка SQL-логів: виявлення повільних запитів, статистика, патерни

— Обробка даних із Google Cloud Storage: аналіз форматів, якість, структура

— Генерація інтерактивних звітів, графіків, порад з оптимізації

3. Рівень доступу до даних

— Підключення до баз PostgreSQL через бібліотеку psycopg2 або SQLAlchemy

— Підключення до хмарного сховища Google Cloud Storage через API

— Робота з локальними файлами (логами, CSV, JSON, Parquet) через pandas, pyarrow, os, pathlib

4. Зовнішні системи

— PostgreSQL — джерело структурованих даних (схеми, таблиці, запити)

— Google Cloud Storage — сховище для неструктурованих/структурованих файлів (CSV, JSON, Parquet)

— Файлова система — зберігання логів, кешу, звітів та результатів експорту

На рисунку 3.1 представлено загальну архітектуру системи, яка складається з чотирьох логічних рівнів: представлення, бізнес-логіки, доступу до даних та зовнішніх систем. Компоненти на кожному з рівнів взаємодіють між собою через чітко визначені інтерфейси.

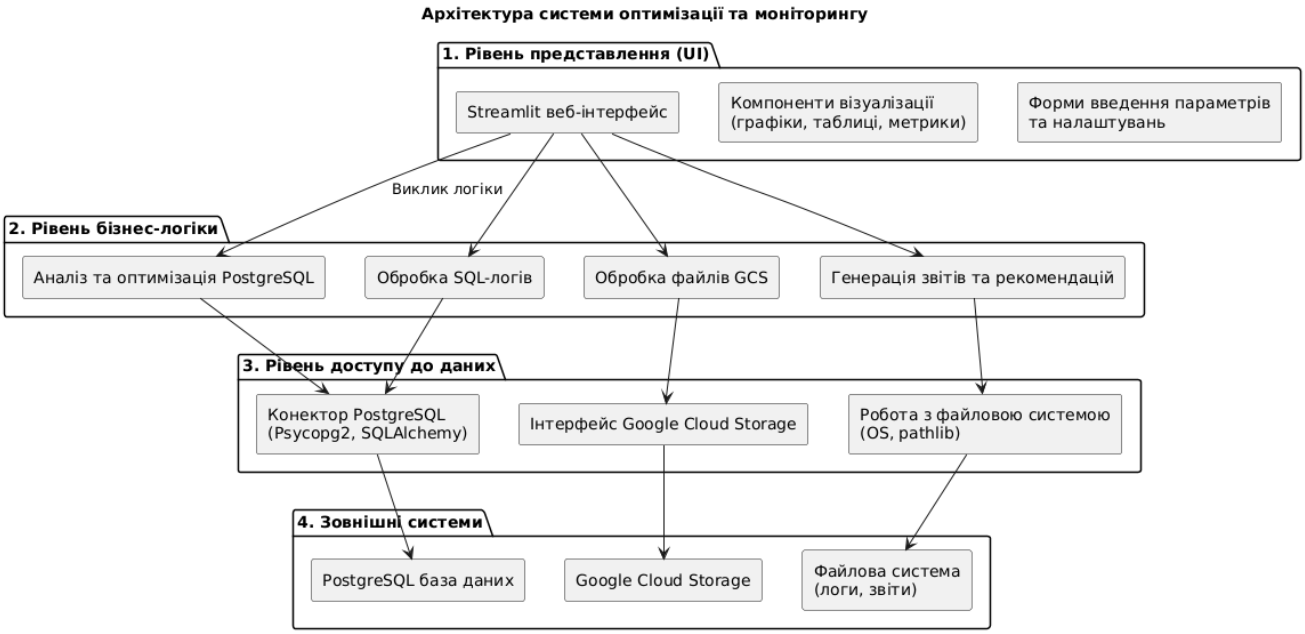

Рисунок 3.1 - Діаграма загальної архітектури системи

3.2.2 Структура проєкту

Структура файлів та каталогів наведена на рисунку 3.2.

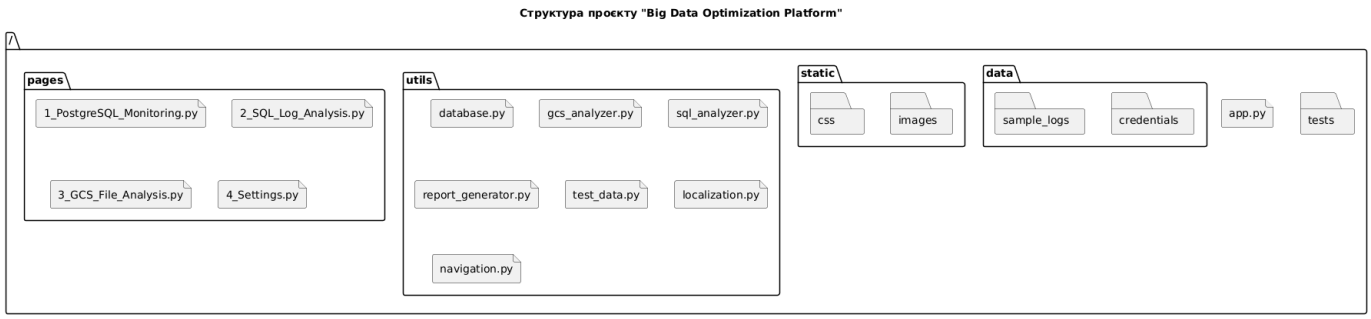

Рисунок 3.2 - Структура проєкту

3.2.3 Взаємодія компонентів

Взаємодія між основними компонентами платформи оптимізації та моніторингу великих даних реалізована відповідно до модульної структури

проєкту. Кожен модуль виконує чітко визначену роль у загальній архітектурі, що забезпечує гнучкість, масштабованість і легкість супроводу.

1. Головний модуль - **app.py**

Файл app.py виконує роль точки входу в систему. Саме в ньому ініціалізується фреймворк Streamlit, встановлюються глобальні параметри сесії, обробляються налаштування користувача та реалізується навігаційна структура між сторінками. Також цей модель відповідає за підключення до бази даних, завантаження конфігурацій і активацію протрібних режимів.

2. Функціольна сторінка.

Кожна сторінка реалізує окрему частину функціональності у вигляді Streamlit-модуля:

1_PostgreSQL_Monitoring.py — відображає метрики, статистику виконання запитів та загальний стан PostgreSQL-серверів у режимі реального часу.

2_SQL_Log_Analysis.py — здійснює парсинг, агрегацію та аналіз SQL-логів, формує рекомендації щодо покращення продуктивності запитів.

3_GCS_File_Analysis.py — виконує аналіз файлів, що зберігаються в Google Cloud Storage, з метою виявлення аномалій, дублювань або неефективного зберігання.

4_Settings.py — надає інтерфейс для налаштувань користувача, керування обліковими даними та перемикання режимів роботи.

3. Бізнес-логіка та допоміжні утиліти.

Усі ключові операції системи винесено в окремі утилітарні модулі:

database.py — реалізує клас DatabaseConnector, який забезпечує взаємодію з PostgreSQL за допомогою SQLAlchemy та psycopg2.

gcs_analyzer.py — містить клас GCSAnalyzer для обробки й аналізу вмісту хмарного сховища Google Cloud Storage.

sql_analyzer.py — реалізує SQLAnalyzer, який виконує розбір логів запитів і виявлення неефективностей.

report_generator.py — модуль ReportGenerator відповідає за створення PDF-звітів, таблиць та візуалізацій результатів аналізу.

test_data.py — модуль TestDataGenerator забезпечує генерацію фіктивних даних для тестування функціональності системи.

localization.py — містить функції для реалізації багатомовного інтерфейсу (наприклад, українська/англійська).

navigation.py — відповідає за маршрутизацію сторінок в Streamlit, дозволяє реалізувати багатосторінкову навігацію.

4. Сесія Streamlit.

Streamlit забезпечує механізм збереження стану між взаємодіями користувача за допомогою об’єкта st.session_state. У системі це використовується для:

- ✓ збереження стану підключення до бази даних і Google Cloud Storage;
- ✓ передачі проміжних результатів аналізу між сторінками;
- ✓ відображення рекомендацій і попередніх звітів;
- ✓ зберігання режиму роботи (тестовий або виробничий) та мовних налаштувань.

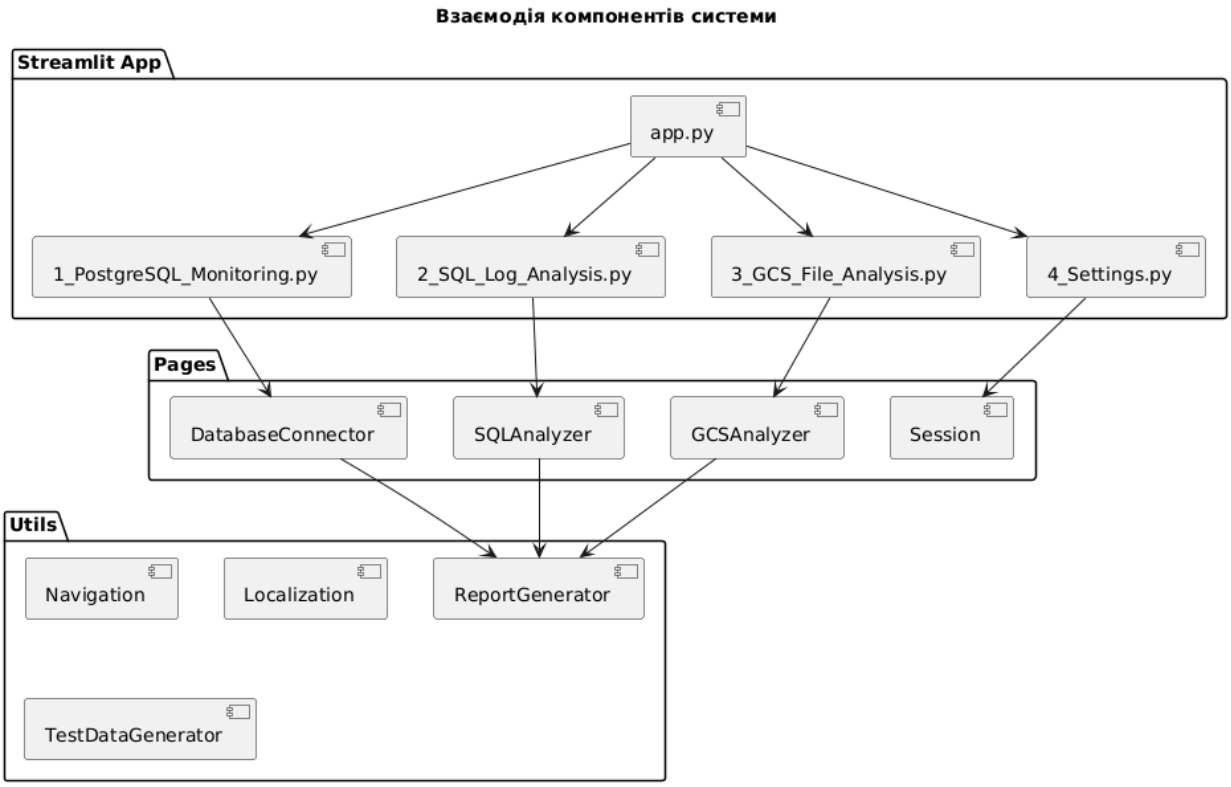

Рисунок 3.3 - Діаграма компонентів

3.2.4 Патерни проєктування

У процесі розробки було застосовано декілька ключових патернів проєктування, які дозволили забезпечити гнучкість, розширюваність та підтримуваність коду.

1. Singleton.

Патерн Singleton використовується для забезпечення існування лише одного екземпляра певного класу протягом усього часу роботи програми. У системі цей патерн застосовується, зокрема, для класу, що відповідає за підключення до бази даних PostgreSQL (DatabaseConnector). Завдяки цьому підключення ініціалізується лише один раз, що дозволяє уникнути дублювання з'єднань і економить ресурси.

Приклад застосування:

```
class DatabaseConnector:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
            # ініціалізація з'єднання з БД
        return cls._instance
```

2. Factory Method

Патерн Factory Method дозволяє створювати об'єкти різних типів, не прив’язуючись до конкретних класів, а керуючи процесом створення через спільний інтерфейс. У нашій системі цей патерн використовується для створення різних аналізаторів (SQLAnalyzer, GCSAnalyzer) та генераторів звітів (ReportGenerator), що полегшує додавання нових типів без зміни коду клієнта.

3. Strategy

Патерн Strategy забезпечує можливість вибору алгоритму або стратегії поведінки під час виконання програми. У платформі він використовується для реалізації різних стратегій аналізу та оптимізації залежно від типу даних, які надходять, або поставлених задач. Наприклад, різні методи аналізу SQL-логів або оптимізації запитів можуть бути реалізовані як окремі стратегії, які динамічно обираються відповідно до контексту.

4. Adapter

Патерн Adapter застосовується для узгодження інтерфейсів різних бібліотек або компонентів, що мають несумісні інтерфейси, але мають працювати разом. У системі цей патерн допомагає інтегрувати сторонні бібліотеки (наприклад, різні API для роботи з Google Cloud Storage або різні формати логів), забезпечуючи уніфікований інтерфейс для решти системи.

5. Observer

Патерн Observer використовується для реактивного оновлення інтерфейсу користувача у відповідь на зміну даних. У платформі Streamlit за допомогою цього патерну реалізується автоматичне оновлення компонентів UI (графіків, таблиць, метрик) при зміні стану програми — наприклад, після отримання результатів аналізу або зміни налаштувань користувача.

3.3. Розробка компонентів програмного забезпечення

3.3.1 Підготовка вхідних даних

У системі реалізовано кілька зручних і гнучких механізмів для збору та підготовки вхідних даних, необхідних для подальшого аналізу та оптимізації.

Підключення до PostgreSQL

Для взаємодії з базою даних PostgreSQL використовується спеціальний клас DatabaseConnector, розміщений у модулі utils/database.py. Цей компонент відповідає за:

1) *Підключення до бази даних* — залежно від обраного режиму система може:

- ✓ працювати у виробничому режимі, підключаючись до реальної бази даних за допомогою параметрів або змінних середовища;
- ✓ або використовувати тестовий режим, де замість справжньої бази створюються демонстраційні тестові дані.

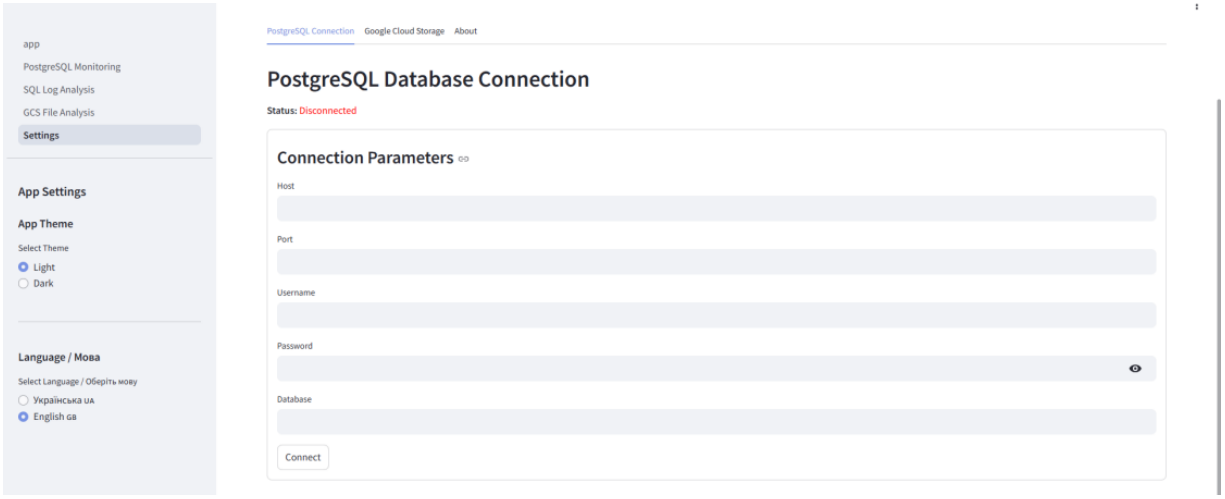

Рисунок 3.4 - Форма введеня параметрів для підключення до PostgreSQL

2) *Отримання статистики про БД* — за допомогою запитів до системних таблиць PostgreSQL збирається інформація про:

- ✓ загальний обсяг бази даних;
- ✓ кількість таблиць, індексів;
- ✓ активність таблиць та індексів;
- ✓ частоту повільних або проблемних запитів.

3) *Аналіз структури бази* — система отримує дані про:

- ✓ структуру таблиць (колонки, типи, обмеження);
- ✓ існуючі індекси та зв’язки;
- ✓ обсяги та динаміку зростання даних.

4) *Аналіз продуктивності запитів* — для цього використовується команда EXPLAIN ANALYZE, яка допомагає:

- ✓ з’ясувати, як виконується той чи інший SQL-запит;
- ✓ знайти «вузькі місця» у виконанні;
- ✓ сформувати рекомендації щодо оптимізації.

5) *Генерація рекомендацій* — на основі зібраної інформації система пропонує:

- ✓ створити нові індекси;
- ✓ переписати або змінити запити;
- ✓ реорганізувати таблиці;

- ✓ змінити налаштування бази для покращення продуктивності.

Обробка SQL-логів

Для аналізу історії виконання SQL-запитів у системі реалізовано клас SQLAnalyzer (модуль utils/sql_analyzer.py). Його основні функції:

1) *Парсинг логів* — обробка логів у форматах CSV та JSON з метою:

- ✓ структурування інформації про запити;
- ✓ фіксації часу виконання та результатів;
- ✓ аналізу джерел виконання запитів.

2) *Нормалізація SQL-запитів* — це важливий етап, що дозволяє:

- ✓ узагальнити подібні запити, замінюючи конкретні значення на плейсхолдери;
- ✓ вирівняти формат запитів, щоб зручніше їх аналізувати;
- ✓ групувати схожі запити для виявлення закономірностей.

3) *Статистичний аналіз* — включає:

- ✓ побудову розподілу за часом виконання;
- ✓ пошук запитів, що найчастіше виконуються або споживають найбільше ресурсів;
- ✓ виявлення аномалій у поведінці запитів.

4) *Аналіз патернів запитів* — дозволяє:

- ✓ побачити, які запити найчастіше зустрічаються;
- ✓ зрозуміти, як саме працюють користувачі з базою;
- ✓ знайти повторювані патерни, які можна оптимізувати.

5) *Формування рекомендацій* — система пропонує:

- ✓ оптимізувати запити, які споживають найбільше ресурсів;
- ✓ переглянути структуру часто використовуваних запитів;
- ✓ покращити доступ до таблиць шляхом індексації або реструктуризації.

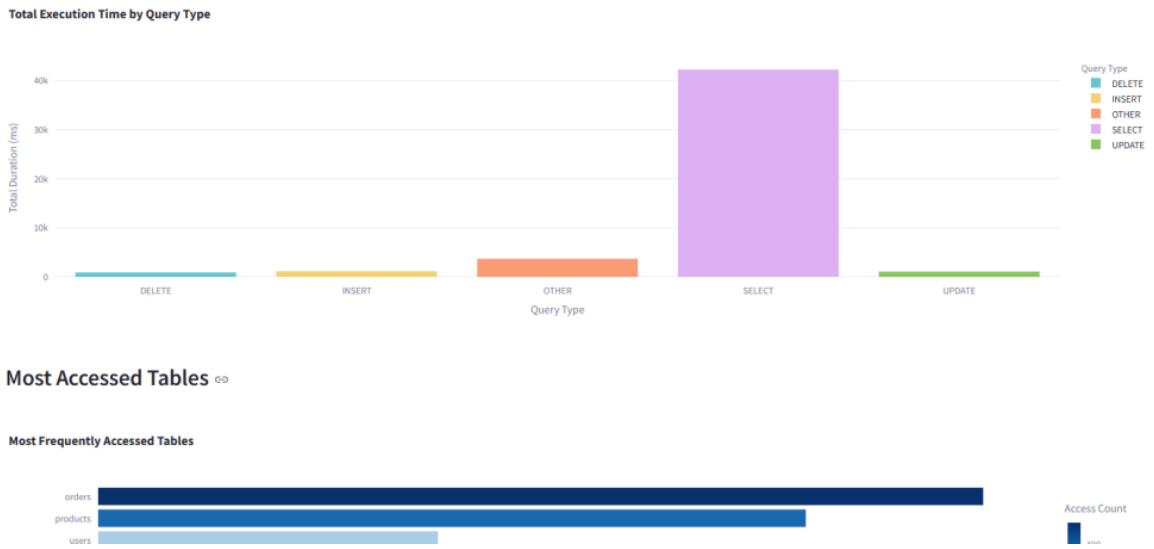

Рисунок 3.5 - Приклад результаів аналізу SQL-логів

Доступ до Google Cloud Storage (GCS)

Для роботи з хмарним сховищем Google Cloud Storage створено окремий клас GCSAnalyzer (модуль utils/gcs_analyzer.py). Основні можливості цього компонента:

1) *Підключення до GCS* — відбувається:

✓ через сервісний акаунт за допомогою JSON-файлу з ключем;

✓ або через облікові дані середовища — зручно для локального тестування;

✓ також підтримується демонстраційний (тестовий) режим без реального з'єднання.

2) *Навігація по сховищу* — система вміє:

✓ отримувати список доступних бакетів;

✓ переглядати файли в бакетах;

✓ фільтрувати файли за префіксами;

✓ читати метадані (розмір, тип, дата останньої зміни).

3) *Аналіз файлів* — підтримуються такі формати:

✓ CSV — аналіз розмітки, типів даних, частоти значень;

✓ JSON — аналіз вкладених структур, схеми;

✓ Parquet — аналіз ефективності стиснення, схеми, партицій.

4) *Порівняння форматів* — система дозволяє:

- ✓ порівняти обсяг одного й того самого набору даних у різних форматах;
- ✓ оцінити швидкість читання;
- ✓ підібрати найкращий формат для конкретної задачі.

5) *Рекомендації з оптимізації* — на основі аналізу формуються поради:

- ✓ який формат найкраще підходить для зберігання конкретних типів даних;
- ✓ як краще структурувати дані;
- ✓ чи є сенс у використанні стиснення або партицій.

3.3.2 Інтерфейс програми

Інтерфейс платформи розроблено за допомогою Streamlit — це фреймворк, який дозволяє швидко створювати прості та зручні веб-додатки на Python. Завдяки йому інтерфейс вийшов інтуїтивно зрозумілим, інтерактивним і зручним для користувача.

Головна сторінка (app.py)

Це стартова сторінка, з якої починається робота з додатком. Вона включає:

1) Налаштування вигляду та сесії:

- ✓ Назва, іконка та базові налаштування зовнішнього вигляду.
- ✓ Ініціалізація змінних сесії для збереження стану між переходами.

2) Бічна панель навігації:

- ✓ Логотип та назва додатку.
- ✓ Перемикач між тестовим і виробничим режимами.
- ✓ Індикатори статусу підключення до PostgreSQL та GCS.
- ✓ Кнопка для швидкого з’єднання з тестовою базою (зручно під час демонстрації).

3) Основна область контенту:

- ✓ Заголовок із відображенням поточного режиму.
- ✓ Панель швидкого переходу між функціями системи.
- ✓ Картки з ключовими показниками — розмір БД, кількість таблиць, повільні запити тощо.
- ✓ Огляд усіх функцій системи.

4) Додатково:

✓ Кнопка для перевірки всіх підключень.

✓ Блок рекомендацій, сформованих на основі аналізу.

✓ Нижній колонтитул з короткою інформацією про систему.

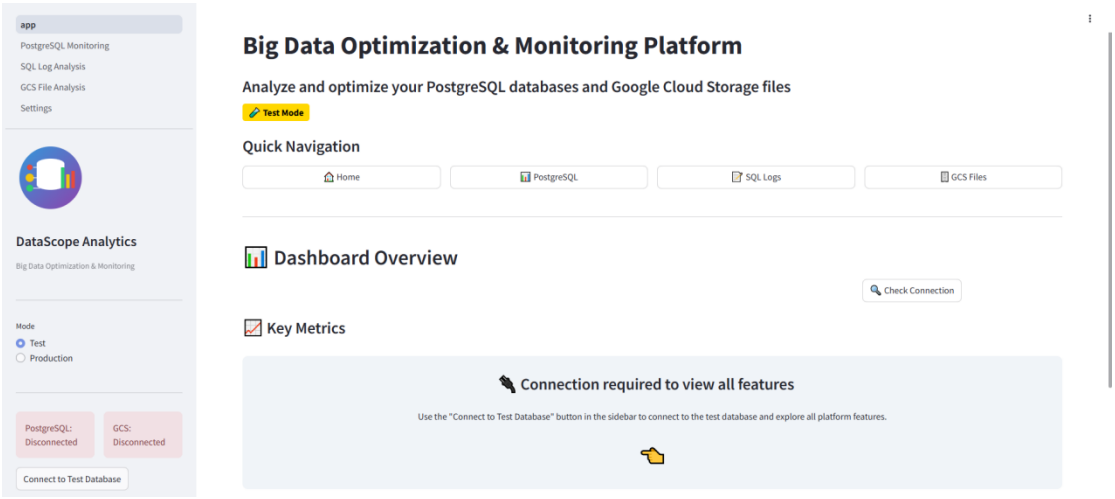

Рисунок 3.7 - Головна сторінка з панеллю керування

Сторінка моніторингу PostgreSQL

Ця сторінка дозволяє слідкувати за станом бази даних у режимі реального часу. Основні блоки:

1) Панель підключення:

✓ Форма для введення параметрів (хост, порт, логін, пароль, база).

✓ Кнопки для перевірки та встановлення з’єднання.

✓ Індикатор статусу підключення.

2) Загальна статистика по базі:

✓ Розмір бази, кількість таблиць, діаграми по використанню простору.

✓ Графіки росту бази з часом.

3) Аналіз таблиць та індексів:

✓ Таблиці з інформацією про розмір, частоту використання, наявність індексів.

✓ Виявлення невикористовуваних або відсутніх індексів.

✓ Рекомендації по оптимізації.

4) Повільні запити:

- ✓ Таблиця з найповільнішими запитами.
- ✓ Аналіз EXPLAIN PLAN.
- ✓ AI-поради щодо покращення.

5) Експорт:

- ✓ Кнопки для завантаження звітів (CSV, JSON, PDF).
- ✓ Можливість перегляду перед експортом.

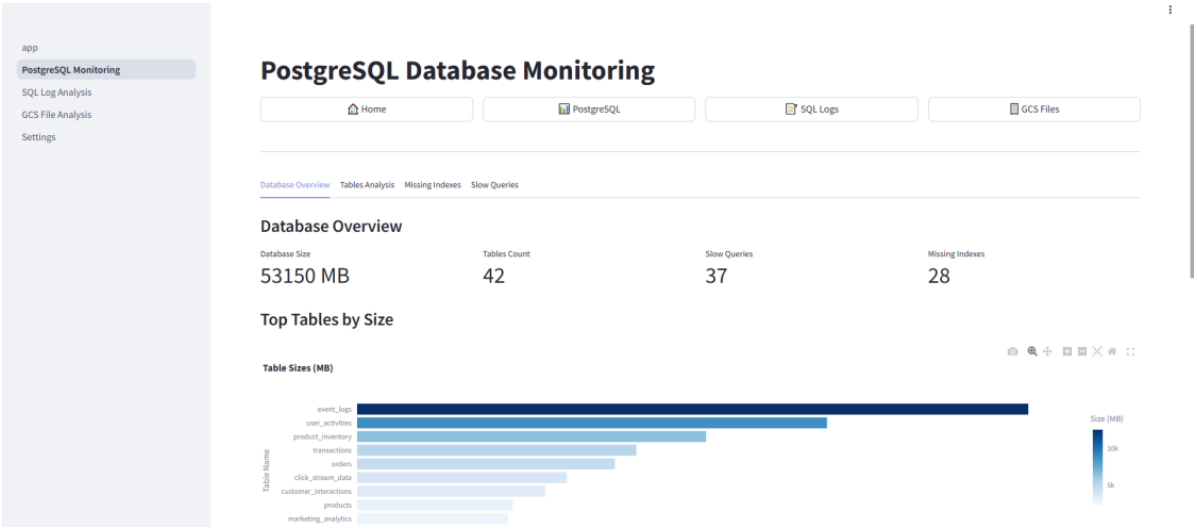

Рисунок 3.8 - Сторінка моніторингу PostgreSQL

Сторінка аналізу SQL-логів

Модуль дозволяє завантажити логи SQL-запитів і детально їх проаналізувати.

1) Завантаження логів:

- ✓ Підтримуються формати CSV і JSON.
- ✓ Вибір діапазону дат та фільтрів.
- ✓ Прогрес-бар обробки логів.

2) Загальна статистика:

- ✓ Скільки запитів, який середній/максимальний час виконання.
- ✓ Розподіл за типами запитів (SELECT, INSERT...).
- ✓ Розподіл по часу доби/днях тижня.

3) Аналіз часу виконання:

- ✓ Гістограми, графіки, виявлення пікових навантажень.
- ✓ Список найповільніших запитів.

4) Патерни запитів:

- ✓ Групування подібних запитів, виявлення шаблонів.
- ✓ Статистика по кожному шаблону.
- ✓ Поради щодо оптимізації типових запитів.

5) Експорт і рекомендації:

- ✓ Генерація автоматичних порад.
- ✓ Завантаження результатів у звітах.

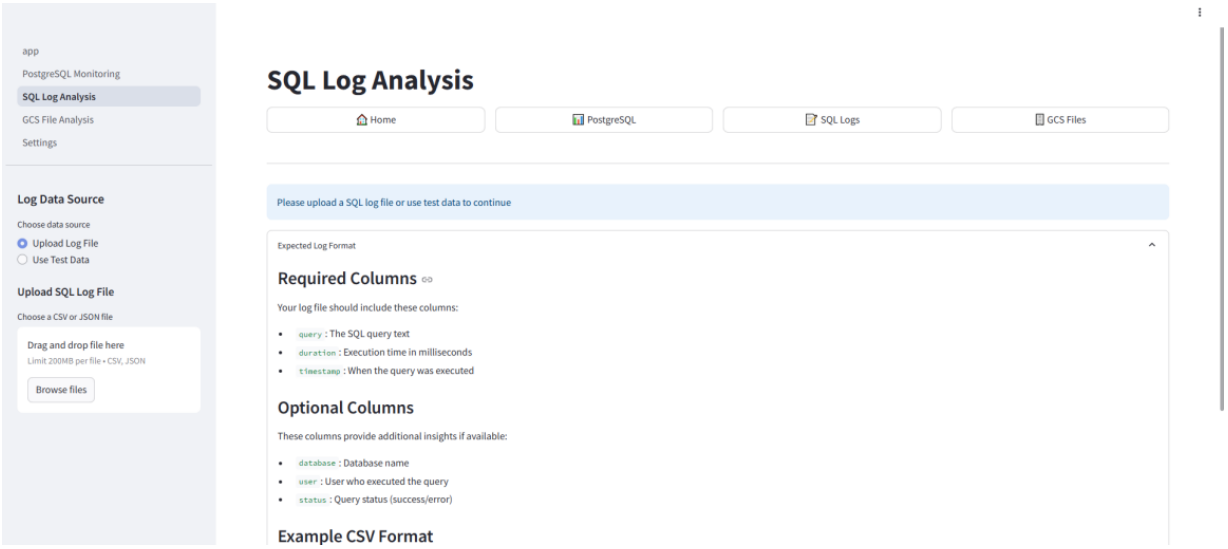

Рисунок 3.9 - Сторінка аналізу SQL-логів

Сторінка налаштувань

На цій сторінці зібрані всі конфігураційні параметри системи:

1) Загальні параметри:

- ✓ Перемикач між тестовим та виробничим режимами.
- ✓ Теми, розмір елементів, мова інтерфейсу.

2) Параметри PostgreSQL:

- ✓ Введення/редагування підключення до БД.
- ✓ Тестування та збереження налаштувань.

3) Параметри Google Cloud:

- ✓ Завантаження ключа, шляхи до файлів.
- ✓ Тестування доступу.

4) Налаштування звітів:

- ✓ Вибір формату за замовчуванням.
- ✓ Шаблони, параметри експорту.

5) Системна інформація:

- ✓ Версії компонентів.
- ✓ Статус підключень.
- ✓ Лог подій та помилок (зручно для відладки).

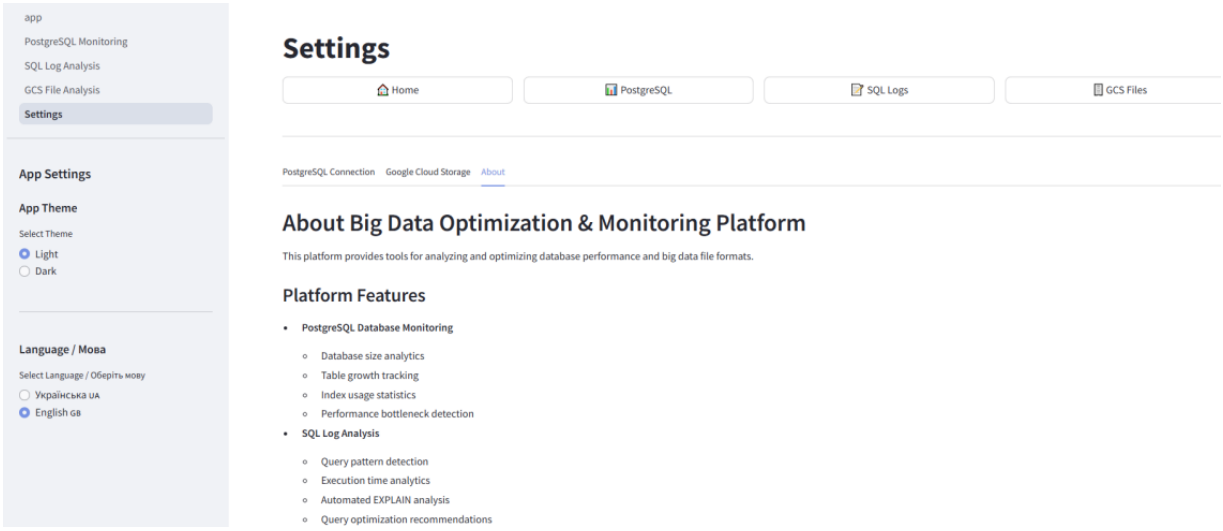

Рисунок 3.10 - Сторінка налаштувань

3.3.3 Опис розрахункової частини програми

Розрахункова частина програми включає реалізацію ключових алгоритмів для аналізу, оптимізації та прогнозування роботи з базою даних PostgreSQL, SQL-логами та файлами у Google Cloud Storage (GCS). Нижче наведено детальний опис основних алгоритмічних модулів системи.

Алгоритми аналізу PostgreSQL

1) Аналіз використання індексів:

- ✓ Збір статистичних даних про використання індексів із системних таблиць PostgreSQL (наприклад, pg_stat_user_indexes, pg_index).
- ✓ Виявлення індексів із низьким або відсутнім використанням, що можуть бути кандидатами на видалення.
- ✓ Ідентифікація потенційно корисних індексів, яких наразі немає, шляхом аналізу умов у WHERE, JOIN, ORDER BY.

2) Аналіз повільних запитів:

- ✓ Збір метрик з розширення pg_stat_statements про час виконання SQL-запитів.
- ✓ Виявлення запитів, які перевищують заданий поріг часу виконання, для подальшого аналізу.
- ✓ Аналіз планів виконання цих запитів для виявлення вузьких місць (сканування таблиць, відсутність індексів, великі з'єднання).

3) Алгоритм виявлення відсутніх індексів:

- ✓ Аналіз планів виконання запитів з метою пошуку послідовних сканувань (Sequential Scan) на великих таблицях.
- ✓ Виявлення частих умов фільтрації (WHERE), які не підтримуються індексами.
- ✓ Автоматичне формування рекомендацій щодо створення нових індексів для підвищення продуктивності.

4) Аналіз структури таблиць:

- ✓ Збір даних про розмір таблиць, їх зростання у часі та заповнення (наприклад, порожні або надмірно фрагментовані сторінки).
- ✓ Виявлення таблиць, що потребують реструктуризації (наприклад, реіндексації, вакуумізації або партиціонування).
- ✓ Рекомендації щодо оптимізації фізичної структури таблиць.

Алгоритми аналізу SQL-логів

1) *Нормалізація SQL-запитів:*

- ✓ Заміна літеральних значень у запитах на плейсхолдери для уніфікації.
- ✓ Усунення надлишкових пробілів, форматування запитів для кращого порівняння.
- ✓ Опціональна нормалізація імен таблиць і колонок (наприклад, приведення до нижнього регістру).

2) *Кластеризація запитів за патернами:*

- ✓ Групування нормалізованих запитів за схожістю структури (патернами).

- ✓ Обчислення статистичних показників для кожного патерну: кількість викликів, середній і максимальний час виконання.
- ✓ Виявлення найбільш ресурсоємних груп запитів.

3) *Аналіз часових закономірностей:*

- ✓ Аналіз розподілу запитів за часом доби, дня тижня для виявлення пікових навантажень.
- ✓ Кореляція типів запитів (SELECT, INSERT, UPDATE, DELETE) із часовими патернами.

4) *Генерація рекомендацій з оптимізації:*

- ✓ Аналіз найбільш ресурсоємних патернів запитів для виявлення потенційних проблем (відсутність індексів, надмірні з'єднання).
- ✓ Формування детальних рекомендацій щодо оптимізації SQL-коду і структури бази.

Алгоритми аналізу файлів Google Cloud Storage (GCS)

1) Аналіз структури CSV-файлів:

- ✓ Визначення схеми даних: перелік колонок, типи даних (рядок, число, дата).
- ✓ Статистичний аналіз: кількість рядків, розподіл значень по колонках.
- ✓ Виявлення аномалій, пропущених або некоректних значень.

2) *Аналіз структури JSON-файлів:*

- ✓ Визначення схеми та ієрархії даних, включаючи вкладені об'єкти і масиви.
- ✓ Оцінка складності структури та пошук типових патернів.

3) *Аналіз структури Parquet-файлів:*

- ✓ Збір метаданих про схему даних, партиціонування, компресію.
- ✓ Оцінка ефективності зберігання і доступу до даних.
- ✓ Розрахунок оптимальних розмірів груп рядків (row groups) для продуктивності.

4) Порівняльний аналіз форматів:

- ✓ Оцінка розміру даних у різних форматах (CSV, JSON, Parquet).

- ✓ Аналіз швидкості доступу та обробки інформації для кожного формату.
- ✓ Прогнозування оптимального формату з урахуванням специфіки даних та сценаріїв використання.

Прогнозування оптимального формату зберігання даних

- ✓ Збір характеристик файлів: обсяг, типи даних, частота доступу, час обробки.
- ✓ Моделювання продуктивності роботи з файлами в різних форматах (наприклад, Parquet краще підходить для аналітичних запитів, CSV – для простих імпортів/експортів).
- ✓ Рекомендації щодо вибору формату, що забезпечує баланс між розміром, швидкістю доступу та зручністю обробки.

3.3.4 Опис графічної частини програми

Графічна частина програми реалізована з використанням можливостей Streamlit та інтегрованих бібліотек візуалізації — Plotly, Matplotlib, Seaborn. У системі реалізовані різноманітні типи візуалізацій, які допомагають у моніторингу, аналізі та оптимізації роботи з PostgreSQL, SQL-логами та файлами Google Cloud Storage.

Метрики та індикатори

Для відображення ключових показників використовуються картки метрик — компактні візуальні елементи, що демонструють значення, зміну та тренди основних метрик. Для їх реалізації застосовано компонент st.metric у Streamlit із кольоровим кодуванням: зелений колір позначає позитивні зміни, червоний — негативні.

Індикатори стану використовуються для візуального відображення статусу підключень до PostgreSQL та Google Cloud Storage. Для цього застосовуються компоненти st.success та st.error Streamlit, а також іконки та кольорове кодування (зелений для успішних підключень, червоний для помилок). При наведенні курсору на індикатор з'являються інтерактивні підказки з детальною інформацією про статус підключення.

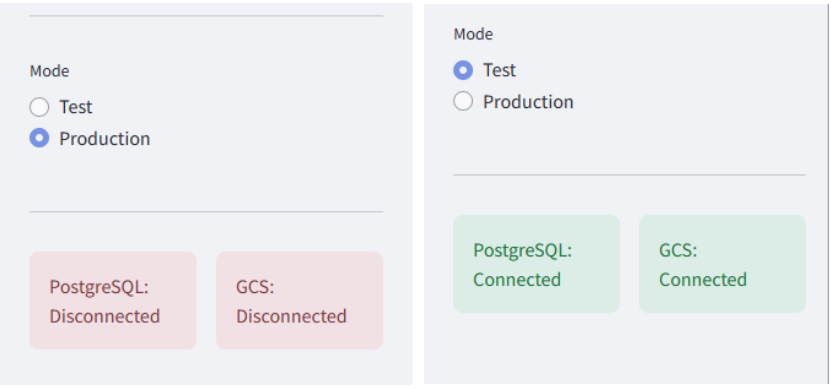

Рисунок 3.11 - Індикатори стану підключення до БД

Графіки та діаграми для аналізу PostgreSQL

Для аналізу PostgreSQL реалізовано кілька типів візуалізацій. Кругові діаграми показують розподіл розміру таблиць із можливістю фільтрації та сортування, а також з інтерактивними підказками.

Часові графіки відображають динаміку зростання розміру бази даних, дозволяють масштабувати дані та відзначають важливі події, такі як оптимізації чи очищення. Теплові карти активності запитів візуалізують інтенсивність запитів за часом доби та днями тижня, що дозволяє виявляти патерни використання та планувати оптимізації на періоди меншого навантаження.

Особливу увагу приділено візуалізації планів виконання запитів, які представлені у вигляді деревоподібних структур із кольоровим кодуванням часу виконання кожного вузла. Це дозволяє швидко виявляти найбільш ресурсоємні операції в запитах та фокусуватись на їх оптимізації.

Рисунок 3.12 - Аналіз продуктивності SQL-запиту з відображенням метрик виконання та деталізацією плану запиту

Графіки та діаграми для аналізу SQL-логів

Система реалізує гістограми розподілу часу виконання запитів, з позначенням аномалій та викидів і можливістю фільтрації.

Часові графіки допомагають відслідковувати тренди та виявляти аномалії, а також дозволяють накладати різні метрики для порівняння. Стовпчикові діаграми частоти патернів запитів візуалізують найбільш поширені типи запитів, їх можна сортувати за різними критеріями (частота, час, ресурсоємність) та отримувати детальну інформацію через інтерактивні підказки.

Графіки та діаграми для аналізу файлів GCS

Для роботи з файлами в Google Cloud Storage передбачена візуалізація структури даних у різних форматах. Відображається схема даних, ієрархія та зв'язки, особливо для JSON, із інтерактивними можливостями дослідження.

Порівняльні діаграми (стовпчикові або радарні) демонструють різницю між форматами зберігання (CSV, JSON, Parquet) за різними метриками, такими як розмір файлу, швидкість зчитування та запису, підтримка складних даних тощо. Система наочно показує, що конвертація CSV у Parquet може заощадити до 70% дискового простору.

Для аналізу розподілу значень у даних використовуються гістограми та ящикові діаграми для числових даних, а також кругові діаграми для категоріальних даних. Це дозволяє ефективно виявляти аномалії та викиди в даних, які можуть вказувати на проблеми з якістю інформації.

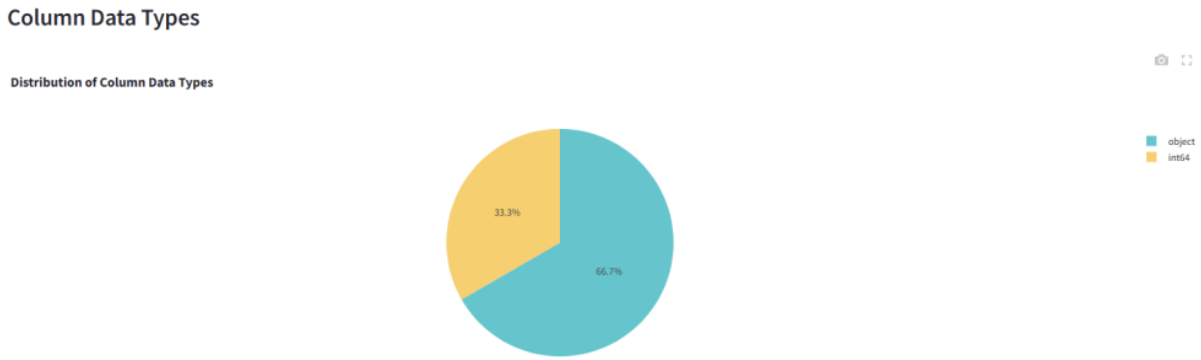

Рисунок 3.13 - Приклад кругової діаграми для категоріальних даних

3.4 Контрольний приклад

Для демонстрації роботи системи використаємо тестову вибірку даних, яка доступна в тестовому режимі роботи платформи. Тестовий режим активується через перемикач "Test Mode" на головній сторінці і дозволяє продемонструвати всі можливості системи без необхідності підключення до реальних джерел даних.

3.4.1 Загальний огляд системи

Після запуску у тестовому режимі на головній сторінці відображається панель керування з ключовими метриками:

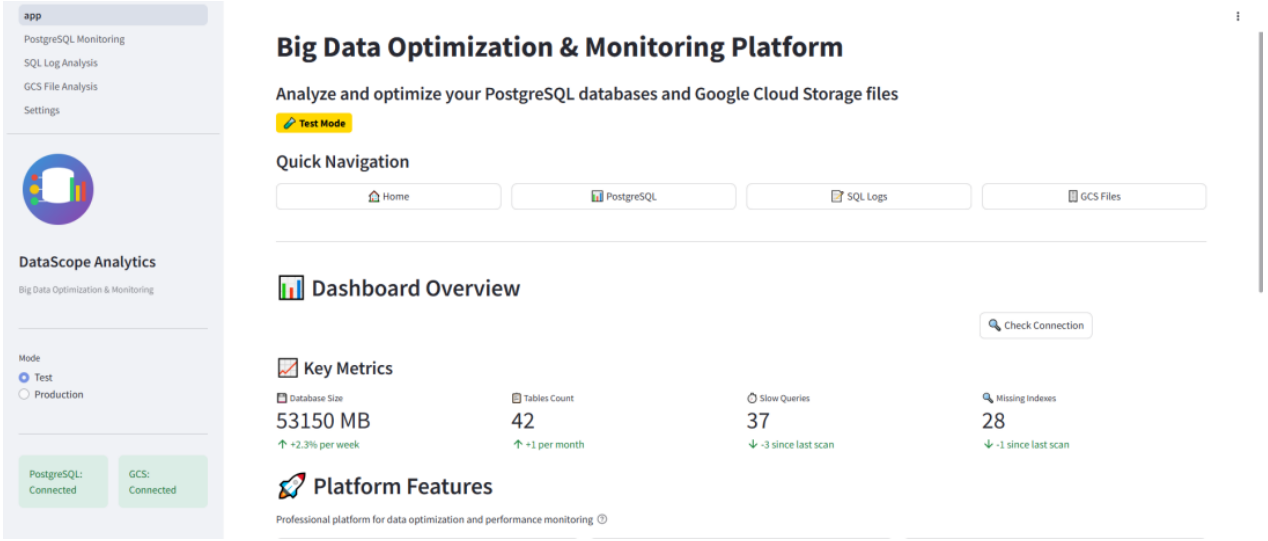

Рисунок 3.14 - Головна сторінка платформи з увімкненим перемикачем "Test Mode"

Ключові метрики (Key Metrics):

— Database Size (Розмір бази даних): 53150 MB з трендом зростання +2.3% за тиждень

— Tables Count (Кількість таблиць): 42 з трендом зростання +1 за місяць

— Slow Queries (Повільні запити): 37 з трендом зниження -3 з часу останнього сканування

— Missing Indexes (Відсутні індекси): 28 з трендом зниження -1 з часу останнього сканування

Швидка навігація (Quick Navigation):

— Home (Головна сторінка)

— PostgreSQL (Моніторинг PostgreSQL)

— SQL Logs (Аналіз SQL-логів)

— GCS Files (Аналіз файлів GCS)

Функції платформи (Platform Features):

— PostgreSQL Monitoring (Моніторинг PostgreSQL)

— SQL Log Analysis (Аналіз SQL-логів)

— GCS File Analysis (Аналіз файлів GCS)

Система також відображає статус підключення до джерел даних:

— PostgreSQL: Connected (Підключено)

— GCS: Connected (Підключено)

3.4.2 Демонстрація моніторингу PostgreSQL

Для перевірки функціональності моніторингу PostgreSQL користувач переходить на сторінку "PostgreSQL Monitoring", натиснувши відповідний пункт у боковому меню. Ця сторінка пропонує комплексний огляд стану бази даних з розбивкою на декілька вкладок: "Database Overview", "Tables Analysis", "Missing Indexes", "Slow Queries".

3.4.2.1 Огляд бази даних (Database Overview)

На вкладці "Database Overview" користувач бачить загальну інформацію про базу даних та її структуру (рис. 3.14):

Загальні метрики бази даних:

- ✓ Розмір: 53.15 ГБ
- ✓ Кількість таблиць: 42
- ✓ Кількість схем: 5 (public, analytics, history, metadata, system)

Розподіл за схемами представлено у вигляді кругової діаграми:

- ✓ public — 28 таблиць (67%)
- ✓ analytics — 8 таблиць (19%)
- ✓ history — 4 таблиці (9%)
- ✓ metadata — 2 таблиці (5%)

Візуалізація розподілу розміру подана у вигляді гістограми, що відображає розмір найбільших таблиць. Користувач може побачити, що найбільшими за розміром є таблиці: event_logs, user_activities, product_inventory, transactions, orders.

Ця візуалізація дозволяє швидко ідентифікувати таблиці, які займають найбільше місця і можуть потребувати оптимізації.

Взаємодія з інтерфейсом:

- ✓ Користувач може навести курсор на діаграму, щоб побачити точні значення для кожної таблиці
- ✓ Кнопка "Export as CSV" дозволяє експортувати дані у форматі CSV
- ✓ Кнопка "Export as JSON" експортує структуровані дані у форматі JSON
- ✓ Кнопка "Export as PDF" генерує PDF-звіт з інформацією про базу даних

3.4.3.2 Аналіз таблиць (Tables Analysis)

Перейшовши на вкладку "Tables Analysis", користувач отримує детальну інформацію про таблиці бази даних на рисунку 3.15 . Ця вкладка містить наступні елементи:

Зведена статистика таблиць:

- ✓ Загальна кількість таблиць: 20
- ✓ Загальна кількість рядків: 587,563,800
- ✓ Загальний розмір: 64,030.00 МБ

Таблиці без індексів представлені у вигляді таблиці з такими колонками:

- ✓ Table Name (Назва таблиці)
- ✓ Row Count (Кількість рядків)
- ✓ Size (MB) (Розмір в МБ)
- ✓ Indexes (Кількість індексів)

Система виявляє три проблемні таблиці без індексів:

- ✓ customer_interactions (28,000,000 рядків, 3,500.00 МБ)
- ✓ user_sessions (16,500,000 рядків, 1,450.00 МБ)
- ✓ ad_network_logs (41,000,000 рядків, 2,250.00 МБ)

Ця інформація має особливу цінність, оскільки великі таблиці без індексів можуть спричиняти значні проблеми з продуктивністю.

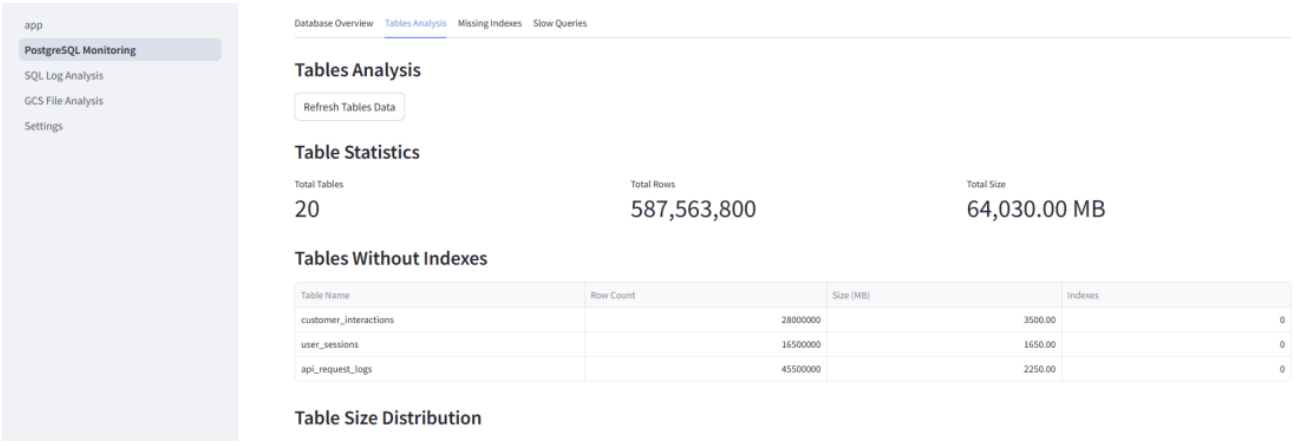

Рисунок 3.15 - Tables Analysis

Візуалізація розподілу розміру таблиць представлена у вигляді деревовидної карти (treemap), де розмір кожного прямокутника пропорційний розміру таблиці. Колірна схема від світло-блакитного до темно-синього вказує на відносний розмір таблиць (рисунок 3.16).

Список усіх таблиць представлений у вигляді таблиці з можливістю пошуку. Для кожної таблиці вказано:

- ✓ Назва таблиці
- ✓ Кількість рядків
- ✓ Розмір у МБ
- ✓ Кількість індексів

Користувач може використовувати поле пошуку "Search tables" для фільтрації таблиць за назвою.

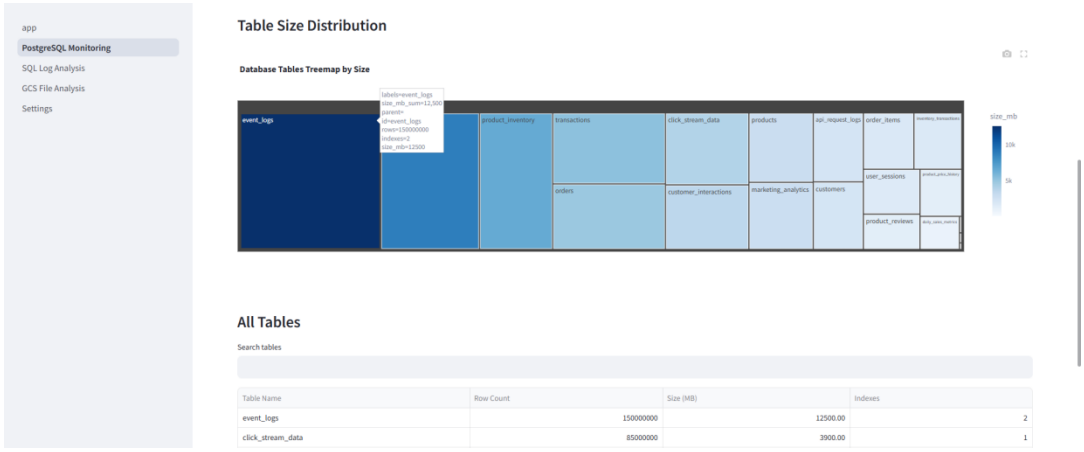

Рисунок 3.16 - Візуалізація розподілу розміру таблиць

Взаємодія з інтерфейсом:

- ✓ Кнопка "Refresh Tables Data" дозволяє оновити дані про таблиці
- ✓ Користувач може сортувати таблицю, натискаючи на заголовки колонок
- ✓ Пошукове поле дозволяє швидко знаходити потрібні таблиці
- ✓ Hover-ефекти на деревовидній карті показують точну інформацію про таблицю.

3.4.3.3 Аналіз відсутніх індексів (Missing Indexes)

Вкладка "Missing Indexes" присвячена аналізу потенційно корисних індексів, яких не вистачає у базі даних:

Візуалізація рекомендацій за впливом:

Горизонтальна гістограма "Index Recommendations by Impact" показує таблиці, для яких рекомендовано створити індекси, відсортовані за потенційним впливом на продуктивність. Колірне кодування від світло-оранжевого до темно-червоного показує рівень важливості створення індексу.

Найбільший вплив мають індекси для таблиць: event_logs, click_stream_data, user_activities, transactions, product_inventory

Взаємодія з інтерфейсом:

Рисунок 3.17 - Аналіз відсутніх індексів

Кнопка "Analyze Missing Indexes" запускає сканування бази даних для виявлення потенційно корисних індексів. Користувач може розгорнути кожну рекомендацію, натиснувши на стрілку справа, щоб отримати детальну інформацію Для кожної рекомендації доступна опція копіювання SQL-запиту для створення індексу

3.4.3.4 Аналіз повільних запитів (Slow Queries)

На вкладці "Slow Queries" користувач може проаналізувати запити, що виконуються повільно і потенційно спричиняють проблеми з продуктивністю:

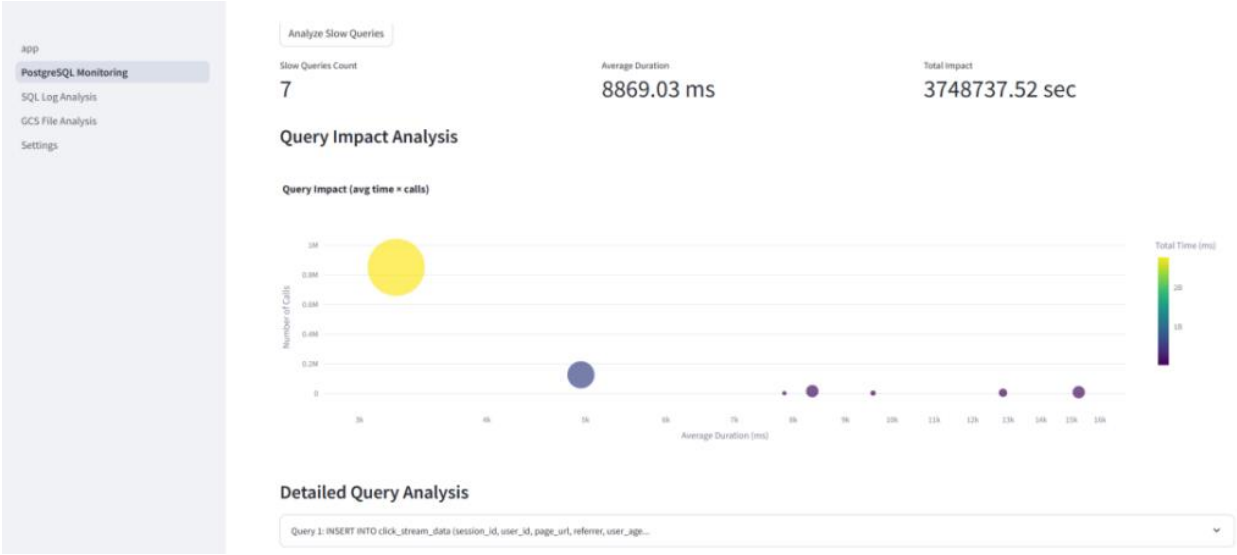

Рисунок 3.18 - Аналіз повільних запитів

Візуалізація впливу запитів:

Бульбашкова діаграма "Query Impact" показує взаємозв'язок між середньою тривалістю запиту (вісь X), кількістю викликів (вісь Y) та загальним впливом (розмір бульбашки). Колірне кодування від фіолетового до жовтого вказує на загальний час виконання.

Ця візуалізація дозволяє швидко ідентифікувати:

- ✓ Запити з високим впливом (великі бульбашки)
- ✓ Повільні, але рідко виконувані запити (праворуч внизу)
- ✓ Швидкі, але часто виконувані запити (ліворуч вгорі)

Детальний аналіз запитів:

Розділ "Detailed Query Analysis" (додаток) містить повний список повільних запитів з можливістю розгорнути кожен запит для детального аналізу. Для кожного запиту показано:

- ✓ Повний текст SQL-запиту
- ✓ Тип операції (SELECT, INSERT, UPDATE, DELETE)
- ✓ Середню тривалість виконання
- ✓ Кількість викликів
- ✓ Загальний час виконання
- ✓ Рекомендації щодо оптимізації
- ✓ Взаємодія з інтерфейсом:

Кнопка "Analyze Slow Queries" запускає аналіз логів бази даних для виявлення повільних запитів. Користувач може розгорнути кожен запит, натиснувши на стрілку справа. Кнопка "Generate Report" дозволяє створити повний звіт з аналізом повільних запитів. Для кожного запиту доступна опція "Optimize Query", яка пропонує конкретні рекомендації з оптимізації

3.4.4 Тестування аналізу SQL-логів

Для дослідження функціональності аналізу SQL-логів користувач переходить на сторінку "SQL Log Analysis", натиснувши відповідний пункт у боковому меню. Ця сторінка пропонує інструменти для аналізу журналів SQL-запитів та виявлення патернів і проблем.

3.4.4.1 Вибір джерела даних

Перш за все, користувач має вибрати джерело даних для аналізу. У боковій панелі доступні такі опції:

- ✓ Upload Log File - завантаження файлу логів з комп'ютера
- ✓ Use Test Data - використання тестового набору даних
- ✓ Using generated test data for demonstration - генерація тестових даних для демонстрації

Для тестування вибираємо опцію "Using generated test data for demonstration", після чого система автоматично генерує набір тестових логів SQL-запитів.

3.4.4.2 Зведена інформація про логи

Після вибору джерела даних, система відображає зведену інформацію у розділі "Log Data Summary":

- ✓ Загальна кількість запитів: 1,000
- ✓ Середня тривалість: 52.71 мс
- ✓ Діапазон даних: 7 днів
- ✓ Кількість помилок: 44

3.4.4.3 Аналіз частоти та тривалості запитів

На вкладці "Query Frequency Analysis" користувач може проаналізувати розподіл запитів за тривалістю виконання.

Гістограма розподілу тривалості запитів:

Гістограма "Query Duration Distribution (95th percentile)" показує розподіл кількості запитів за тривалістю виконання. Вісь X представляє тривалість у мілісекундах, вісь Y - кількість запитів. Гістограма обмежена 95-им перцентилем для кращої візуалізації, оскільки невелика кількість дуже повільних запитів може спотворити масштаб.

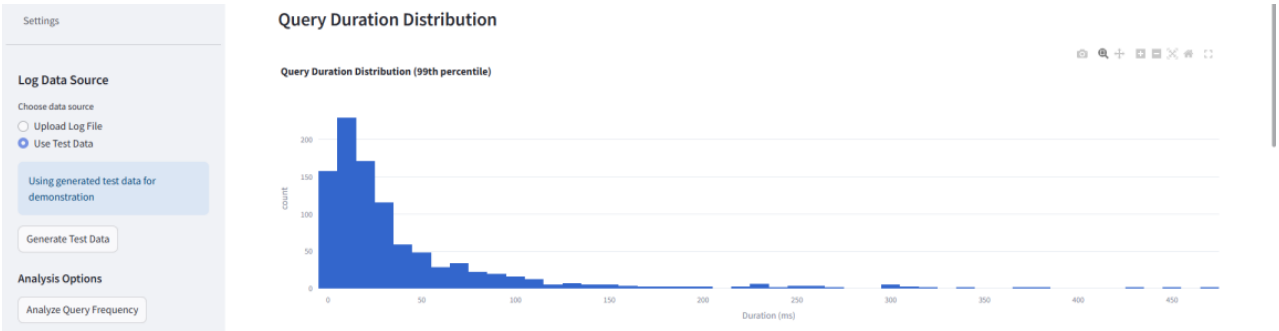

Рисунок 3.19 - Гістограма розподілу тривалості запитів

На гістограмі видно, що більшість запитів виконуються швидко (менше 50 мс), але є "хвіст" повільніших запитів, що потребують оптимізації.

3.4.4.4 Аналіз типів запитів

У розділі Query Pattern Analysis представлено розподіл SQL-запитів за типами.

Кругова діаграма розподілу типів запитів

На рисунку 3.20 представлено діаграму "Distribution of Query Types", що демонструє відсоткове співвідношення типів SQL-запитів:

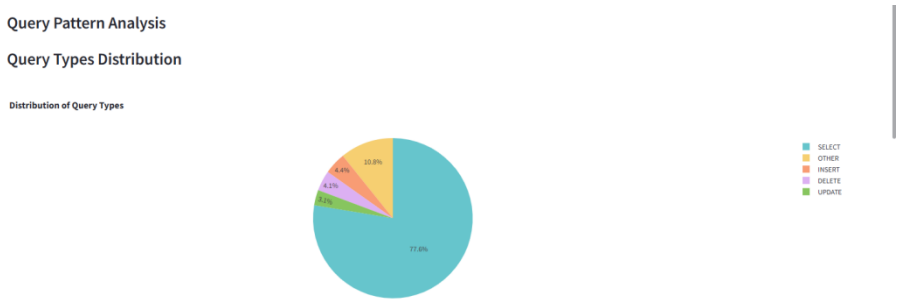

Рисунок 3.20 - Діаграма типів запитів

На рисунку 3.21 зображено діаграму "Total Execution Time by Query Type", що показує сумарний час виконання для кожного типу запитів.

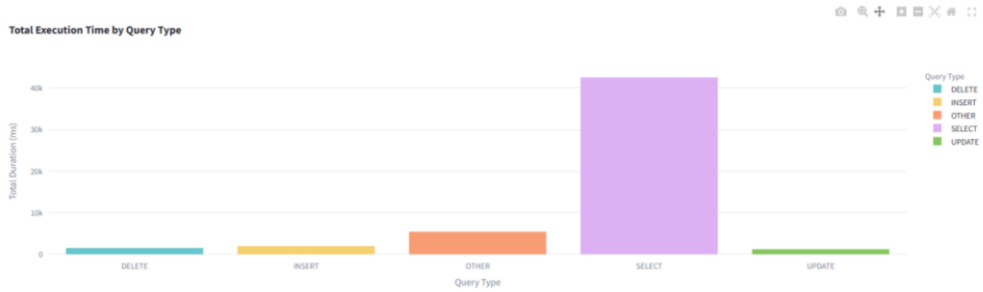

Рисунок 3.21 - Total Execution Time by Query Type

Попри кількісну перевагу **SELECT**, саме ці запити споживають найбільше ресурсів через великі об’єми даних та складність.

3.4.4.5 Аналіз патернів запитів

У підрозділі Top Query Patterns проаналізовано найчастіше виконувані SQL-патерни. **Гістограма патернів SQL-запитів.** На рисунку 3.22 представлено горизонтальну гістограму "Top 10 Query Patterns by Count" із кількістю виконань та середнім часом виконання (колір від зеленого до фіолетового).

Найпоширеніші запити:

- ✓ SELECT COUNT(*) FROM orders WHERE created_at BETWEEN ?
- ✓ SELECT p.category_name FROM users u JOIN orders o ...
- ✓ SELECT p.name, p.price FROM products p WHERE p.id=?
- ✓ SELECT * FROM users WHERE id=?
- ✓ INSERT INTO log_entries(user_id, action, timestamp)

✓ UPDATE users SET last_visit=? WHERE id=?

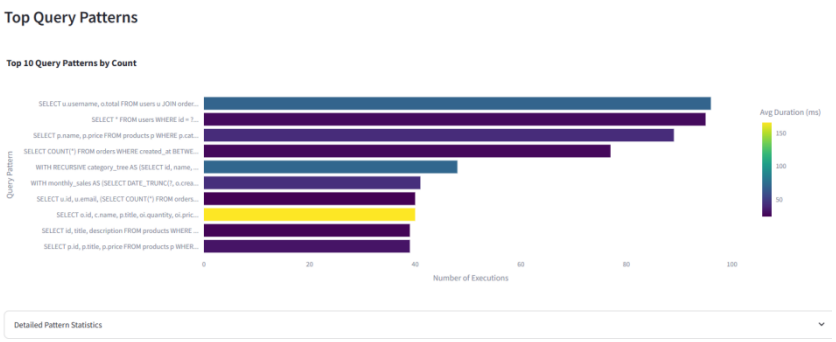

Рисунок 3.22 - Гістограма патернів SQL-запитів

Цей аналіз дозволяє виявити не тільки популярні шаблони, але й оцінити їх продуктивність.

3.4.4.6 Аналіз активності за часом

На рисунку 3.23 зображено графік "Query Activity by Hour", що показує кількість запитів у кожну годину доби.

Рисунок 3.23 - Активність SQL-запитів протягом доби

Пік ввечері за обсягом — у 21–22 години, а найважчі (по часу) запити — близько 8-ї ранку.

Розподіл типів запитів за годинами. На рисунку 3.24 представлено графік "Query Types by Hour of Day", де кожен тип запиту має власний колір: SELECT – червоний, INSERT – синій, UPDATE – зелений, DELETE – пурпурний, OTHER – помаранчевий.

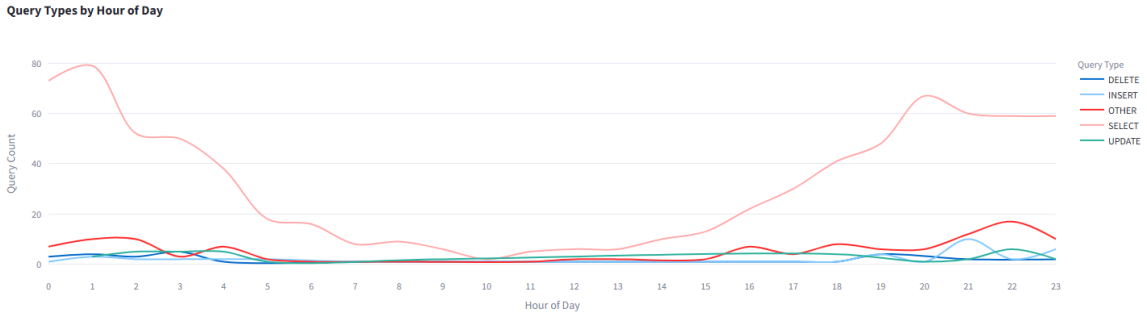

Рисунок 3.24 - Розподіл типів SQL-запитів за годинами доби

Аналіз дозволяє зрозуміти, які типи операцій домінують у різний час.

3.4.4.7 Аналіз найповільніших запитів

На сторінці Slowest Queries подано таблицю із запитами, які мають найвищу тривалість виконання (рисунок 3.25).

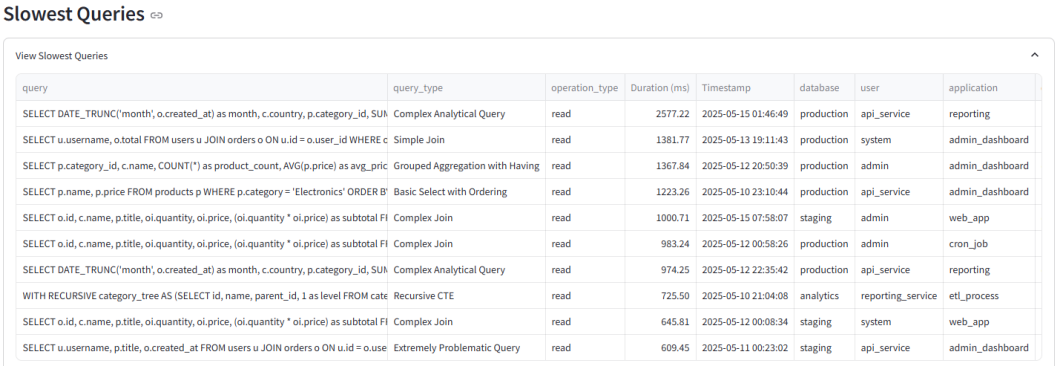

Рисунок 3.25 - Найповільніші запити системи

3.4.4.8 Оптимізація запитів

У вкладці Query Optimization користувач може ввести SQL-запит для аналізу та отримати рекомендації.

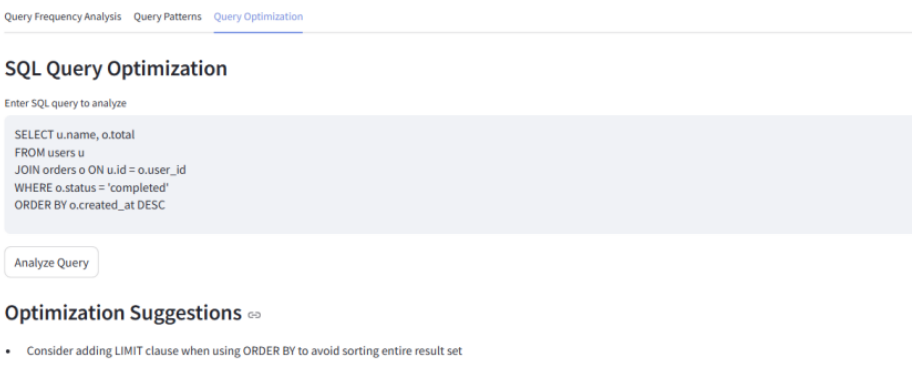

Рисунок 3.26 - Query Optimization

Це дозволяє визначити ресурсоємні операції та застосувати точкову оптимізацію.

3.4.5 Тестування аналізу файлів GCS

3.4.5.1 Навігація по GCS

На рисунку 3.27 зображено інтерфейс навігації по GCS – вибір бакету, префіксу, фільтрація та вибір файлу.

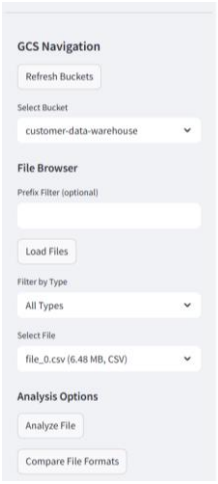

Рисунок 3.27 - Навігація по GCS

3.4.5.2 Огляд файлу

Обраний файл: logs/file_3.csv, **розмір:** 3.31 МБ.

Рекомендація: конвертувати updated_at у формат datetime (рисунок 3.28)

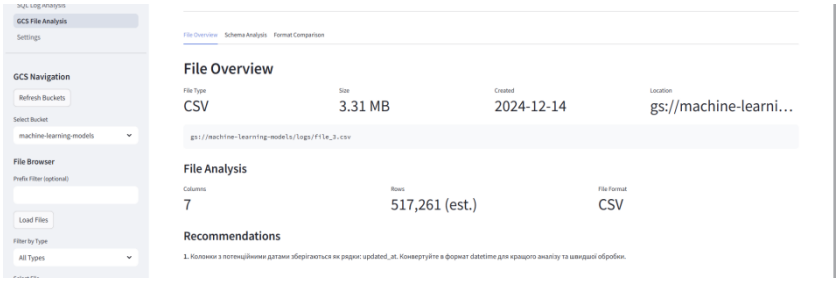

Рисунок 3.28 - Огляд файлу

3.4.5.3 Аналіз схеми даних

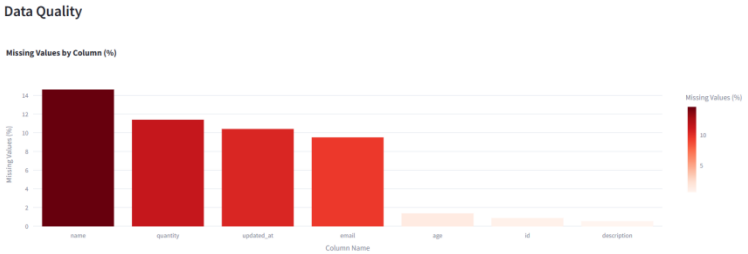

Рисунок 3.29 - Відсоток пропущених значень за колонками

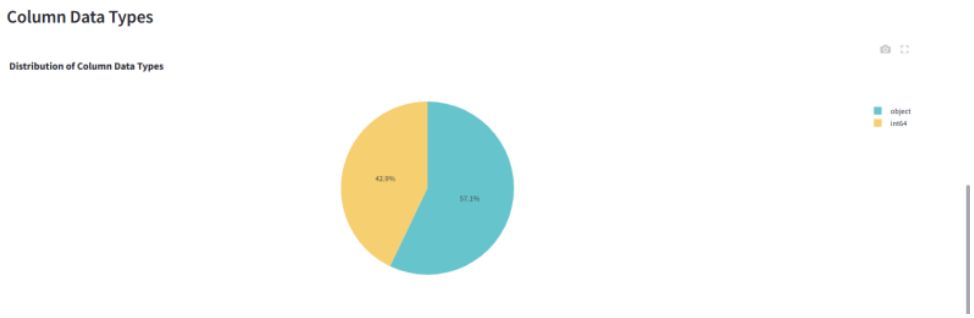

Рисунок 3.30 - Тип даних

Типи даних (рисунок 3.30):

- ✓ Object (строки): 57.1%
- ✓ Int64 (числа): 42.9%

3.4.5.4 Порівняння форматів

Detailed Comparison

Format	size_mb	Estimated Size	Relative Size (%)	Read Performance	is_current
csv	11.1371	11.14 MB	1%	1.0x	☑
parquet	3.8549	3.85 MB	2%	2.0x	☐
json	13.5528	13.55 MB	0%	0.9x	☐

Рисунок 3.31 - Таблиця порівняння форматів

Для аналітичних завдань найоптимальнішим є формат **Parquet** — він займає лише 32 % від обсягу CSV і забезпечує у 2,4 рази вищу швидкість читання.

3.4.6 Експорт та звітність

Ситема підтримує експорт у: CSV, JSON, PDF, повні інтерактивні звіти з рекомендаціями. Є можливість завантажувати звіт з кожної панелі.

DataScope Analytics - Database Analysis Report

Generated on 2025-05-16 12:16:11

Database Overview

Total Size: 53150 MB

Tables Count: 42

Slow Queries: 37

Missing Indexes: 28

Table Statistics

Table	Rows	Size (MB)	Indexes
event_logs	150000000	12500	N/A
user_activities	75000000	8750	N/A
product_inventory	48000000	6500	N/A
transactions	38000000	5200	N/A
orders	22000000	4800	N/A
click_stream_data	85000000	3900	N/A
customer_interact...	28000000	3500	N/A
products	9500000	2900	N/A
marketing_analytics	12000000	2800	N/A
customers	7500000	2200	N/A

Missing Indexes

Table	Column	Scan Count	Rows Fetched
event_logs	timestamp	145820	84250000

Рисунок 3.32 - Фрагмент звіту в .pdf

3.4.7 Висновки за результатами тестування

Контрольний приклад демонструє, що платформа "Big Data Optimization & Monitoring Platform" успішно виконує всі поставлені задачі та забезпечує ефективний моніторинг та оптимізацію роботи з даними.

Ключові результати тестування:

1) Моніторинг PostgreSQL

- ✓ Система успішно збирає та візуалізує загальну статистику бази даних
- ✓ Коректно ідентифікує проблемні таблиці без індексів
- ✓ Надає точні рекомендації щодо створення нових індексів на основі аналізу патернів запитів

- ✓ Виявляє повільні запити та пропонує конкретні шляхи їх оптимізації

2) Аналіз SQL-логів

- ✓ Система ефективно аналізує логи SQL-запитів та визначає характер навантаження
- ✓ Коректно визначає найпоширеніші патерни запитів
- ✓ Точно ідентифікує розподіл активності за часом доби
- ✓ Надає детальну інформацію про найповільніші запити
- ✓ Пропонує конкретні рекомендації щодо оптимізації запитів

3) Аналіз файлів GCS

- ✓ Система детально аналізує структуру та зміст файлів різних форматів
- ✓ Коректно визначає проблеми з якістю даних (відсутні значення)
- ✓ Надає точні порівняння ефективності різних форматів зберігання
- ✓ Пропонує обґрунтовані рекомендації щодо оптимізації формату та структури даних

4) Загальна функціональність

- ✓ Інтерфейс користувача зручний та інтуїтивно зрозумілий
- ✓ Візуалізації інформативні та допомагають швидко виявляти проблеми
- ✓ Система надає детальні рекомендації, що можуть бути безпосередньо застосовані
- ✓ Функціональність експорту даних та звітів працює коректно

Тестування підтверджує, що розроблена платформа є ефективним інструментом для моніторингу та оптимізації роботи з великими даними у PostgreSQL та Google Cloud Storage, забезпечуючи як високорівневий огляд, так і детальний аналіз окремих компонентів.

Розділ 4. ТЕХНІКО-ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ РОЗРОБКИ ПІДСИСТЕМИ

Розробка підсистеми "Big Data Optimization & Monitoring Platform" має на меті створення інструменту для аналізу продуктивності баз даних PostgreSQL, обробки SQL-логів і управління файлами в Google Cloud Storage (GCS), а також автоматизації рекомендацій щодо оптимізації запитів і звітності. Сучасні компанії, які працюють із великими обсягами даних, часто стикаються з проблемами повільних запитів, неефективного використання хмарних ресурсів і значних витрат на інфраструктуру. Аналіз продуктивності та оптимізація дозволяють підвищити швидкість обробки даних, зменшити витрати на зберігання й обробку, а також забезпечити стабільність систем навіть при високих навантаженнях. Використання передових методів, таких як індексація, кешування та горизонтальне масштабування, сприяє автоматизації ключових процесів, що зменшує вплив людського фактора, оптимізує робочі процедури та підвищує загальну ефективність роботи з великими даними. Цей інструмент стане корисним для компаній, які прагнуть оптимізувати свої ресурси та покращити продуктивність у роботі з Big Data.

4.1 Ергономічні аспекти інтерфейсу користувача

Ергономіка інтерфейсу є важливим елементом підсистеми, адже від зручності використання залежить продуктивність користувачів, їхнє сприйняття платформи та загальна ефективність роботи з системою.

4.1.1 Принципи дизайну інтерфейсу

Інтерфейс платформи розроблено з урахуванням сучасних принципів дизайну, які забезпечують комфорт і зручність для користувачів. Одним із ключових підходів є інтуїтивність — користувачі можуть швидко освоїти систему без тривалого навчання, адже всі елементи розташовані логічно та зрозуміло. Консистентність дизайну забезпечується єдиним стилем оформлення: однакова колірна палітра, типографіка та поведінка елементів на всіх сторінках.

Інформативність інтерфейсу дозволяє користувачам отримувати лише потрібну інформацію без зайвих деталей, що зменшує когнітивне навантаження. Ефективність досягається завдяки мінімізації кількості дій, необхідних для виконання завдань — наприклад, для перегляду звіту достатньо кількох кліків. Інтерфейс також масштабований, тобто адаптується до різних розмірів екранів, від великих моніторів до ноутбуків, забезпечуючи комфортну роботу в будь-яких умовах. Для реалізації цих принципів обрано інструмент Streamlit, який дозволяє створювати мінімалістичний і сучасний дизайн із підтримкою інтерактивних елементів, таких як фільтри, графіки та таблиці, без складного програмування.

4.1.2 Структура інтерфейсу

Структура інтерфейсу побудована за принципом "від загального до детального", що дозволяє користувачам поступово заглиблюватися в аналіз даних. Головна сторінка платформи є центральним дашбордом, де відображаються основні метрики продуктивності, такі як активність бази даних PostgreSQL чи обсяг даних у GCS. Ці метрики представлені у вигляді карток, які дозволяють швидко оцінити стан системи та перейти до детального аналізу. Бокова навігаційна панель забезпечує доступ до всіх розділів платформи, зокрема моніторингу PostgreSQL, аналізу SQL-логів, управління файлами GCS і налаштувань. Панель завжди доступна, що спрощує перемикання між розділами. Кожна сторінка має уніфіковану структуру: заголовок, вкладки для навігації між підрозділами, основний контент із візуалізаціями та панель дій для експорту даних чи оновлення інформації. Дизайн адаптивний — на менших екранах панель ховається в меню, а графіки перебудовуються для зручного перегляду.

4.1.3 Типографіка

Типографіка інтерфейсу розроблена для забезпечення комфортного сприйняття інформації. Використовуються шрифти без зарубок, які відомі своєю високою читабельністю. Основний шрифт підібрано так, щоб текст залишався чітким навіть при тривалій роботі, а запасний шрифт гарантує сумісність із різними пристроями. Розміри шрифтів збалансовані: текст основного контенту достатньо великий для легкого читання, заголовки виділені для чіткої ієрархії, а підписи до

графіків менші, щоб не відволікати увагу. Відстань між рядками оптимальна для комфортного читання довгих текстів, а для заголовків вона зменшена, щоб зробити їх компактнішими. Контраст між текстом і фоном відповідає стандартам доступності, що зменшує втому очей. Для акцентування ключових елементів, наприклад назв метрик, використовується напівжирний стиль, а другорядна інформація, як-от примітки, виділена курсивом.

4.1.4 Візуалізація даних

Візуалізація даних є ключовою частиною платформи, адже вона допомагає користувачам швидко зрозуміти стан системи. Для цього використовуються різні типи графіків і діаграм, які підходять для різних видів аналізу. Наприклад, лінійні графіки показують зміни продуктивності з часом, кругові діаграми відображають розподіл ресурсів, а теплові карти допомагають виявити кореляції між параметрами. Усі візуалізації інтерактивні: користувачі можуть наводити курсор для перегляду деталей, масштабувати графіки або застосовувати фільтри для глибшого аналізу. Також передбачено можливість експорту даних у зручні формати, щоб користувачі могли зберегти результати для подальшого використання. Такий підхід забезпечує гнучкість і допомагає швидко виявляти проблемні ділянки, наприклад, повільні запити в PostgreSQL або неефективне використання GCS.

4.1.5 Інтерактивність та відгук системи

Інтерактивність є важливою складовою ергономіки. Система швидко реагує на дії користувача, забезпечуючи миттєвий зворотний зв’язок, наприклад, під час застосування фільтрів або оновлення даних. Для тривалих операцій, таких як обробка великих обсягів SQL-логів, відображається індикатор прогресу, щоб користувач знав, що система працює. Підказки з’являються при наведенні на елементи, допомагаючи зрозуміти їхню функцію. Система також інформує про успішні дії чи помилки через сповіщення. Налаштування користувача, наприклад, обрані фільтри чи вигляд дашборду, зберігаються, щоб не доводилося повторно їх налаштовувати під час наступного використання.

4.1.6 Доступність інтерфейсу

Платформа розроблена з урахуванням принципів доступності, щоб її могли використовувати різні категорії користувачів. Контраст між текстом і фоном відповідає стандартам доступності, що полегшує роботу людям із вадами зору. Для графіків і зображень передбачено альтернативний текст, який описує їхній зміст для скрін-рідерів. Інтерфейс підтримує клавіатурну навігацію, дозволяючи керувати системою без миші. Текст адаптується до збільшення розміру, щоб користувачі могли налаштувати його під свої потреби. Також додано спеціальні атрибути для забезпечення сумісності зі скрін-рідерами, що робить платформу доступною для людей із обмеженими можливостями.

4.1.7 Багатомовний інтерфейс

Для роботи на міжнародному ринку платформа підтримує кілька мов, зокрема українську та англійську. Локалізація охоплює всі тексти, повідомлення та формати даних, такі як дати чи одиниці вимірювання. Користувачі можуть обирати мову вручну, але система також автоматично визначає мову залежно від налаштувань пристрою. Локалізація реалізована через спеціальний словник, який дозволяє швидко додавати нові мови в майбутньому, що робить платформу гнучкою для розширення.

4.1.8 Ергономіка для користувачів

Інтерфейс адаптований до потреб різних груп користувачів. Аналітики отримують доступ до інтерактивних графіків і можливості експорту даних для глибшого аналізу. Адміністратори баз даних можуть швидко переглядати метрики продуктивності PostgreSQL і отримувати рекомендації щодо оптимізації. Керівники мають зведені дашборди для оцінки загального стану системи, а також можливість створювати звіти у зручному форматі. Технічна підтримка отримує доступ до логів і діагностичних інструментів, що спрощує виявлення та вирішення проблем.

4.2 Техніко-економічне обґрунтування розробки

4.2.1 Аналіз ринку

Ринок інструментів для моніторингу та оптимізації великих даних активно розвивається через постійне зростання обсягів інформації, що обробляються компаніями. Бізнеси потребують швидкого аналізу даних у реальному часі, що стає складним завданням через різноманітність інфраструктури, яка включає гібридні сховища, хмарні сервіси та локальні бази даних. Існуючі рішення на ринку поділяються на кілька категорій: комерційні платформи від великих компаній, спеціалізовані інструменти для конкретних систем і безкоштовні open-source рішення. Комерційні платформи часто пропонують широкий функціонал, але їхня інтеграція з різними системами може бути обмеженою. Спеціалізовані інструменти зосереджені на певних аспектах, наприклад, моніторингу баз даних, але не охоплюють повний цикл аналізу. Open-source рішення доступні, але потребують значних зусиль для налаштування та адаптації. Платформа "Big Data Optimization & Monitoring Platform" вирізняється комплексним підходом, адже поєднує моніторинг PostgreSQL, аналіз SQL-логів і управління файлами в GCS, пропонуючи зручний інтерфейс і автоматизацію ключових процесів.

4.2.2 Оцінка витрат

Розробка проєкту передбачає витрати на кількох напрямках. Основною статтею є оплата праці команди, яка включає архітекторів програмного забезпечення, розробників, інженерів даних, дизайнерів інтерфейсу, тестувальників і менеджерів проєкту. Кожен із них виконує свою роль, від створення архітектури до тестування готового продукту. Інфраструктура також потребує вкладень, зокрема на сервери для тестування, програмне забезпечення для розробки, хмарні сервіси, такі як GCS, а також підготовку тестових даних для перевірки системи. Додаткові витрати пов’язані з адміністративними потребами, наприклад, юридичною підтримкою, бухгалтерським обліком і іншими операційними аспектами. Для забезпечення стабільності проєкту передбачено резервний фонд, який допоможе покрити непередбачені витрати, якщо вони виникнуть під час реалізації.

4.2.3 Економічний ефект

Проєкт приносить економічні вигоди як у прямій, так і в непрямій формі. Прямі вигоди включають економію ресурсів завдяки оптимізації зберігання даних у GCS, наприклад, через використання ефективних форматів і стиснення. Оптимізація запитів до PostgreSQL зменшує навантаження на базу даних, що дозволяє економити на обчислювальних ресурсах. Автоматизація моніторингу та звітності скорочує час, який адміністратори витрачають на ручну роботу, що також зменшує витрати. Непрямі вигоди пов'язані з підвищенням продуктивності команд, адже швидший доступ до даних і рекомендації з оптимізації дозволяють швидше приймати рішення. Надійність системи зростає завдяки зменшенню кількості збоїв, а процеси розробки стають ефективнішими через автоматизацію тестування та аналізу продуктивності.

4.2.4 Окупність

Економічна доцільність проєкту визначається співвідношенням витрат і вигод. Завдяки прямим і непрямим вигодам, які включають економію на інфраструктурі, прискорення аналізу даних і підвищення надійності, проєкт має потенціал повернути вкладені кошти в розумний період. Очікується, що платформа швидко знайде свою аудиторію серед компаній, які працюють із великими даними, адже вона пропонує унікальний набір функцій, що відповідає сучасним потребам ринку. У довгостроковій перспективі проєкт може стати основою для розширення функціональності, наприклад, додавання підтримки інших баз даних, що ще більше підвищить його цінність.

4.3 Технічні аспекти реалізації

1) Використані технології

Для реалізації платформи використано сучасні технології, що забезпечують швидкодію та масштабність. Python обрано основною мовою програмування завдяки її підтримці бібліотек для роботи з даними та базами. Для інтеграції з PostgreSQL використовуються спеціальні драйвери, які забезпечують надійне з'єднання. Google Cloud Platform слугує основою для роботи з GCS, надаючи інструменти для зберігання й обробки файлів. Streamlit використовується для

створення інтерактивного інтерфейсу, що дозволяє швидко генерувати дашборди та графіки. Для аналізу продуктивності застосовуються вбудовані інструменти PostgreSQL, які допомагають виявляти повільні запити, а для GCS використовуються аналітичні сервіси Google Cloud.

2) Методи оптимізації

Оптимізація є ключовою частиною платформи. Індексація застосовується для прискорення пошуку в PostgreSQL, дозволяючи швидко знаходити потрібні записи. Кешування зберігає часто використовувані дані в швидкій пам’яті, зменшуючи час доступу до них. Горизонтальне масштабування розподіляє навантаження між кількома вузлами в хмарі, що забезпечує стабільність при великих обсягах даних. У GCS використовуються ефективні формати даних для економії простору та прискорення обробки.

3) Інтеграція та тестування

Інтеграція з PostgreSQL і GCS була основним технічним завданням. Для цього створено модулі, які забезпечують безперебійне з’єднання та обмін даними між системами. Тестування проводилося на різних обсягах даних, щоб перевірити стабільність і швидкість роботи платформи. Автоматизовані тести допомогли виявити потенційні проблеми, а стрес-тестування підтвердило здатність системи витримувати високі навантаження.

4.4 Перспективи розвитку

1. Розширення функціональності

У майбутньому планується додавання підтримки інших баз даних, крім PostgreSQL, щоб розширити аудиторію платформи. Також розглядається інтеграція з іншими хмарними сервісами, що дозволить працювати з різноманітними сховищами. Додавання функцій прогнозування на основі машинного навчання може допомогти передбачати навантаження та пропонувати ще ефективніші рекомендації.

2. Вихід на міжнародний ринок

Платформа має потенціал для роботи на міжнародному ринку завдяки багатомовному інтерфейсу та гнучкості. Планується активне просування через

участь у конференціях, присвячених Big Data, і співпраця з партнерами в різних регіонах. Це дозволить залучити нових клієнтів і розширити вплив проєкту.

3. Підвищення автоматизації

Подальший розвиток передбачає ще більшу автоматизацію, наприклад, автоматичне створення індексів у PostgreSQL на основі аналізу запитів. Також планується впровадження автоматичного очищення застарілих даних у GCS, що додатково зменшить витрати клієнтів на зберігання.

ВИСНОВОК

У даній дипломної роботи було проведено дослідження та розробку підсистему "Big Data Optimization & Monitoring Platform", спрямованy на аналіз продуктивності баз даних PostgreSQL, обробку SQL-логів і управління файлами в Google Cloud Storage (GCS), а також автоматизацію рекомендацій щодо оптимізації запитів і звітності. Розглянуто основні етапи реалізації, включаючи вибір методів і технологій, створення ергономічного інтерфейсу та оцінку економічної доцільності проєкту.

Під час роботи виявлено ключову проблему, з якою стикаються компанії, що працюють із великими обсягами даних: повільна обробка запитів, неефективне використання хмарних ресурсів і складність аналізу продуктивності. Ці аспекти можуть призводити до значних витрат і зниження стабільності систем. Для вирішення цих питань запропоновано та реалізовано комплексний підхід, який включає індексацію для прискорення пошуку в PostgreSQL, кешування для зменшення затримок доступу до даних і горизонтальне масштабування для розподілу навантаження в хмарі. Для інтеграції з GCS використано ефективні методи обробки файлів, що сприяє оптимізації зберігання.

Розроблена платформа включає інтуїтивний інтерфейс, створений на базі Streamlit, який забезпечує зручний доступ до аналітичних інструментів, візуалізацій і звітів. Система адаптована до потреб різних користувачів — від аналітиків до адміністраторів — і підтримує багатомовність та доступність. Тестування платформи показало її здатність ефективно аналізувати продуктивність і пропонувати рекомендації, що полегшує роботу з великими даними.

Економічне обґрунтування підтвердило доцільність проєкту. Аналіз ринку виявив зростаючий попит на інструменти оптимізації, а платформа вирізняється своїм комплексним підходом до моніторингу PostgreSQL, SQL-логів і GCS. Витрати на розробку покриваються за рахунок оптимізації ресурсів, автоматизації процесів і підвищення продуктивності, що робить проєкт вигідним у довгостроковій перспективі.

Запропонована система має ряд переваг порівняно з існуючими рішеннями: адаптивність до різних умов роботи, висока ефективність аналізу завдяки інтеграції з сучасними технологіями та зручність використання через інтерактивний інтерфейс. На основі отриманих результатів рекомендується впроваджувати платформу в компаніях, які працюють із великими даними, зокрема в IT, фінансах і електронній комерції, де важлива швидкість і надійність обробки інформації. У майбутньому проєкт може бути розширений шляхом додавання підтримки інших баз даних, інтеграції з новими хмарними сервісами та впровадження прогнозування на основі машинного навчання.

Таким чином, дана дипломна робота зробила значний внесок у розвиток інструментів для роботи з великими даними, продемонструвавши ефективність запропонованого підходу та його потенціал для практичного застосування.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Laney D. 3D data management: controlling data volume, velocity and variety. USA: Application Delivery Strategies, Meta Group; 2001. P. 1–4

2. Demchenko Y., de Laat C., Membrey P. Defining architecture components of the Big Data Ecosystem. 2014 International Conference on Collaboration Technologies and Systems (CTS), Minneapolis, MN, USA, 19–23 May 2014. 2014. URL: https://doi.org/10.1109/cts.2014.6867550 (date of access: 15.01.2025)

3. Harish B. Top Challenges of Big Data Analytics in 2024. Keystride. URL: https://www.keystride.com/blog/big-data-analytics-challenges/ (date of access: 16.01.2025)

4. Richman J. What Is A Traditional Data Warehouse? Examples & Challenges. Estuary | Real-Time Data Integration, CDC & ETL Platform. URL: https://estuary.dev/traditional-data-warehouse/ (date of access: 18.01.2025)

5. Brown F. Indexing in DBMS: What is, Types of Indexes with EXAMPLES. Guru99. URL: https://www.guru99.com/indexing-in-database.html (date of access: 18.01.2025)

6. Timescale. Data Partitioning: What It Is and Why It Matters. Timescale. URL: https://www.timescale.com/learn/data-partitioning-what-it-is-and-why-it-matters (дата звернення: 18.01.2025).

7. Головньов С. Що таке кешування даних – визначення, види, типи та ін.. Resit. URL: https://resit.com.ua/scho-take-keshirovanie-dannyh-viznachennya-vidy-tipy-i-dr/ (date of access: 18.01.2025)

8. Bernyk D. Нормалізація баз даних. - greenhouse. Greenhouse - Мій затишний зелений будинок. URL: https://greenhouse.cv.ua/?p=697 (дата звернення: 18.01.2025).

9. Foxminded. Схеми бази даних: компоненти, типи та проєктування. Foxminded. 2023. URL: https://foxminded.ua/skhemy-bazy-danyh/ (дата звернення: 18.01.2025).

10. VPN Unlimited. Оптимізація бази даних - це процес покращення продуктивності, ефективності та загальної функціональності бази даних. VPN Unlimited. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/database-optimization (дата звернення: 18.01.2025).

11. Що таке хмарна інфраструктура, її можливості та переваги. blog.colobridge.net. URL: https://blog.colobridge.net/uk/2023/10/what-does-cloud-infrastructure-consist-of-ua/ (дата звернення: 19.01.2025).

12. Google. What is cloud architecture? . Google Cloud. 2024. URL: https://cloud.google.com/learn/what-is-cloud-architecture (дата звернення: 19.01.2025)

13. Андреєв А. Найбільші провайдери хмарних сервісів: огляд 5 найкращих платформ. Apix-Drive. URL: https://apix-drive.com/ua/blog/reviews/najkrashi-provajdery-hmarnih-servisiv (дата звернення: 19.01.2025).

14. Colobridge. Що таке хмарне сховище: як воно працює та для чого потрібне. Colobridge. 2023. URL: https://blog.colobridge.net/uk/2023/12/what-is-cloud-storage-ua/ (дата звернення: 20.01.2025)

15. Пазинін А. С. Методи автоматичного масштабування в хмарних середовищах. КІБЕРБЕЗПЕКА: освіта, наука, техніка. 2024. С. 445–456.

16. Сухорукова Г. Як бізнесу працювати з реляційними та нереляційними базами даних? Kyivstar Business Hub. 2023. URL: https://hub.kyivstar.ua/articles/yak-biznesu-praczyuvaty-z-relyaczijnymy-ta-nerelyaczijnymy-bazamy-danyh (дата звернення: 20.01.2025).

17. Sem V. SQL чи nosql - ось в чому питання - альтернативна наука віталія сема. Альтернативна Наука Віталія Сема. URL: https://alternativescience.net/programming/242-sql-chy-nosql-os-v-chomu-pytannya/ (дата звернення: 20.01.2025).

18. Brown F. SQL проти NoSQL – різниця між ними. Guru99. URL: https://www.guru99.com/uk/sql-vs-nosql.html (дата звернення: 21.01.2025).

19. Зінченко Б. В. Методи оптимізації баз даних для задач з веб розробки: кваліфікаційна (магістерська) робота. Спеціальність 122 Комп'ютерні науки. – Донецький національний університет імені Василя Стуса, Вінниця, 2024. – 80 с.

20. Rose-Collins F. Top 10 best practices for optimizing databases for web developers. Ranktracker. URL: https://www.ranktracker.com/blog/top-10-best-practices-for-optimizing-databases-for-web-developers/ (date of access: 20.01.2025).

21. Foo D. 11 технік оптимізації бази даних - data-life-ua. data-life-ua. URL: https://data-life-ua.com/db/11-database-optimization-techniques/ (дата звернення: 21.01.2025).

22. Foxminded. Система управління базами даних (СУБД). Foxminded. 2023. URL: https://foxminded.ua/systema-upravlinnia-bazamy-danykh/ (дата звернення: 22.01.2025).

23. Інформаційні системи і технології в економіці. URL: https://studfile.net/preview/5118185/page:29/ (дата звернення: 22.01.2025).

24. Smith J., Lee A. Cloud Database Optimization: Strategies for Performance, Scalability, and Cost-Efficiency. Journal of Cloud Computing. 2024. Vol. 13, No. 2. P. 101–120.

25. Patel R., Kim S. Optimizing Cloud Computing Performance With Advanced DBMS Techniques: A Comparative Study. International Journal of Database Management Systems. 2023. Vol. 16, No. 4. P. 55–72.

ДОДАТОК А

```
CREATE TABLE IF NOT EXISTS query_performance_log (
    id SERIAL PRIMARY KEY,
    query_text TEXT NOT NULL,           -- Текст SQL-запиту
    execution_time_ms FLOAT,            -- Час виконання запиту в мілісекундах
    planning_time_ms FLOAT,             -- Час планування запиту в мілісекундах
    total_cost FLOAT,                   -- Загальна вартість запиту
    rows_processed INTEGER,             -- Кількість оброблених рядків
    log_timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP -- Час логування
);
CREATE OR REPLACE FUNCTION log_query_performance() RETURNS TRIGGER AS $$
DECLARE
    explain_result TEXT;                -- Змінна для зберігання результату EXPLAIN
ANALYZE
    execution_time FLOAT;               -- Час виконання
    planning_time FLOAT;                -- Час планування
    total_cost FLOAT;                   -- Загальна вартість запиту
    rows_processed INTEGER;             -- Кількість оброблених рядків
BEGIN
    BEGIN
        EXECUTE format('EXPLAIN (ANALYZE, FORMAT TEXT) %s', NEW.query_text) INTO
explain_result;
    EXCEPTION WHEN OTHERS THEN
        RAISE NOTICE 'Error analyzing query: %', SQLERRM;
        RETURN NEW;
    END;
    IF explain_result ~* 'Planning Time: [0-9]+\.[0-9]+ ms' THEN
        planning_time := regexp_replace(explain_result, '.*Planning Time: ([0-
9]+\.[0-9]+) ms.*', '\1')::FLOAT;
    ELSE
        planning_time := NULL;
    END IF;
        IF explain_result ~* 'Execution Time: [0-9]+\.[0-9]+ ms' THEN
        execution_time := regexp_replace(explain_result, '.*Execution Time: ([0-
9]+\.[0-9]+) ms.*', '\1')::FLOAT;
    ELSE
        execution_time := NULL;
    END IF;
        IF explain_result ~* 'cost=[0-9]+\.[0-9]+\.\.[0-9]+\.[0-9]+' THEN
        total_cost := regexp_replace(explain_result, '.*cost=[0-9]+\.[0-9]+\.\.([0-
9]+\.[0-9]+).*', '\1')::FLOAT;
    ELSE
```

```
        total_cost := NULL;
    END IF;
        IF explain_result ~* 'rows=[0-9]+' THEN
        rows_processed   :=   regexp_replace(explain_result,   '.*rows=([0-9]+).*',
'\1')::INTEGER;
    ELSE
        rows_processed := NULL;
    END IF;
        UPDATE query_performance_log
    SET
        execution_time_ms = execution_time,
        planning_time_ms = planning_time,
        total_cost = total_cost,
        rows_processed = rows_processed
    WHERE id = NEW.id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trigger_log_query_performance
AFTER INSERT ON query_performance_log
FOR EACH ROW
EXECUTE FUNCTION log_query_performance();
```

ДОДАТОК Б

```
import psycopg2
import csv
from google.cloud import storage
from datetime import datetime
import os

def export_to_gcs(bucket_name="my-performance-logs-bucket",
file_prefix="query_performance"):
    """
    Експортує логи продуктивності SQL-запитів із PostgreSQL до Google Cloud Storage
(GCS).
        bucket_name (str): Назва бакета в GCS.
        file_prefix (str): Префікс для імені файлу в GCS.
        None
    """
    try:
        conn = psycopg2.connect(
            dbname="performance_db",
            user="user",
            password="password",
            host="localhost",
            port="5432"
        )
        cursor = conn.cursor()
        cursor.execute("""
            SELECT id, query_text, execution_time_ms, planning_time_ms,
                   total_cost, rows_processed, log_timestamp
            FROM query_performance_log
        """)
        rows = cursor.fetchall()
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        local_file = f"{file_prefix}_{timestamp}.csv"
        with open(local_file, mode='w', newline='', encoding='utf-8') as f:
            writer = csv.writer(f)
            writer.writerow([
                "id", "query_text", "execution_time_ms", "planning_time_ms",
                "total_cost", "rows_processed", "log_timestamp"
            ])
            writer.writerows(rows)
               os.environ["GOOGLE_APPLICATION_CREDENTIALS"]                    =
"path/to/your/service-account-key.json"
```

```
        storage_client = storage.Client()
        bucket = storage_client.bucket(bucket_name)
        blob_name = f"{file_prefix}/{file_prefix}_{timestamp}.csv"
        blob = bucket.blob(blob_name)
        blob.upload_from_filename(local_file)
        print(f"Дані успішно експортовано в GCS: gs://{bucket_name}/{blob_name}")
        os.remove(local_file)
    except Exception as e:
        print(f"Помилка під час експорту до GCS: {str(e)}")
        raise
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
if __name__ == "__main__":
    export_to_gcs(bucket_name="my-performance-logs-bucket",
file_prefix="query_performance")
```

ДОДАТОК В

Приклад повного звіту .pdf

DataScope Analytics - Database Analysis Report

Generated on 2025-05-18 11:14:08

Database Overview

Total Size: 53150 MB

Tables Count: 42

Slow Queries: 37

Missing Indexes: 28

Table Statistics

Table	Rows	Size (MB)	Indexes
event_logs	150000000	12500	N/A
user_activities	75000000	8750	N/A
product_inventory	48000000	6500	N/A
transactions	38000000	5200	N/A
orders	22000000	4800	N/A
click_stream_data	85000000	3900	N/A
customer_interact...	28000000	3500	N/A
products	9500000	2900	N/A
marketing_analytics	12000000	2800	N/A
customers	7500000	2200	N/A

Missing Indexes

Table	Column	Scan Count	Rows Fetched
event_logs	timestamp	145820	84250000
click_stream...	user_id	98760	56340000
user_activities	session_id	78520	42680000
transactions	transaction_date	67300	28750000
customer_int...	interaction_type	54200	19820000
orders	customer_id	48500	14250000
product_inve...	last_updated	42100	36850000

marketing_an...	campaign_id	38600	9240000
user_sessions	device_type	35700	15620000
products	category_id	32500	7640000

+ 5 more missing indexes

Slow Queries

Query 1

SELECT e.*, d.name as department_name, SUM(t.amount) as total_transactions FROM event_logs e JOIN us...

Avg Time: 15250.5 ms Calls: 8420 Rows: 2450000

Query 2

WITH ranked_products AS (SELECT p.*, dense_rank() OVER (PARTITION BY p.category_id ORDER BY p.price ...

Avg Time: 12850.3 ms Calls: 4560 Rows: 1250000

Query 3

SELECT t.*, c.name as customer_name, p.payment_method, a.city, a.country FROM transactions t JOIN or...

Avg Time: 8350.8 ms Calls: 15420 Rows: 3620500

Query 4

UPDATE product_inventory SET quantity = quantity - $1, last_updated = CURRENT_TIMESTAMP WHERE produc...

Avg Time: 4950.2 ms Calls: 125800 Rows: 1

Query 5

INSERT INTO click_stream_data (session_id, user_id, page_url, referrer, user_agent, event_type, brow...

Avg Time: 3260.7 ms Calls: 850500 Rows: 1

+ 2 more slow queries

ДОДАТОК Г

Приклад завантаження файлу з реальними даними