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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Аналіз стратегій покращення продуктивності реляційних систем керування базами даних на прикладі PostgreSQL

Виконав: студент VI курсу, групи СНм-61
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Чекановський А.Б.
(підпис) (прізвище та ініціали)

Керівник _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Никитюк В.В.
(підпис) (прізвище та ініціали)

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

Рецензент _____ Кульчицький Т.Р.
(підпис) (прізвище та ініціали)

Тернопіль
2024

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

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

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
« 26 » грудня 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Магістр
(назва освітнього ступеня)

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

Студенту Чекановському Андрію Богдановичу
(прізвище, ім'я, по батькові)

1. Тема роботи Аналіз стратегій покращення продуктивності реляційних систем керування базами даних на прикладі PostgreSQL

Керівник роботи Боднарчук Ігор Орестович, к.т.н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 29 » листопада 2024 року № 4/7-1141

2. Термін подання студентом завершеної роботи 27 грудня 2024р.

3. Вихідні дані до роботи Наукові публікації про стратегії покращення продуктивності реляційних систем керування базами даних, їх аналіз та застосування

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз предметної області. 2 Стратегії покращення продуктивності систем керування базами даних. 3 Практичне дослідження стратегій оптимізації. 4 Охорона праці та безпека в надзвичайних ситуаціях. Висновки. Перелік використаних джерел. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титульний слайд. 2. Мета та задачі дослідження. 3. Актуальність. 4. Аналіз предметної області. 5. Проектування схеми БД. 6. Індексація. 7. Оптимізація запитів. Нормалізація та денормалізація. 8. Матеріалізовані подання. Управління ключами. 9. Конфігурація БД. Оптимізація апаратних ресурсів. 10. Опис експерименту та тестового середовища. 11. Вплив індексування. 12. Управління ключами. Послідовний ID та UUID. 13. Конфігурація БД. 14. Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Сенчишин В.С., доцент		
Безпека в надзвичайних ситуаціях	Клепчик В.М., ст. викладач		

7. Дата видачі завдання 28 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	29.11.2024	*Виконано*
2.	Підбір наукових джерел щодо стратегій покращення	29.11.2024-04.12.2024	*Виконано*
	продуктивності реляційних систем керування базами		
	даних		
3.	Опрацювання наукових публікацій та збір даних по	29.11.2024-04.12.2024	*Виконано*
	темі роботи		
4.	Виконання дослідження згідно мети кваліфікаційної	05.12.2024-09.12.2024	*Виконано*
	роботи		
5.	Оформлення розділу «Аналіз предметної області»	05.12.2024-09.12.2024	*Виконано*
6.	Оформлення розділу «Стратегії покращення	05.12.2024-09.12.2024	*Виконано*
	продуктивності систем керування базами даних»		
7.	Оформлення розділу «Практичне дослідження	05.12.2024-09.12.2024	*Виконано*
	стратегій оптимізації»		
8.	Виконання завдання до підрозділу «Охорона праці»	02.12.2024-09.12.2024	*Виконано*
9.	Виконання завдання до підрозділу «Безпека в	02.12.2024-09.12.2024	*Виконано*
	надзвичайних ситуаціях»		
10.	Оформлення кваліфікаційної роботи	10.12.2024-11.12.2024	*Виконано*
11.	Нормоконтроль	12.12.2024-13.12.2024	*Виконано*
12.	Перевірка на плагіат	16.12.2024	*Виконано*
13.	Попередній захист кваліфікаційної роботи	17.12.2024	*Виконано*
14.	Захист кваліфікаційної роботи	27.12.2024	

Студент _____ Чекановський А.Б.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Аналіз стратегій покращення продуктивності реляційних систем керування базами даних на прикладі PostgreSQL // Кваліфікаційна робота освітнього рівня «Магістр» // Чекановський Андрій Богданович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНм-61 // Тернопіль, 2024 // С. 83, рис. – 29, табл. – 19, кресл. – 14, додат. – 3, бібліогр. – 65.

Ключові слова: база даних, оптимізація, продуктивність, аналіз, ресурс, метрика, конфігурація.

Кваліфікаційна робота присвячена аналізу стратегій покращення продуктивності реляційних СКБД на прикладі PostgreSQL.

У першому розділі кваліфікаційної роботи описані значення продуктивності баз даних у сучасних інформаційних системах, ключові показники продуктивності, а також фактори, що впливають на неї.

У другому розділі кваліфікаційної роботи розглянуто архітектуру PostgreSQL, життєвий цикл виконання запитів, ключові елементи плану запиту. Проаналізовано проектування схеми бази даних, детально описано індексацію та її типи. Описано нормалізацію та денормалізацію, а також наведено ефективність матеріалізованих представлень. Описано управління ключами, роль реплікації та шардингу. Проаналізовано вплив конфігураційних параметрів, а також ефективність використання апаратних ресурсів і RAID рівнів.

У третьому розділі кваліфікаційної роботи описано тестове середовище, конфігурацію бази даних і використані інструменти для аналізу продуктивності. Проаналізовано результати впровадження різних типів індексів на продуктивність запитів, оптимізацію ключів, а також вплив змін конфігураційних параметрів на загальну ефективність роботи системи. Проведено відповідні експерименти.

ANNOTATION

Analysis of strategies for the Performance Improving of Relational Database Management Systems Using PostgreSQL as an Example // The educational level "Master" qualification work // Chekanovskyi Andrii Bogdanovich // Ternopil Ivan Pulyuy National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, SNm-61 group // Ternopil, 2024 // P. 83, fig. - 29, tables - 19, posters - 14, annexes - 3, ref. - 65.

Key words: database, optimization, performance, analysis, resource, metric, configuration.

The thesis is devoted to analyzing strategies for improving the performance of relational database management systems (RDBMS), with PostgreSQL as a case study.

The first chapter describes the significance of database performance in modern information systems, key performance indicators, and the factors influencing them.

The second chapter examines the architecture of PostgreSQL, the lifecycle of query execution, and the essential components of query plans. It analyzes database schema design, provides detailed insights into indexing and its types, and describes normalization, denormalization, and the efficiency of materialized views. Key management, the role of replication, and sharding are also explored. The chapter further evaluates the impact of configuration parameters and the effectiveness of utilizing hardware resources and RAID levels.

The third chapter presents the test environment, database configuration, and tools used for performance analysis. It assesses the impact of implementing various types of indexes on query performance, key optimization, and the influence of changes to configuration parameters on overall system efficiency. Corresponding experiments were conducted.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

БД – База даних.

СКБД – Система керування базами даних.

ACID (англ. Atomicity, Consistency, Isolation, Durability) – Атомарність, узгодженість, ізоляція, довговічність.

BRIN (англ. Block Range Index) – Індекс діапазону блоків.

ERD (англ. Entity-Relationship Diagram) – Діаграма сутність-зв'язок.

GIN (англ. Generalized Inverted Index) – Узагальнений інвертований індекс.

GIST (англ. Generalized Search Tree) – Узагальнене дерево пошуку.

HDD (англ. Hard Disk Drive) – Жорсткий диск.

JSONB (англ. JavaScript Object Notation Binary) – Бінарне представлення JSON.

NVMe (англ. Non-Volatile Memory Express) – Інтерфейс для швидкісного доступу до твердотільних накопичувачів.

PK (англ. Primary Key) – Первинний ключ.

RAM (англ. Random Access Memory) – Оперативна пам'ять.

RAID (англ. Redundant Array of Independent Disks) – Надлишковий масив незалежних дисків.

SP-GIST (англ. Space-Partitioned Generalized Search Tree) – Узагальнене дерево пошуку із розбиттям простору.

SQL (англ. Structured Query Language) – Мова структурованих запитів.

SSD (англ. Solid State Drive) – Твердотільний накопичувач.

TID (англ. Tuple Identifier) – Ідентифікатор кортежу.

TPS (англ. Transactions Per Second) – Кількість транзакцій за секунду.

ULID (англ. Universally Unique Lexicographically Sortable Identifier) – Універсальний унікальний лексикографічно сортований ідентифікатор.

UUID (англ. Universally Unique Identifier) – Універсальний унікальний ідентифікатор.

WAL (англ. Write-Ahead Logging) – Журнал передзапису.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах високих навантажень на системи керування базами даних, зокрема в середовищах з великим обсягом транзакцій та запитів, зростає потреба в ефективних стратегіях покращення їхньої продуктивності. Проблеми з масштабованістю, продуктивністю і надійністю можуть суттєво вплинути на бізнес-процеси і якість обслуговування користувачів.

Проблеми якості та продуктивності роботи СКБД досліджували багато фахівців. Так, у роботі Римарчука О. [1] розглянуто підхід до підвищення продуктивності баз даних в умовах обчислювального кластеру. Запропоновано симбіоз мов програмування для оптимізації міжмодульної взаємодії та міграції процесів між серверами. Цей підхід дозволяє ефективно масштабувати обчислювальні ресурси. У статті Дерень А. [2] запропоновано модель оцінки якості СКБД, яка дозволяє обирати оптимальне рішення на основі уніфікованих критеріїв, таких як функціональна повнота, масштабованість і сумісність. Шаряк В. [3] у своєму дослідженні аналізує архітектури баз даних і пропонує методи оптимізації на основі таких характеристик, як швидкість пошуку даних і надлишковість. Також він наводить критерії ефективності та фактори, які впливають на підвищення ефективності БД.

Актуальність теми полягає в необхідності дослідити та узагальнити існуючі стратегії покращення продуктивності реляційних систем керування базами даних, щоб забезпечити їх оптимальне застосування у сучасних умовах.

Мета і задачі дослідження. Мета дослідження полягає в оцінці ефективності існуючих стратегій покращення продуктивності СКБД і визначенні їхнього впливу на ключові аспекти продуктивності. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

- проаналізувати значення оптимізації баз даних;
- дослідити фактори впливу на продуктивність баз даних;
- розробити методологію тестування та тестове середовище;
- здійснити практичне тестування стратегій оптимізації;

– систематизувати отримані результати.

Об’єкт дослідження: реляційні СКБД, зокрема PostgreSQL.

Предмет дослідження: стратегії і підходи до покращення продуктивності СКБД, включаючи методи оптимізації запитів, індексування, кешування, реплікацію, конфігурування та інші техніки.

Наукова новизна одержаних результатів полягає в комплексній оцінці впливу існуючих стратегій на різні аспекти продуктивності баз даних.

Практичне значення одержаних результатів: практичне значення результатів дослідження полягає в наданні рекомендацій для фахівців з управління базами даних і розробників програмного забезпечення. Розроблені рекомендації можуть бути використані для покращення продуктивності баз даних, зменшення затримок, підвищення надійності систем і оптимізації витрат на ресурси.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Значення продуктивності баз даних у сучасних інформаційних системах

Бази даних є ключовим компонентом будь-якої інформаційної системи, і їх ефективність безпосередньо впливає на загальну продуктивність системи. База даних – це впорядкована колекція структурованої інформації або даних, які зазвичай зберігаються в електронному вигляді на комп'ютерній системі. Її функціонування зазвичай забезпечується системою керування базами даних (СКБД). У комплексі дані, СКБД і пов’язані з ними програми утворюють систему бази даних, яку часто скорочено називають базою даних [4].

Сучасні інформаційні системи стикаються з величезними обсягами даних та складними запитами, що створює необхідність у постійній оптимізації для забезпечення їхньої швидкої і ефективної роботи.

Бази даних сприяють розвитку бізнесу у різних напрямках. Вони допомагають у вдосконаленні управління інформацією про персонал, економлять час, забезпечують ефективне управління даними клієнтів та інше. Використання SQL-баз даних і продумане управління БД дозволяють суттєво покращити продуктивність компанії. Таким чином, бази даних допомагають максимально використовувати можливості для вдосконалення бізнес-процесів і досягнення стратегічних цілей [5].

Ефективна організація та пошук даних є основою стабільної роботи баз даних. Завдяки структурованому зберіганню забезпечується швидкий і зручний доступ до необхідної інформації. Використання методів, таких як індексація й оптимізація запитів, значно покращує швидкість отримання даних. Також застосування технологій, як-от розділення на секції та кластеризація, сприяє ще більшій продуктивності. Ці підходи мінімізують час, необхідний для пошуку, що, у свою чергу, підвищує загальну ефективність роботи бази даних [6].

Скорочення й оптимізація процесів управління даними є вирішальними для покращення бізнес-аналітики. Завдяки впровадженню надійної бази даних

компанії можуть централізувати інформацію, забезпечити її точність і узгодженість. Це дозволяє аналітикам та керівникам оперативно отримувати потрібну інформацію для прийняття обґрунтованих рішень. Крім того, якісно спроектована база даних може автоматизувати рутинні задачі, як-от введення й очищення даних, заощаджуючи час та ресурси. Таким чином, компанії оптимізують свої операції та підвищують конкурентоспроможність у сучасному цифровому середовищі [6].

Аналітика в реальному часі відіграє важливу роль у сучасному бізнесі. Вона дозволяє компаніям отримувати цінні інсайди та швидко приймати рішення. Аналізуючи дані в момент їх надходження, бізнеси можуть виявляти тенденції, фіксувати аномалії та адаптуватися до змін ринкових умов. Це сприяє підвищенню ефективності операцій, покращенню досвіду клієнтів і збереженню конкурентної переваги.

В умовах постійного збільшення обсягів даних компанії потребують надійних систем, здатних виконувати складні запити та забезпечувати точні результати [6]. Знання переваг і недоліків різних стратегій оптимізації допомагає уникнути проблем і забезпечити ефективність роботи системи. Зокрема, це дозволяє досягти зменшення часу виконання запитів, зниження навантаження на сервери і підвищення швидкості відгуку системи. Неправильне застосування стратегій оптимізації може не лише не поліпшити, а навіть погіршити продуктивність системи, що може призвести до затримок, перевантаження серверів і збоїв у роботі.

1.2 Показники продуктивності баз даних

Продуктивність бази даних визначає, наскільки ефективно система бази даних може обробляти запити, транзакції та інші операції. Це включає швидкість, оперативність та загальну ефективність операцій з базою даних [7].

Показники продуктивності баз даних дають змогу оцінювати швидкість виконання запитів, ефективність використання ресурсів та надійність системи.

До ключових показників можна віднести наступні:

- час відгуку;
- пропускна здатність;
- затримка;
- використання процесора;
- використання пам’яті;
- дисковий ввід/вивід;
- кешування;
- ефективність індексів;
- статистика блокувань;
- стан реплікації;
- аналіз плану виконання запитів.

Розуміння ключових метрик, таких як час відгуку, пропускна здатність, утилізація ресурсів і затримка, допомагає ідентифікувати вузькі місця та оптимізувати базу даних для задоволення зростаючих потреб користувачів. Ці показники також відіграють важливу роль у виборі підходів до оптимізації, налаштування апаратного забезпечення та оцінки впливу нових технологій на продуктивність.

Час відгуку відображає проміжок часу від моменту надсилання запиту до бази даних до отримання результатів. У реальному часі, наприклад, у фінансових операціях чи під час аварійних реагувань, навіть різниця в кілька мілісекунд може бути критичною [8].

Пропускна здатність визначає обсяг даних, які система може ефективно обробити за певний час. Зазвичай її вимірюють у транзакціях на секунду (TPS) або операціях на секунду. Висока пропускна здатність свідчить про здатність бази даних швидко обробляти великі обсяги запитів [8].

Затримка – ця метрика відображає середній час, необхідний для обробки однієї транзакції. Вона включає затримки, спричинені введенням/виведенням, блокуванням ресурсів та іншими факторами. Менше значення затримки вказує на швидшу обробку транзакцій.

Використання процесора – це показник, який відображає відсоток ресурсів процесора зайнятий базою даних. Високе завантаження може свідчити про неефективні запити або недостатність апаратних ресурсів [9].

Використання пам’яті – це показник, який визначає обсяг оперативної пам'яті, що використовується базою даних. Адекватний обсяг пам’яті важливий для кешування даних і прискорення запитів. Надмірне використання може спричинити обмін сторінками, що знижує продуктивність [9].

Дисковий ввід/вивід охоплює операції читання та запису на диск, де зберігаються дані. Високий рівень дискових операцій може свідчити про неефективне кешування. Частий доступ до диска сповільнює роботу системи [8].

Коефіцієнт попадання в кеш – цей показник визначає ефективність вилучення даних із кешу [10]. У PostgreSQL він відображає відсоток сторінок, отриманих із кешу спільної пам'яті. Високе значення означає менше звернень до диска [11].

Ефективність індексів – відображає, наскільки ефективно використовуються індекси для пришвидшення запитів. Добре спроектовані індекси можуть значно скоротити час виконання запитів, але поганий дизайн або надмірне використання можуть сповільнити систему [9].

Статистика блокувань – PostgreSQL застосовує механізми блокування для підтримки цілісності даних і управління паралельністю. Моніторинг таких метрик, як конфлікти блокувань, тупикові ситуації та тайм-аути, допомагає діагностувати проблеми [12].

Стан реплікації – для систем із реплікацією моніторинг цього показника забезпечує консистентність і доступність даних. Наприклад, затримка реплікації відображає різницю між основною та реплікованою базами даних [12].

Аналіз плану виконання запитів – включає перевірку стратегії виконання запиту базою даних. Це допомагає виявити неефективності та оптимізувати продуктивність шляхом налаштування запитів [9].

1.3 Фактори, що впливають на продуктивність баз даних

Фактори, що впливають на продуктивність баз даних, можна розділити на кілька категорій, кожна з яких відіграє важливу роль у забезпеченні швидкості та ефективності роботи системи:

1. Апаратні ресурси. Швидкість і кількість ядер процесора впливають на обробку запитів та виконання складних обчислень. Достатній обсяг оперативної пам’яті важливий для кешування даних та запитів. Тип і швидкість дисків (HDD, SSD, NVMe) визначають, як швидко дані зчитуються та записуються. Мережа, в свою чергу, визначає швидкість передачі даних між клієнтами та сервером може стати вузьким місцем для продуктивності [13].

2. Архітектура бази даних. Підбір оптимальних типів даних і їх адаптація до ваших конкретних потреб може забезпечити ефективніше використання пам'яті, пришвидшення виконання запитів і загальне підвищення продуктивності бази даних [14]. Правильне індексування прискорює пошук даних, але надлишок індексів може створювати додаткове навантаження при записі даних. Баланс між нормалізацією та денормалізацією впливає на швидкість виконання запитів. Розподіл даних між кількома вузлами забезпечує масштабованість і доступність, але додає складності.

3. Програмні аспекти. Некоректно написані або неоптимізовані SQL-запити можуть уповільнювати виконання. Ефективне використання кешу, як в оперативній пам’яті, так і в спеціалізованих інструментах (наприклад, Redis), значно прискорює роботу. Аналіз та оптимізація плану виконання запитів можуть усунути «вузькі місця».

4. Паралелізм та блокування. У PostgreSQL або інших системах, блокування може виникати через одночасний доступ до одних і тих самих даних, що впливає на швидкість [13]. Також часта зміна стану транзакцій або тривалі транзакції можуть створювати затримки.

5. Масштабованість і обсяги даних. Великі обсяги даних можуть значно уповільнювати операції бази даних, якщо не застосовуються ефективні підходи до управління даними, такі як архівація старих записів або поділ таблиць на

менші сегменти. Щоб подолати ці виклики, можна використовувати масштабування, яке передбачає два основних підходи: вертикальне – збільшення ресурсів одного сервера і горизонтальне – розподіл навантаження між кількома серверами, що працюють разом.

6. Системні параметри та конфігурація. Коригування різних налаштувань дозволяє оптимізувати роботу бази даних і підвищити її загальну продуктивність [14]. Такі параметри, як розмір буферів, частота контрольних точок, налаштування WAL впливають на продуктивність. Крім того, версія програмного забезпечення також має значення: оновлення можуть містити оптимізації, що підвищують ефективність, але водночас створюють ризики сумісності, які потрібно враховувати.

7. Безпека та резервне копіювання. Шифрування даних підвищує рівень захисту, проте може знижувати швидкість доступу. Процеси створення резервних копій повинні бути ретельно оптимізованими, щоб звести до мінімуму їхній вплив на основні операції бази даних, забезпечуючи при цьому надійність та доступність даних.

1.4 Висновок до першого розділу

У першому розділі кваліфікаційної роботи було висвітлено значення продуктивності баз даних у сучасних інформаційних системах, визначено основні показники продуктивності та проаналізовано фактори, які впливають на ефективність їхньої роботи.

2 СТРАТЕГІЇ ПОКРАЩЕННЯ ПРОДУКТИВНОСТІ СИСТЕМ КЕРУВАННЯ БАЗАМИ ДАНИХ

2.1 Внутрішня структура та механізми PostgreSQL

PostgreSQL – об'єктно-реляційна система управління базами даних із підтримкою ACID-транзакцій. Вона забезпечує автоматично оновлювані та матеріалізовані подання, тригери, зовнішні ключі та збережені процедури. PostgreSQL сумісна з основними операційними системами, такими як Windows, Linux, macOS, і підходить для різних завдань – від роботи на окремих машинах до сховищ даних чи багатокористувацьких сервісів [15].

2.1.1 Ключові аспекти архітектури PostgreSQL

Архітектура PostgreSQL дотримується моделі клієнт-сервер (див. рис. 2.1). Його основна програма працює як сервіс, відповідальний за визначення структур даних, зберігання даних і обробку запитів [16]. Цей демон прослуховує підключення від клієнтів, які можуть пройти автентифікацію, а потім надсилати інструкції на сервер.

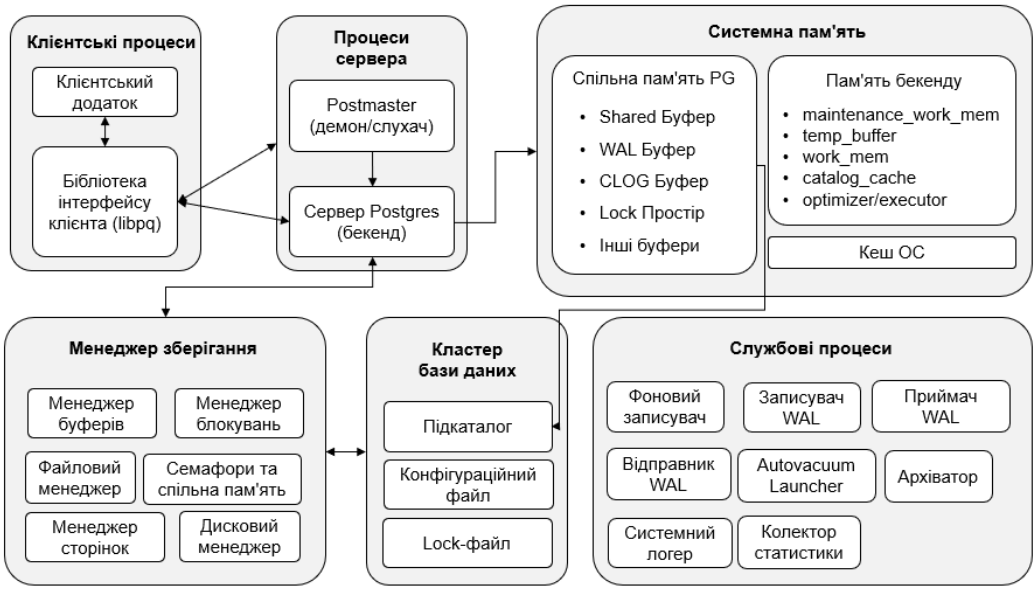


Рисунок 2.1 – Узагальнена архітектура PostgreSQL

Спільна пам’ять PostgreSQL базується на двох основних компонентах: спільному буфері (shared buffer) та буфері WAL [17].

Спільний буфер використовується як кеш для операцій введення-виведення. Процеси читають і записують дані безпосередньо в цей буфер. Якщо потрібно отримати кортеж (запис), процес звертається до спільного буфера. Часто використовувані дані зберігаються в ньому, що мінімізує звернення до диска. Дискові операції виконуються лише тоді, коли даних немає в буфері, що значно підвищує продуктивність і скорочує час відгуку.

WAL – це журнал усіх транзакцій, які виконуються в базі даних. Він забезпечує відмовостійкість PostgreSQL, дозволяючи відновлювати незавершені транзакції після збою системи та зберігати цілісність бази. Для підвищення швидкодії використовується буфер WAL, де дані тимчасово зберігаються перед записом на диск. Ці записи виконує фоновий процес [17].

PostgreSQL підтримує чотири основні типи процесів:

1. Postmaster-процес – головний контролер, який керує підключеннями клієнтів і запускає внутрішні процеси. Він обробляє запити на підключення та забезпечує зв’язок між клієнтом і сервером.

2. Бекенд-процеси – запускаються для кожного клієнтського підключення. Вони відповідають за виконання запитів і обробку транзакцій.

3. Фонові процеси – виконують системні завдання, не пов’язані з конкретними клієнтами. Серед них: обслуговування кешу, оновлення журналів, системне адміністрування.

4. Клієнтські процеси – безпосередньо взаємодіють із додатками, забезпечуючи підключення та обмін даними.

2.1.2 Життєвий цикл запиту

Механізм виконання запитів PostgreSQL доволі складний, але розуміння його роботи допомагає максимально ефективно використовувати базу даних. Запит, отриманий сервером для виконання, проходить кілька етапів (див. рис. 2.2).

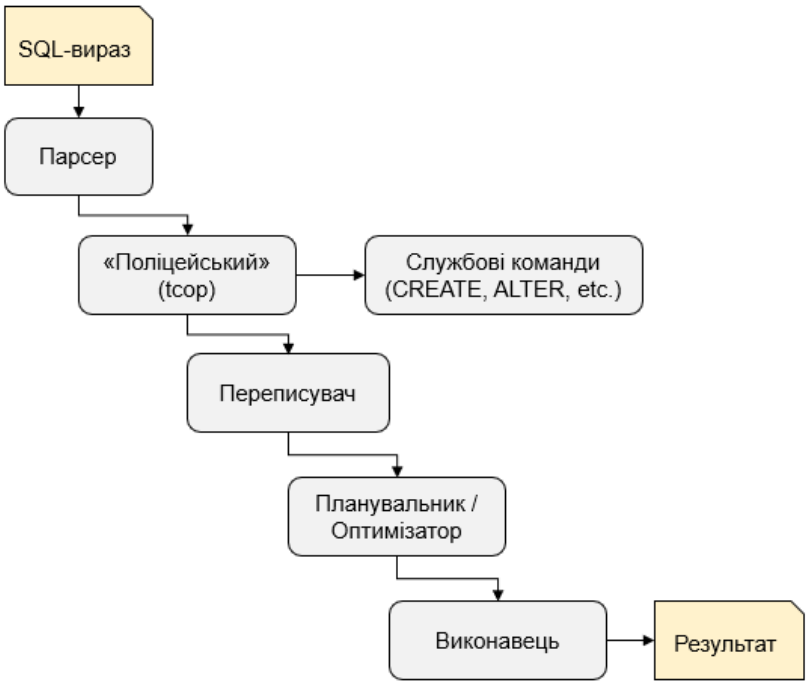


Рисунок 2.2 – Етапи виконання запиту

На першому етапі спеціальний компонент – парсер – аналізує SQL-запит. Він перевіряє його на наявність синтаксичних помилок. У разі помилки виконання запиту припиняється. Якщо запит коректний, парсер розбиває його на основні частини, наприклад: список задіяних таблиць і колонок, умови фільтрації, сортування тощо [18].

Далі «поліцейський» відділяє службові команди (ALTER, CREATE, DROP, GRANT тощо) від решти [19]. Після цього, запит переходить до етапу переписування. На цьому етапі переписувач застосовує синтаксичні правила для перетворення вихідного SQL-запиту у форму, яку буде виконано. Зокрема, тут застосовуються правила та тригери. Після завершення цього етапу формується остаточний запит, який переходить до наступного кроку [18].

На етапі оптимізації запит обробляється оптимізатором, завданням якого є знайти найшвидший спосіб доступу до даних. Це може включати використання індексів, прямого доступу або інших методів. Оптимізатор має обрати найефективніший метод із мінімальними витратами часу й ресурсів, що є досить складним завданням [18].

Після вибору оптимального способу доступу до даних запит переходить до етапу виконання. Виконанням займається компонент-виконавець, який безпосередньо звертається до сховища та виконує операції читання, запису або зміни даних, використовуючи методи, визначені оптимізатором [18].

2.1.3 План запиту

План запиту – це послідовність вузлів (або кроків), які використовуються базою даних для доступу до даних. Розуміння плану запиту є ключовим для розуміння продуктивності запиту.

Кожен вузол має певний тип і може мати один або більше дочірніх вузлів. Які саме дочірні вузли залежать від типу вузла. Різні типи вузлів поводяться по-різному, але загальний механізм однаковий: батьківський вузол отримує дані зі своїх дочірніх вузлів рядок за рядком. Діти або створюють дані безпосередньо (наприклад, зчитуючи їх із таблиць), або отримують від своїх дітей [20].

Щоб побудувати план виконання, PostgreSQL аналізує запит і оцінює кілька можливих варіантів його виконання. Для цього система розраховує вартість кожного базового вузла, а потім передає ці значення вгору по дереву, щоб визначити загальну вартість внутрішніх вузлів і, зрештою, кореневого вузла. PostgreSQL оцінює кілька планів виконання та обирає найбільш економний на основі заданих налаштувань вартості, доступних індексів і евристичних правил, заснованих на зібраній статистиці бази даних [20].

Типи вузлів можна умовно розділити на три категорії [21]:

– вузли сканування: створюють рядки на основі даних таблиці;

– вузли з'єднання: об'єднують рядки з дочірніх вузлів;

– інші вузли: забезпечують широкий спектр функціоналу (наприклад, агрегація, обмеження, групування тощо).

Вузли сканування представляють методи, які використовує база даних для отримання рядків із джерела даних під час виконання запиту. Кожен вузол сканування в плані виконання відповідає певній стратегії доступу для отримання рядків із таблиці або пов’язаних з нею індексів.

Поширені типи вузлів сканування:

- послідовне сканування;
- сканування за індексом;
- сканування лише за індексом;
- сканування за bitmap-індексом.

Кожен із цих методів сканування є корисним залежно від запиту та інших параметрів, таких як кардинальність таблиці, селективність, вартість дискового вводу-виводу, вартість випадкового доступу та послідовного доступу [22].

Послідовне сканування, передбачає послідовне читання всіх записів на всіх сторінках таблиці. Наприклад, якщо таблиця містить 100 сторінок, а на кожній сторінці знаходиться по 1000 записів, то під час послідовного сканування буде оброблено 100 * 1000 записів, щоб знайти ті, які відповідають рівню ізоляції та присудка (predicate clause). Таким чином, навіть якщо потрібний лише один запис, все одно буде перевірено 100 тисяч записів [22]. Цей метод зазвичай використовується, коли в таблиці відсутні індекси на стовпцях [23].

Сканування за індексом використовує структуру даних, що відповідає певному індексу, для швидкого знаходження записів, які відповідають умовам запиту [22]. Сканування за індексом складається з двох кроків: перший – отримання розташування рядка з індексу, а другий – збір фактичних даних з купи або сторінок таблиці [24].

Незважаючи на те, що сканування за індексом таке ефективне, ми не можемо його завжди використовувати. Це пояснюється тим, що кожна операція має свою ціну. У випадку сканування за індексом здійснюється випадковий доступ до даних, оскільки для кожного запису, знайденого в індексі, потрібно виконати окремий запит до heap-сховища. Натомість послідовне сканування працює з послідовним доступом до даних, яке зазвичай приблизно в чотири рази швидше за випадковий доступ [22].

Тобто, коли вибірковість висока, послідовне сканування є кращим варіантом, оскільки сканування всієї таблиці за один прохід більш ефективне. Якщо запит повертає лише кілька записів, індекс дозволяє швидко знайти потрібні дані, тому тут сканування за індексом буде ефективнішим.

Сканування лише за індексом є особливим типом сканування за індексом. Якщо запит стосується лише стовпців, які входять до індексу, PostgreSQL достатньо розумний, щоб уникнути звернення до heap-сховища і отримати всю необхідну інформацію безпосередньо з індексу [18].

У випадку сканування на основі bitmap-індексу, PostgreSQL створює в пам’яті бітову карту («bitmap»), що вказує на місця записів, які відповідають умовам запиту. Ця бітова карта потім використовується для доступу до відповідних записів [18].

2.1.4 Роль команди «EXPLAIN»

Якщо потрібно проаналізувати повне дерево плану, його можна вивести в журнал сервера, активувавши параметр «debug_print_plan». Однак на практиці зазвичай достатньо переглянути текстову версію плану, яку надає команда «EXPLAIN» [25].

Команда «EXPLAIN» додається перед запитом, щоб показати, які операції PostgreSQL планує виконати, скільки рядків кожна з них зачіпає та скільки часу вони можуть займати. Це основний інструмент для діагностики продуктивності, який допомагає виявити, що уповільнює виконання запиту [26].

«EXPLAIN» оцінює вартість кожної операції у дереві виконання, не виконуючи сам запит. Завдяки тому, що PostgreSQL постійно збирає статистику щодо запитів, ці оцінки є досить точними для виявлення проблем з продуктивністю. Якщо ж потрібно отримати точні дані про вартість, виконуючи запит, використовується команда «EXPLAIN ANALYZE» [26].

2.2 Проектування ефективної схеми бази даних

Проектування ефективної схеми бази даних є важливим для ефективного управління даними. Добре спроектована схема забезпечує організацію складних даних, вирішує питання цілісності даних, продуктивності, масштабованості та безпеки, зменшує дублювання даних і покращує можливості для їх аналізу.

Важливо правильно вибрати типи даних для кожного стовпця, оскільки це може вплинути на ефективність зберігання та обробки даних. Наприклад, використання більш компактних типів даних для числових значень або дат може зменшити обсяг пам'яті, яку потрібно для зберігання цих даних.

Проектування бази даних повинно враховувати майбутній ріст обсягу даних. Система повинна бути готовою до збільшення навантаження без значного перероблення. Проектування – це не одноразовий процес, тому як тільки надходять нові вимоги до додатку або зміни в ньому, необхідно регулярно переглядати і вдосконалювати модель бази даних, щоб вона відповідала цим вимогам [27].

Визначення правильних зв'язків між таблицями допомагає зберегти логічну структуру даних і забезпечує швидкий доступ до необхідної інформації за допомогою запитів з об'єднанням таблиць.

Використання діаграм сутностей та зв’язків (ERD) допомагає візуалізувати структури даних і виявляти потенційні проблеми на етапі проектування. Це дозволяє зрозуміти, як різні елементи бази даних взаємодіють між собою і чи є логічні помилки в їхньому зв’язку [27].

2.3 Індексація

Індекс бази даних – це структура даних, яка дозволяє прискорити пошук інформації в таблиці за рахунок створення додаткових записів і використання додаткового простору для підтримки цих структур. Індекси дозволяють швидко знаходити потрібні дані без необхідності переглядати кожний рядок таблиці під час запиту [28].

Оскільки індекси впорядковані, їх також можна використовувати для прискорення операцій сортування, якщо потрібно отримати значення з певної впорядкованої частини таблиці [29].

Існує кілька типів індексів бази даних. Найбільш популярні з них:

- B-дерево;
- Хеш-індекс;

- GIST;
- GIN;
- BRIN.

Незалежно від типу індексу, всі вони виконують однакову базову функцію: пов’язують ключ (наприклад, значення індексованої колонки) з рядками таблиці, що містять цей ключ. Рядки ідентифікуються за допомогою TID, який включає номер блоку у файлі та позицію рядка в цьому блоці. Це дозволяє швидко знаходити потрібні рядки за відомим ключем або частиною інформації про нього, без повного сканування таблиці [30].

Хоча індекси прискорюють доступ до даних, вони вимагають додаткових витрат на обслуговування. Під час кожної операції вставки, видалення або оновлення рядків таблиці індекси також повинні оновлюватися в межах тієї ж транзакції.

У базах даних із частими оновленнями чи вставками індекси можуть стати надто «дорогими» у використанні. Тому важливо уникати надмірного індексування, знаходячи баланс між кількістю індексів і навантаженням на записи [30].

Коли не варто створювати індекси:

- поля з малою кількістю унікальних значень (низька кардинальність). Наприклад, індексація колонки булевої колонки «isActive» (значення 0 або 1) часто неефективне, оскільки індекс мало допомагає у фільтрації;
- рідко використовувані колонки: якщо колонка майже не використовується у запитах, індексація не дає суттєвих переваг, але збільшує витрати на обслуговування;
- часті вставки/оновлення: якщо колонка часто оновлюється, підтримка індексу може суттєво уповільнити операції «INSERT» та «UPDATE».

2.1.1 Індекс B-дерева

В інформатиці B-дерево – це самобалансуюча деревовидна структура даних, яка зберігає відсортовані дані та дозволяє здійснювати пошук,

послідовний доступ, вставки та видалення за логарифмічний час. B-дерево узагальнює бінарне дерево пошуку [31].

У контексті баз даних B-дерево широко використовується для створення індексів. Візуальне представлення даного типу індексу в PostgreSQL подано на рисунку 2.3.

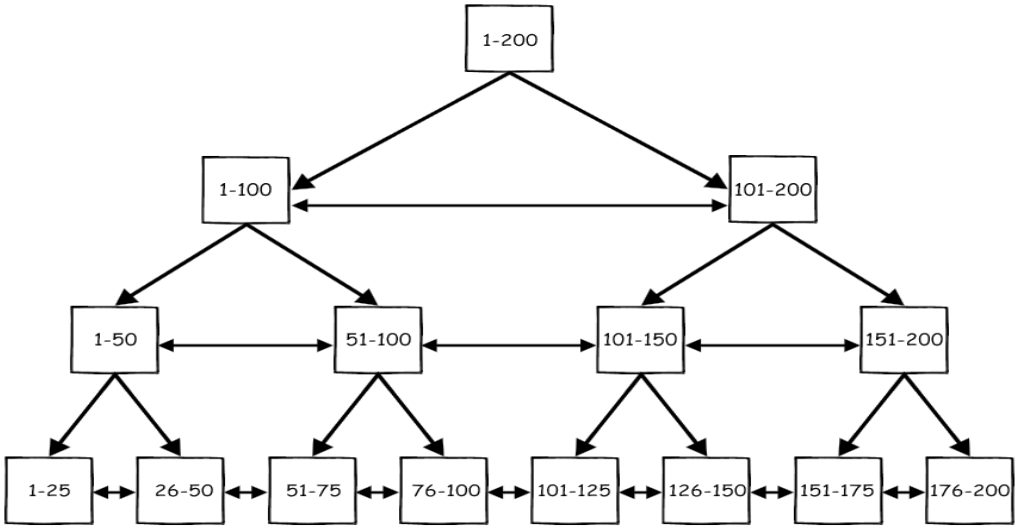


Рисунок 2.3 – Візуальне представлення індексу B-дерева у PostgreSQL

Найперша сторінка індексу є метасторінкою, яка посилається на корінь індексу. Внутрішні вузли розташовані під коренем, а листові сторінки – у самому нижньому рядку. Листові вузли містять посилання на рядки таблиці (TID) [32].

У PostgreSQL індекси на основі B-дерева допомагають оптимізувати запити, що включають порівняння за операторами «<», «<=», «=», «>=», або «>». Це робить B-дерева одним із ключових інструментів для швидкого доступу до впорядкованих даних. Вони також підтримують запити з конструкціями «BETWEEN» та «IN», та можуть обробляти запити, які шукають «NULL» значення в полі. B-дерева працюють з «LIKE» та «ILIKE», якщо шаблон починається з конкретного значення. Наприклад, «col LIKE 'abc%» підтримується, тоді як «col LIKE '"%xyz'» – ні [33].

Індекси B-дерева дозволяють повертати дані в упорядкованому вигляді. Це може бути корисним, якщо запит вимагає сортування. Це не завжди швидше, ніж просте сканування та сортування, але часто корисно [33].

Ключові властивості індексу B-дерева [32]:

– збалансованість: усі листові сторінки однаково віддалені від кореня дерева. Це забезпечує однаковий час доступу до будь-якого значення, незалежно від його розташування;

– багатогілковість: кожна сторінка (базова одиниця зберігання даних у PostgreSQL, типовий розмір сторінки – 8 КБ) дерева містить велику кількість записів, що зменшує глибину дерева. Навіть для дуже великих таблиць глибина зазвичай не перевищує 4–5 рівнів;

– сортування та з'єднання сторінок: дані індексу впорядковані у невід'ємному порядку, а сторінки одного рівня з'єднані двонаправленим списком. Це дозволяє швидко отримувати впорядковані набори даних, обходячи сторінки без повернення до кореня дерева.

2.1.2 Хеш-індекс

Хеш-індекси – це спеціалізована структура даних, призначена для швидкого пошуку одного запису за умовою рівності [34]. Хеш-індекси зберігають 32-бітний хеш-код, отриманий із значення індексованої колонки. Таким чином, ці індекси підтримують лише прості порівняння на рівність. Оптимізатор запитів використовує хеш-індекс у випадках, коли в умовах запиту присутній оператор рівності для індексованої колонки [33].

Через певні обмеження та той факт, що транзакційна підтримка хеш-індексів була додана відносно недавно, вони ще не набули широкого застосування в більшості баз даних PostgreSQL. Проте, завдяки їхній ефективності у пошуку окремих записів, хеш-індекси можуть бути дуже корисними в середовищах, де основним завданням є швидкий доступ до окремих даних [34].

2.1.3 GIST-індекс

Індекси GIST не є одним конкретним видом індексу, а радше інфраструктурою, в рамках якої можна реалізувати різноманітні стратегії індексації [33]. Їхня основна ідея полягає у створенні модульної платформи, де можна визначати оператори та функції для індексації даних різних структур. Наприклад, SP-GIST є просторовим індексом, який використовується в географічних застосунках [18].

Структура дерева GIST складається зі сторінок вузлів, які в свою чергу містять індексні записи. Листові вузли містять записи, що зазвичай включають предикат (логічний вираз) і посилання на відповідний запис у таблиці. Індексовані дані повинні відповідати цьому предикату. Внутрішні вузли містять записи з предикатом і посиланням на дочірній вузол. Усі дані в дочірньому піддереві мають задовольняти умови цього предиката [35].

Іншими словами, предикат внутрішнього вузла є узагальненим предикатом для всіх дочірніх вузлів. Ця властивість замінює звичайне впорядкування даних, яке використовується у B-деревах, і надає GIST-індексам більшої гнучкості в роботі з нестандартними структурами даних [35].

2.1.4 GIN-індекс

GIN – це тип індексу, який замість вказівки на окремий рядок у таблиці пов’язує кілька значень або навіть масиви значень. Цей індекс часто використовується в сценаріях повнотекстового пошуку, де індексується текст із багатьма повторюваними ключами (наприклад, однаковими словами чи термінами), які посилаються на різні місця (наприклад, однакове слово в різних фразах або рядках) [18].

У сучасних програмах часто застосовуються складні типи даних, такі як JSONB, масиви або функції повнотекстового пошуку, які не підтримуються простими B-деревами. Наприклад, B-дерево не підходить для індексації стовпців типу JSONB, де потрібен доступ до окремих компонентів. Тут на допомогу

приходить GIN. Індекс GIN створений для роботи з подільними типами даних, забезпечуючи швидкий пошук елементів масивів, лексем тексту тощо [36].

Для підвищення ефективності повнотекстового пошуку в базах даних використовують особливий вид індексування – триаграмні індекси. Дані індекси побудовані на основі GIN або GiST. Вони працюють шляхом розбиття тексту на менші одиниці, які називаються триаграмами, і збереження їх в індексі. Це дозволяє базі даних уникати посимвольного пошуку в тексті та швидше знаходити потрібну інформацію [37].

PostgreSQL підтримує триаграмні індекси через розширення «pg_trgm». Проте, щоб використовувати такий індекс, необхідно виконати дві умови [38]:

- створений індекс повинен бути індексом GIN або GiST;
- для індексу повинен бути визначений відповідний клас операторів..

Однією з головних переваг триаграмних індексів є можливість їх використання у запитах з умовами «LIKE» або «ILIKE», що дозволяє уникнути складної конфігурації повнотекстового пошуку або змін у запитах, забезпечуючи водночас високу продуктивність.

2.1.5 BRIN-індекс

BRIN – це індекс, який працює на основі діапазонів значень у блоках пам'яті. Кожен блок зберігає лише мінімальне та максимальне значення, і індекс записує ці пари для кожного блоку на диску. Коли виконується запит, індекс вказує на блок, де може бути знайдене шукане значення, але для кожного запису цього блоку все одно потрібно здійснити перевірку [18].

Цей тип індексу менш точний, ніж B-дерево, і тому має назву «втратний» що означає, що він не завжди дає абсолютно точні результати – можуть бути деякі втрати. Проте, порівняно з іншими індексами, BRIN є набагато меншим за розміром, оскільки зберігає лише кілька значень для кожного блоку [18].

BRIN-індекси є особливо корисними, коли існує висока кореляція між фізичним порядком даних на диску і їх значенням у стовпці, наприклад, для таблиць з часовими мітками або значеннями, що зберігаються у порядку

зростання чи спадання. BRIN-індекс – це невелика таблиця, яка пов’язує діапазон значень із діапазоном сторінок у порядку таблиці (див. рис. 4). Створення індексу вимагає лише одного сканування таблиці, тому порівняно зі створенням такої структури, як B-дерево, це дуже швидко [39].

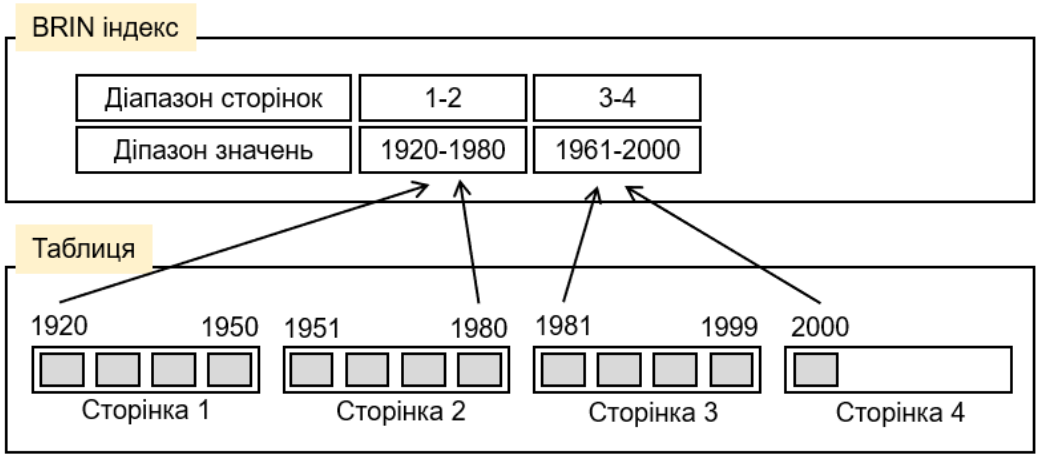


Рисунок 4 – BRIN-індекс

BRIN займає значно менше місця порівняно з іншими індексами, оскільки зберігаються тільки мінімальні та максимальні значення для блоків [39].

2.4 Оптимізація запитів

Оптимізація запитів є ключовим аспектом систем керування базами даних, що полягає у виборі найефективнішої стратегії виконання запиту. Це дозволяє зменшити використання системних ресурсів та підвищити швидкість отримання результатів. Одним з основних завдань є мінімізація витрат часу та пам'яті при виконанні запиту, що досягається через оптимізацію процесу пошуку, обробки та повернення даних [40].

Перевантаження запитів вайлд-кардами (наприклад, SELECT *) може знижувати їх ефективність, оскільки запитує зайві стовпці. Натомість слід вказувати лише ті стовпці, які дійсно необхідні для виконання запиту [41].

СКБД можуть мати труднощі з оптимізацією запитів, що включають багато з'єднань, підзапитів або складних умов. У таких випадках корисно спростити

запити, розбиваючи їх на менші, більш зосереджені частини для покращення продуктивності [42].

Порядок з'єднань також може впливати на продуктивність запитів у PostgreSQL, особливо при складних запитах з великими наборами даних.

Також важливо правильно вибирати тип з'єднання – «INNER JOIN», «LEFT JOIN» чи інші, і стратегії з'єднання – наприклад, вкладені цикли або хеш-з'єднання, оскільки це може вплинути на ефективність запиту. Хоча PostgreSQL зазвичай самостійно вибирає найкращу стратегію, впливати на її вибір можна за допомогою переписування запиту або підказок. Таким чином, застосування цих стратегій може значно покращити швидкість запитів і зменшити навантаження на ресурси системи [42].

При з'єднанні таблиць корисно починати з менших таблиць або тих, що фільтруються за допомогою умов, щоб зменшити розмір проміжних результатів на ранніх етапах. Такий підхід може знизити використання пам'яті та процесорних ресурсів, обмежуючи кількість даних, що оброблятимуться в подальших з'єднаннях. Планувальник запитів зазвичай оптимізує порядок з'єднань, оцінюючи вартість виконання. Проте в деяких випадках, наприклад, при складних запитах з численними з'єднаннями або недостатніх статистичних даних, може бути корисним вручну вказати порядок з'єднань або використовувати підказки, такі як «JOIN LATERAL».

Переписування запитів може значно покращити швидкість виконання запитів. Використання оператора «EXISTS» замість «IN» для підзапитів часто працює ефективніше, оскільки «EXISTS» припиняє пошук, як тільки знаходить перший збіг, що економить час виконання, в той час як «IN» перевіряє всі можливі значення. Також важливо обмежувати область підзапитів, розміщуючи їх у «WHERE» або «JOIN», а не у «FROM», оскільки це допомагає уникнути створення тимчасових таблиць та зменшує складність запиту.

Ще одним важливим аспектом є зменшення кількості вкладених запитів. Якщо це можливо, варто розгортати складні підзапити, щоб зменшити час обробки і покращити використання індексів. Крім того, для агрегацій варто розумно використовувати «HAVING» – тільки для фільтрації після агрегації.

Якщо фільтрацію можна застосувати до агрегації, краще це зробити на етапі «WHERE».

2.5 Нормалізація та денормалізація бази даних

Нормалізація – це метод проектування бази даних, що зменшує дублювання даних та усуває небажані проблеми, такі як аномалії при вставці, оновленні та видаленні [43].

Хоча нормалізація та денормалізація загалом вважається правилом для проектування реляційних баз даних, іноді проектувальники можуть звертатися до денормалізації бази даних, щоб покращити продуктивність [44].

Денормалізація може бути описана як процес зменшення ступеня нормалізації з метою покращення продуктивності обробки запитів при ретельному розумінні вимог до додатку. База даних може швидко обробляти операції читання, зменшуючи кількість з’єднань таблиць, необхідних для отримання даних [45].

Додавання зайвих стовпців у таблицю, як процес денормалізації, може усунути потребу в з’єднаннях і покращити продуктивність запитів. Однак адміністраторам слід бути обережними, щоб переконатися, що додані стовпці не компрометують цілісність даних [45].

2.6 Матеріалізовані представлення

Матеріалізовані представлення – це попередньо обчислені та збережені результати SQL-запиту, які базуються на одній або декількох таблицях. Вони є окремими об'єктами бази даних, які займають власний дисковий простір у СКБД [46].

Матеріалізовані представлення прискорюють виконання запитів, зберігаючи попередньо обчислені результати складних запитів. Це усуває потребу виконувати ресурсоємні обчислення на вихідних таблицях, що забезпечує швидкий доступ до результатів. Завдяки цьому їх часто

використовують для побудови звітів і отримання аналітичних інсайдів, оскільки подальші запити виконуються безпосередньо на цих представленнях [46].

Основна відмінність від звичайних представлень полягає в тому, що матеріалізовані представлення зберігають результати на диску. Виконання складних запитів може бути дорогим з точки зору ресурсів. Попереднє обчислення дозволяє працювати лише з підготовленим набором даних, без необхідності повторного виконання оригінального запиту [47].

Матеріалізовані представлення ефективні для складних і часто виконуваних запитів на великих обсягах даних, особливо коли дані змінюються рідко, оскільки витрати на оновлення таких представлень є мінімальними [46].

2.7 Управління ключами

Послідовні ідентифікатори, UUID і ULID є поширеними форматами первинного ключа.

Послідовний ідентифікатор, часто ціле число з автоматичним збільшенням, є послідовним і простим, що дозволяє легко посилатися на нього, але його легко передбачити. Такі ідентифікатори ефективні для індексування та зберігання. Їх послідовний характер забезпечує впорядкованість вставок індексів, що мінімізує розбиття сторінок у B-деревах і прискорює як читання, так і запис. Однак у розподілених системах центральність зростаючого цілого числа може створити вузькі місця.

UUID – це 128-бітні значення, часто представлені у вигляді рядків із 36 символів, що забезпечують високу унікальність у системах і програмах, але мають додаткові витрати на зберігання та індексування [48]. UUID через їх високу випадковість широко розподіляють значення в індексах, що призводить до непослідовних вставок. Ця випадковість спричиняє часті розбиття сторінок і високе використання пам’яті, що впливає на швидкість вставки, особливо у великих базах даних. Крім того, їхній більший розмір вимагає більше місця для зберігання та може збільшити час, необхідний для пошуку. Незважаючи на це,

UUID корисні у розподілених системах, адже пропонують сильну унікальність, що є неоціненним у децентралізованих середовищах.

ULID – це новіший формат, який поєднує унікальність UUID із лексикографічним сортуванням, що підвищує ефективність індексування та зберігання записів. ULID надають компроміс – вони зберігають глобальну унікальність, як UUID, але лексикографічно сортуються за часом створення, тобто вони частково зберігають порядок вставки [49]. Таке впорядкування на основі часу мінімізує розбиття сторінок, покращуючи продуктивність індексування та пошуку порівняно з UUID, і водночас підтримує розподілені та глобально унікальні ключі.

2.8 Реплікація та шардинг

Коли таблиця стає надто великою, може знизитись її продуктивність. У цьому випадку на допомогу приходить реплікаця даних. Реплікація даних полягає в розбитті великої таблиці на менші. Для клієнтської програми сервер все одно буде виглядати так, ніби вона працює з однією таблицею, однак наявність менших таблиць означає менші індекси, які мають більше шансів залишатися в пам'яті, що покращує продуктивність обробки даних. Крім того, менші таблиці дозволяють процесам вакуумування працювати швидше, оскільки вони мають менший обсяг для обробки, що зменшує час їх виконання [18].

Коли є кілька реплік бази даних, запити на читання можуть бути розподілені між різними репліками. Це зменшує навантаження на основну базу даних (майстер-сервер) і дозволяє обробляти більше запитів одночасно. Реплікація також може бути налаштована так, щоб забезпечити географічно розподілені сервери, що дозволяє зменшити затримки для користувачів, які знаходяться далеко від основного сервера.

Шардинг є трохи складнішим, ніж реплікація. Коли обсяг даних, які зберігаються, досягає певного розміру, масштабування стає великою проблемою продуктивності. Шардинг – це рішення, яке створено, щоб протидіяти цьому, розбиваючи базу даних на окремі сегменти на кількох машинах [50].

Кожен шард зберігає частину даних, що дозволяє розподіляти навантаження на кілька серверів. Замість того, щоб один сервер обробляв всі запити, дані розподіляються між кількома шардовими серверами. Це дозволяє кожному серверу працювати з меншими обсягами даних, зменшуючи час на обробку запитів. Крім того, коли дані розподілені між кількома шардовими серверами, запити можуть бути оброблені паралельно на різних серверах. Це дозволяє значно прискорити обробку запитів, оскільки кожен сервер працює лише з частиною даних.

2.9 Конфігурація бази даних

Однією з поширених помилок є використання серверів баз даних із параметрами за замовчуванням. Хоча ці налаштування тестувалися для певних середовищ, вони можуть бути неоптимальними для конкретних програм [51].

Конфігурація бази даних безпосередньо впливає на продуктивність, оптимізуючи розподіл пам'яті, ввід/вивід на диск та обробку запитів. Налаштування пам'яті дозволяє більш ефективно кешувати дані, що зменшує доступ до диску і пришвидшує операції читання. Оптимізація параметрів вводу/виводу на диск дозволяє планувальнику запитів враховувати доступні ресурси, що сприяє кращим стратегіям виконання запитів. Крім того, тонке налаштування параметрів, що стосуються ЦП, допомагає у паралельній обробці, що робить складні запити швидшими. В цілому, налаштування цих параметрів забезпечує ефективне використання системних ресурсів PostgreSQL, покращуючи час відгуку запитів і дозволяючи краще справлятися з великими навантаженнями.

Серед поширених параметрів для налаштування в PostgreSQL є: «shared_buffers», «effective_cache_size», «wal_buffers», «work_mem», «maintenance_work_mem», «synchronous_commit».

Кешування – це збереження даних у пам’яті (RAM) для швидшого доступу. Однією з основних причин низького рівня кешування таблиці є те, що базі даних не вистачає місця в її внутрішньому буфері кешу. Цей буфер використовується

для завантаження рядків таблиці з диска в пам'ять, і якщо місця в ньому недостатньо, сервер бази даних змушений постійно отримувати дані з диска [52].

Для баз даних PostgreSQL розмір кешу налаштовується через параметр «shared_buffers». Він визначає, скільки пам'яті машини можна виділити для зберігання даних у пам'яті. За замовчуванням це значення дуже низьке (128 МБ), оскільки деякі ядра не підтримують більше без зміни налаштувань ядра. Більше пам'яті може бути корисним для більших баз, але існує межа оптимізації, оскільки PostgreSQL також використовує кеш операційної системи. Чим більше пам'яті виділяється для «shared_buffers», тим менше пам'яті буде доступно для кешу операційної системи [52].

Параметр «effective_cache_size» відображає загальний обсяг дискового простору, доступного для кешування бази даних. Це значення зазвичай налаштовується як сума «shared_buffers» та розміру кешу диска операційної системи після запуску бази даних. Це зазвичай більше половини загальної системної пам'яті на типовому сервері бази даних. Цей параметр не виділяє пам'ять безпосередньо, а служить орієнтовним значенням для планувальника запитів, вказуючи, що ймовірно буде доступно для кешування [29].

PostgreSQL записує свої записи WAL у буфери, і потім ці буфери вивантажуються на диск. За замовчуванням розмір буфера, визначений параметром «wal_buffers», становить 16 МБ. Однак, якщо у багато одночасних підключень, то збільшення цього значення може покращити продуктивність [50]. «wal_buffers» покращує продуктивність у PostgreSQL, зменшуючи частоту і затримку записів на диск під час операцій з великою кількістю записів. Коли зміни в даних спочатку зберігаються в «wal_buffers», вони накопичуються там тимчасово, а не записуються відразу на диск. Така буферизація дозволяє обробляти кілька змін за один раз, зменшуючи витрати на ввід/вивід для кожної операції запису.

Параметр «work_mem» визначає обсяг пам'яті, який використовується запитами для операцій сортування [29]. Оскільки «work_mem» виділяється для кожної операції в межах кожної сесії, великі значення можуть призвести до надмірного використання пам'яті при великій кількості одночасних підключень.

Тому, якщо багато користувачів, які виконують операції сортування, система виділятиме пам'ять «work_mem» * n, де n – кількість операцій сортування для всіх користувачів.

«maintenance_work_mem» – це параметр пам'яті, що використовується для обслуговуючих задач. За замовчуванням його значення складає 64 МБ. Встановлення великого значення допомагає в таких завданнях, як «VACUUM», «RESTORE», «CREATE INDEX», «ADD FOREIGN KEY» та «ALTER TABLE» [53].

Параметр «synchronous_commit» визначає, що підтвердження транзакції буде очікувати запису WAL на диск, перш ніж повернути клієнту повідомлення про успішне завершення. Це компроміс між продуктивністю та надійністю. Якщо у додатку продуктивність важливіша за надійність, параметр «synchronous_commit» можна вимкнути. У такому разі між повідомленням про успіх і фактичним записом даних на диск виникає часовий розрив. Якщо сервер аварійно завершить роботу, дані можуть бути втрачені, навіть якщо клієнт отримав підтвердження. Вимкнення цієї опції дозволяє швидше завершувати транзакції, оскільки запис WAL-файлів на диск виконується асинхронно, але при цьому знижується надійність [53].

Також можна впливати на плани виконання запитів, змінюючи параметри оптимізатора. Наприклад, якщо для запиту використовується послідовне сканування, можна відключити його командою «SET ENABLE_SEQSCAN» у значення «off». Проте це не гарантує, що оптимізатор повністю виключить цей оператор, але він розглядатиме його як значно дорожчий [51].

Окрім цього, можна змінювати параметри вартості, такі як «CPU_OPERATOR_COST», «CPU_INDEX_TUPLE_COST», «CPU_TUPLE_COST», «RANDOM_PAGE_COST», та «EFFECTIVE_CACHE_SIZE», щоб впливати на роботу оптимізатора.

Щоб налаштувати параметри PostgreSQL відповідно до апаратного забезпечення, можна скористатися інструментом pgTune, який генерує оптимальні значення на основі конфігурації вашого обладнання. Проте будь-які зміни слід ретельно перевіряти, щоб уникнути можливого погіршення

продуктивності [51]. Варто також враховувати, чи використовується сервер виключно для бази даних або виконує інші завдання. У разі спільного використання ресурсу, коли на одному сервері працюють як база даних, так і інші сервіси, параметри потрібно налаштовувати з урахуванням цього [53].

2.10 Оптимізація апаратних ресурсів

Апаратне забезпечення відіграє ключову роль у продуктивності будь-якої бази даних. Від швидкості процесора до обсягу оперативної пам’яті, від типу накопичувача до пропускної здатності мережі – усі ці компоненти впливають на швидкість, надійність і масштабованість системи.

Процесор має значний вплив на продуктивність бази даних, особливо якщо у запитах використовується багато обчислень. Найшвидший спосіб отримати інструкції для бази даних – це безпосередньо через процесор. Розмір кешу L2 і L3 є ключовим фактором у цьому процесі, і чим більший їхній обсяг, тим краще [54].

Навіть якщо ресурси вашого процесора здаються надмірними, вони можуть бути корисними для таких цілей [54]:

- зменшення потреб у введенні/виведенні завдяки стисненню даних;
- компенсація вузьких місць у продуктивності пам’яті та сховища;
- підвищення безпеки шляхом стиснення резервних копій бази даних.

Також, чим більше пам’яті доступно для кешування, тим швидше процесор оброблятиме її, уникаючи простоїв, викликаних очікуванням операцій введення/виведення.

Додаткова пам’ять забезпечує такі переваги [54]:

- зниження вимог до введення/виведення;
- розширення розмірів буферних пулів;
- зменшення частоти контрольних точок.

Якщо програма орієнтована на інтенсивні операції читання чи запису, продуктивність значно покращується при використанні швидших накопичувачів, таких як NVMe або SSD [55].

Операції WAL можуть стати вузьким місцем у базах даних із високою інтенсивністю запису. Тому, використання окремого й швидкого накопичувача для WAL допомагає уникнути цієї проблеми, таким чином розділяючи диск WAL і диск даних [55].

Мережа є теж важливим елемент інфраструктури бази даних. Перевантаження мережі може призвести до затримок, падіння продуктивності, втрати пакетів і навіть до збоїв серверів. Під час налаштування мережі слід враховувати не лише локальний хост, а й клієнтів, які підключаються до бази даних із різних частин світу. Для додатків, чутливих до затримок, важливо розміщувати сервер додатків і базу даних якнайближче географічно [54].

2.11 Використання RAID рівнів

RAID – це стандартний підхід для вирішення обмежень продуктивності та надійності окремих жорстких дисків. Масив RAID об'єднує кілька дисків, зазвичай з однаковими характеристиками, в один набір, який працює як єдиний диск, але з покращеною продуктивністю, надійністю або тим і іншим [29].

Найшвидший рівень RAID – це RAID 0. Він розподіляє дані між кількома дисками без резервування, що дозволяє досягти високих швидкостей читання та запису. Кожен диск обробляє лише частину даних одночасно, що максимізує продуктивність. Але RAID 0 не має захисту від відмов: якщо хоча б один диск вийде з ладу, всі дані будуть втрачені.

Для середовищ, де потрібна і висока продуктивність, і надійність, найкращим варіантом є RAID 10. Він поєднує переваги RAID 0 – стріпування і RAID 1 – віддзеркалення. Дані розподіляються між дисками і одночасно віддзеркалюються на інших, що дає високу продуктивність і надійний захист. Але для цього потрібно мінімум чотири диски, і половина простору буде використана для резервування.

RAID 10 часто вибирають для баз даних і додатків з високими вимогами, оскільки він поєднує швидкість і надійність. RAID 0 підходить тільки для некритичних даних, де не потрібен захист від відмов.

2.12 Висновок до другого розділу

У другому розділі роботи було розглянуто ключові стратегії покращення продуктивності систем керування базами даних із фокусом на PostgreSQL. Описано внутрішню архітектуру цієї СКБД, що є основою для її оптимізації. Зокрема, модель клієнт-сервер, роль спільної пам’яті, та механізми виконання запитів. Увага була приділена життєвому циклу запиту та вибору оптимального плану виконання, який забезпечує економічне використання ресурсів.

Проектування ефективної схеми бази даних визнано однією з фундаментальних стратегій оптимізації. Також підкреслено важливість нормалізації для забезпечення цілісності даних і зменшення дублювання, а також денормалізації, що може бути корисною в певних сценаріях для підвищення швидкості запитів.

Індексація в PostgreSQL була виділена як важливий інструмент для прискорення операцій пошуку та доступу до даних. Розглянуто різні типи індексів (B-дерево, GIST, GIN тощо).

Крім того, увагу приділено практикам оптимізації запитів, використанню матеріалізованих представлень для прискорення операцій, та управлінню ключами, яке впливає на структуру та масштабованість бази даних. Реплікація та шардинг розглядаються як важливі методи розподілу даних, що забезпечують як продуктивність, так і надійність.

Окремо проаналізовано апаратну оптимізацію та вибір налаштувань RAID, обсяг оперативної пам’яті та інших апаратних ресурсів. Правильна конфігурація параметрів PostgreSQL доповнює ефективність використання бази даних.

Таким чином, розділ дає чітке уявлення про багаторівневий підхід до покращення продуктивності систем керування базами даних, акцентуючи увагу на комплексній оптимізації архітектури, запитів, апаратних і програмних ресурсів.

3 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ СТРАТЕГІЙ ОПТИМІЗАЦІЇ

3.1 Опис експерименту та тестового середовища

Суть експерименту полягає у практичному застосуванні стратегій оптимізації СУБД PostgreSQL, зборі метрик та проведенні порівняльного аналізу результатів до та після експерименту.

Для проведення тестувань, PostgreSQL було розгорнуто у Docker-контейнері. Docker – це відкрита платформа для розробки, доставки та запуску програм. Він дозволяє відокремити програми від інфраструктури. Контейнер, в свою чергу, це ізольований процес з усіма файлами, які йому потрібні для запуску [56].

Управляти контейнерами зручно, особливо при проведенні експериментів. До прикладу, після маніпуляцій з конфігураційними файлами сервісу, який запущений всередині контейнеру та необхідності відновити початкову конфігурацію, ми можемо з легкістю повторно створити контейнер , що поверне первинний стан сервісу. До того ж, додатковий рівень ізоляції, який забезпечує контейнеризація, дозволить більш точно провести експеримент.

Для дослідження буде використовуватись PostgreSQL версії 17.0. Характеристики гостьової системи: процесор – Intel Core i7-8750H, 2.20GHz; кількість ядер – 6; RAM – 16.0 ГБ, DDR4; SSD.

3.1.2 Структура БД та конфігураційні параметри СУБД

Щоб створити схему бази даних, яка може ефективно продемонструвати вплив різних стратегій оптимізації, розробимо таблиці з атрибутами, які піддаються різним типам індексування, методів оптимізації запитів, керування ключами і тд. Створимо схему БД гіпотетичної платформи електронної комерції, яка включає клієнтів, замовлення та продукти. На рисунку 3.1 продемонстровано ER-модель бази даних.

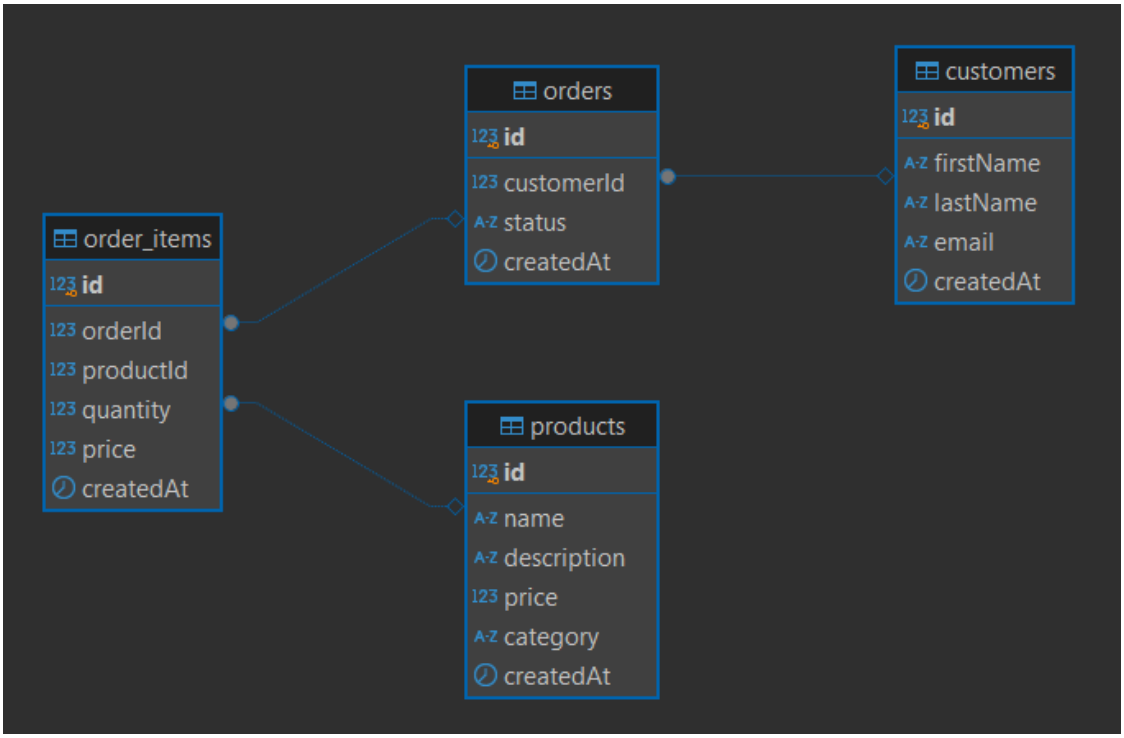


Рисунок 3.1 – ER-модель бази даних

Таблиця «customers» (клієнти) зберігає інформацію про клієнтів. Опис таблиці подано у таблиці 3.1.

Таблиця 3.1 – Опис таблиці «customers»

Поле	**Тип даних**	**Опис**
id	SERIAL	Унікальний ідентифікатор клієнта, первинний ключ
firstName	VARCHAR	Ім'я клієнта
lastName	VARCHAR	Прізвище клієнта
email	VARCHAR	Електронна пошта клієнта. Поле унікальне, щоб уникнути дублікатів
createdAt	TIMESTAMP	Дата і час створення запису. Значення за замовчуванням – поточний час

Таблиця «products» (продукти) зберігає інформацію про товари. Опис таблиці подано у таблиці 3.2.

Таблиця 3.2 – Опис таблиці «products»

Поле	Тип даних	Опис
id	SERIAL	Унікальний ідентифікатор продукту, первинний ключ
name	VARCHAR	Назва продукту
description	TEXT	Опис продукту
price	NUMERIC	Ціна продукту з двома десятковими знаками
category	VARCHAR	Категорія, до якої належить продукт
createdAt	TIMESTAMP	Дата і час створення запису. Значення за замовчуванням – поточний час

Таблиця «orders» (продукти) зберігає інформацію про замовлення. Опис таблиці подано у таблиці 3.3.

Таблиця 3.3 – Опис таблиці «orders»

Поле	Тип даних	Опис
id	SERIAL	Унікальний ідентифікатор замовлення, первинний ключ
customerId	INTEGER	Ідентифікатор клієнта (зовнішній ключ до таблиці customers)
status	VARCHAR	Статус замовлення. Обмеження: лише «pending», «shipped», «delivered», або «cancelled»
createdAt	TIMESTAMP	Дата і час створення замовлення. Значення за замовчуванням – поточний час

Таблиця «order_items» (деталі замовлень) зберігає інформацію про товари, що входять до замовлень. Опис таблиці подано у таблиці 3.4.

Таблиця 3.4 – Опис таблиці «order_items»

Поле	Тип даних	Опис
id	SERIAL	Унікальний ідентифікатор запису, первинний ключ
orderId	INTEGER	Ідентифікатор замовлення (зовнішній ключ до таблиці orders)
productId	INTEGER	Ідентифікатор продукту (зовнішній ключ до таблиці products)
quantity	INTEGER	Кількість одиниць продукту в замовленні
price	NUMERIC	Ціна одиниці продукту в замовленні
createdAt	TIMESTAMP	Дата і час створення запису. Значення за замовчуванням – поточний

Зв’язки між таблицями:

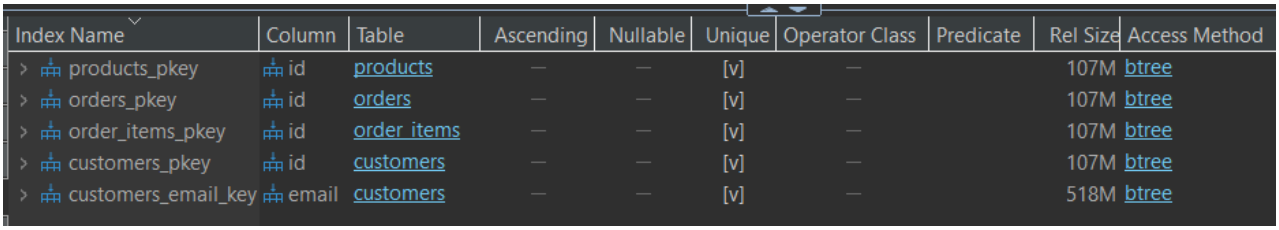
- customers та orders: один клієнт може мати багато замовлень;
- orders та order_items: одне замовлення може включати багато товарів;
- products та order_items: один продукт може входити до кількох замовлень.

Кожна з таблиць була наповнена 5 млн. записів, що показано на рисунку 3.2.

Table Name	Object ID	Owner	Tablespace	Row Count Estimate
customers	16,567	postgres	pg_default	5,000,561
order_items	16,603	postgres	pg_default	4,999,991
orders	16,589	postgres	pg_default	5,000,012
products	16,579	postgres	pg_default	4,999,902

Рисунок 3.2 – Демонстрація кількості записів у таблицях

На рисунку 3.3 також показані наявні індекси. Оскільки кожна таблиця має первинний ключ, а поле «email» додатково забезпечене унікальним обмеженням, у результаті було створено 5 індексів.



Index Name	Column	Table	Ascending	Nullable	Unique	Operator Class	Predicate	Rel Size	Access Method
products_pkey	id	products	—	—	[v]	—		107M	btree
orders_pkey	id	orders	—	—	[v]	—		107M	btree
order_items_pkey	id	order_items	—	—	[v]	—		107M	btree
customers_pkey	id	customers	—	—	[v]	—		107M	btree
customers_email_key	email	customers	—	—	[v]	—		518M	btree

Рисунок 3.3 – Наявні індекси

Ключові конфігураційні параметри серверу бази даних, які встановлено за замовчуванням подано на рисунку 3.4. Для простоти отримання та візуалізації цих даних було використано інструмент pgHero.

Setting	Value
max_connections	100
shared_buffers	128MB
effective_cache_size	4GB
maintenance_work_mem	64MB
checkpoint_completion_target	0.9
wal_buffers	4MB
default_statistics_target	100
random_page_cost	4
effective_io_concurrency	1
work_mem	4MB
huge_pages	try
min_wal_size	80MB
max_wal_size	1GB

Рисунок 3.4 – Ключові конфігураційні параметри серверу БД

3.1.3 Наповнення таблиць даними

Для наповнення бази даних тестовими даними для таблиць «customers», «products», «orders» і «order_items», було реалізовано скрипт на мові JavaScript. Скрипт використовує транзакції для пришвидшення вставки. З'єднання з базою встановлюється за допомогою клієнта PostgreSQL, а для генерації випадкових даних застосовується бібліотека «faker». Реалізація скрипта подана в додатку Б.

Для кожної таблиці дані вставляються пакетами, що зменшує кількість індивідуальних запитів до бази. Кожен пакет вставляється в рамках однієї транзакції, що суттєво підвищує продуктивність порівняно з виконанням окремих запитів. Наприклад, таблиця «customers» заповнюється випадковими іменами, прізвищами та унікальними email-адресами. У таблиці «products» додаються товари з випадковими назвами, описами, цінами та категоріями. Для таблиці «orders» генеруються замовлення, пов'язані з клієнтами, а «order_items» поєднує замовлення з продуктами, включаючи кількість та ціну.

Після завершення кожної операції транзакція завершується і клієнт закриває з’єднання. Цей підхід дозволяє швидко наповнити базу великим обсягом даних для тестування та оцінки продуктивності.

3.1.4 Інструменти та засоби для аналізу продуктивності

DBeaver – це універсальний інструмент для роботи з базами даних та їх адмініструванням, який підтримує різні СУБД, включаючи PostgreSQL. За допомогою нього зручно виконувати SQL-запити, аналізувати результати, створювати розширення, адмініструвати таблиці, індекси і т.д.

psql – це консольний клієнт, який дозволяє взаємодіяти з базами даних PostgreSQL. Він пропонує широкий набір мета-команд та функціональність, схожу на Shell, що спрощує написання скриптів і автоматизацію різних операцій [57]. У рамках автоматизованого процесу він забезпечує гнучкий спосіб виконання SQL-запитів і отримання інформації про базу даних.

pgBench – це утиліта для тестування продуктивності PostgreSQL. Вона використовується для оцінки швидкодії бази даних за допомогою навантажувальних тестів, зокрема стандартних сценаріїв транзакцій або кастомних тестів для перевірки конкретних ситуацій.

Bash-скрипти – в контексті проведення експерименту, дані скрипти застосовуються для автоматизації роботи з psql та pgBench. Вони дозволяють виконувати попереднє налаштування, запускати навантажувальні тести та

аналізувати результати. Це забезпечує зручність у повторюваних сценаріях тестування та налаштування бази даних.

JS-скрипти – в рамках дослідження, ці скрипти використовуються для масового наповнення таблиць тестовими даним.

3.2 Заходи для забезпечення достовірності результатів

Для забезпечення достовірності результатів експериментів важливо проводити моніторинг фонових процесів і проводити їх в однакових умовах. Наприклад, відкриті додатки або важкі фонові завдання можуть значно вплинути на час виконання запиту. Наявність додаткових програм додає суттєвий шум до вимірювань [58]. Це означає, що під час вимірювання продуктивності не можна дозволяти, щоб один тест проводився з меншими ресурсними можливостями, ніж інший.

Також, один з основних чинників, що впливає на продуктивність бази даних, це кеш. Коли працюємо з базою даних, багато даних потрапляють до кешу як самої системи управління базою даних, так і операційної системи, що дозволяє наступним запитам виконуватися значно швидше Для забезпечення точності замірів слід уникати ситуацій, коли запити частково використовують кеш, а частково читають дані з диска. Найпростішим способом забезпечити рівні умови для всіх тестів є очищення кешу перед кожним запуском. Наприклад, можна перезапустити службу PostgreSQL або вручну скинути кеш операційної системи.

Виконання тестів на великому обсязі даних також важливе для достовірності результатів. Якщо тести виконуються на запитах, які займають менше мілісекунди, навіть декількох запусків таких запитів буде недостатньо для отримання точних результатів. Проте саме по собі виконання тестування кілька разів допоможе зменшити вплив випадкових помилок або шуму [58].

Якщо тести включають видалення рядків, потрібно звертати увагу на процес вакуумування, який очищує таблиці від видалених даних. Якщо це не контролювати, автоочищення таблиць під час тестів може значно спотворити результати через додаткові операції введення-виведення [58].

3.3 Вплив індексування на продуктивність запитів

Здійснимо аналіз впливу індексів на швидкодію виконання запитів. Першим типом індексу, який ми протестуємо, буде індекс B-дерева, застосований до поля «email» у таблиці «customers». Оскільки цей індекс було створено під час ініціалізації таблиці, для проведення порівняльного аналізу спершу видалимо його. На рисунку 3.5 продемонстровано список індексів після видалення індексу на полі «email».

	tablename	indexname	indexdef
1	customers	customers_pkey	CREATE UNIQUE INDEX customers_pkey ON public.customers USING btree (id)
2	order_items	order_items_pkey	CREATE UNIQUE INDEX order_items_pkey ON public.order_items USING btree (id)
3	orders	orders_pkey	CREATE UNIQUE INDEX orders_pkey ON public.orders USING btree (id)
4	products	products_pkey	CREATE UNIQUE INDEX products_pkey ON public.products USING btree (id)

Рисунок 3.5 – Список індексів після видалення індексу з поля «email»

Перевіримо швидкість виконання запиту на вибірку рядка з таблиці без індексу на полі «email» (див. лістинг 3.1).

Лістинг 3.1 – Запит на вибірку даних з таблиці «customers»

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT *
FROM customers c
WHERE c.email = 'medhurst24950004999cydney.windler45@hotmail.com';
```

На рисунку 3.6 можна побачити план виконання даного запиту.

	QUERY PLAN
1	Gather (cost=1000.00..88689.77 rows=1 width=66) (actual time=1366.854..1368.928 rows=1 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	Buffers: shared read=61648
5	-> Parallel Seq Scan on customers c (cost=0.00..87689.67 rows=1 width=66) (actual time=1121.245..1345.034 rows=0 loops=3)
6	Filter: ((email)::text = 'medhurst24950004999cydney.windler45@hotmail.com'::text)
7	Rows Removed by Filter: 1666666
8	Buffers: shared read=61648
9	Planning:
10	Buffers: shared hit=73 read=27
11	Planning Time: 9.498 ms
12	Execution Time: 1368.954 ms

Рисунок 3.6 – План виконання запиту на вибірку даних без індексу

В результаті видно, що запит використовував паралельне послідовне сканування, що вказує на те, що PostgreSQL намагався обробити великий обсяг даних за допомогою кількох робочих процесів. Однак через відсутність індексу на полі «email», система була змушена виконати повний перегляд таблиці, перевіряючи кожен рядок.

Тепер виконаємо запит декілька разів та заповнимо таблицю 3.5 результатами.

Таблиця 3.5 – Результати виконання запиту на вибірку даних без індексу

Спроба	1	2	3	4	5
Час виконання (мс)	1368.954	1344.979	1359.356	1346.983	1303.180

Переходимо до створення індексу. Оскільки поле «email» в реальних застосунках повинно бути унікальним, додаємо обмеження «UNIQUE», а PostgreSQL автоматично створить індекс B-дерева:

```
ALTER TABLE customers ADD CONSTRAINT unique_email UNIQUE (email);
```

На рисунку 3.7 можна побачити план виконання даного запиту.

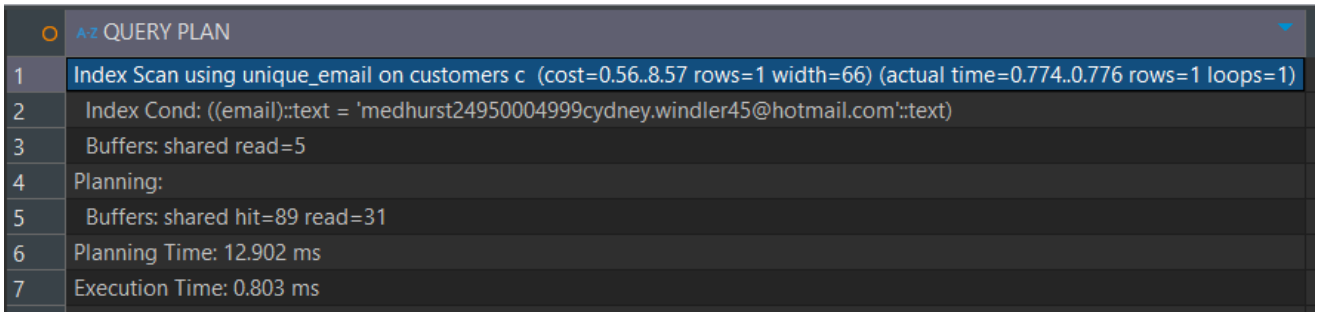


Рисунок 3.7 – План виконання запиту на вибірку даних з індексом

В результаті виконання запиту бачимо, що використовується сканування за індексом «unique_email» на таблиці «customers». Це означає, що запит використовує індекс для пошуку запису. Час виконання запиту складає 0.803 мс.

Виконаємо запит декілька разів та заповнимо таблицю 3.6 результатами.

Таблиця 3.6 – Результати виконання запиту на вибірку даних з індексом B-дерева

Спроба	1	2	3	4	5
Час виконання (мс)	0.803	0.619	0.691	0.674	0.755

На стовпчиковій діаграмі нижче можна візуально побачити різницю у швидкості виконання запитів на вибірку (див. рис. 3.8).

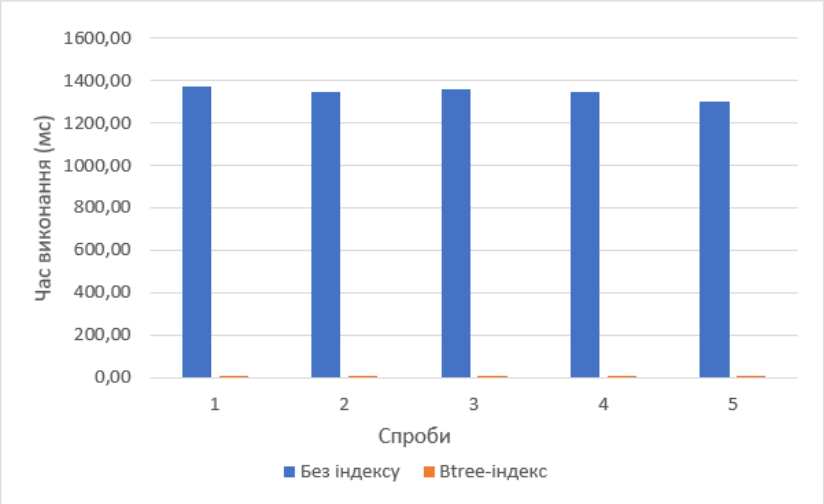


Рисунок 3.8 – Діаграма часу виконання запитів з індексом B-дерева та без нього

Оскільки індекси є окремою структурою, вони теж займають дисковий простір. Для порівняння скільки місця займає сама таблиця та скільки ресурсів витрачається на індекс, виконаємо запит, який подано у лістингу 3.2.

Лістинг 3.2 – Запит на отримання інформації про розмір таблиці та індексу

```
SELECT
    pg_size_pretty(pg_relation_size('customers')) AS table_size,
    pg_size_pretty(pg_relation_size('unique_email')) AS
idx_unique_email_size;
```

На рисунку 3.9 подано результати запиту.

table_size	idx_unique_email_size
482 MB	310 MB

Рисунок 3.9 – Розмір таблиці та створеного індексу

Розмір таблиці «customers» становить 482 МБ, а розмір індексу – 310 МБ. Це свідчить про те, що індекс за полем «email» займає значну частину місця на диску, що є типовим для індексів, які працюють з великими обсягами даних та забезпечують унікальність записів.

Перейдемо до тестування хеш-індексу. Застосуємо його теж для поля «email». Хеш-індекс може бути ефективнішим, оскільки він дозволяє здійснити прямий пошук за хешем значення, що значно прискорює виконання запиту. Створимо даний індекс:

```
CREATE INDEX IF NOT EXISTS idx_customers_email_hash ON customers
USING hash (email);
```

Перевіримо швидкість виконання запиту на вибірку рядка з таблиці без індексу на полі «email». Запит буде наступним:

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT *  FROM customers c
WHERE c.email = 'medhurst24950004999cydney.windler45@hotmail.com';
```

На рисунку 3.10 можна побачити план виконання даного запиту.

	QUERY PLAN
1	Index Scan using idx_customers_email_hash on customers c (cost=0.00..8.02 rows=1 width=66) (actual time=1.035..1.037 rows=1 loops=1)
2	Index Cond: ((email)::text = 'medhurst24950004999cydney.windler45@hotmail.com'::text)
3	Buffers: shared read=3
4	Planning:
5	Buffers: shared hit=105 read=11
6	Planning Time: 13.354 ms
7	Execution Time: 1.174 ms

Рисунок 3.10 – План виконання запиту на вибірку даних із застосуванням хеш-індексу

У статистиці запиту видно, що використовується сканування індексу. Час виконання запиту з хеш-індексом (1.174 мс) виявився дещо повільнішим у порівнянні з індексом B-дерева, де час був близько 0.8 мс.

Виконаємо запит декілька разів та заповнимо таблицю 3.7 результатами.

Таблиця 3.7 – Результати виконання запиту на вибірку даних із хеш-індексом

Спроба	1	2	3	4	5
Час виконання (мс)	1.174	1.072	0.836	0.717	1.109

На рисунку 3.11 зображено діаграму, де порівнюється швидкість вибірки, з індексом B-дерева та з хеш-індексом.

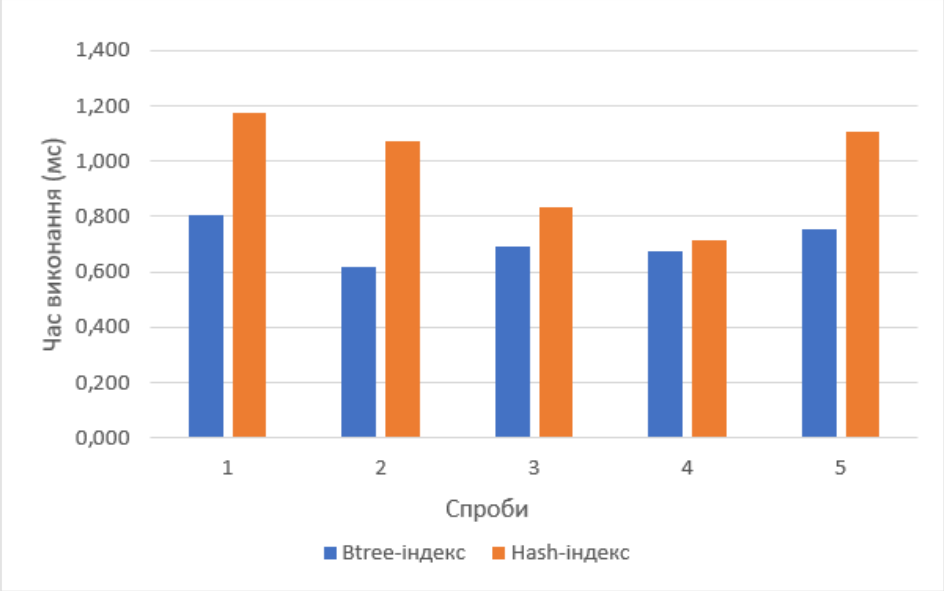


Рисунок 3.11 – Швидкість виконання запиту з індексом B-дерева та з хеш-індексом

Розмір хеш-індексу становить 128 МБ (див. рис. 12)

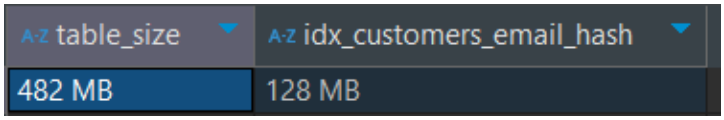


Рисунок 3.12– Розмір таблиці та хеш-індексу

Проте, варто зазначити, що даний індекс не підтримує унікальність за замовчуванням та не підтримує діапазонні пошукові запити (наприклад, знайти всі email-адреси, що починаються з певного рядка).

Наступним кроком протестуємо GIN-індекс. Цей тип індексу особливо корисний для повнотекстового пошуку. Його робота базується на списку токенів, що створюються для кожного значення у полі, яке індексується. Виконаємо скрипт, який поверне записи з таблиці «products», де поле опису починається текстом «Discover the elastic new Shirt» (див. лістинг 3.4).

Лістинг 3.4 – Запит на пошук тексту з фіксованим префіксом

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT * FROM products p
WHERE description
ILIKE 'Discover the elastic new Shirt%';
```

На рисунку 3.13 подано план виконання даного запиту.

	QUERY PLAN
1	Gather (cost=1000.00..123164.17 rows=500 width=122) (actual time=1129.277..8958.281 rows=5 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	Buffers: shared read=96072
5	-> Parallel Seq Scan on products p (cost=0.00..122114.17 rows=208 width=122) (actual time=3571.772..8877.399 rows=2 loops=3)
6	Filter: (description ~~* 'Discover the elastic new Shirt%'::text)
7	Rows Removed by Filter: 1666665
8	Buffers: shared read=96072
9	Planning:
10	Buffers: shared hit=75 read=16
11	Planning Time: 18.444 ms
12	JIT:
13	Functions: 6
14	Options: Inlining false, Optimization false, Expressions true, Deforming true
15	Timing: Generation 1.770 ms (Deform 0.724 ms), Inlining 0.000 ms, Optimization 1.196 ms, Emission 22.164 ms, Total 25.130 ms
16	Execution Time: 9018.919 ms

Рисунок 3.13 – План виконання запиту на пошук тексту з фіксованим префіксом

Задіяно послідовне паралельне сканування, що дозволяє кільком потокам одночасно виконувати послідовне сканування таблиці. Загальний час виконання – 9 секунд, що свідчить про те, що запит не оптимізований для текстового пошуку.

Протестуємо даний запит повторно декілька разів. Результати подані у таблиці 3.8.

Таблиця 3.8 – Результати виконання запиту на пошук тексту з фіксованим префіксом без індексу

Спроба	1	2	3	4	5
Час виконання (мс)	9018.919	8627.087	8679.067	8819.715	9321.250

Проаналізуємо запит пошуку підрядка (див. лістинг 3.5).

Лістинг 3.5 – Запит пошуку підрядка

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT * FROM products p
WHERE description
ILIKE '%elastic new Shirt%';
```

На рисунку 3.14 подано план виконання даного запиту.

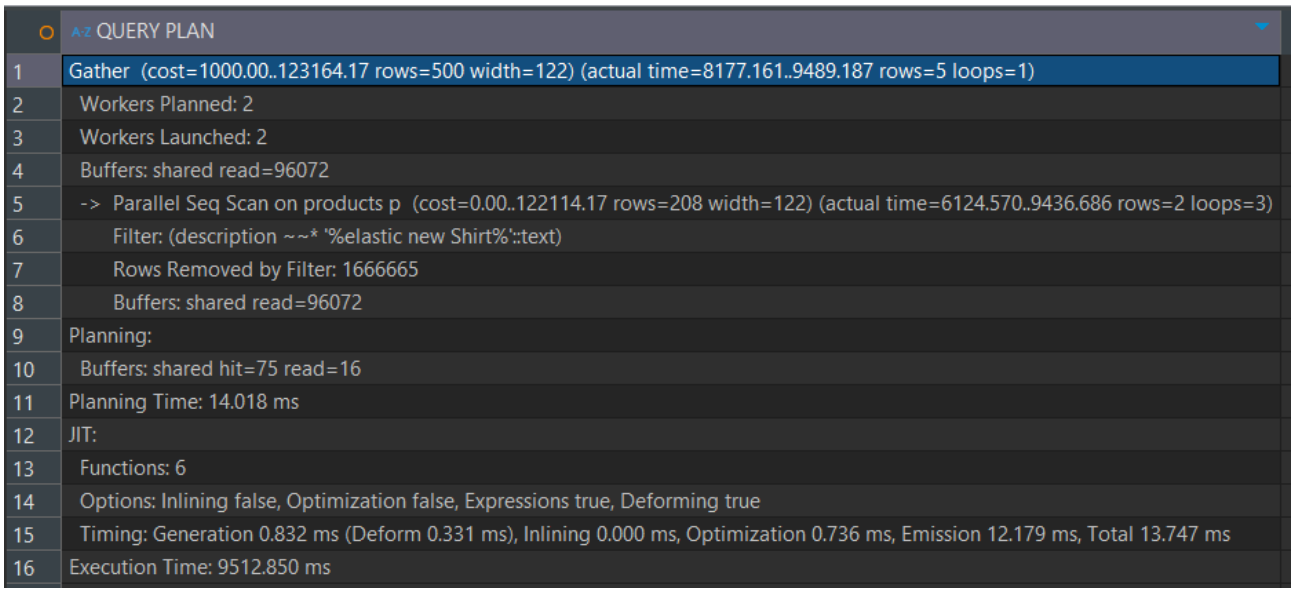


Рисунок 3.14 – План виконання запиту на пошук підрядка

Запит тривав приблизно 9,5 секунд, де більша частина припала на сканування таблиці. Результати повторних 5 запусків запиту подані у таблиці 3.9.

Таблиця 3.9 – Результати виконання запиту на пошук підрядка без індексу

Спроба	1	2	3	4	5
Час виконання (мс)	9512.850	8693.049	9082.125	8903.656	8724.129

Запити виконуються неефективно. Використаємо розширення «pg_trgm» та застосуємо «gin_trgm_ops» для створення GIN-триаграмного індексу:

```
CREATE EXTENSION IF NOT EXISTS pg_trgm;
CREATE INDEX IF NOT EXISTS idx_products_description_gin ON
products USING gin (description gin_trgm_ops);
```

Виконаємо такий ж запит на пошук тексту, що починається на «Discover the elastic new Shirt» за полем «description». На рисунку 3.15 подано план виконання запиту.

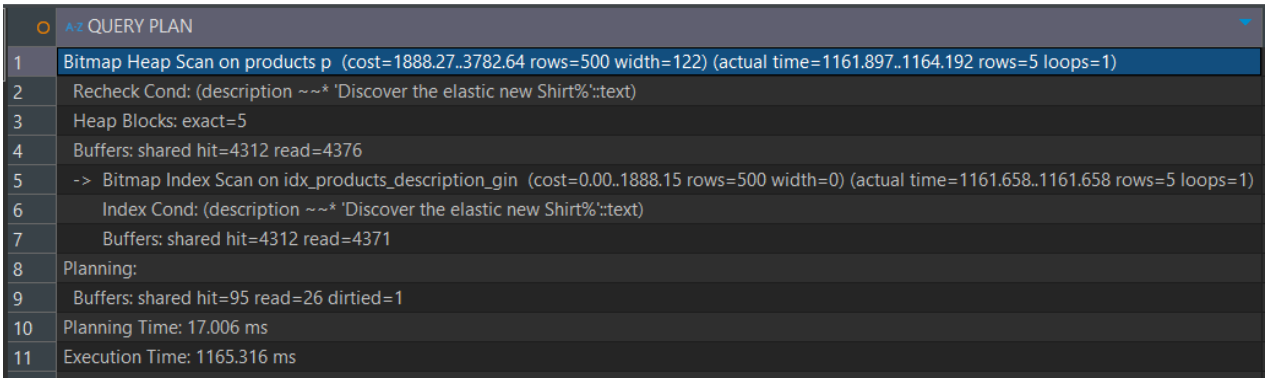


Рисунок 3.15 – План виконання запиту на пошук тексту із застосуванням GIN-індексу

На першому етапі GIN-індекс ефективно ідентифікує відповідні триаграми. Це дозволило знайти всі потенційно релевантні записи (5 рядків). Потім система перевіряє фізичні рядки, щоб підтвердити, що вони відповідають запиту. Зчитано лише 5 блоків таблиці. Загальний час виконання – 1165.316 мс.

Результати повторних 5 запусків запиту подані у таблиці 3.10.

Таблиця 3.10 – Результати виконання запиту на пошук тексту з фіксованим префіксом із застосуванням індексу

Спроба	1	2	3	4	5
Час виконання (мс)	1165.316	1199.516	1699.409	1181.674	1172.194

Проте, розмір GIN-індексу становить 493 МБ, що навіть перевищує обсяг самої таблиці (див. рис. 3.16).

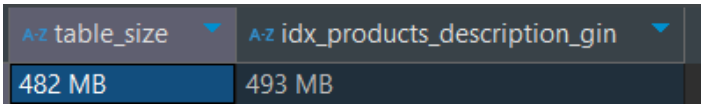


Рисунок 3.16 – Розмір GIN-індексу

GIN-індекси займають багато місця, оскільки вони детально індексують текст для швидкого пошуку.

Далі протестуємо пошук підрядка «elastic new Shirt» з наявним індексом. План виконання запиту подано на рисунку 3.17.

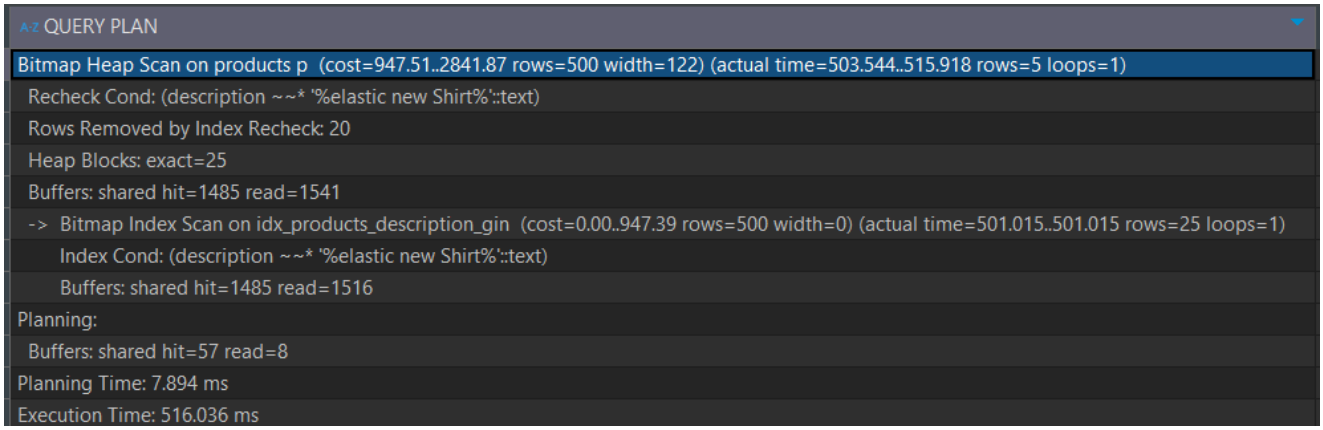


Рисунок 3.17 – План виконання запиту на пошук підрядка із застосуванням GIN-індексу

План виконання запиту демонструє, що використання GIN-індексу суттєво прискорює пошук підрядка в тексті. GIN-індекс обробив запит за 516 мс, виконавши сканування за bitmap-індексом, який знайшов 25 потенційно відповідних рядків, і потім bitmap-heap сканування, що перевірило відповідність цих рядків умовам запиту.

Результати повторних 5 запусків запиту подані у таблиці 3.11.

Таблиця 3.11 – Результати виконання запиту на пошук підрядка із застосуванням GIN-індексу

Спроба	1	2	3	4	5
Час виконання (мс)	516.036	518.184	446.174	453.831	456.832

На рисунку 3.18 зображено діаграму, де порівнюється швидкість пошуку тексту без GIN-індексу та з індексом.

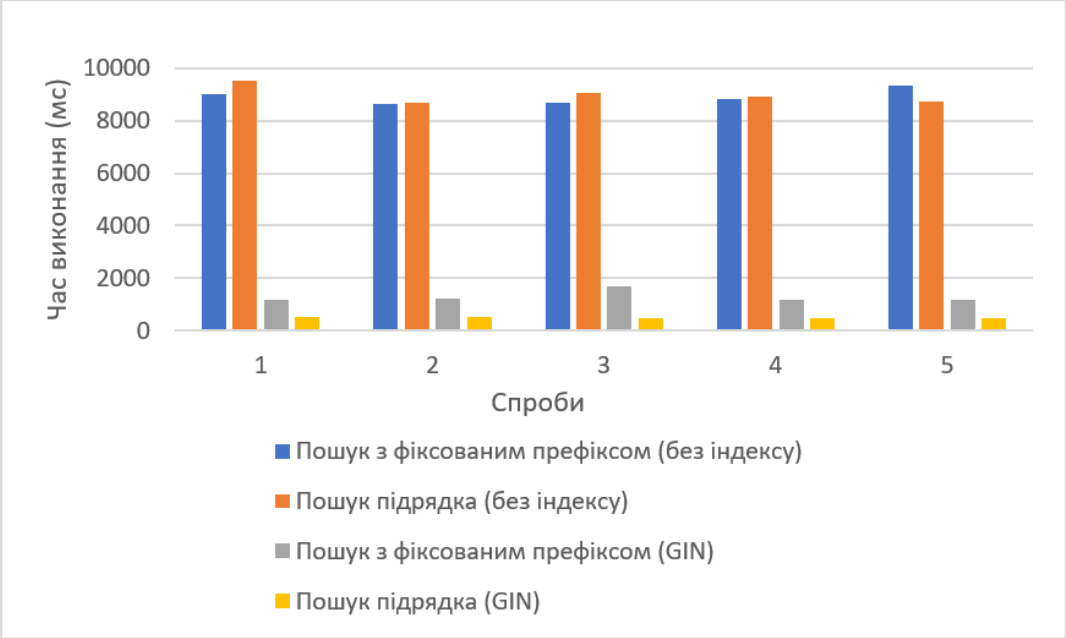


Рисунок 3.18 – Діаграма швидкості пошуку тексту без GIN-індексу та з індексом

Також протестуємо BRIN-індекс. Оскільки цей індекс ефективно працює з великими таблицями, де записи розподілені по діапазонах і значення в полі змінюються поступово, він може значно покращити продуктивність запитів до таких таблиць. Один з важливих аспектів, який підтверджує ефективність BRIN-індексу, це кореляція значень у полі з фізичним порядком рядків у таблиці.

Для підтвердження кореляції між значеннями в полі і фізичним порядком можна використовувати таблицю «pg_stats». Вона містить статистичні дані, які використовує планувальник запитів PostgreSQL для вибору оптимального індексу або способу доступу до даних (див. рис. 3.19).

	attname	correlation
1	id	1
2	customerId	0.002155113
3	status	0.2505585
4	createdAt	1

Рисунок 3.19 – Кореляція полів таблиці «orders»

В колонці «correlation» можна побачити кореляцію для кожного з полів таблиці. Обираємо поле «createdAt» для тестування індексу, адже воно має високу кореляцію – 1.

Виконаємо запит для підрахунку кількості записів в таблиці orders, де значення в полі «createdAt» знаходяться в діапазоні між 1 січня 2024 року і 2 січня 2024 року:

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT COUNT(*) FROM orders o
WHERE "createdAt" BETWEEN '2024-01-01' AND '2024-01-02';
```

План виконання запиту подано на рисунку 3.20.

	QUERY PLAN
1	Gather (cost=1000.00..102099.70 rows=10497 width=25) (actual time=2259.887..2269.720 rows=10000 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	Buffers: shared read=68800
5	-> Parallel Seq Scan on orders o (cost=0.00..100050.00 rows=4374 width=25) (actual time=2208.148..2211.449 rows=3333 loops=3)
6	Filter: (("createdAt" >= '2024-01-01 00:00:00'::timestamp without time zone) AND ("createdAt" <= '2024-01-02 00:00:00'::timestamp without time zone))
7	Rows Removed by Filter: 1663333
8	Buffers: shared read=68800
9	Planning:
10	Buffers: shared hit=54 read=18 dirtied=1
11	Planning Time: 14.979 ms
12	JIT:
13	Functions: 6
14	Options: Inlining false, Optimization false, Expressions true, Deforming true
15	Timing: Generation 2.639 ms (Deform 1.199 ms), Inlining 0.000 ms, Optimization 1.180 ms, Emission 25.299 ms, Total 29.118 ms
16	Execution Time: 2291.241 ms

Рисунок 3.20 – План виконання запиту з вибіркою даних за діапазоном дат

Загальний час виконання запиту склав 2291.241 мс. Результати повторних 5 запусків запиту подані у таблиці 3.12.

Таблиця 3.12 – Результат виконання запитів з вибіркою даних у діапазоні дат без індексу

Спроба	1	2	3	4	5
Час виконання (мс)	2291.241	2383.324	2475.473	2802.100	2595.787

Тепер створимо BRIN-індекс:

```
CREATE INDEX IF NOT EXISTS idx_orders_created_at_brin ON orders
USING brin ("createdAt");
```

Тепер виконаємо запит із створеним BRIN-індексом. План виконання подано на рисунку 3.21.

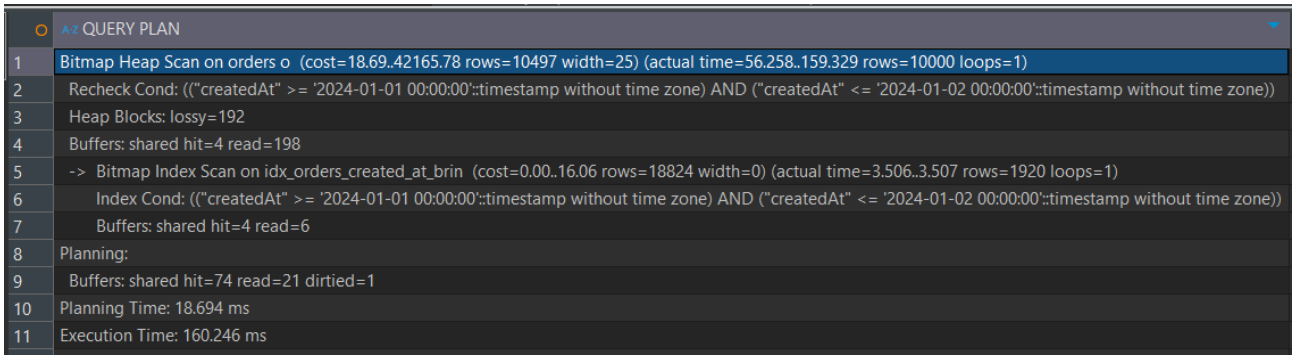


Рисунок 3.21 – План виконання запиту із застосуванням BRIN-індексу

План виконання запиту показує, як система використовує BRIN-індекс для пошуку записів у таблиці orders на основі діапазону значень у колонці createdAt. Застосування BRIN-індексу дозволило значно зменшити обсяг даних, що потрібно сканувати. Час виконання 160.246 мс є хорошим результатом.

Повторимо запит, а результати занесемо в таблицю 3.13.

Таблиця 3.13 – Результати виконання запиту із BRIN-індексом

Спроба	1	2	3	4	5
Час виконання (мс)	160.246	97.963	162.887	131.289	126.853

Розмір BRIN-індексу є невеликим – 32 КБ, адже зберігаються лише мінімальні і максимальні значення для блоків даних (див. рис. 3.22).

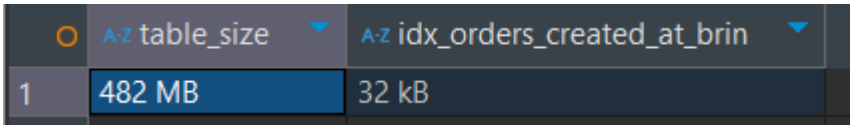


Рисунок 3.22 – Розмір BRIN-індексу

На рисунку 3.23 зображена діаграма, де візуально помітно різницю у швидкості виконання запитів із застосуванням BRIN-індексу та без нього.

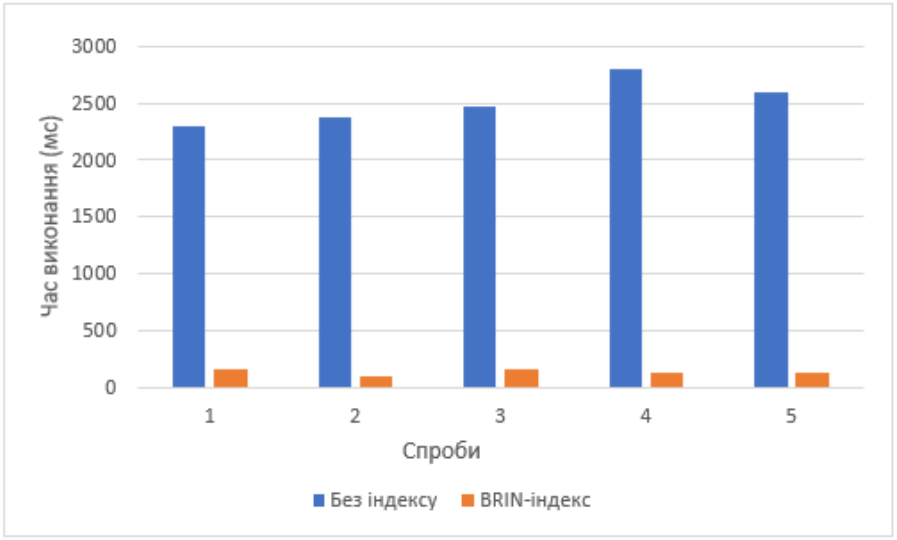


Рисунок 3.23 – Діаграма швидкості виконання запиту без BRIN-індексу та з індексом

3.4 Управління ключами

Порівняємо два підходи до використання первинних ключів у PostgreSQL: послідовні ідентифікатори та універсально унікальні ідентифікатори (UUID).

Для експерименту було створено дві додаткові таблиці: «orders_serial», де первинний ключ реалізовано через послідовний ідентифікатор, та «orders_uuid», де первинний ключ базується на типі UUID. Дослідимо як різниця у типах первинного ключа впливає на швидкодію вставки. Вставки виконувалися для обсягів у 5000, 10000, 50000 та 100000 рядків.

Для кожного етапу вставки використовувався скрипт, який:

1. Вимірює час виконання вставки з моменту початку операції до завершення.

2. Фіксує розмір таблиці після вставки, щоб оцінити обсяг даних у базі.

3. Визначає розмір індексу для кожної таблиці, щоб порівняти вплив різних типів ключів.

Для тестування використовувався pgbench. Вставка даних виконувалася за допомогою власного SQL-скрипта, у якому було визначено інструкції для вставки (див. лістинг 3.6).

Лістинг 3.6 – Скрипт для тестування швидкості вставки даних

```
pgbench --no-vacuum --transactions=$((milestones[$milestone_idx] / 
PGBENCH_CLIENTS)) --client=$PGBENCH_CLIENTS \
      --file=$transaction_script --host=$DB_HOST --port=$DB_PORT -
-username=$DB_USER --dbname=$DB_NAME \
      --report-per-command >> $pgbench_log_file 2>> 
$pgbench_error_log
```

Повний лістинг даного коду подано у додатку В.

Метрики, які фіксувалися для кожного етапу:

- кількість рядків, вставлених у таблицю;
- час виконання вставки (у мілісекундах);
- розмір таблиці після вставки (у КБ);
- розмір індексу після вставки (у КБ).

Протестуємо спочатку швидкодію вставки у таблицю з послідовним первинним ключем. Результати подано у таблицю 3.14.

Таблиця 3.14 – Результати вимірювань швидкості вставок записів у таблицю з послідовним первинним ключем

Кількість вставок	**Загальний час (мс)**	**Розмір таблиці (КБ)**	**Розмір індексу (КБ)**
5000	12286	408	128
10000	35422	752	344
50000	352507	2776	1120
100000	1304805	5704	2208

Для генерації значень UUID таблиці «orders_uuid» використовувалася функція «gen_random_uuid», яка є частиною розширення «pgcrypto» у PostgreSQL:

```
CREATE TABLE orders_uuid (
  id uuid DEFAULT gen_random_uuid(), …
);
```

Результати вставок у таблицю з UUID наведено у таблиці 3.15.

Таблиця 3.15 – Результати вимірювань швидкості вставок записів у таблицю з первинним ключем UUID

Кількість вставок	**Загальний час (мс)**	**Розмір таблиці (КБ)**	**Розмір індексу (КБ)**
5000	9163	352	224
10000	42719	1048	608
50000	545453	3376	1984
100000	1830120	6760	4072

Візуальне представлення результатів подано на рисунках 3.24 – 3.26.

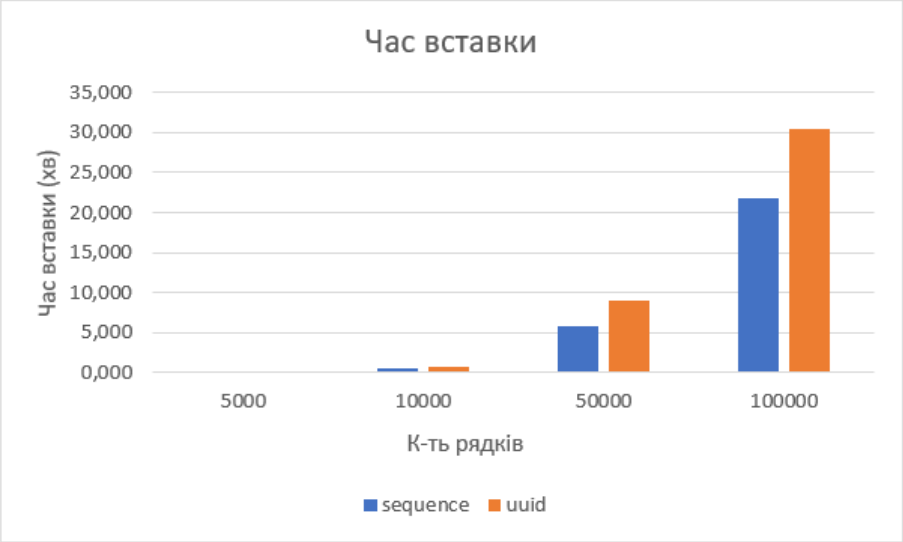


Рисунок 3.24 – Порівняльна діаграма швидкодії вставки у таблиці з послідовним PK та UUID PK

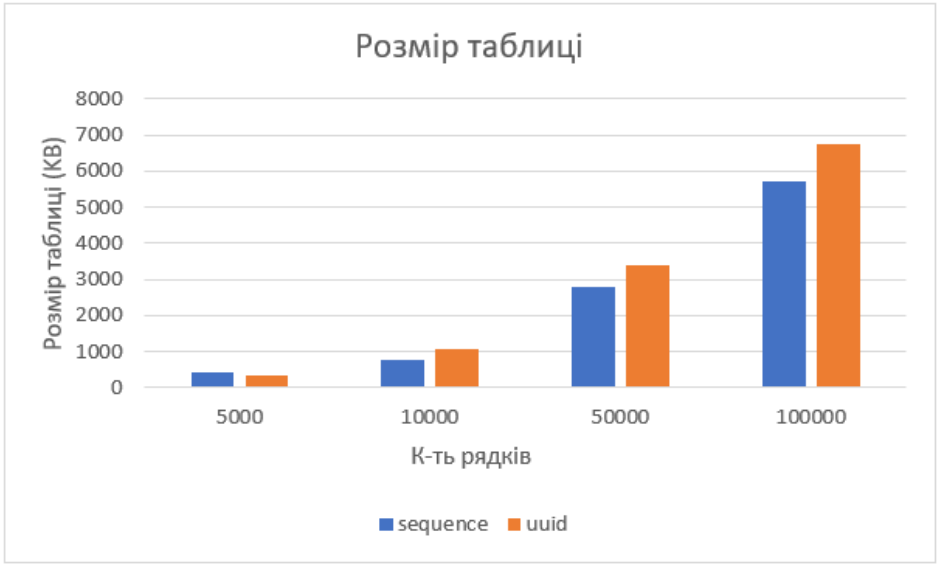


Рисунок 3.25 – Порівняльна діаграма розміру таблиць з послідовним PK та UUID PK

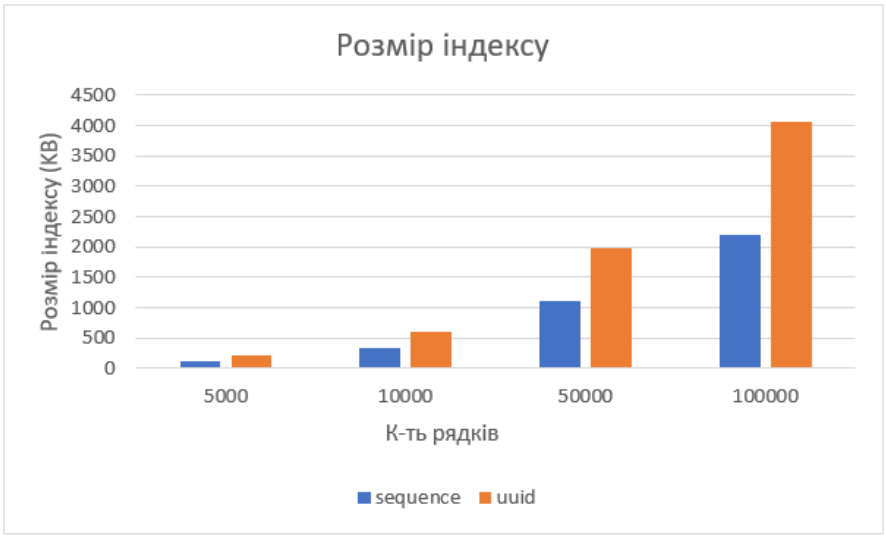


Рисунок 3.26 – Порівняльна діаграма розміру індексу таблиці з послідовним PK та UUID PK

Результати тестування показали, що таблиці з послідовним первинним ключем забезпечують швидшу вставку та менший розмір таблиці та індексу, особливо при великих обсягах даних. Наприклад, для 100,000 записів час вставки становив 1,304,805 мс, а розмір таблиці та індексу – 5,704 КБ і 2,208 КБ відповідно.

У випадку з UUID ключем спостерігається швидша вставка для невеликих обсягів даних (9,163 мс для 5,000 записів). Однак із зростанням обсягу даних час

вставки збільшується суттєво (1,830,120 мс для 100,000 записів), а таблиця та індекс займають більше місця (6,760 КБ та 4,072 КБ відповідно).

3.5 Конфігурація бази даних

Параметри PostgreSQL, такі як «shared_buffers», «wal_buffers», «maintenance_work_mem», і «work_mem», дозволяють оптимізувати використання пам’яті, буферів журналу транзакцій та інших ресурсів для досягнення більшої продуктивності. Проведемо налаштування цих параметрів, тестуючи їхній вплив на різні операції бази даних.

Перейдемо до параметра «shared_buffers», який визначає обсяг пам’яті, зарезервованої для кешування сторінок бази даних у пам’яті. Хоча налаштування цього параметра дуже корисне для вибірки даних, воно також забезпечує суттєві покращення продуктивності для інтенсивних операцій запису – дозволяє накопичувати більшу кількість змін в пам’яті та дозволяючи більшій кількості «брудних» сторінок записуватись на диск одночасно. Тому для тестування впливу «shared_buffers» використовується скрипт із виконанням транзакцій, які включають отримання даних, створення та оновлення (див. лістинг 3.7).

Лістинг 3.7 – Скрипт транзакції із операціями отримання, створення та оновлення даних

```
\set customer_id random(1, 500000)
\set product_id random(1, 500000)
\set quantity random(1, 10)
\set order_status random(1, 4)
\set order_id random(1, 500000)

BEGIN;
SELECT * FROM customers WHERE id = :customer_id;
INSERT INTO orders ("customerId", status) VALUES (:customer_id,
'pending');
INSERT INTO order_items ("orderId", "productId", quantity, price)
VALUES (:order_id, :product_id, :quantity, (SELECT price FROM
products WHERE id = :product_id));
UPDATE orders
SET status = CASE WHEN :order_status = 1 THEN 'shipped'
                  WHEN :order_status = 2 THEN 'delivered'
```

Продовження лістингу 3.7

```
                WHEN :order_status = 3 THEN 'cancelled'
                ELSE 'pending' END
WHERE id = :order_id;
COMMIT;
```

Метриками, які враховувались в тестуванні є TPS та середня затримка для обробки однієї транзакції. pgbench виконує транзакції, визначені у скрипті, протягом 300 секунд (див. лістинг 3.8).

Лістинг 3.8 – Скрипт запуску транзакцій

```
pgbench -f transaction.sql --jobs=2 --time=300 --client=10 \
  --file=$transaction_script --host=$DB_HOST --port=$DB_PORT --
username=$DB_USER --dbname=$DB_NAME \
  --report-per-command >> $pgbench_log_file 2>> $pgbench_error_log
```

Для кожного значення «shared_buffers» – 64 МБ, 128 МБ, 256 МБ, 512 МБ та 1024 МБ, результати фіксувалися у таблицю 3.16.

Таблиця 3.16 – Результати заміру метрик при різних значеннях «shared_buffers»

shared_buffers	64 МБ	128 МБ	256 МБ	512 МБ	1024 МБ
TPS	34.498	40.947	119.792	129.142	145.097
Середня затримка (мс)	289.867	244.213	83.477	77.434	68.919

Значення TPS суттєво зросло з підвищенням розміру «shared_buffers». Це вказує на те, що більший обсяг пам'яті дозволив зберігати більше даних у кеші, зменшуючи звернення до диску. В свою чергу, середня затримка знижувалася зі збільшенням розміру цього параметру.

Далі дослідимо вплив зміни параметру «wal_buffers», що визначає розмір буфера для журналу транзакцій (WAL). Цей параметр впливає на продуктивність запису, особливо при великих обсягах транзакцій. Для тестування використовується pgbench у поєднанні зі згенерованим скриптом для вставки даних. Кількість операцій вставки у одній транзакції становить 10000.

Тестування проводилося для значень 4MB, 8MB та 16MB, і фіксувалися основні метрики: TPS, загальна кількість завершених транзакцій протягом тестового періоду, загальна кількість успішно вставлених записів. Результати тестувань подані у таблиці 3.17.

Таблиця 3.17 – Результати заміру метрик з різними значеннями «wal_buffers»

wal_buffers	4 МБ	8 МБ	16 МБ
TPS	1.823	2.402	2.868
Кількість завершених транзакцій	286	375	434
Кількість вставлених записів	5500000	7280000	8740000

При збільшенні розміру «wal_buffers» спостерігалося зростання TPS, а також збільшення кількості завершених транзакцій та кількості вставок за той самий проміжок часу, що корелює зі збільшенням TPS.

Перейдемо до параметра «maintenance_work_mem», який впливає на операції обслуговування бази даних, такі як створення індексів, злиття таблиць і виконання вакуумування. Збільшення цього значення може суттєво зменшити час для інтенсивних операцій обслуговування. Скрипт тестування передбачає створення індексу на таблиці «products»:

```
psql -U postgres -d "strategies-pg" -c '\timing on' -c 'CREATE
INDEX IF NOT EXISTS idx_products_name_description ON products
(name, description);'
```

Тестування виконувалося із включеним вимірюванням часу – «\timing on» для кожного значення параметра (16 МБ, 32 МБ, 128 МБ, 256 МБ). Результати вимірювань подано у таблицю 3.18.

Таблиця 3.18 – Результати заміру метрик з різними значеннями «maintenance_work_mem»

maintenance_work_mem	16 МБ	32 МБ	128 МБ	256 МБ
Тривалість створення індексу (мс)	96871.386	96955.503	88228.657	85055.091

Збільшення значення «maintenance_work_mem» до призвело до суттєвого зменшення часу, необхідного для створення індексу.

Розглянемо тепер параметр «work_mem», який визначає обсяг пам’яті для виконання операцій сортування та з’єднання. Для перевірки використовується відповідний SQL-запит із сортуванням та з’єднанням (див. лістинг 3.15):

Лістинг 3.15 – Запит на вибірку даних із сортуванням та join-операцією

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT p.id, p.name, p.price, p.category, p."createdAt", oi.quantity
FROM products p
JOIN order_items oi ON oi."productId" = p.id
ORDER BY p.price DESC, p."createdAt" DESC
LIMIT 10;
```

Час виконання запиту вимірювався для різних значень параметра «work_mem» – 500 КБ, 2 МБ, 4 МБ, 8 МБ (таблиця 3.19).

Таблиця 3.19 – Результати заміру метрик при різних значеннях «work_mem»

work_mem	500 КБ	2 МБ	4 МБ	8 МБ
Час виконання (мс)	56047.062	42612.235	26913.250	21845.333

Збільшення розміру «work_mem» суттєво знижує час виконання запиту. Це свідчить про те, що більший обсяг пам’яті дозволяє обробляти дані більш ефективно, мінімізуючи кількість звернень до диску.

3.6 Висновок до третього розділу

У третьому розділі було виконано практичне дослідження ефективності різних стратегій оптимізації бази даних PostgreSQL. Експеримент проводився у середовищі Docker для ізоляції та забезпечення повторюваності результатів. База даних створена за допомогою схеми гіпотетичної платформи електронної комерції, з таблицями для клієнтів, замовлень і продуктів. Середовище було наповнене тестовими даними.

У ході експерименту досліджено вплив різних типів індексів, зокрема B-дерева, GIN, BRIN і триграмних індексів, на продуктивність виконання запитів. Проведено тестування з індексами та без них, із визначенням їхнього впливу на час виконання запитів та використання дискового простору.

Також було порівняно два підходи до управління первинними ключами: послідовні ідентифікатори та UUID. Результати демонструють різницю у швидкодії вставки та використанні ресурсів при різних обсягах даних.

Особливу увагу приділено оптимізації конфігураційних параметрів PostgreSQL, таких як «shared_buffers» і «work_mem», щоб оцінити їхній вплив на продуктивність системи. Дотримання умов проведення експерименту, включаючи ізоляцію середовища та мінімізацію впливу кешу, забезпечило більшу достовірність отриманих результатів.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

4.1.1 Вимоги безпеки щодо організації робочих місць

Організація робочого місця передбачає комплекс заходів із його обладнання засобами та предметами праці, а також упорядкування їх розташування. Обслуговування робочого місця означає забезпечення його необхідними ресурсами, інструментами, матеріалами та послугами для ефективного виконання робочих завдань. Основною метою організації робочого місця є забезпечення високої якості та економічної ефективності виконання виробничих завдань у встановлені терміни. Це досягається завдяки повному використанню обладнання, раціональному розподілу робочого часу, впровадженню сучасних методів праці, створенню безпечних та комфортних умов [59].

Специфіка виробництва визначає додаткові фактори, що впливають на організацію робочих місць, такі як баланс фізичної та розумової праці чи рівень відповідальності. Під час проектування робочих місць враховуються санітарно-гігієнічні умови, включаючи освітлення, температуру, вологість, шум, вібрацію, запиленість тощо.

Вимоги до організації робочого місця включають [59]:

– характеристику робочого місця: визначає його фізичні параметри (площа, висота, обсяг), відповідність специфіці діяльності та ергономічність, яка мінімізує зайві фізичні зусилля та ризик стомлення;

– загальні принципи його організації: забезпечення ефективності через чітке планування, раціональне розташування предметів праці та дотримання принципів безпеки;

– оснащення робочого місця: включає необхідне обладнання, інструменти, меблі та допоміжні засоби, які відповідають нормативам безпеки та зручності;

– оптимальне розташування інструментів, матеріалів та обладнання: забезпечує легкий доступ до часто використовуваних предметів, зменшує кількість зайвих рухів і час на виконання завдань;

– опис методів роботи та передових способів виконання завдань: визначає ефективні підходи до виконання операцій, використання нових технологій і зменшення трудомісткості;

– умови праці: включають параметри мікроклімату, освітлення, рівень шуму та інші фактори, які впливають на комфорт і продуктивність працівників;

– вимоги безпеки та охорони праці: регламентують правила користування обладнанням, забезпечення пожежної безпеки та мінімізацію ризиків травматизму;

– документацію, необхідну для роботи: охоплює інструкції, регламенти, технічні описи, формуляри, які використовуються під час виконання робочих процесів.

Під час оцінювання робочого місця слід перевірити відповідність розмірів робочих інструментів пропорціям рук і тіла людини. Важливо визначити зони, зручні та незручні для виконання виробничих операцій, і забезпечити їх узгодження з моторним полем [60]. Правильне облаштування згідно зі встановленими стандартами створює комфортні умови для працівника. Це досягається за рахунок можливості регулювання висоти крісла, кута нахилу або наявності підставки для ніг, а також параметрів робочої поверхні, таких як її висота та розміри [61]. Усі робочі дії повинні здійснюватися в межах моторного поля: зони оптимальної досяжності, легкої досяжності або простого доступу, залежно від частоти й точності рухів.

Робочий простір має бути організований так, щоб забезпечити комфортне положення тіла працівника, безпеку виконання завдань і високу продуктивність праці [60]. Робота в незручних позах, які викликають швидку втому, має бути максимально мінімізована або допускатися лише у виняткових випадках.

Основні правила організації робочого місця [61]:

– на робочому місці повинні бути лише необхідні предмети, без надлишкових об’єктів, які можуть заважати;

– об'єкти, що використовуються часто, слід розташовувати ближче до працівника, ніж ті, що застосовуються рідко;

– для зручності доступу лівою рукою предмети повинні знаходитися зліва, а для доступу правою – справа;

– якщо робота потребує одночасного використання обох рук, розташування предметів має забезпечувати легкий доступ обома руками;

– робочий простір повинен залишатися впорядкованим і не захараченим;

– організація повинна гарантувати працівнику оптимальний огляд робочої зони.

Елементи для відображення інформації мають розміщуватися в межах інформаційного поля робочого місця, враховуючи частоту використання, важливість даних і вимоги до точності та швидкості їх зчитування.

Важливим аспектом забезпечення безпеки праці користувачів комп’ютерів є правильне розташування відеотермінала, клавіатури та принтера на робочому місці. Екран повинен бути встановлений так, щоб забезпечити зручність зорового спостереження у вертикальній площині під кутом ±30° від лінії зору користувача. Оптимальні умови для розпізнавання цифр і символів досягаються, коли верхній край екрану знаходиться на рівні очей, а погляд спрямований вниз на центр монітора. Екран має бути нахилений назад під кутом 20° від вертикалі. Відстань між екраном і очима залежить від розміру діагоналі [60]:

– 35 см – 60–70 см;

– 43 см – 70 см;

– 48 см – 80 см.

Клавіатура повинна мати достатньо простору для переміщення та поворотів на робочій поверхні. Її положення і кут нахилу слід налаштовувати відповідно до уподобань користувача. Кут нахилу клавіатури може варіюватися в межах 5–10° [60].

Одним із ключових інструментів покращення організації праці на робочих місцях є їх атестація. Це процес комплексної оцінки, спрямований на визначення відповідності робочого місця сучасним науково-технічним і організаційним стандартам. Мета атестації полягає у створенні оптимальних умов для

ефективної роботи працівників, що передбачає забезпечення комфортної виробничої обстановки, підвищення продуктивності праці та формування справедливої системи оплати [59].

Основні критерії атестації робочих місць:

– умови праці та техніка безпеки: аналізуються параметри мікроклімату, освітлення, рівень шуму, ергономічність робочого місця, дотримання норм охорони праці;

– техніко-технологічні аспекти: оцінюється технічне обладнання, його сучасність, планування робочого простору, обслуговування та прогресивність технологічних процесів;

– рівень організації праці: враховуються темп виконання завдань, рівень зайнятості, продуктивність і інтенсивність роботи;

– економічна ефективність: визначаються витрати, доходи, якість виконуваних робіт та їхній кінцевий результат.

Результатом атестації є складання паспорта робочого місця – документа, який об’єднує всі вимоги до організації, включаючи умови праці, технічне обладнання та економічні аспекти. Такий підхід дозволяє систематизувати інформацію і встановити єдині стандарти для подальшого вдосконалення [59].

4.1.2 Вимоги до профілактичних медичних оглядів для працівників за ПК

Працівники, які працюють з ПК, повинні пройти обов'язкові медичні огляди для забезпечення їхнього здоров'я та працездатності під час виконання завдань, що вимагають тривалого використання комп'ютерної техніки. До роботи на ПК допускаються лише ті особи, які не мають медичних протипоказань [60].

Існує два види медичних оглядів:

– попередні медичні огляди – це обов'язкові огляди, що проводяться під час прийому на роботу. Метою є виявлення стану здоров'я працівника перед початком роботи з ВДТ, а також оцінка його здатності виконувати роботу, яка

може впливати на здоров'я (наприклад, тривала робота за комп’ютером, що може призвести до проблем з очима, хребтом тощо). Під час цього огляду перевіряється загальний стан організму, а також здійснюється оцінка функціональних можливостей органів зору;

– періодичні медичні огляди – ці огляди проводяться регулярно протягом трудової діяльності працівника, в залежності від умов праці. Згідно з наказом Міністерства охорони здоров'я України № 45 від 31.03.1994 року, такі медичні огляди мають проводитися не рідше ніж раз на два роки. Вони включають в себе огляди фахівцями різних спеціальностей. В разі наявності медичних показань, можуть бути залучені й інші спеціалісти, такі як кардіолог, ендокринолог або психіатр.

Основні критерії для визначення придатності до роботи з відеодисплейними терміналами [62]:

– гострота зору – це один з найважливіших критеріїв, оскільки тривала робота за комп’ютером може призвести до погіршення зору. Працівники повинні мати достатню гостроту зору для комфортної роботи за дисплеєм;

– показники рефракції – це здатність очей правильно фокусувати зображення. У разі виявлення проблем з рефракцією, працівникові можуть бути рекомендовані коригувальні засоби, наприклад, окуляри;

– стан акомодації – це здатність ока змінювати фокусну відстань при роботі з різними об'єктами. Якщо акомодація порушена, працівнику може бути важко зосередитися на екрані комп’ютера, що призводить до стомлення та болю в очах. Перевірка акомодації є важливим етапом медичного огляду;

– функціональність бінокулярного апарату – здатність обох очей працювати разом для створення тривимірного зображення. Порушення бінокулярного зору можуть призвести до дискомфорту при роботі за дисплеєм та головного болю;

– загальний стан здоров’я – крім стану органів зору, важливо враховувати й інші аспекти здоров'я працівника. Хронічні захворювання, такі як гіпертонія або цукровий діабет, можуть впливати на здатність працювати за ПК і вимагати регулярного медичного контролю.

Жінки, які працюють за дисплеєм, зобов’язані проходити обов'язкові огляди акушера-гінеколога раз на два роки. У разі встановлення вагітності або в період годування груддю, жінки не допускаються до виконання робіт, що передбачають тривале використання комп'ютерних терміналів [60]. Це обумовлено можливими ризиками для здоров’я матері та дитини, такими як стрес, перевантаження очей і проблеми з поставою.

Дотримання вимог щодо медичних оглядів має супроводжуватися відповідними профілактичними заходами, які повинні бути впроваджені на робочих місцях. Це включає [62]:

- використання зручних і ергономічних робочих місць, що дозволяють мінімізувати навантаження на органи зору та хребет;
- регулярні перерви під час роботи з комп'ютером;
- забезпечення належного освітлення на робочому місці;
- використання коригувальних засобів, таких як спеціальні окуляри для роботи з комп'ютерами;
- застосування вправ для зниження втоми очей та покращення кровообігу в області шиї та спини.

Дотримання цих вимог, разом із проведенням регулярних медичних оглядів, дозволить зберегти здоров’я працівників і зменшити ризики розвитку професійних захворювань, таких як синдром комп'ютерного зору, проблеми з поставою та болі в спині.

4.2 Безпека в надзвичайних ситуаціях

4.2.1 Проведення рятувальних та інших невідкладних робіт на об’єкті господарської діяльності в осередку ураження

Рятувальні та інші невідкладні роботи здійснюються з метою порятунку людей та надання допомоги постраждалим, локалізації та ліквідації аварій, створення умов для подальшого відновлення діяльності об'єкта.

Види рятувальних робіт [63]:

– проведення розвідки маршрутів до осередку ураження та об'єкта виконання робіт;

– локалізація та гасіння пожеж;

– пошук і порятунок людей з-під завалів або зруйнованих споруд;

– постачання повітря у завалені захисні укриття;

– відкриття завалених захисних споруд і евакуація осіб, що перебувають у них;

– надання першої медичної допомоги постраждалим та їх евакуація до медичних закладів;

– виведення населення із небезпечних зон до безпечних місць;

– проведення санітарної обробки людей, знезараження одягу, техніки, будівель, території, продовольства та води.

Невідкладні роботи в інтересах порятунку людей включають [63]:

– прокладання проходів через завали та заражені території;

– локалізацію та ліквідацію аварій на комунально-енергетичних і технологічних мережах;

– відновлення пошкоджених ліній зв’язку;

– зміцнення або знесення небезпечних конструкцій, що загрожують виконанню рятувальних робіт;

– знешкодження та утилізацію боєприпасів і вибухонебезпечних предметів.

Для проведення рятувальних і невідкладних робіт залучаються сили цивільного захисту, зокрема [64]:

– оперативно-рятувальні служби цивільного захисту;

– воєнізовані аварійно-рятувальні загони;

– спеціалізовані та невоєнізовані формування.

На об’єкті створюються формування із працівників, які поділяються за призначенням [63]:

– формування загального призначення: зведені рятувальні загони, команди, групи, які виконують основні рятувальні роботи;

– формування службового призначення: протипожежні команди, аварійно-технічні загони, санітарні дружини тощо. Ці формування спеціалізуються на виконанні окремих завдань і можуть використовуватися для посилення загальних формувань.

4.2.2 Вплив виробничого середовища на працездатність та здоров'я користувачів комп'ютерів

Хімічні речовини у повітрі робочої зони зазвичай не є характерними для працівників, які працюють за комп'ютерами. Однак, за певних умов, можливий їхній вплив. Наприклад, під час активного використання копіювальної техніки у приміщенні може з'являтися озон, який має гостру дію на організм людини. Також хімічні речовини можуть виділятися з матеріалів, використаних для ремонту чи оздоблення приміщень, таких як синтетичні покриття для підлоги чи полімери на стінах. Найпоширенішою серед них є формальдегід, який може викликати алергічні реакції [65].

Шум у робочій зоні може також впливати на працівників. Для користувачів комп'ютерів допустимий рівень шуму становить 50 дБА, тоді як у приміщеннях з великою кількістю робочих місць цей показник часто перевищується через телефонні розмови чи інші джерела звуку.

Щодо неіонізуючого випромінювання, сучасні рідкокристалічні монітори практично не створюють електромагнітного випромінювання, однак залишаються джерелом електростатичного поля. Його рівень залежить від вологості повітря та чистоти робочого місця [65].

Мікроклімат у приміщеннях також є важливим фактором. У теплий період температура повинна бути в межах 22-25 °С, а в холодний – 21-24 °С, із відносною вологістю 40-60%. Проте у багатьох офісах вологість повітря знижена, а температура часто перевищує норму через нагрівання деталей комп'ютерів. Це може впливати на самопочуття, викликати дискомфорт у слизових оболонках та збільшувати запиленість повітря.

Робоча поза є ще одним значущим фактором. Неправильна організація робочого місця може призводити до остеохондрозу, захворювань кисті рук та загальної втоми. Важливо забезпечити правильну поставу, оптимальну висоту столу й стільця, а також враховувати відстань до монітора [65].

Напруженість праці під час роботи за комп'ютером часто викликає перевтому очей через тривале зосередження уваги. Відсутність перерв, неправильне освітлення, невідповідна яскравість чи мерехтіння екрана можуть погіршувати зір і сприяти розвитку захворювань очей.

Таким чином, дотримання гігієнічних норм та правильної організації робочого місця значно зменшують негативний вплив виробничого середовища на здоров'я користувачів комп'ютерів.

4.3 Висновок до четвертого розділу

У результаті виконання четвертого розділу було проаналізовано основні вимоги безпеки щодо організації робочих місць. Було встановлено, що ефективна організація робочого простору є ключовим чинником забезпечення комфортних умов праці, зниження травматизму та підвищення продуктивності. Значна увага приділялася фізіологічним і психофізіологічним параметрам працівника, дотриманню стандартів ергономіки, а також правильному розташуванню обладнання та інструментів для оптимізації роботи. Крім того, визначено основні принципи профілактичних медичних оглядів для працівників, які працюють із ПК, для забезпечення їхнього здоров'я.

Також у розділі було розглянуто заходи безпеки в надзвичайних ситуаціях, зокрема проведення рятувальних робіт. Окремо досліджено вплив виробничого середовища на працездатність працівників, які використовують комп'ютери. Результати підкреслили важливість забезпечення санітарно-гігієнічних умов, уникнення впливу шкідливих хімічних речовин і створення безпечного робочого середовища. Ці заходи дозволяють мінімізувати ризики для здоров'я працівників і сприяють підвищенню загальної ефективності праці.

ВИСНОВКИ

У результаті виконання дипломної роботи було досліджено та оцінено ефективність різних стратегій оптимізації продуктивності реляційних систем керування базами даних на прикладі PostgreSQL. У роботі теоретично обґрунтовано значення продуктивності баз даних у сучасних інформаційних системах, наведено ключові показники продуктивності та проаналізовано фактори, що на неї впливають. Окрему увагу приділено архітектурі PostgreSQL, включаючи стадії виконання запитів, механізми генерації планів запитів, типи вузлів та їхню роль у продуктивності системи. Акцентовано на важливості використання команди «EXPLAIN» для аналізу виконання запитів.

У дослідженні детально розглянуто поширені стратегії оптимізації, зокрема проектування схеми бази даних, індексацію (типи B-дерево, GIN, GIST, BRIN, хеш-індекси), нормалізацію та денормалізацію, матеріалізовані представлення, а також вибір ефективних форматів первинних ключів, таких як послідовні ідентифікатори, UUID та ULID. Окремо висвітлено підходи до реплікації та шардингу для розподілу навантаження, а також значення конфігурування бази даних, включаючи параметри «shared_buffers», «effective_cache_size», «wal_buffers», «work_mem», «maintenance_work_mem», «synchronous_commit». Розглянуто вплив оптимізації апаратних ресурсів, зокрема використання рівнів RAID.

Практичні експерименти підтвердили ефективність застосованих стратегій. Проаналізовано вплив різних типів індексів, зокрема при повнотекстовому пошуку, та виконано порівняння метрик із застосуванням і без застосування індексів. Також продемонстровано використання UUID у порівнянні з послідовними ідентифікаторами. Налаштування параметрів PostgreSQL дозволило суттєво підвищити продуктивність системи.

Практичне значення дослідження полягає в його застосуванні для оптимізації реальних інформаційних систем. Отримані результати та рекомендації спрямовані на зменшення затримок і ефективного використання ресурсів, що сприяє підвищенню якості обслуговування кінцевих користувачів.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Римарчук О. Підвищення продуктивності системи баз даних в середовищі обчислювального кластеру / Римарчук О. // Матеріали VI всеукраїнської студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання.“, 25-26 квітня 2013 року — Т. : ТНТУ, 2013 — Том 1. — С. 108.

2. Дерень А. Дослідження якості систем керування базами даних / Дерень А. // Матеріали III Всеукраїнської студентської науково-технічної конференції „Природничі та гуманітарні науки. Актуальні питання“, 22-23 квітня 2010 року — Т. : ТНТУ, 2010 — Том 1. — С. 68.

3. Шаряк В. Методи дослідження системних характеристик моделей бази даних / В. Шаряк // Вісник ТДТУ. — Т. : ТДТУ, 2008. — Том 13. — № 2. — С. 116–121.

4. What Is a Database?. Oracle | Cloud Applications and Cloud Platform. URL: https://www.oracle.com/database/what-is-database/ (дата звернення: 07.06.2024).

5. How Databases Can Help Improve Business Performance - The European Business Review. The European Business Review. URL: https://www.europeanbusinessreview.com/how-databases-can-help-improve-business-performance/ (дата звернення: 09.06.2024)

6. Team O. The Importance of Databases in Today's Digital World. OptimizDBA. URL: https://optimizdba.com/the-importance-of-databases-in-todays-digital-world/ (дата звернення: 10.06.2024).

7. Scalability and Performance: Different but Crucial Database Management Capabilities. *Database Trends and Applications*. URL: https://www.dbta.com/Editorial/Think-About-It/Scalability-and-Performance-Different-but-Crucial-Database-Management-Capabilities-161866.aspx (дата звернення: 11.06.2024).

8. The Most Important Database Performance Metrics - DBPLUS Better Performance. DBPLUS Better Performance. URL:

https://dbplus.tech/en/2024/06/06/the-most-important-database-performance-metrics/ (дата звернення: 11.06.2024).

9. Database Performance Metrics to Know for Intro to Database Systems. Focused study guides for every class | Fiveable. URL: https://library.fiveable.me/lists/database-performance-metrics (дата звернення: 13.06.2024).

10. Cache Hit Ratio Metric | DigitalOcean Documentation. Docs Home :: DigitalOcean Documentation. URL: https://docs.digitalocean.com/glossary/cache-hit-ratio (дата звернення: 15.06.2024).

11. Understanding PostgreSQL's Cache Hit Ratio | Redgate. Redgate. URL: https://www.red-gate.com/hub/product-learning/redgate-monitor/understanding-postgresqls-cache-hit-ratio (дата звернення: 16.06.2024).

12. Key Metrics and Tools for Effective PostgreSQL Monitoring. Percona Database Performance Blog. URL: https://www.percona.com/blog/key-metrics-and-tools-for-effective-postgresql-monitoring (дата звернення: 25.06.2024).

13. What factors affect database performance?. Dragonfly - The Fastest In-Memory Data Store. URL: https://www.dragonflydb.io/faq/what-factors-affect-database-performance (дата звернення: 28.06.2024).

14. Team O. Ways to Optimize Your Database Performance. OptimizDBA.com. URL: https://optimizdba.com/ways-to-optimize-your-database-performance/ (дата звернення: 30.06.2024).

15. Contributors to Wikimedia projects. PostgreSQL - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/PostgreSQL (дата звернення: 01.07.2024).

16. Understanding PostgreSQL architecture and attributes. Prisma's Data Guide. URL: https://www.prisma.io/dataguide/postgresql/getting-to-know-postgresql (дата звернення: 03.07.2024).

17. Instaclustr. Understanding the Fundamentals of PostgreSQL® Architecture. Medium. URL: https://instaclustr.medium.com/understanding-the-fundamentals-of-postgresql-architecture-98ce3672376f (дата звернення: 04.07.2024).

18. Ferrari L., Pirozzi E. Learn PostgreSQL - Second Edition: Use, manage and build secure and scalable databases with PostgreSQL 16 : Навчальний посібник. 2nd ed. Birmingham : Packt Publishing, 2023. 745 с.

19. PostgreSQL: ANALYZE and optimizer statistics. CYBERTEC PostgreSQL | Services & Support. URL: https://www.cybertec-postgresql.com/en/postgresql-analyze-and-optimizer-statistics/ (дата звернення: 07.07.2024).

20. The Basics of Postgres Query Planning · pganalyze. Postgres performance at any scale | PostgreSQL Tuning - pganalyze. URL: https://pganalyze.com/docs/explain/basics-of-postgres-query-planning (дата звернення: 12.07.2024).

21. Monitoring Postgres EXPLAIN plans · pganalyze. Postgres performance at any scale | PostgreSQL Tuning - pganalyze. URL: https://pganalyze.com/docs/explain (дата звернення: 15.07.2024).

22. An Overview of the Various Scan Methods in PostgreSQL. Severalnines. URL: https://severalnines.com/blog/overview-various-scan-methods-postgresql (дата звернення: 18.07.2024).

23. Kelkar S. Understanding a Postgres query plan. Medium. URL: https://sskelkar.medium.com/understanding-a-postgres-query-plan-75e8e1dc7d35 (дата звернення: 19.07.2024).

24. One Index, Three Different PostgreSQL Scan Types: Bitmap, Index, and Index Only. Percona Database Performance Blog. URL: https://www.percona.com/blog/one-index-three-different-postgresql-scan-types-bitmap-index-and-index-only (дата звернення: 21.07.2024).

25. Rogov E. PostgreSQL 14 Internals : Навчальний посібник. Moscow : ДМК Пресс, 2023. 548 с.

26. Explaining PostgreSQL EXPLAIN | Timescale. PostgreSQL ++ for time series and events | Timescale. URL: https://www.timescale.com/learn/explaining-postgresql-explain (дата звернення: 31.07.2024).

27. The Importance of Database Performance for Developers | MySQL Database Performance | performed without giving any access to your production data.

MySQL CONSULTANT DBA CANADA. URL: https://ericvanier.com/the-importance-of-database-performance-for-developers (дата звернення: 04.08.2024).

28. Contributors to Wikimedia projects. Database index - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Database_index (дата звернення: 06.08.2024).

29. Smith G. PostgreSQL 9.0 High Performance: Accelerate your PostgreSQL system and avoid the common pitfalls that can slow it down : Навчальний посібник. Birmingham : Packt Publishing, 2010. 468 с.

30. Indexes in PostgreSQL – 1. Postgres Professional. URL: https://postgrespro.com/blog/pgsql/3994098 (дата звернення: 12.08.2024).

31. Contributors to Wikimedia projects. B-tree - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/B-tree (дата звернення: 13.08.2024).

32. Indexes in PostgreSQL – 4 (Btree). Postgres Professional. URL: https://postgrespro.com/blog/pgsql/4161516 (дата звернення: 17.08.2024).

33. Index Types. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/current/indexes-types.html (дата звернення: 21.08.2024).

34. An Introduction to B-Tree and Hash Indexes in PostgreSQL. SolarWinds THWACK Community. URL: https://thwack.solarwinds.com/groups/data-driven/b/blog/posts/an-introduction-to-b-tree-and-hash-indexes-in-postgresql (дата звернення: 22.08.2024).

35. Indexes in PostgreSQL – 5 (GiST). Postgres Professional. URL: https://postgrespro.com/blog/pgsql/4175817 (дата звернення: 24.08.2024).

36. Full Text Search in Milliseconds with Rails and PostgreSQL. pganalyze. URL: https://pganalyze.com/blog/full-text-search-ruby-rails-postgres (дата звернення: 29.08.2024).

37. Holt D. Database Indexes: Trigram. Dan The Engineer. URL: https://dantheengineer.com/database-indexes-trigram/ (дата звернення: 30.08.2024).

38. Fast Search Using PostgreSQL Trigram Text Indexes. The most-comprehensive AI-powered DevSecOps platform | GitLab. URL:

https://about.gitlab.com/blog/2016/03/18/fast-search-using-postgresql-trigram-indexes (дата звернення: 01.09.2024).

39. Postgres Indexing: When Does BRIN Win? | Crunchy Data Blog. Crunchy Data. URL: https://www.crunchydata.com/blog/postgres-indexing-when-does-brin-win (дата звернення: 03.09.2024).

40. Query Optimization. Dremio. URL: https://www.dremio.com/wiki/query-optimization (дата звернення: 05.09.2024).

41. Guide to PostgreSQL Performance | Timescale. PostgreSQL ++ for time series and events | Timescale. URL: https://www.timescale.com/learn/guide-to-postgresql-performance (дата звернення: 06.09.2024).

42. Blaze V. Maximizing PostgreSQL: Advanced Techniques for Better Performance. Medium. URL: https://blog.devgenius.io/maximizing-postgresql-advanced-techniques-for-better-performance-5ff46f3f982c (дата звернення: 07.09.2024).

43. Нормалізація СУБД: приклад бази даних 1NF, 2NF, 3NF. Guru99. URL: https://www.guru99.com/uk/database-normalization.html (дата звернення: 10.09.2024).

44. Kumar Rath M., Kumar Mishra S. Denormalisation : Towards Improved Performance. Srusti Management Review. 2012. Т. 5, № 2. С. 145–152.

45. Gratas B. Denormalization in Databases: Pros, Cons, and Techniques. InvGate ITSM. URL: https://blog.invgate.com/denormalization-in-databases (дата звернення: 14.09.2024).

46. Materialized Views: Precompute with Postgres | Hydra. *Hydra*. URL: https://www.hydra.so/blog-posts/2023-05-30-materialized-views-precompute-with-postgres (дата звернення: 17.09.2024).

47. Svojanovsky T. Unlocking PostgreSQL’s Potential: Exploring Views vs. Materialized Views. Medium. URL: https://blog.stackademic.com/unlocking-postgresqls-potential-exploring-views-vs-materialized-views-8ef1ef76d14f (дата звернення: 25.09.2024).

48. UUID Insecurities. HackTricks. URL: https://book.hacktricks.xyz/ua/pentesting-web/uuid-insecurities (дата звернення: 29.09.2024).

49. Jonathan N. What is ULID and why should you start using it?. DEV Community. URL: https://dev.to/nejos97/what-is-ulid-and-why-should-you-start-using-it-14j9 (дата звернення: 30.09.2024).

50. Advantages of Sharding vs Replication. Double Cloud. URL: https://double.cloud/blog/posts/2022/09/advantages-of-sharding-vs-replication/ (дата звернення: 04.10.2024).

51. How to Benchmark PostgreSQL Performance. Severalnines. URL: https://severalnines.com/blog/benchmarking-postgresql-performance/ (дата звернення: 09.10.2024).

52. Li C. How to Boost PostgreSQL Cache Performance. Medium. URL: https://redfin.engineering/how-to-boost-postgresql-cache-performance-8db383dc2d8f (дата звернення: 10.10.2024).

53. PostgreSQL Performance Tuning: Optimizing Database Parameters for Maximum Efficiency. Percona Database Performance Blog. URL: https://www.percona.com/blog/tuning-postgresql-database-parameters-to-optimize-performance (дата звернення: 11.10.2024).

54. Drljaca B. Hardware Optimization: Best Practices for Database Performance Boost. dzone.com. URL: https://dzone.com/articles/hardware-optimization-best-practices-for-database (дата звернення: 12.10.2024).

55. PostgreSQL Performance Tuning: Optimize Your Database Server. EDB. URL: https://www.enterprisedb.com/postgres-tutorials/introduction-postgresql-performance-tuning-and-optimization (дата звернення: 14.10.2024).

56. What is Docker?. Docker Documentation. URL: https://docs.docker.com/get-started/docker-overview (дата звернення: 15.10.2024).

57. psql. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/9.0/app-psql.html (дата звернення: 17.10.2024).

58. How To Benchmark PostgreSQL Queries Well. Tangram Vision | Sensor Tools & Perception Infrastructure. URL: https://www.tangramvision.com/blog/how-to-benchmark-postgresql-queries-well (дата звернення: 18.10.2024).

59. Організація праці на робочому місці. Вимоги безпеки щодо організації робочих місць. Vseosvita. URL: https://vseosvita.ua/lesson/orhanizatsiia-pratsi-na-robochomu-mistsivymohy-bezpeky-shchodo-orhanizatsii-robochykh-mists-341443.html (дата звернення: 21.10.2024).

60. Голінько В. І. Охорона праці в галузі інформаційних технологій: навч. посіб. / В. І. Голінько, М. Ю. Іконніков, Я. Я. Лебедєв; М-во освіти і науки України, Держ. вищий навч. закл. "Нац. гірн. ун-т". - Дніпропетровськ: НГУ, 2015. - 246 с.

61. Вимоги безпеки щодо організації робочих місць. Buklib. URL: https://buklib.net/books/31185/ (дата звернення: 21.10.2024).

62. Гандзюк М.П. Основи охорони праці: Підручник. 4-е вид./Гандзюк М.П., Желібо Є.П., Халімовський М.О. - Київ: Каревела, 2008. – 384с.

63. Техноекологія та цивільна безпека. Частина «Цивільна безпека»: Навчальний посібник; укл.: Стручок В. С. Тернопіль: ФОП Паляниця В.А., 2022. 150 с.

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

65. Умови праці працівників, які використовують у роботі персональні комп'ютери. Zolochiv.Net. URL: https://zolochiv.net/umovy-pratsi-pratsivnykiv-iaki-vykorystovuiut-u-roboti-personal-ni-komp-iutery/ (дата звернення: 25.10.2024).

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18-19 грудня 2024 року

ТЕРНОПІЛЬ
2024

УДК 001
М34

ПРОГРАМНИЙ КОМІТЕТ

ОРГАНІЗАЦІЙНИЙ КОМІТЕТ

Матеріали XII науково-технічної конфції «Інформаційні моделі, системи та технології» Тернопільського національного технічного університету імені Івана Пулюя, (Тернопіль, 18–19 грудня 2024 р.). – Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2024.
М34

Адреса оргкомітету: ТНТУ ім. І. Пулюя, м. Тернопіль, вул. Руська, 56, 46001,
тел. (0352) 52-41-33, факс (0352) 25-49-83.
E-mail: conffis2024@gmail.com
Редагування, оформлення та верстка: Надія Крива

СЕКЦІЇ КОНФЕРЕНЦІЇ, ЯКІ ПРЕДСТВЛЕНІ В ЗБІРНИКУ

- Математичне моделювання
- Інформаційні системи та технології, кібербезпека
- Комп'ютерні системи та мережі
- Програмна інженерія та моделювання складних розподілених систем
- Новітні фізико-технічні та освітні технології

В збірнику надруковано тези доповідей XII науково-технічної конференції «Інформаційні моделі, системи та технології» (Тернопіль, 18–19 грудня 2024 р.) за такими науковими напрямками: математичне моделювання; інформаційні системи та технології, кібербезпека; комп'ютерні системи та мережі; програмна інженерія та моделювання складних розподілених систем; новітні фізико-технічні та освітні технології.

Розрахований на науковців, викладачів та студентів вузів.

За зміст тез та дотримання норм академічної доброчесності відповідальність несе автор.

УДК 004.65
А. Б. Чекановський; К. Б. Швирло
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

СТРАТЕГІЇ ПОКРАЩЕННЯ ПРОДУКТИВНОСТІ РЕЛЯЦІЙНИХ СИСТЕМ КЕРУВАННЯ БАЗАМИ ДАНИХ

UDC 004.65
A. B. Chekanovskyi; K. B. Shvyrlo

STRATEGIES FOR IMPROVING PERFORMANCE OF RELATIONAL DATABASE MANAGEMENT SYSTEMS

У сучасних інформаційних системах продуктивність баз даних є критичною, адже затримки у виконанні запитів, надмірне споживання ресурсів та ускладнення масштабування можуть впливати на швидкість обробки інформації, зручність роботи користувачів та загальну продуктивність бізнесу. З огляду на це, виникає потреба у впровадженні рішень, які дозволять не лише покращити швидкість і надійність доступу до даних, але й забезпечити гнучкість та стійкість баз даних в умовах підвищених навантажень.

Індексація є основною технікою для прискорення пошуку та вибірки даних. Вона включає одиночні індекси для прискорення виконання запитів, що здійснюють фільтрацію чи сортування на основі одного стовпця, складені індекси, що забезпечують оптимізацію складніших запитів, а також спеціалізовані структури, такі як GIN (Generalized Inverted Index) та GiST (Generalized Search Tree) індекси, які забезпечують ефективну роботу з нетиповими даними або специфічними типами пошуку [1].

Грамотна побудова SQL-запитів, а також використання відповідних інструментів для аналізу та оптимізації коду також дозволяють скоротити час виконання складних запитів і зменшити навантаження на базу даних. Правильне налаштування кешу дозволяє суттєво знизити затрати на обчислення і скоротити час виконання запитів, зменшуючи навантаження на сервер бази даних. Кешування є ефективним способом зменшення кількості звернень до бази даних при однотипних запитах, забезпечуючи швидкий доступ до часто використовуваних даних.

Нормалізація бази даних допомагає уникнути надлишковості даних та оптимізувати структуру. Проте, в деяких випадках, денормалізація може бути також доцільною, особливо коли потрібно мінімізувати кількість JOIN-операцій у запитах для прискорення їх виконання [2].

Реплікація та фрагментування (шардінг) допомагають розподіляти дані між кількома вузлами або серверами, забезпечуючи балансування навантаження та зменшення часу доступу до даних [3]. Апаратні ресурси, зокрема об’єм пам'яті, потужність процесора, тип дискової підсистеми, також грають важливу роль в оптимізації, а для систем з інтенсивним використанням дискової підсистеми, використання різних рівнів RAID може істотно підвищити продуктивність баз даних.

Отже, впровадження вищенаведених стратегій покращення продуктивності реляційних баз даних дозволяє суттєво підвищити ефективність роботи систем.

Література

1. Recommendations for optimizing data. Microsoft Learn. URL: https://learn.microsoft.com/en-us/azure/well-architected/performance-efficiency/optimize-data-performance.
2. Udasi A. Denormalized Data Explained. Zenduty Blog. URL: https://www.zenduty.com/blog/data-denormalization.
3. Learn When to Use Database Replica and Database Sharding for Systems Design. Java Challengers. URL: https://javachallengers.com/database-replica-and-sharding.

Додаток Б

Скрипт для заповнення бази даними

```
const pg = require("pg");
const { faker } = require("@faker-js/faker");

const { Client } = pg;

const client = new Client({
  user: "postgres",
  host: "localhost",
  database: "strategies-pg",
  password: "postgres",
  port: 5433,
});

async function insertCustomers(batchSize = 1000, totalRows =
10000) {
  await client.connect();
  console.log("Inserting customers...");
  try {
    await client.query("BEGIN");
    for (let i = 0; i < totalRows; i += batchSize) {
      const values = [];
      const placeholders = [];

      for (let j = 0; j < batchSize; j++) {
        const firstName = faker.person.firstName();
        const lastName = faker.person.lastName();
        const email = faker.internet.email();

        values.push(
          firstName,
          lastName,
          `${lastName}${i}${j}${email}`.toLowerCase()
        );

        const rowPlaceholder = `($${values.length - 2}, $${
          values.length - 1
        }, $${values.length})`;

        placeholders.push(rowPlaceholder);
      }

      const query = `
        INSERT INTO customers (firstName, lastName, email)
        VALUES ${placeholders.join(", ")}
      `;

      await client.query(query, values);
    }
```

```
    await client.query("COMMIT");
    console.log("Customers inserted.");
  } catch (error) {
    console.error("Error inserting customers:", error);
    await client.query("ROLLBACK");
  }
}

async function insertProducts(batchSize = 1000, totalRows = 5000)
{
  console.log("Inserting products...");
  try {
    await client.query("BEGIN");
    for (let i = 0; i < totalRows; i += batchSize) {
      const values = [];
      const placeholders = [];

      for (let j = 0; j < batchSize; j++) {
        const name = faker.commerce.productName();
        const description = faker.commerce.productDescription();
        const price = faker.commerce.price({ min: 1, max: 500,
dec: 2 });
        const category = faker.commerce.department();

        values.push(name, description, price, category);

        const rowPlaceholder = `($${values.length - 3}, $${
          values.length - 2
        }, $${values.length - 1}, $${values.length})`;

        placeholders.push(rowPlaceholder);
      }

      const query = `
        INSERT INTO products (name, description, price, category)
        VALUES ${placeholders.join(", ")}
      `;
      await client.query(query, values);
    }
    await client.query("COMMIT");
    console.log("Products inserted.");
  } catch (error) {
    console.error("Error inserting products:", error);
    await client.query("ROLLBACK");
  }
}

async function insertOrders(batchSize = 1000, totalRows = 20000) {
  console.log("Inserting orders...");
  try {
    await client.query("BEGIN");

    const customersCountRes = await client.query(
      "SELECT COUNT(*) FROM customers;"
    );
```

```
    const customersCount = customersCountRes.rows[0].count;
    console.log("Customers amount:", customersCount);
    for (let i = 0; i < totalRows; i += batchSize) {
      const values = [];
      const placeholders = [];

      for (let j = 0; j < batchSize; j++) {
        const customerId = faker.number.int({ min: 1, max:
customersCount });

        const status = faker.helpers.arrayElement([
          "pending",
          "shipped",
          "delivered",
          "cancelled",
        ]);

        values.push(customerId, status);

        const rowPlaceholder = `($${values.length - 1},
$${values.length})`;

        placeholders.push(rowPlaceholder);
      }

      const query = `
        INSERT INTO orders (customerId, status)
        VALUES ${placeholders.join(", ")}
      `;

      await client.query(query, values);
    }
    await client.query("COMMIT");
    console.log("Orders inserted.");
  } catch (error) {
    console.error("Error inserting orders:", error);
    await client.query("ROLLBACK");
  }
}

async function insertOrderItems(batchSize = 1000, totalRows =
40000) {
  console.log("Inserting order items...");
  try {
    await client.query("BEGIN");

    const ordersCountRes = await client.query("SELECT COUNT(*)
FROM orders;");
    const ordersCount = ordersCountRes.rows[0].count;
    console.log("Orders amount:", ordersCount);
    const productsCountRes = await client.query("SELECT COUNT(*)
FROM orders;");
    const productsCount = productsCountRes.rows[0].count;
    console.log("Products amount:", productsCount);
```

```
    for (let i = 0; i < totalRows; i += batchSize) {
      const values = [];
      const placeholders = [];

      for (let j = 0; j < batchSize; j++) {
        const orderId = faker.number.int({ min: 1, max:
ordersCount });
        const productId = faker.number.int({ min: 1, max:
productsCount });
        const quantity = faker.number.int({ min: 1, max: 10 });
        const unitPrice = faker.commerce.price({ min: 1, max: 500,
dec: 2 });

        values.push(orderId, productId, quantity, unitPrice);

        const rowPlaceholder = `($${values.length - 3}, $${
          values.length - 2
        }, $${values.length - 1}, $${values.length})`;

        placeholders.push(rowPlaceholder);
      }

      const query = `
        INSERT INTO order_items (orderId, productId, quantity,
price)
        VALUES ${placeholders.join(", ")}
      `;

      await client.query(query, values);
    }
    await client.query("COMMIT");
    console.log("Order items inserted.");
  } catch (error) {
    console.error("Error inserting order items:", error);
    await client.query("ROLLBACK");
  } finally {
    await client.end();
  }
}

async function runInserts() {
  await insertCustomers(5000, 5_000_000);
  await insertProducts(5000, 5_000_000);
  await insertOrders(5000, 5_000_000);
  await insertOrderItems(5000, 5_000_000);
}

runInserts();
```

Додаток В

Скрипт для збирання метрик під час вставки даних залежно від типу первинного ключа

```
#!/bin/bash

PGBENCH_CLIENTS=10
DB_NAME="strategies-pg"
DB_USER="postgres"
DB_HOST="strategies-pg-db"
DB_PORT="5432"

table="orders_serial"
transaction_script="transaction.sql"
pgbench_log_file="serial_pgbench_log.csv"
log_file="serial_insert_performance.csv"
pgbench_error_log="serial_pgbench_error_log.csv"
milestones=(5000 10000 50000 100000)

echo "Row Count,Insert Time (milliseconds),Table Size (KB),Index
Size (KB)" > $log_file
echo "" > $pgbench_log_file
echo "" > $pgbench_error_log

echo "
\set customerId random(1, 1000000)
\set status random(1, 4)
INSERT INTO $table (\"customerId\", status)
VALUES (:customerId,
        CASE :status
            WHEN 1 THEN 'pending'
            WHEN 2 THEN 'shipped'
            WHEN 3 THEN 'delivered'
            ELSE 'cancelled'
        END
);
" >transaction.sql

recreate_table() {
  PGPASSWORD="postgres" psql -h $DB_HOST -p $DB_PORT -U $DB_USER -
d $DB_NAME -c "DROP TABLE IF EXISTS $table;"

  PGPASSWORD="postgres" psql -h $DB_HOST -p $DB_PORT -U $DB_USER -
d $DB_NAME -Xc "
    CREATE TABLE $table (
      id SERIAL PRIMARY KEY,
      \"customerId\" INTEGER REFERENCES customers(id),
      status VARCHAR(50) CHECK (status IN ('pending', 'shipped',
'delivered', 'cancelled')),
      \"createdAt\" TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
    );
  "
  echo "Recreated the table $table"
}

get_table_size() {
  size=$(PGPASSWORD="postgres" psql -h $DB_HOST -p $DB_PORT -U
$DB_USER -d $DB_NAME -t -c "SELECT pg_relation_size('$table') /
1024;")
  echo $size
}

get_index_size() {
  local pkey="${table}_pkey"
  size=$(PGPASSWORD="postgres" psql -h $DB_HOST -p $DB_PORT -U
$DB_USER -d $DB_NAME -t -c "SELECT pg_relation_size('$pkey') /
1024;")
  echo $size
}

clear_table() {
  PGPASSWORD="postgres" psql -h $DB_HOST -p $DB_PORT -U $DB_USER -
d $DB_NAME -c "DELETE FROM $table;"
  echo "Cleared the table $table"
}

insert_and_log_performance() {
  local milestone_idx=0
  start_time=$(date +%s%3N)

  while  [ "$milestone_idx" -lt "${#milestones[@]}" ]; do
    PGPASSWORD="postgres" pgbench --no-vacuum --
transactions=$((milestones[$milestone_idx] / PGBENCH_CLIENTS)) --
client=$PGBENCH_CLIENTS --file=$transaction_script --host=$DB_HOST
--port=$DB_PORT --username=$DB_USER --dbname=$DB_NAME
--report-per-command >> $pgbench_log_file 2>> $pgbench_error_log

    end_time=$(date +%s%3N)
    insert_time=$((end_time - start_time))
    table_size=$(get_table_size)
    index_size=$(get_index_size)
    echo
"${milestones[$milestone_idx]},$insert_time,$table_size,$index_siz
e" >> $log_file
    echo "Inserted ${milestones[$milestone_idx]} rows in
$insert_time milliseconds, Table size: $table_size KB, Index size:
$index_size KB"
    milestone_idx=$((milestone_idx + 1))
    clear_table
  done
}

recreate_table
insert_and_log_performance
```