Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

Оптимізація роботи СКБД PostgreSQL

Текстова частина до курсової роботи
за спеціальністю „Комп'ютерні науки та інформаційні технології" - 122

Керівник курсової роботи
ст. викладач
Захоженко П.О.

_____
(підпис)
"____" _____ 2020 року

Виконав студент КНІТ-4
Чумак В. В.
"____" _____ 2020 року

Київ 2020

Зміст

Анотація

У цій роботі описуються як загальні методи оптимізації роботи баз даних, так і специфічні види оптимізації роботи PostgreSQL. Робота присвячена дослідженню способів пришвидшити роботу даної СКБД на великих об’ємах даних. Це досягається методами загальної оптимізації (налаштування параметрів ОС та параметрів PostgreSQL) та специфічної оптимізації (індекси, оптимізація запитів).

Для демонстрації роботи СКБД була обрана наступна схема: два Docker контейнери з однаковими версіями PostgreSQL та демонстраційними базами даних. Один із оптимізованими налаштуваннями, інший – з параметрами за замовчуванням. Результатом роботи є «бенчмарки» – показники швидкості роботи запитів до обох баз, демонстрація приросту швидкості роботи.

Після проведення тестів будуть створені графіки приросту оптимізації, які покажуть чи стали покращення дієвим та наскільки вони підвищили швидкість роботи системи.

Вступ

В еру цифрових технологій людство все частіше стикається з проблемою обробки великих обсягів даних. Якщо декілька десятиліть тому між швидкодією та використаною пам’яттю обиралось друге, то зараз зберігання великих обсягів даних не є проблемою. Основною проблемою виявилась неможливість швидко та зручно їх обробляти та аналізувати. Маніпуляції з великими обсягами даних часто займають купу часу, що просто не припустимо. Це стосується будь-яких сховищ даних, зокрема і баз даних.

У сучасному світі інформаційні системи, реалізовані з використанням баз даних, знайшли застосування практично у всіх сферах діяльності людини. Очевидно, що для підвищення ефективності та швидкості їх роботи, необхідна оптимізація найпоширеніших операцій, якими є пошук даних та здійснення доступу до інформації. Це дозволить заощадити витрати ресурсів і часу для обробки великих об’ємів даних. Така оптимізація включає в себе максимальне прискорення доступу до інформації і вибір найкращого способу пошуку даних.

Для прискорення роботи на великих обсягах інформації використовуються різноманітні методи оптимізації, часто евристичні алгоритми. Серед найпопулярніших методів: оптимізація на рівні ОС, специфічні налаштування СКБД, використання індексів, розробка коректної схеми БД, використання нормалізації, правильне написання запитів.

У цій роботі буде розглянута конкретна СКБД – PostgreSQL та основні методи оптимізації її роботи. PostgreSQL досить популярна СКБД, але, важливіше те, що вона більш гнучка, ніж її аналоги. З точки зору простоти адміністрування, порівнюючи, наприклад, з MySQL, PostgreSQL програє, і це не тому, що там все гірше продумано, просто там більші можливості для налаштувань. Дана СКБД дозволяє налаштовувати все, що потрібно для конкретної задачі, це така собі «зброя», якою потрібно вміти користуватися.

1. Методи загальної оптимізації.

1.1. Оптимізація параметрів ОС

Думаючи про прискорення роботи якоїсь програми або сервісу, перше, що спадає на думку – це прискорення ОС на якій цей застосунок працює. СКБД записують і читають дані з диску, операційні системи в цьому відіграють роль провідника, який дозволяє зберегти деякі дані в швидкій пам’яті, щоб їх можна було легко дістати. Зазвичай на серверах використовують дистрибутиви Linux, тому саме про прискорення цієї ОС йтиметься мова.

Для налаштувань параметрів ОС Linux існує файл sysctl.conf, який відповідає за управління окремими параметрами ядра, безпеки, мережевої підсистеми, дозволяє визначати такі значення як: розмір сегменту роздільної пам’яті, обмеження на кількість запущених процесів і тд.

Перше, що варто налаштувати – це підкачка пам’яті swap. В ОС Linux за це відповідає параметр vm.swappiness. За замовчуванням значення цього параметру відповідає 60%, а це означає, якщо 40% (100-60) оперативної пам’яті зайнято, то система записує дані на диск. Це важливо для систем з малою кількістю оперативки, але, маючи, наприклад, 16гб оперативної пам’яті, починати запис на диск при заповнених 6.4гб – не дуже вдала ідея, тому варто зменшити значення цього параметру, наприклад, до 10, а у випадку, коли на машині лише база даних значення може взагалі бути близьким до нуля.

Наступним для налаштування є параметр overcommit_memory. Він відповідає за стратегію overcommit та може мати 3 значення:

- 0 – OVERCOMMIT_GUESS – евристичний підхід до розподілу пам’яті. В ньому виділяється стільки пам’яті, скільки потрібно процесу, але, наприклад, в swap потрапляють лише ті сторінки, які ним використовуються. Оскільки пам’ять виділяється на основі

евристичного алгоритму, а не точного, це може призвести до перевищення допустимого навантаження на пам’ять

- 1 – OVERCOMMIT_ALWAYS – overcommit пам’яті є завжди. Ядро не опрацьовує перерозподіл пам’яті, при цьому імовірність перевищення навантаження виростає, як і продуктивність
- 2 – OVERCOMMIT_NEVER – без overcommit – відмова обробки запитів, які запитують пам’ять, сумарний розмір якої перевищує swap+ram*overcommit_ratio/100, де swap – розмір підкачки, ram – кількість оперативної пам’яті, overcommit_ratio – значення параметру, яке за замовчуванням рівне 50

Вибір стратегії також залежить від того, чи СКБД сама на сервері, адже якщо ні, то може виникнути ситуація, коли інші процеси «з’їдять» стільки пам’яті, що ОС вб’є процес СКБД, тому, якщо база не сама на машині, overcommit краще вимкнути, задля уникнути помилок out of memory.

Наступним методом оптимізації на рівні ОС може стати прискорення операцій введення/виведення. Як і в більшості операцій з базами даних, ефективність дискових операцій вважається одним із способів підвищення продуктивності та оптимізації. Існує декілька загальних технологій, як, наприклад, стискання даних. При роботі з великою кількістю інформації, стискання застосовується не лише до даних в таблицях, але й до журналів, які зберігають інформацію про самі дані. Деякі з методів стискання і справді прості, як наприклад відступів чи хвостових нулів, або об’єднання декількох журнальних записів в один. Деякі системи використовують автоматично, наприклад, відкладений запис (write-behind), випереджальне читання (read-ahead), попередня вибірка (prefetch).

У сценарії write-behind записи додаються до джерела даних після налаштованої затримки. Це дозволяє краще налаштувати роботу під час пікових ситуацій, коли система не встигає записати дані через їх кількість. В сучасних

дискових пристроях підтримуються власні черги команд, тому записи виконуються краще.

Випереджальне читання – стратегія організації операцій введення та виведення, за якої запити на читання блоків, розташованих за поточною областю читання, видаються одночасно із запитами на читання поточних блоків. Таке читання зазвичай використовується при скануванні. Розглянемо приклад для кращого розуміння:

Нехай диск має час доступу близько 5.5мс, тобто 182 операції введення виведення (IO) за секунду. Нехай системі потрібно записати блок розміром в 256KB, тоді максимальна пропускна здатність прямого запису складе:

182*256 = 46545 KB/s (46 MB/s)

Якщо застосувати стратегію read-ahead, цей результат можна значно покращити. Уявімо, що система відправила запит на блок номер 500, тоді при включеному режимі випереджального читання система одразу запросить і блоки 501, 502, 503 та 504. Таким чином їй не доведеться чекати 5.5мс для запиту блока 501, а це значно пришвидшить алгоритм послідовного читання.

Попередня вибірка застосовується не лише до видобутку даних, але й до операцій їх оновлення. Prefetch, як і попередні методи, не приводить до підвищення продуктивності, а лише зменшує час реакції чи затримку виконання окремих операцій, що може призвести до підвищення пропускної здатності.

Для пришвидшення операцій введення/виведення в Linux можна налаштувати свого планувальника. Для прикладу розглянемо планувальник CFQ(Completely Fair Queuing). Він рівномірно розподіляє ресурси між процесами, відповідно до пріоритетів. Процеси діляться на три класи: realtime, best effort та idle. Пріоритет можна змінити вручну за допомогою команди ionice. За замовчуванням процеси обслуговуються в порядку черги best effort. Процеси класу realtime мають найвищий пріоритет, тому його варто використовувати з обережністю. Останніми обслуговуються бездіяльні процеси (idle).

1.2. Налаштування PostgreSQL

Стандартна версія PostgreSQL більше розрахована на високу сумісність, а не на продуктивність. Тому, для кращої роботи, СКБД повинна бути налаштована, відповідно до задач, які вона має виконувати. Налаштуванням можна керувати різними способами, але як правило, зміни відбуваються за допомогою ALTER SYSTEM (ALTER DATABASE) або безпосередньо у конфігураційному файлі – *postgresql.conf*.

Для автоматизації налаштування існують спеціалізовані інструменти, які дозволяють автоматично виявити «слабкі» місця продуктивності, проаналізувати конфігурацію і надати рекомендації щодо налаштування, аналізувати журнали для створення звітів про ефективність. Серед таких : db Forge Studio для PostgreSQL, Postgresqltuner.pl, PgBadger, pgMustard.

Параметри для налаштування мають різні рівні доступу та гнучкості, для цього їх було розділено на відповідні рівні:

- Postmaster – вимагає перезавантаження серверу
- Sighup – потребує сигналу HUP(працюючий сервер перезавантажить конфігураційний файл без припинення роботи)
- User – значення змінюється лише в рамках сеансу і набуває чинності лише в рамках даного сеансу
- Internal – встановлюється тільки під час компіляції, використовується в основному для довідок
- Backend – налаштування, які повинні бути встановлені до початку сеансу
- Superuser – налаштування, які можуть бути встановлені під час роботи сервера, але лише суперюзером

Самі параметри налаштування приймають значення одного з п’яти типів:

- Логічний – значення можуть задаватись рядками on, off, true, false, yes, no, 1,0
- Рядок – значення обмежується апострофами
- Число (ціле або з рухомою комою) – задаються у звичайних числових форматах, підтримується шістнадцятковий (0x) та вісімковий (0) формати
- Число з одиницею вимірювання – пишеться в форматі числа та рядка, яке символізує формат (наприклад 3kb, 30s, 512mb)
- Перерахування (enumeration) – записуються рядками через кому

Для налаштування СКБД існує десяток параметрів, значення яких можна змінювати або в конфігураційному файлі, або за допомогою командного рядка.

Деякі з параметрів для налаштувань:

- listen_addresses – для створення мережевої підсистеми, яка буде складатись із більше, ніж одного комп’ютера (сервер знаходиться на окремому комп’ютері)
- max_connections – параметр, який відповідає за максимальну кількість одночасних з’єднань, які обслуговуватиме сервер
- shared_buffers – параметр, який відповідає за кількість виділеної пам’яті для кешування даних. Рекомендується використовувати 15-25% всієї доступної оперативної пам’яті
- effective_cache_size – параметр, що відповідає за кількість пам’яті для дискового кешування
- checkpoint_segments – розмір сегменту, на які ділиться інформація при записі в базу даних, чим менша кількість сегментів – тим скоріше проходить запис
- work_mem – збільшення значення цього параметру дозволяє швидше виконувати операції сортування та вибірки даних. Параметр виділяє задану кількість пам’яті під кожну таку операцію

- maintaince_work_mem – параметр, який відповідає за кількість пам’яті виділеної для різноманітних статистичних та управляючих процесів (VACUUM, CREATE INDEX, ALTER TABLE ADD FOREIGN KEY)
- wal_buffers – об’єм пам’яті, який буде використано для буферизації даних, ще не записаних на диск. Рекомендується збільшувати в системах з великою кількістю записів.
- synchronous_commit – включення/виключення синхронного запису в log-файл після кожної транзакції. Захищає від можливої втрати даних, але накладає обмеження на пропускну здатність сервера.
- autovacuum – відповідає за включення/виключення автоматичної очистки. По замовчуванню – включено. Для роботи автоматичної очистки обов’язково має бути ввімкненим параметр track_counts
- max_prepared_transactions – задає максимальне число транзакцій, які можуть знаходитись в режимі підготовки

Отже, налаштування СКБД PostgreSQL може бути дуже гнучким та зручним, кожен з параметрів дозволяє змінити роботу сервера для спеціалізованої задачі. Виходячи з вимог до роботи СКБД та застосованих до неї схем баз даних, будь-які налаштування можна змінити задля оптимізації та пришвидшення роботи.

2. Методи специфічної оптимізації

2.1. Схема БД

Не менш дієвим способом оптимізації роботи будь-якої СКБД є правильна організація роботи з даними. Для зберігання інформації в базі даних використовуються схеми баз даних, які будуються, виходячи з потреб системи та замовника. Для розумного проектування баз даних використовуються, так звані, нормальні форми – властивості відношення в реляційній моделі даних, які характеризують його з точки зору надмірності. Нормалізація (приведення до нормальних форм) – процес розбиття реляції відповідно до алгоритму.

Всього існує десять нормальних форм, але застосування їх всіх не є обов'язковим, адже кожна задача має своє вирішення, а тому приведення до певної форми може не задовільнити бізнес логіку застосунку. Кожна наступна нормальна форма зберігає властивості попередньої та покращує їх. На практиці найчастіше застосовуються перші три нормальні форми:

- 1НФ – збережені дані на перетині рядків і стовпчиків повинні мати одне значення, а таблиці не повинні мати рядків, що повторюються
- 2НФ – кожен стовпчик, який не є ключем повинен залежати від первинного ключа
- 3НФ – кожен стовпчик, який не є ключем, повинен залежати тільки від первинного ключа

Нормалізація дозволяє позбутись не лише надлишкової інформації, а й надлишкових операцій для її обробки (запису, читання, видалення або оновлення), адже її метою є позбутись дублювання даних. Нормалізація покликана запобігти аномаліям при роботі з базою даних. Аномалії – ситуації, які призводять до протиріч у роботі бази даних, або суттєво ускладнюють її обробку. Аномалії діляться на три групи:

- Аномалії модифікації – оновлення одних даних призводить до оновлення інших
- Аномалії видалення – при видаленні певного кортежу може зникнути інформація, яка напряму не зв’язана з даним кортежем
- Аномалії запису – ситуація, коли дані не можна записати в таблиці, бо вони не є повними, або коли для запису в таблицю треба переглядати всі дані цієї таблиці

Такі аномалії не лише ускладнюють роботу бази даних, але й роблять її ненадійною та нестійкою. Саме процес проектування бази даних за допомогою нормальних форм дозволяє уникнути та запобігти вищевказаним аномаліям. Такий процес вважається ітераційним і полягає у послідовному переведенні реляції (таблиці) з 1НФ у нормальну форму більш високого порядку.

Для специфічних задач в базі даних існує зворотний процес – денормалізація. Адже нормальні форми переважно потрібні для безпроблемного запису, але іноді в таблиці свідомо надається надлишкова інформація для швидшого читання даних. Цей процес полягає не лише в дублюванні даних в реляціях, а й в дублюванні цілих таблиць, об’єднанні декількох таблиць в одну. Це свідома ціна при записі, яку платять для швидшого читання даних. Звичайно, такий метод вимагає і більше пам’яті для зберігання, і більше часу на оновлення даних, створення індексів, але все ще – він дозволяє виграти при видобуванні інформації, саме тому є таким специфічним.

Вибір в сторону нормалізації або денормалізації потрібно робити, виходячи з глибокого аналізу предметної області та технічного завдання системи, адже кожен з методів корисний у тій чи іншій ситуації. Для правильного вибору варто провести аналіз затрат та виграшів, побудувавши схему, яка покаже, що розробники виграють і втрачають при одному варіанті, а що – при іншому.

2.2. Індекси

Індекс – об’єкт бази даних, створений задля підвищення ефективності видобутку інформації. Використовуючи індекси сервер баз даних може знаходити потрібну інформацію швидше, але з ними також пов’язане додаткове навантаження на СКБД, саме тому їх варто використовувати обдумано.

Уявімо, що існує таблиця з довільними атрибутами, та ключем – id. Ця таблиця часто використовується для видобутку інформації з неї за допомогою запитів, а, отже, якщо система не підготовлена до подібної роботи, їй доведеться щоразу проходити таблиця повністю та шукати потрібний рядок. Якщо таблиця містить велику кількість рядків, а запити до неї видобувають лише декілька з них, або навіть жодного, очевидно, що таке сканування не є ефективним. Саме в такий момент індекси стають в нагоді.

Для створення індексу в СКБД PostgreSQL потрібно застосувати наступну команду:

CREATE INDEX table1_id_index ON table1 (id);

де: table1_id_index – назва індексу, table1 – назва таблиці, id – колонка по якій створюється індекс.

Після створення індексу жодних додаткових дій не потрібно, адже система сама буде оновлювати його після запису в таблицю, або, навпаки, видалення, а, також, використовувати його в запитах, де це буде ефективніше, ніж сканування всієї таблиці.

СКБД PostgreSQL підтримує декілька типів індексів: Б-дерево (B-tree), хеш, GiST, SP-GiST, GIN та BRIN. Тип індексу залежить від алгоритму, який лежить в його основі. За замовчуванням команда CREATE INDEX створює індекс типу Б-дерево, який вважається оптимальним у більшості випадків.

Індекси, як і ключі таблиць, можуть бути складеними і складатись з декількох стовпчиків таблиці. Такі індекси вважаються ефективними, коли постійні запити пошуку по таблиці проходять не лише по ключу, або якомусь одному полю, а по декількох одразу.

Ще одна корисна можливість індексів – застосування для сортування даних. Це дозволяє врахувати команду ORDER BY, не виконуючи додаткове сортування. З усіх типів індексів, таку можливість мають лише ті, що построєні на Б-деревах. Адже, за замовчуванням елементи Б-дерева зберігаються в порядку зростання, а це означає, що при скануванні індексу можна врахувати порядок, заданий ORDER BY.

При створенні індексу можна вказати порядок, в якому будуть зберігатись елементи. Для цього достатньо додати його в дужки, де вказано колонку з таблиці, наприклад:

CREATE INDEX table1_id_index ON table1 (id DESC NULLS LAST);

Враховуючи, що кортежі в базі даних можуть бути NULL, вказуємо також їхній порядок зберігання.

Варто зазначити, що використовувати індекси потрібно дуже обережно, адже при великій кількості даних сповільнюється запис в таблицю (через потребу оновити індекс). Самі індекси можуть займати досить великий обсяг місця на диску, а, відповідно, в пам’яті, за умови, що вони кешуються. При інтенсивному записі в таблицю, дані її індексів не завжди знаходяться на тій сторінці, на якій повинні, тому утворюються «пропуски» і доводиться робити дефрагментацію індексів, а в такі періоди СКБД працює повільніше.

Отже, індекси справді прискорюють видобування даних з бази, але і тут доводиться чимось жертвувати. Варто пам’ятати, що погані ті індекси, які не використовуються, або використовуються дуже рідко.

2.3. Оптимізація запитів

Не зважаючи на важливість правильного проектування схеми бази даних, не менш важливим аспектом є написання запитів. Ця задача полягає на плечі програміста, адже запити мають бути написані таким чином, щоб СКБД могла вибирати більш ефективні способи знаходження даних.

В основі мови SQL лежить реляційна алгебра Кодда. Саме через це часто легше сформувати запит за допомогою реляційної алгебри, а вже потім писати його на SQL.

Розглянемо дві множини R1 та R2. Нехай обидві множини мають по 10000 кортежів, та існує умова F, якій відповідають лише 100 кортежів з обох множин. Якщо застосувати предикат F до об’єднання множин (UNION) наступним чином:

sF(R1 U R2),

то, після об’єднання, ми отримаємо множину, яка буде складатись із 20000 елементів, і до якої буде застосовуватись предикат. Якщо ж записати цю дію, змінивши порядок операцій:

sF(R1) U sF(R2),

то після вибірки залишаться дві множини по 100 кортежів, які необхідно буде об’єднати. Враховуючи, що об’єднання виконується шляхом сортування даних, для видалення дублів, і проміжний результат має зберігатись в пам’яті, то даний порядок операцій набагато виграшний за перший, хоча на перший погляд це може не здатись таким очевидним.

Сучасні СКБД мають власні оптимізатори запитів до бази даних, де такі прості проблеми, як описані вище, враховуються і виправляються. Так оптимізатори будують декілька можливих планів виконання запиту і, за

допомогою евристичних методів, вибирають кращий. Такі методи дозволяють значно звузити множину всіх можливих планів виконання, адже ті з них, які вважаються неефективними, оптимізатор одразу виключає з множини. Один з прикладів евристичних методів: хорошим вважається той план, в якому вибірка відбувається на ранніх етапах виконання. Евристика часто базується на роботі з операціями реляційної алгебри Кодда.

Робота таких оптимізаторів часто називається оптимізацією за вартістю. Вартість – оцінка очікуваного часу виконання запиту, в яку можуть враховуватись різні фактори: кількість необхідних ресурсів пам’яті, час процесора, час операцій дискового вводу і тд. Вартість плану виконання також визначається на основі інформації про розподіл даних по таблицях, до яких звертається запит. Така інформація називається статистикою і зберігається в словнику даних. Для однієї реляції статистика може містити:

- Кількість блоків даних, виділених таблиці
- Кількість записів в таблиці
- Середню довжину запису
- Кількість індексованих значень
- Середню кількість записів у блоці

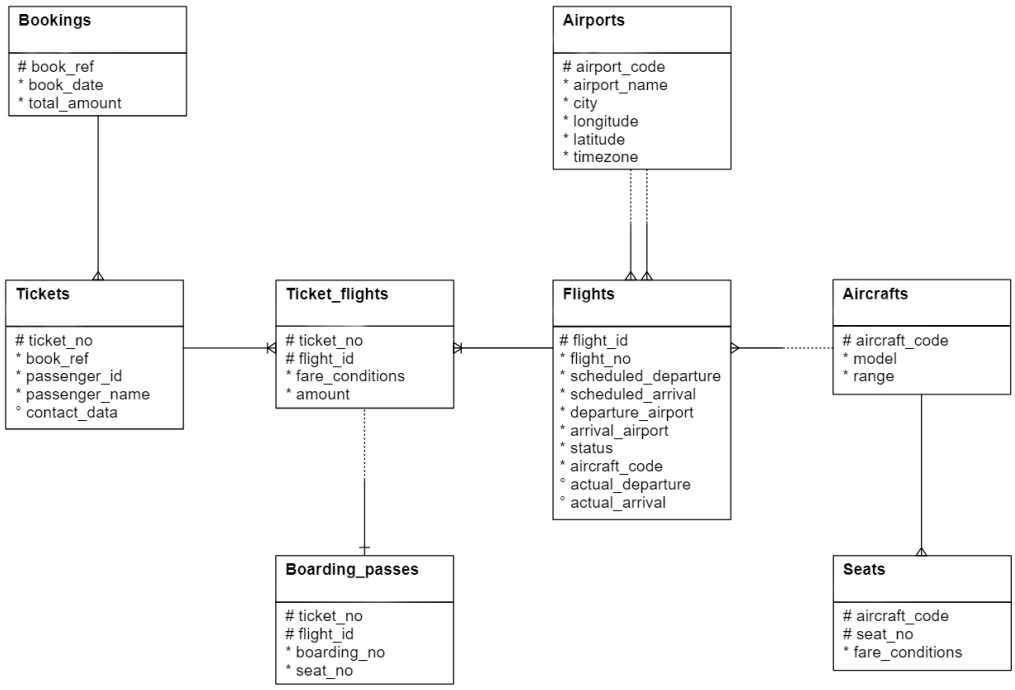
Написання запитів може сильно залежати від предметної області та особливо схеми даних. Якщо враховувати ці фактори, можна розробити специфічні запити, які будуть працювати швидше, адже, хоч СКБД і мають оптимізатори, вони не завжди можуть правильно визначити наскільки один запит кращий від іншого. Саме тому правильне написання запитів – це ще одна складова оптимізації роботи з базою даних.

3. Практична реалізація.

3.1. Підготовка середовища

Для демонстрації оптимізації роботи СКБД було вирішено використати два Docker контейнери зі встановленими однаковими версіями PostgreSQL, з яких один – з оптимізованими параметрами, а інший – з параметрами за замовчуванням.

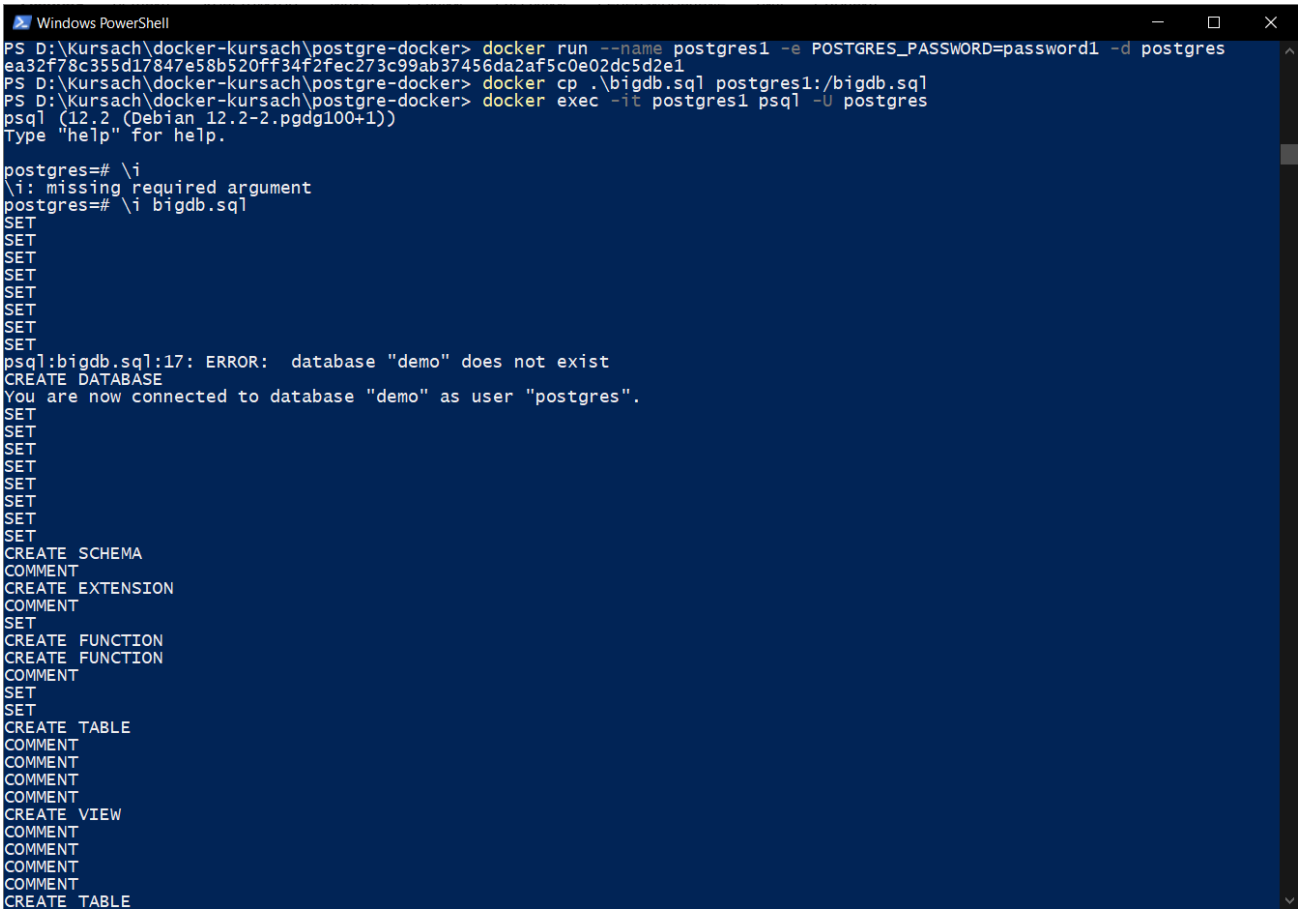
Для заповнення бази тестовими даними використовується демонстраційна база даних авіаперельотів.



Зображення 1

3.1.1. Встановлення та налаштування

Створимо два docker контейнери з однаковою версією PostgreSQL та наповнимо їх даними.



Зображення 2

Робиться за допомогою наступних команд:

- docker run – створення та запуск контейнера
- docker cp – копіювання файлу з жорсткого диску до контейнера
- docker exec – запуск команди в контейнері
- \i bigdb.sql – виконання файлу з даними

Аналогічно налаштовуємо і другий контейнер. Контейнер postgres1 – буде оптимізованим, а postgres2 – з параметрами за замовчуванням.

3.1.2. Написання тестових запитів

1) Для кожного пасажира вивести кількість аеропортів, в які він прибував

```
SELECT passenger_name, COUNT(airport_code)
FROM ((tickets INNER JOIN ticket_flights ON
tickets.ticket_no=ticket_flights.ticket_no)
INNER JOIN flights ON ticket_flights.flight_id=flights.flight_id)
INNER JOIN airports ON flights.departure_airport=airports.airport_code
GROUP BY passenger_name;
```

2) Для кожного літака вивести кількість пасажирів, яку він перевіз

```
SELECT model, COUNT(ticket_flights.flight_id)
FROM (aircrafts INNER JOIN flights ON aircrafts.aircraft_code =
flights.aircraft_code ) INNER JOIN ticket_flights ON flights.flight_id =
ticket_flights.flight_id
GROUP BY model;
```

3) Для кожного пасажира порахувати суму коштів, яку він витратив на квитки

```
SELECT passenger_name, SUM(ticket_flights.amount)
FROM tickets INNER JOIN ticket_flights ON
tickets.ticket_no=ticket_flights.ticket_no
GROUP BY passenger_name;
```

4) Для кожного аеропорту порахувати кількість літаків, яку він прийняв

```
SELECT airport_name, COUNT(flights.aircraft_code)
FROM (airports INNER JOIN flights ON flights.departure_airport =
airports.airport_code) INNER JOIN aircrafts ON
flights.aircraft_code=aircrafts.aircraft_code
GROUP BY airport_name
ORDER BY COUNT(flights.aircraft_code);
```

5) Вибірка даних з найбільшої таблиці (ticket_flights)

```
SELECT * FROM ticket_flights;
```

3.1.2. Налаштування для оптимізації

Для оптимізації роботи системи змінимо налаштування підкачки, встановивши параметру vm.swappiness значення 10. Таким чином ми жертвуємо 90% оперативної пам’яті для швидких операцій.

Для прискорення роботи СКБД в опрацюванні запитів, налаштуємо деякі з параметрів:

- shared_buffers = 1024MB – визначає кількість пам’яті виділеної для кешування даних
- work_mem = 40MB – кількість пам’яті буде виділено на операції сортування та вибору даних
- maintenance_work_mem = 150MB – кількість пам’яті для статистичних і управляючих процесів
- effective_cache_size = 4GB – допомагає планувальнику визначити кількість вільної пам’яті для дискового кешування

Такі значення параметрів були вибрані шляхом підбору та результатів бенчмарків. Для конкретно цієї бази даних вони показали найвищі цифри приросту швидкої.

Під час написання запитів використовуємо операції INNER JOIN замість декартового добутку, щоб прискорити вибірку даних, також створимо індекси по основних колонках таблиць для швидшої обробки. Індекси додаємо лише в таблиці, які найчастіше використовуються в запитах, щоб не навантажувати СКБД зайвими, які використовуватись не будуть.

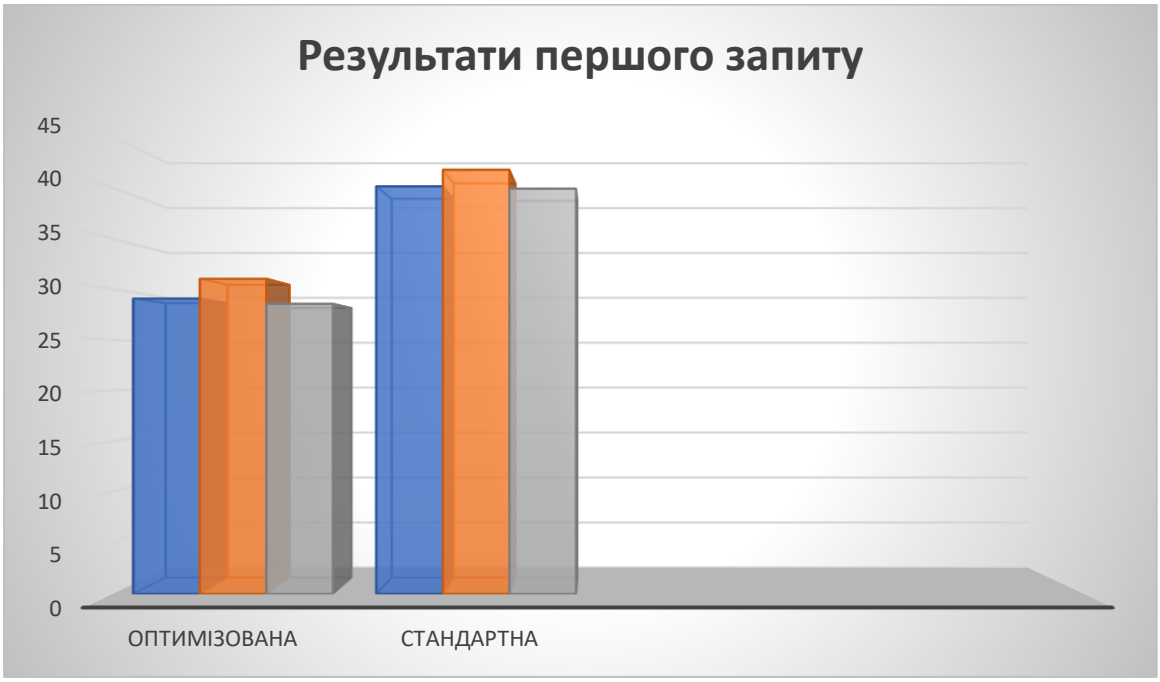
Аналізуючи схему бази даних, можна помітити, що не у всіх таблицях дотримано умов нормальних форм. Наприклад, в таблиці Tickets, пасажир вписаний атрибутами, хоча краще було б відокремити його в свою таблицю.

Оскільки схема взята для прикладу, змінювати її не будемо. Залишається спиратись на оптимізацію запитів, роботи СКБД та ОС.

Для швидшого запису даних змінимо параметр checkpoint_segments. Адже він визначає розміри частин, на які ділиться інформація при записі до бази даних.

3.2. Результати

Протестувавши запити, написані вище, можна зробити певну статистику. Кожен запит тестується по три рази і для результату обирається середній показник. Для кращої візуалізації представимо результати у вигляді графіків:



Діаграма 1

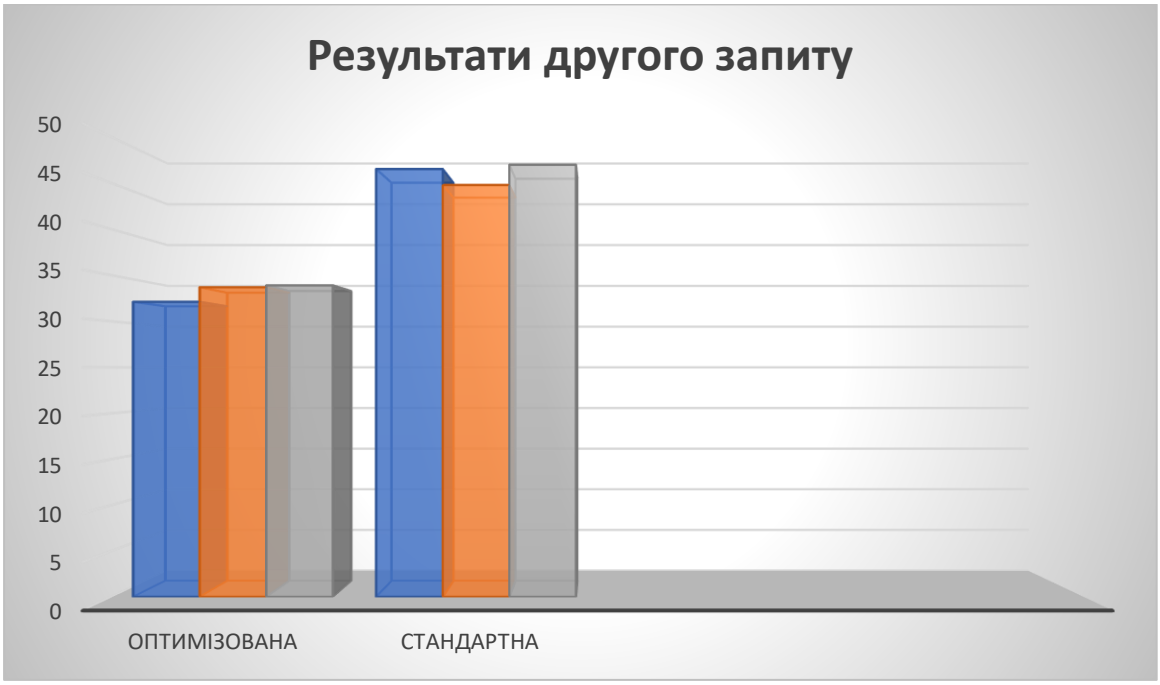
Оптимізована база даних дала результат 29.058с, база із стандартними налаштуваннями показала результат 40.122с. Оптимізація показала приріст швидкості 27.6%. Виконання запиту:

```
demo=# SELECT passenger_name, COUNT(airport_code)
FROM ((tickets INNER JOIN ticket_flights ON tickets.ticket_no=ticket_flights.ticket_no)
INNER JOIN flights ON ticket_flights.flight_id=flights.flight_id)
INNER JOIN airports ON flights.departure_airport=airports.airport_code
GROUP BY passenger_name;
     passenger_name      | count
-------------------------+-------
 SABINA ILINA            |     2
 ARSENIY ZHUKOV          |    37
 RAMAZAN KUZMIN          |     3
 ILMIR FEDOTOV           |     4
 EDGAR GUSEV             |     4
 ELVIRA NIKIFOROVA       |    59
 TALGAT VOROBEV          |     9
 SALAVAT NAZAROV         |     8
 SVETLANA BOGDANOVA      |   990
 LIYA ISAEVA             |     8
 SABINA GRISHINA         |     3
 FLYURA KOMAROVA         |    10
 FELIKS VOLKOV           |    10
 KRISTINA MARTYNOVA      |   169
 RIFAT MOROZOV           |    16
 ARTEM DMITRIEV          |   495
 RUSTYAM KUZNECOV        |    10
 VALERIY VLASOV          |   674
 RAFIK KUZNECOV          |    61
 RUSHANIYA ZOTOVA        |     8
 ZAKHAR KISELEV          |    34
 ELLA DMITRIEVA          |     7
 FANIL KARPOV            |    14
 RITA MELNIKOVA          |    10
 ANZHELIKA SIDOROVA      |    59
 GULFIYA MOROZOVA        |    29
 TAISIYA KRASNOVA        |   113
 PELAGEYA NIKOLAEVA      |    38
 SABINA SCHERBAKOVA      |     4
 KIRILL KOZLOV           |   460
 LARISA KUZMINA          |   506
 SAMIRA ANTONOVA         |    15
 ILONA KAZAKOVA          |     1
 VIKTOR TIKHONOV         |  1301
 RUZALIYA SOROKINA       |     8
 LIDIYA POTAPOVA         |   421
 GEVORG SOROKIN          |     6
 VADIM TIMOFEEV          |   164
 NATALIYA FROLOVA        |   389
 EVGENIY MELNIKOV        |  1367
 RAILYA NOVIKOVA         |     8
 ANTONINA ZOTOVA         |   225
 NIKITA MIRONOV          |   491
 LUIZA TIKHONOVA         |    14
 ANTONINA KULIKOVA       |   326
 RUMIYA FILIPPOVA        |     5
 ANAIT MARTYNOVA         |     6
Time: 29058.414 ms (00:29.058)
```

Зображення 3

```
demo=# SELECT passenger_name, COUNT(airport_code)
demo-# FROM ((tickets INNER JOIN ticket_flights ON tickets.ticket_no=ticket_flights.ticket_no)
demo(# INNER JOIN flights ON ticket_flights.flight_id=flights.flight_id)
demo-# INNER JOIN airports ON flights.departure_airport=airports.airport_code
demo-# GROUP BY passenger_name;
     passenger_name      | count
-------------------------+-------
 SABINA ILINA            |     2
 ARSENIY ZHUKOV          |    37
 RAMAZAN KUZMIN          |     3
 ILMIR FEDOTOV           |     4
 EDGAR GUSEV             |     4
 TALGAT VOROBEV          |     9
 ELVIRA NIKIFOROVA       |    59
 SALAVAT NAZAROV         |     8
 SVETLANA BOGDANOVA      |   990
 LIYA ISAEVA             |     8
 SABINA GRISHINA         |     3
 FLYURA KOMAROVA         |    10
 FELIKS VOLKOV           |    10
 KRISTINA MARTYNOVA      |   169
 RIFAT MOROZOV           |    16
 ARTEM DMITRIEV          |   495
 RUSTYAM KUZNECOV        |    10
 RAFIK KUZNECOV          |    61
 VALERIY VLASOV          |   674
 ZAKHAR KISELEV          |    34
 RUSHANIYA ZOTOVA        |     8
 ELLA DMITRIEVA          |     7
 FANIL KARPOV            |    14
 RITA MELNIKOVA          |    10
 ANZHELIKA SIDOROVA      |    59
 GULFIYA MOROZOVA        |    29
 TAISIYA KRASNOVA        |   113
 PELAGEYA NIKOLAEVA      |    38
 SABINA SCHERBAKOVA      |     4
 KIRILL KOZLOV           |   460
 LARISA KUZMINA          |   506
 SAMIRA ANTONOVA         |    15
 ILONA KAZAKOVA          |     1
 VIKTOR TIKHONOV         |  1301
 RUZALIYA SOROKINA       |     8
 LIDIYA POTAPOVA         |   421
 GEVORG SOROKIN          |     6
 VADIM TIMOFEEV          |   164
 NATALIYA FROLOVA        |   389
 EVGENIY MELNIKOV        |  1367
 RAILYA NOVIKOVA         |     8
 ANTONINA ZOTOVA         |   225
 NIKITA MIRONOV          |   491
 LUIZA TIKHONOVA         |    14
 ANTONINA KULIKOVA       |   326
 RUMIYA FILIPPOVA        |     5
 ANAIT MARTYNOVA         |     6
Time: 40122.499 ms (00:40.122)
```

Зображення 4



Діаграма 2

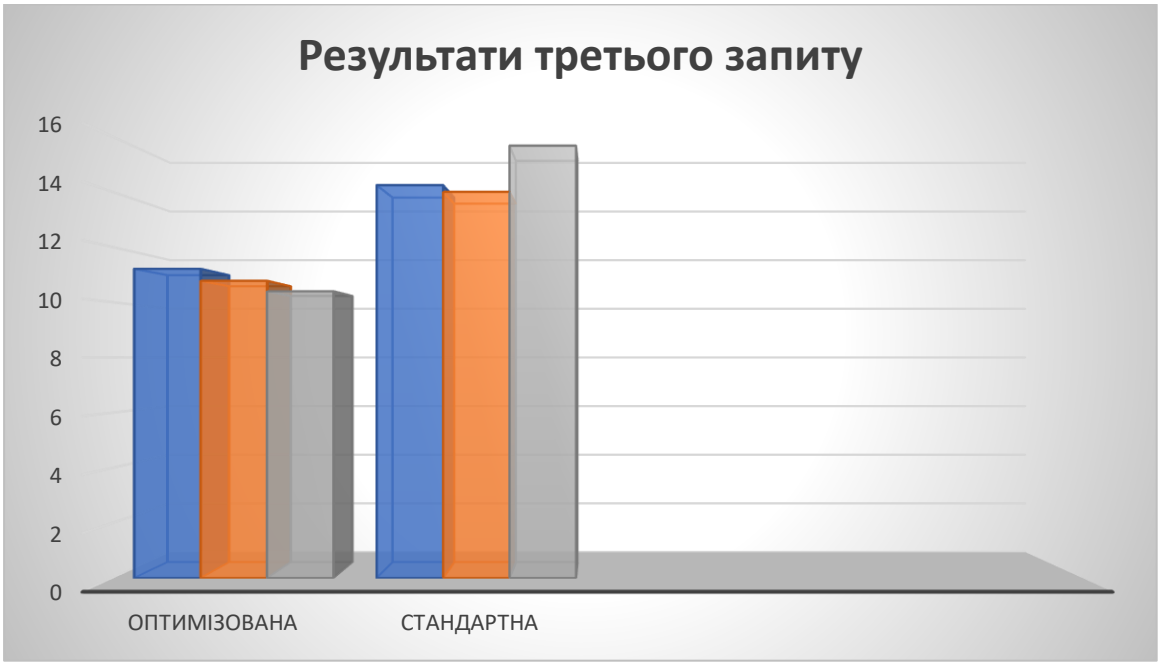
Оптимізована база даних дала результат 33.7с, база із стандартними налаштуваннями показала результат 46.526с. Оптимізація показала приріст швидкості 27.56%. Виконання запиту:

```
demo=# SELECT model, COUNT(ticket_flights.flight_id)
demo-# FROM (aircrafts INNER JOIN flights ON aircrafts.aircraft_code = flights.aircraft_code ) INNER JOIN ticket_flights
 ON flights.flight_id = ticket_flights.flight_id
demo-# GROUP BY model;
        model         |  count
----------------------+---------
 Bombardier CRJ-200   | 1221893
 Sukhoi Superjet-100  | 2931472
 Airbus A319-100      |  430904
 Boeing 767-300       | 1001296
 Cessna 208 Caravan   |  117638
 Boeing 737-300       |  688080
 Airbus A321-200      |  826773
 Boeing 777-300       | 1173796
(8 rows)

Time: 46526.031 ms (00:46.526)
```

Зображення 5

```
demo=# SELECT model, COUNT(ticket_flights.flight_id)
FROM (aircrafts INNER JOIN flights ON aircrafts.aircraft_code = flights.aircraft_code ) INNER JOIN ticket_flights ON fli
ghts.flight_id = ticket_flights.flight_id
GROUP BY model;
        model         |  count
----------------------+---------
 Bombardier CRJ-200   | 1221893
 Sukhoi Superjet-100  | 2931472
 Airbus A319-100      |  430904
 Boeing 767-300       | 1001296
 Cessna 208 Caravan   |  117638
 Boeing 737-300       |  688080
 Airbus A321-200      |  826773
 Boeing 777-300       | 1173796
(8 rows)

Time: 33699.814 ms (00:33.700)
```

Зображення 6

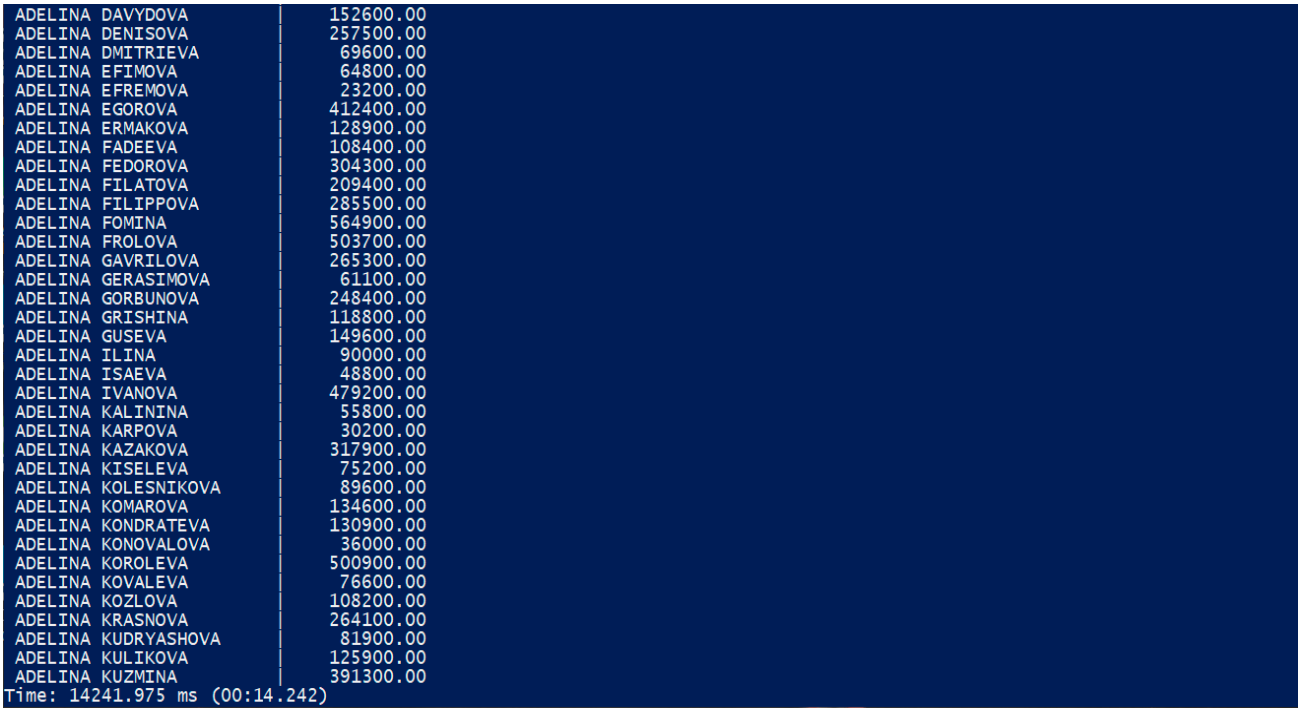


Діаграма 3

Оптимізована база даних дала результат 10.778с, база із стандартними налаштуваннями показала результат 14.242с. Оптимізація показала приріст швидкості 24.32%. Виконання запиту:

```
 ADELINA DAVYDOVA        |    152600.00
 ADELINA DENISOVA        |    257500.00
 ADELINA DMITRIEVA       |     69600.00
 ADELINA EFIMOVA         |     64800.00
 ADELINA EFREMOVA        |     23200.00
 ADELINA EGOROVA         |    412400.00
 ADELINA ERMAKOVA        |    128900.00
 ADELINA FADEEVA         |    108400.00
 ADELINA FEDOROVA        |    304300.00
 ADELINA FILATOVA        |    209400.00
 ADELINA FILIPPOVA       |    285500.00
 ADELINA FOMINA          |    564900.00
 ADELINA FROLOVA         |    503700.00
 ADELINA GAVRILOVA       |    265300.00
 ADELINA GERASIMOVA      |     61100.00
 ADELINA GORBUNOVA       |    248400.00
 ADELINA GRISHINA        |    118800.00
 ADELINA GUSEVA          |    149600.00
 ADELINA ILINA           |     90000.00
 ADELINA ISAEVA          |     48800.00
 ADELINA IVANOVA         |    479200.00
 ADELINA KALININA        |     55800.00
 ADELINA KARPOVA         |     30200.00
 ADELINA KAZAKOVA        |    317900.00
 ADELINA KISELEVA        |     75200.00
 ADELINA KOLESNIKOVA     |     89600.00
 ADELINA KOMAROVA        |    134600.00
 ADELINA KONDRATEVA      |    130900.00
 ADELINA KONOVALOVA      |     36000.00
 ADELINA KOROLEVA        |    500900.00
 ADELINA KOVALEVA        |     76600.00
 ADELINA KOZLOVA         |    108200.00
 ADELINA KRASNOVA        |    264100.00
 ADELINA KUDRYASHOVA     |     81900.00
 ADELINA KULIKOVA        |    125900.00
 ADELINA KUZMINA         |    391300.00
Time: 10777.966 ms (00:10.778)
```

Зображення 7



ADELINA DAVYDOVA	152600.00
ADELINA DENISOVA	257500.00
ADELINA DMITRIEVA	69600.00
ADELINA EFIMOVA	64800.00
ADELINA EFREMOVA	23200.00
ADELINA EGOROVA	412400.00
ADELINA ERMAKOVA	128900.00
ADELINA FADEEVA	108400.00
ADELINA FEDOROVA	304300.00
ADELINA FILATOVA	209400.00
ADELINA FILIPPOVA	285500.00
ADELINA FOMINA	564900.00
ADELINA FROLOVA	503700.00
ADELINA GAVRILOVA	265300.00
ADELINA GERASIMOVA	61100.00
ADELINA GORBUNOVA	248400.00
ADELINA GRISHINA	118800.00
ADELINA GUSEVA	149600.00
ADELINA ILINA	90000.00
ADELINA ISAEVA	48800.00
ADELINA IVANOVA	479200.00
ADELINA KALININA	55800.00
ADELINA KARPOVA	30200.00
ADELINA KAZAKOVA	317900.00
ADELINA KISELEVA	75200.00
ADELINA KOLESNIKOVA	89600.00
ADELINA KOMAROVA	134600.00
ADELINA KONDRATEVA	130900.00
ADELINA KONOVALOVA	36000.00
ADELINA KOROLEVA	500900.00
ADELINA KOVALEVA	76600.00
ADELINA KOZLOVA	108200.00
ADELINA KRASNOVA	264100.00
ADELINA KUDRYASHOVA	81900.00
ADELINA KULIKOVA	125900.00
ADELINA KUZMINA	391300.00

Time: 14241.975 ms (00:14.242)

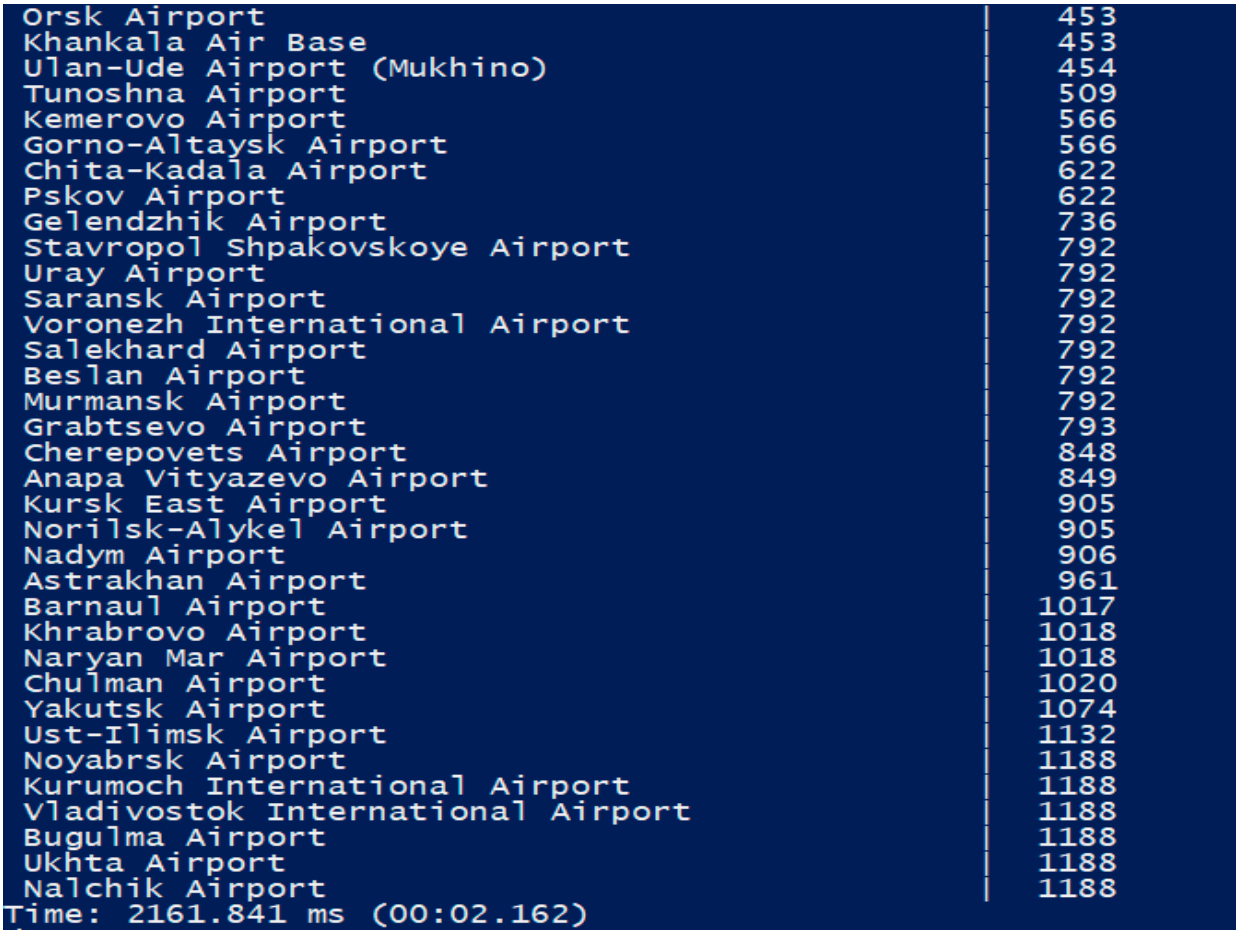
Зображення 8

Діаграма 4

Оптимізована база даних дала результат 1.185с, база із стандартними налаштуваннями показала результат 2.162с. Оптимізація показала приріст швидкості 45.18%. Виконання запиту:

```
Orsk Airport                                |   453
Khankala Air Base                           |   453
Ulan-Ude Airport (Mukhino)                  |   454
Tunoshna Airport                            |   509
Kemerovo Airport                            |   566
Gorno-Altaysk Airport                       |   566
Chita-Kadala Airport                        |   622
Pskov Airport                               |   622
Gelendzhik Airport                          |   736
Stavropol Shpakovskoye Airport              |   792
Uray Airport                                |   792
Saransk Airport                             |   792
Voronezh International Airport              |   792
Salekhard Airport                           |   792
Beslan Airport                              |   792
Murmansk Airport                            |   792
Grabtsevo Airport                           |   793
Cherepovets Airport                         |   848
Anapa Vityazevo Airport                     |   849
Kursk East Airport                          |   905
Norilsk-Alykel Airport                      |   905
Nadym Airport                               |   906
Astrakhan Airport                           |   961
Barnaul Airport                             |  1017
Khrabrovo Airport                           |  1018
Naryan Mar Airport                          |  1018
Chulman Airport                             |  1020
Yakutsk Airport                             |  1074
Ust-Ilimsk Airport                          |  1132
Noyabrsk Airport                            |  1188
Kurumoch International Airport              |  1188
Vladivostok International Airport           |  1188
Bugulma Airport                             |  1188
Ukhta Airport                               |  1188
Nalchik Airport                             |  1188
Time: 1184.608 ms (00:01.185)
```

Зображення 9



Orsk Airport	453
Khankala Air Base	453
Ulan-Ude Airport (Mukhino)	454
Tunoshna Airport	509
Kemerovo Airport	566
Gorno-Altaysk Airport	566
Chita-Kadala Airport	622
Pskov Airport	622
Gelendzhik Airport	736
Stavropol Shpakovskoye Airport	792
Uray Airport	792
Saransk Airport	792
Voronezh International Airport	792
Salekhard Airport	792
Beslan Airport	792
Murmansk Airport	792
Grabtsevo Airport	793
Cherepovets Airport	848
Anapa Vityazevo Airport	849
Kursk East Airport	905
Norilsk-Alykel Airport	905
Nadym Airport	906
Astrakhan Airport	961
Barnaul Airport	1017
Khrabrovo Airport	1018
Naryan Mar Airport	1018
Chulman Airport	1020
Yakutsk Airport	1074
Ust-Ilimsk Airport	1132
Noyabrsk Airport	1188
Kurumoch International Airport	1188
Vladivostok International Airport	1188
Bugulma Airport	1188
Ukhta Airport	1188
Nalchik Airport	1188

Time: 2161.841 ms (00:02.162)

Зображення 10

Діаграма 5

Оптимізована база даних дала результат 9.269с, база із стандартними налаштуваннями показала результат 16.032с. Оптимізація показала приріст швидкості 42.18%. Виконання запиту:

```
| 0005434863576 |      65557 | Business         |  49700.00
| 0005432801416 |       9604 | Business         | 150400.00
| 0005434880642 |      94798 | Business         |  49700.00
| 0005432801652 |       9613 | Business         | 150400.00
| 0005435858548 |     179085 | Business         |  90500.00
| 0005432802115 |       9577 | Business         | 150400.00
| 0005435471897 |     117290 | Business         | 199300.00
| 0005432802620 |       9595 | Business         | 150400.00
| 0005435818157 |     116920 | Business         | 199800.00
| 0005433808342 |     199975 | Business         | 117400.00
| 0005434865104 |      94734 | Business         |  49700.00
| 0005433807549 |      93852 | Business         | 117400.00
| 0005434857581 |      65663 | Business         |  49700.00
| 0005433807783 |      93851 | Business         | 117400.00
| 0005435834448 |      38766 | Business         | 199300.00
| 0005432801017 |     129423 | Business         | 150400.00
| 0005435469437 |     117207 | Business         | 199300.00
| 0005432801025 |     129403 | Business         | 150400.00
| 0005432687669 |       7540 | Business         | 115000.00
| 0005432801779 |     129428 | Business         | 150400.00
| 0005434855003 |      94987 | Business         |  49700.00
| 0005433812669 |     199973 | Business         | 117400.00
| 0005433551212 |      76832 | Business         | 105900.00
| 0005432802557 |     129418 | Business         | 150400.00
| 0005435472226 |      39078 | Business         | 199300.00
| 0005433812709 |      93879 | Business         | 117400.00
| 0005434879921 |      95068 | Business         |  49700.00
| 0005433812898 |      93880 | Business         | 117400.00
| 0005435821901 |     116930 | Business         | 199800.00
| 0005433813844 |      36332 | Business         |  99800.00
| 0005433551130 |     164217 | Business         | 105900.00
| 0005433813939 |      36355 | Business         |  99800.00
| 0005435830570 |      38918 | Business         | 199300.00
| 0005433813651 |      89930 | Business         |  99800.00
| 0005433555864 |     163947 | Business         | 105900.00
| 0005433814019 |      89727 | Business         |  99800.00
| 0005432947223 |     163988 | Business         | 105900.00
Time: 16032.022 ms (00:16.032)
```

Зображення 11

0005434863576	65557	Business	49700.00
0005432801416	9604	Business	150400.00
0005434880642	94798	Business	49700.00
0005432801652	9613	Business	150400.00
0005435858548	179085	Business	90500.00
0005432802115	9577	Business	150400.00
0005435471897	117290	Business	199300.00
0005432802620	9595	Business	150400.00
0005435818157	116920	Business	199800.00
0005433808342	199975	Business	117400.00
0005434865104	94734	Business	49700.00
0005433807549	93852	Business	117400.00
0005434857581	65663	Business	49700.00
0005433807783	93851	Business	117400.00
0005435834448	38766	Business	199300.00
0005432801017	129423	Business	150400.00
0005435469437	117207	Business	199300.00
0005432801025	129403	Business	150400.00
0005432687669	7540	Business	115000.00
0005432801779	129428	Business	150400.00
0005434855003	94987	Business	49700.00
0005433812669	199973	Business	117400.00
0005433551212	76832	Business	105900.00
0005432802557	129418	Business	150400.00
0005435472226	39078	Business	199300.00
0005433812709	93879	Business	117400.00
0005434879921	95068	Business	49700.00
0005433812898	93880	Business	117400.00
0005435821901	116930	Business	199800.00
0005433813844	36332	Business	99800.00
0005433551130	164217	Business	105900.00
0005433813939	36355	Business	99800.00
0005435830570	38918	Business	199300.00
0005433813651	89930	Business	99800.00
0005433555864	163947	Business	105900.00
0005433814019	89727	Business	99800.00
0005432947223	163988	Business	105900.00

Time: 9269.145 ms (00:09.269)

Зображення 12

Діаграма 6

Результати показують, що, в залежності від складності запиту, кожна база даних дає свій результат і, ці числа відрізняються доволі сильно.

Перестановка таблиць при об’єднанні не впливає на швидкість виконання, скоріше за все це пов’язано з тим, що СКБД має власний оптимізатор запитів, який і перевіряє такі кейси.

Оскільки база наповнена великою кількістю інформації, параметри СКБД, які були підвищені, надали більше місця для обробки даних на швидкій пам’яті. Стандартно налаштовані значення цих параметрів, занадто малі, щоб швидку опрацьовувати такі блоки інформації.

Отже, оптимізація пройшла успішно, адже у всіх випадках було отримано доволі непоганий приріст швидкодії. Особливо це помітно у запитах з найбільшими таблицями по кількості рядків.

Висновки

Після проведеної роботи стає зрозумілим, наскільки важливо максимально покращувати роботу будь-чого, не лише СКБД. Адже для отримання найкращого результату завжди потрібні лише найвдаліші ідеї та найшвидші алгоритми.

Вивчивши документацію та деякі відкриті джерела стосовно цієї теми, можна побачити, що не так і легко оптимізувати швидкодію системи. В даній роботі були розглянуті найбільш поширені методи, але якщо заглибитись в цю тему далі, її можна вивчати роками.

Проблема швидкості та пам’яті досі не вирішена, хтось віддає перевагу першому, бо має необмежену кількість ресурсів, хтось же, навпаки, намагається зекономити вільне місце, але жертвує часом.

З розглянутих методів не всі були використані у практичній частині, а лише ті, які показали найкращий результат. Адже для кожної системи, схеми і навіть запитів можна підібрати свої параметри, які будуть кращими у конкретних випадках.

Результати показали цілком солідний приріст швидкодії, а, враховуючи, що даних було на декілька гігабайтів, можна тільки уявити, наскільки важлива подібна оптимізація у системах із десятками, а то і сотнями гігабайтів інформації.

Проведена робота допомогла зрозуміти цінність оптимізації та набратись навичок її впровадження. Вивчені методи оптимізації можна і потрібно застосовувати в реальних проектах, а не лише в навчальних цілях, адже це масштабує будь-яку систему та робить її комфортнішою в користуванні.

Список використаних джерел

1. Документація PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/docs/
2. Параметри налаштування PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.postgresql.org/wiki/Tuning_Your_PostgreSQL_Server
3. Параметри налаштування PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://ruhighload.com/%D0%A2%D1%8E%D0%BD%D0%B8%D0%BD%D0%B3+%D0%B1%D0%B0%D0%B7%D1%8B+postgres
4. Демонстраційна база даних про авіаперельоти [Електронний ресурс] – Режим доступу до ресурсу: https://postgrespro.com/education/demodb
5. Оптимізація СКБД [Електронний ресурс] – Режим доступу до ресурсу: https://www.percona.com/blog/2018/08/31/tuning-postgresql-database-parameters-to-optimize-performance/
6. Аналіз і налаштування роботи PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://stackify.com/postgresql-performance-tutorial/
7. Оптимізація роботи з інформацією [Електронний ресурс] – Режим доступу до ресурсу: https://cyberleninka.ru/article/n/optimizatsiya-raboty-s-informatsiey-v-bazah-dannyh/viewer
8. Нормалізація [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/254773/
9. Основи баз даних [Електронний ресурс] – Режим доступу до ресурсу: https://mipt.ru/dnbic/content/db.pdf
10. Нормалізація [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Database_normalization
11. Алгоритми роботи індексів у базах даних[Електронний ресурс] – Режим доступу до ресурсу: http://citforum.ck.ua/database/articles/b-tree_indexes/

12. Оптимізація роботи дискових операцій [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/127703/
13. Документація Docker [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/
14. Параметри ОС Linux [Електронний ресурс] – Режим доступу до ресурсу: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/index