Керівник наукової роботи:
Захоженко Павло Олегович
Виконала студентка:
Живіцька Єлизавета Олексіївна

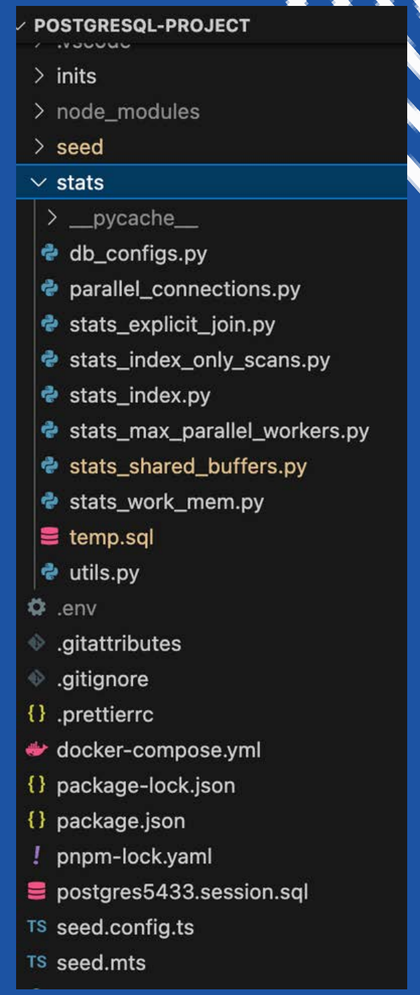
Оптимізація роботи СУБД PostgreSQL: від рівня сервера до рівня запитів

Пост ановка задачі

- Дослідити та проаналізувати параметри конфігурації PostgreSQL сервера
- Визначити оптимальні значення параметрів конфігурації для оптимізації PostgreSQL сервера
- Дослідити та проаналізувати способи оптимізації запитів до бази даних PostgreSQL

Технологічний ст ек

- Docker (postgres зображення)
- Snaplet (бібліотека для автоматичної генерації seed даних)
- typescript (для роботи зі Snaplet)
- npm (менеджмент скриптів та інсталювання Snaplet)
- python (тестування серверу та малювання графіків)



План робот и

1. Оптимізація серверу

- work_mem
- shared_buffers
- max_parallel_workers_per_gather

1. Оптимізація запитів

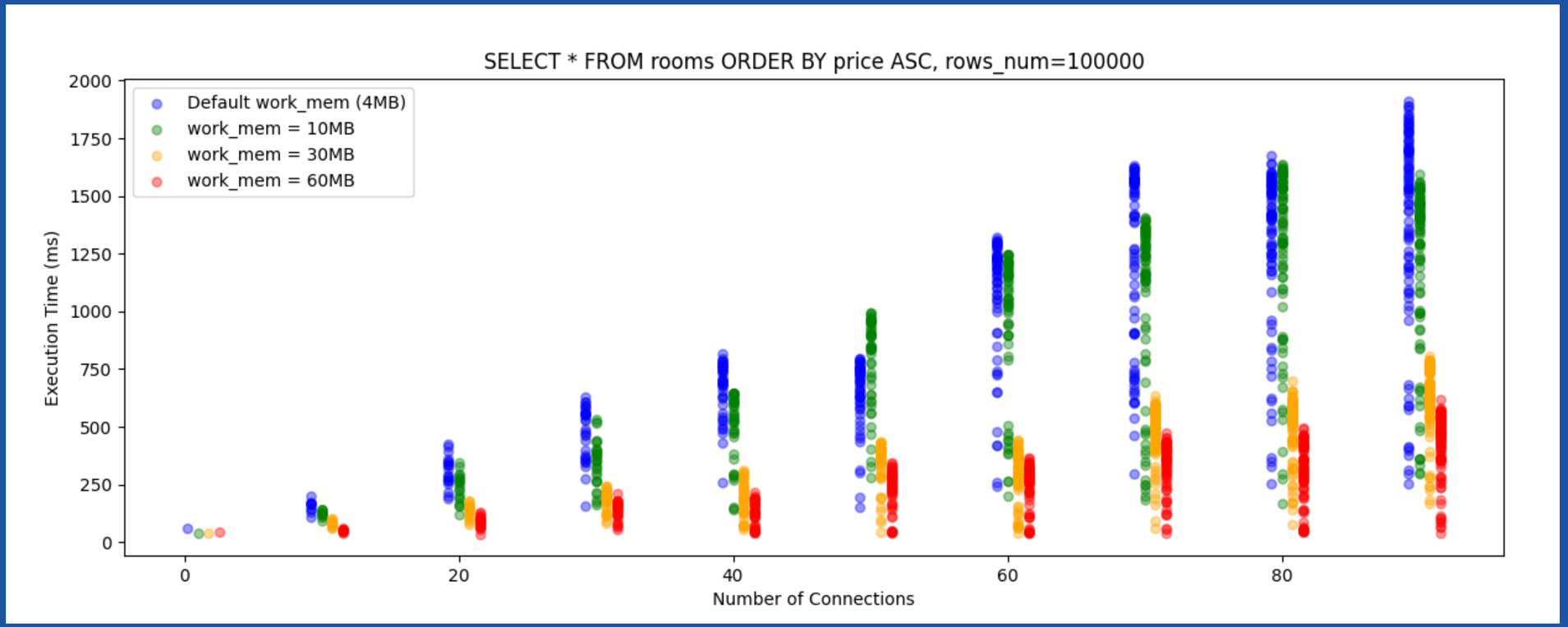
- явні та неявні JOIN
- index
- index only scans

Опт имізація сервера

work_mem

- базовий максимальний обсяг пам'яті, який використовується операцією сортуванням в запиті
- значення за замовчуванням - 4МБ
- важливо враховувати параметр max_connections для оцінки можливого максимального обсягу зайнятої пам’яті
- рекомендується становлювати значення:

  (25 % * RAM) / max_connections



- Значення work_mem за замовчуванням у 4 МБ є недостатнім для ефективної обробки великих наборів даних та високого рівня паралелізму.
- Зі збільшенням під’єднань розмір використовуваної пам’яті збільшується
- Збільшення work_mem до 10МБ, 30МБ і особливо 60МБ призводить до зменшення часу виконання, що свідчить про кращу продуктивність.

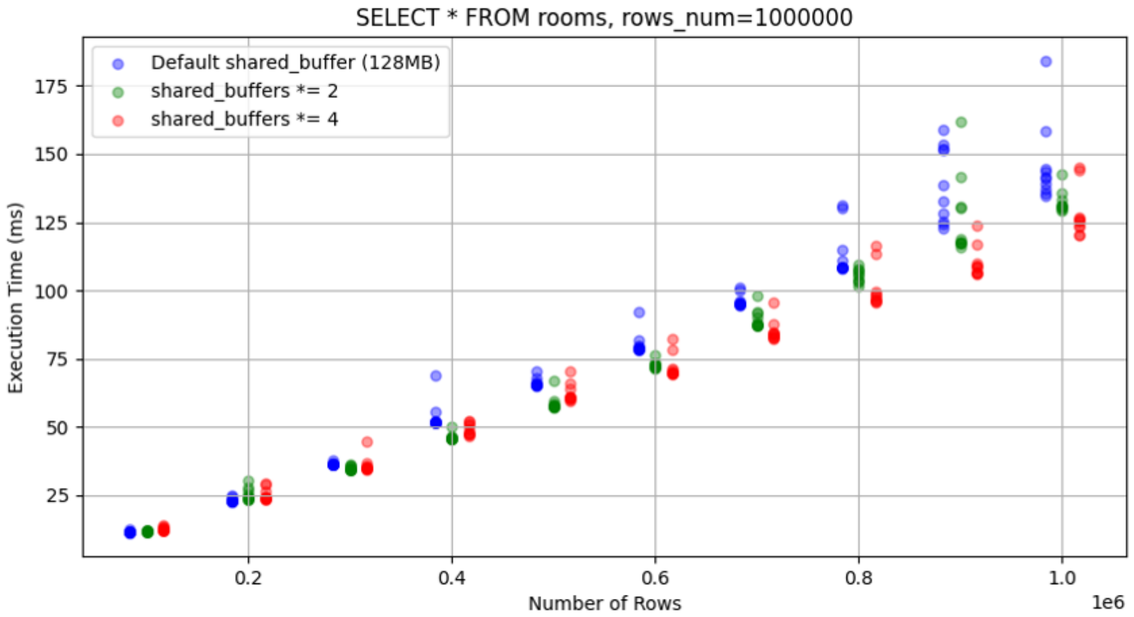
Оптимізація сервера

shared_buffers

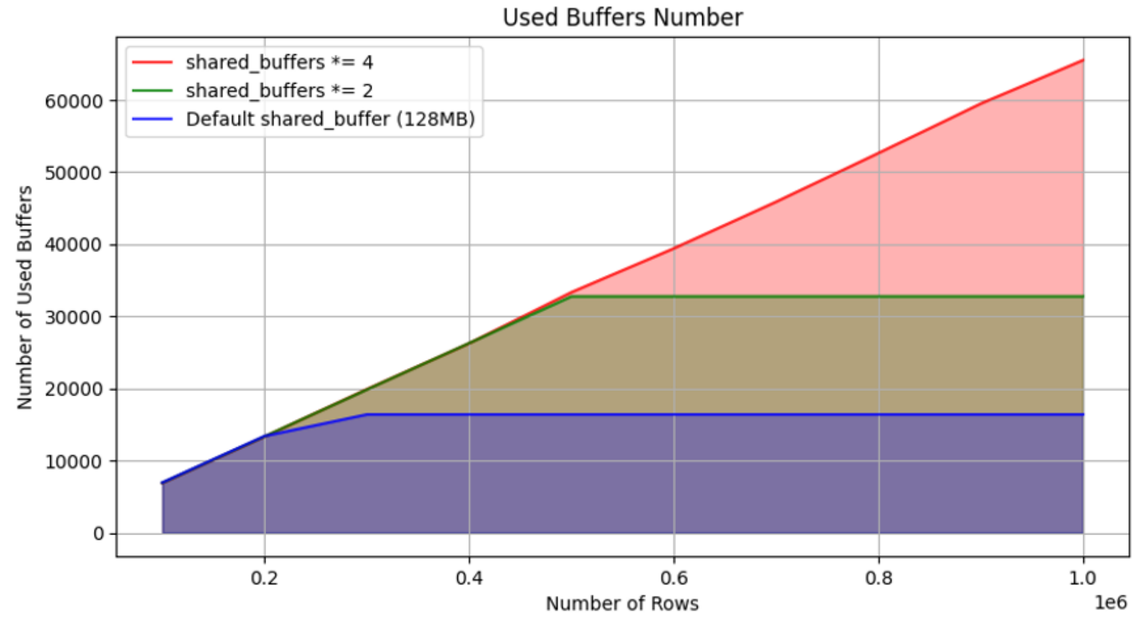
- визначає, скільки пам'яті виділяється PostgreSQL для кешування даних
- значення за замовчуванням - 128 МБ
- зберігає лічильники використання найбільш використовуваних даних
- взаємодіє з кешем ОС
- рекомендується становлювати значення:

25 % * RAM

Час виконання запиту



Кількість використаних буферів



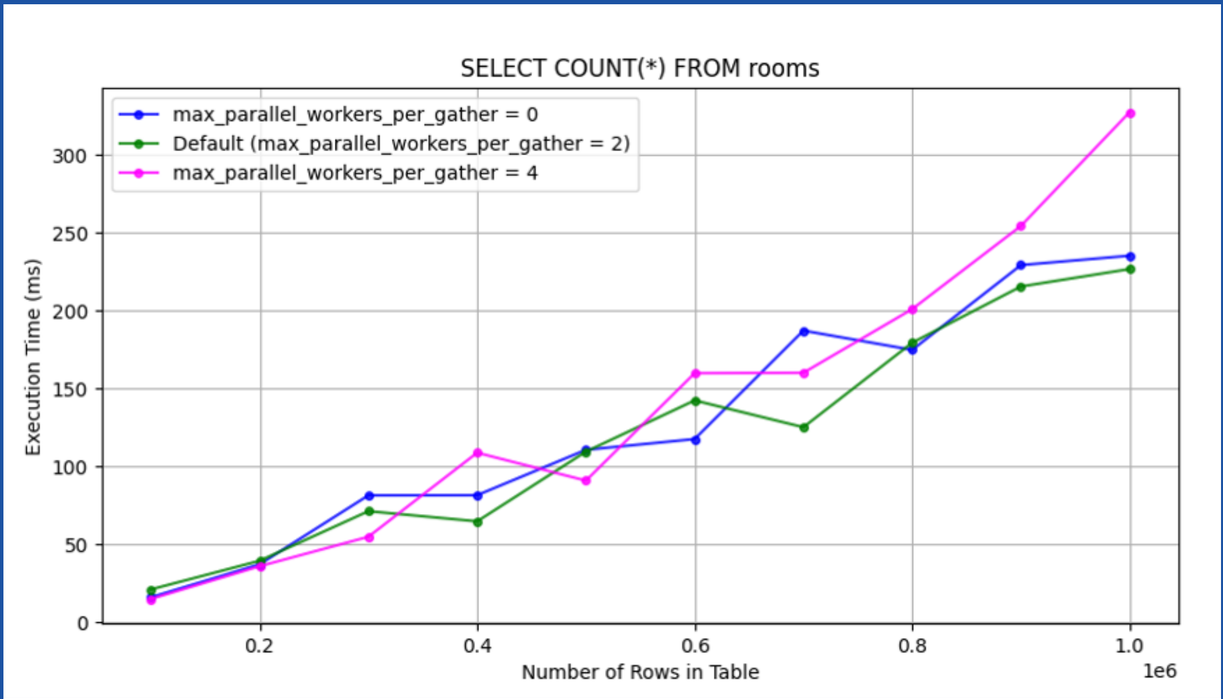
- Збільшення shared_buffers від 128 Мб до 256 МБ і 512 МБ скорочується час виконання для SELECT * запита, оскільки доступно більше буферів, що вказує на те, що підвищена ємність буфера дозволяє більш ефективно кешувати та зменшувати виведення з диска.
- Зі збільшенням кількості рядків, що повертає запит, збільшується кількість використаних буферів
- Налаштування за замовчуванням (синій) досягає своєї межі раніше порівняно з більшими налаштуваннями буфера (зелений і червоний).

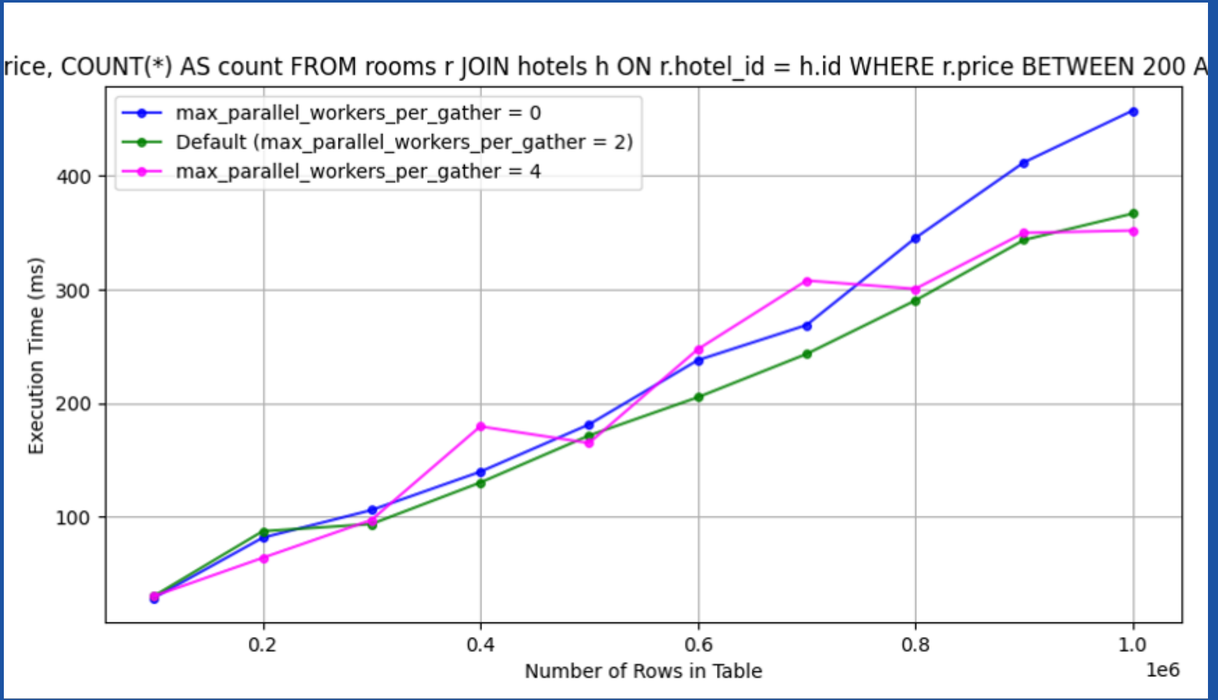
Опт имізація сервера

max_parallel_workers_per_gather

- встановлює максимальну кількість паралельних робочих процесів, які може використовувати один запит з використанням вузла `Gather` або `Gather Merge`
- значення за замовчуванням - 2
- збільшення параметру збільшує споживання ресурсів, тому його слід налаштовувати з обережністю

```
SELECT h.id, h.name, r.price, COUNT(*) AS count
FROM rooms r JOIN hotels h ON r.hotel_id = h.id
WHERE r.price BETWEEN 200 AND 500 GROUP BY h.id, r.price
```





- Налаштування за замовчуванням max_parallel_workers_per_gather = 2 показує найкращі показники для всіх запитів
- Збільшення кількості паралельних робітників до 4 вводить додаткові накладні витрати, що може призвести до більш повільних виконанньь порівняно з налаштуваннями за замовчуванням
- Використання паралельних працівників, як правило, покращує продуктивність, особливо для більших наборів даних та більш складних запитів

Оптимізація запитів

Явні та неявні JOIN

Неявний JOIN

```
SELECT *
FROM rooms r, bookings b, users
u, profiles p
WHERE u.id = b.user_id
AND r.id = b.room_id
AND p.user_id = u.id;
```

Явний JOIN

```
SELECT *
FROM (users u LEFT JOIN
 (bookings b LEFT JOIN rooms r on
b.room_id = r.id) ON b.user_id = u.id)
LEFT JOIN profiles p ON p.user_id =
u.id;
```

Оптимізація запитів

Явні та неявні JOIN

Неявний JOIN

І виконання:

profiles->bookings->rooms->users

Execution Time: 14.029 ms

Planning Time: 3.624 ms

ІІ виконання:

rooms->bookings->users->profiles

Execution Time: 17.035 ms

Planning Time: 4.157 ms

Явний JOIN

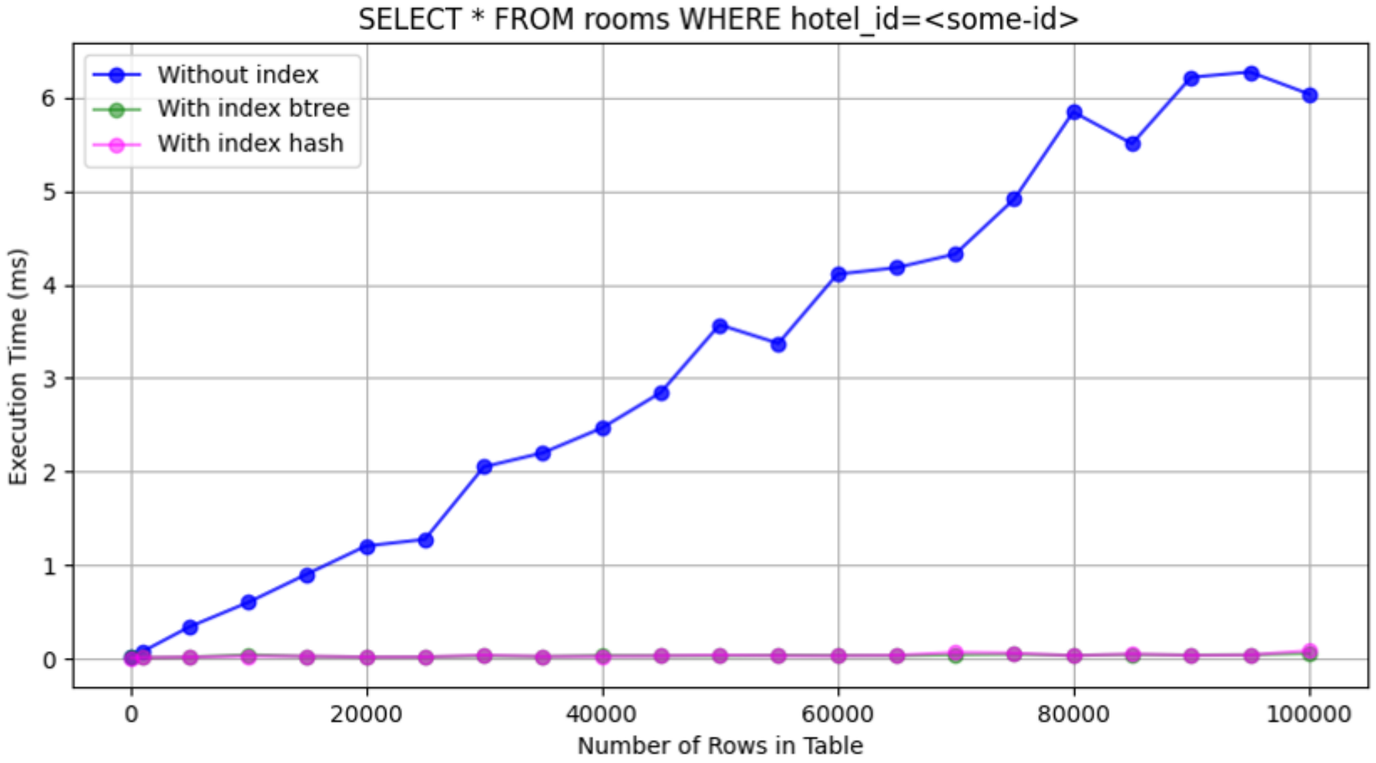
rooms->bookings->users->profiles

Execution Time: 14.954 ms

Planning Time: 0.714 ms

Опт имізація запит ів

Індекси

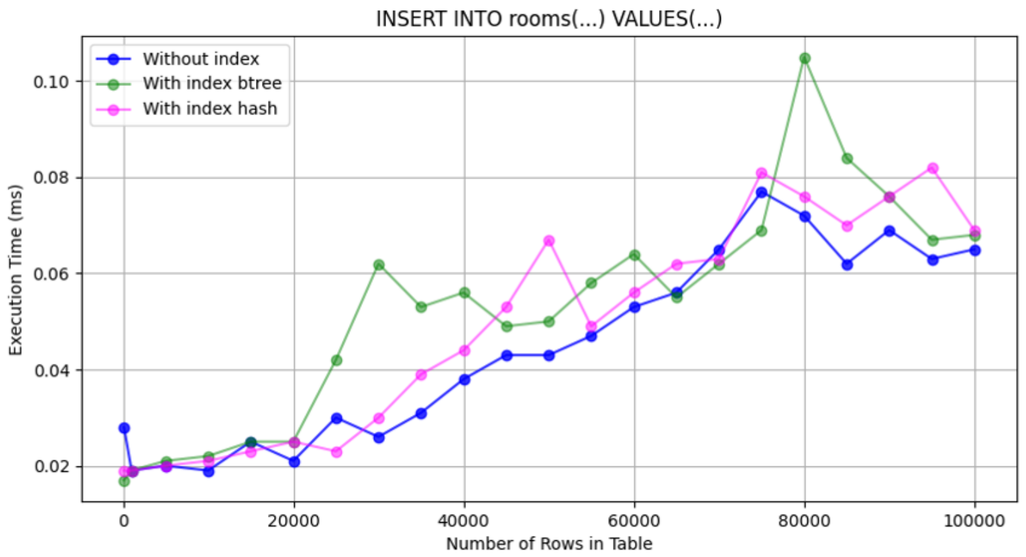


- Без індексів час виконання значно збільшується зі збільшенням кількості рядків, демонструючи круту тенденцію до зростання
- З індексами час виконання залишається незмінно низьким і майже рівним у всіх рядках, що демонструє їхню ефективність

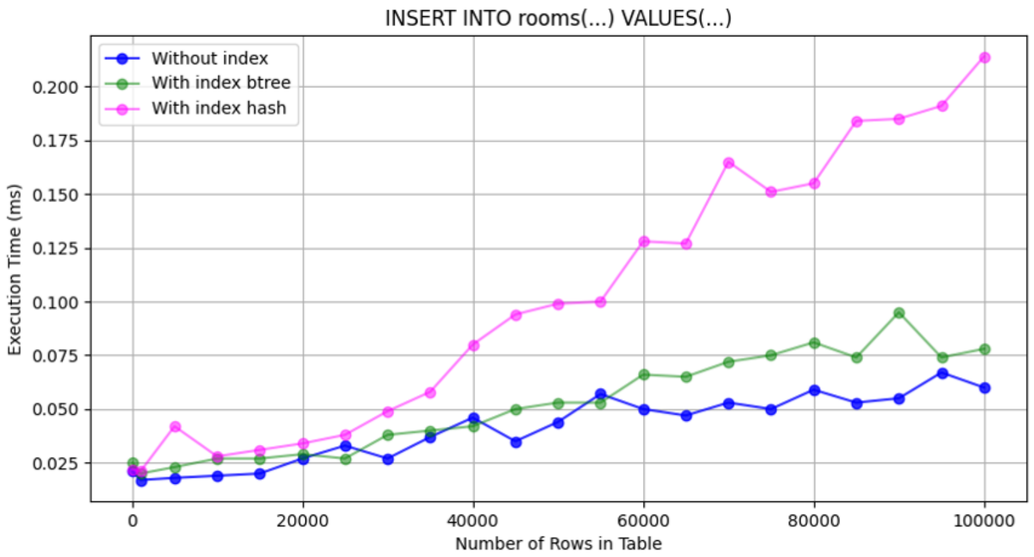
Оптимізація запитів

Індекси та операції запису

з 1 додатковим індексом



з 4 додатковими індексами



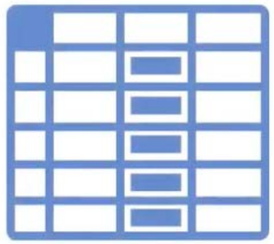
- Велика кількість індексів значно збільшує час операції вставки
- Індекси типу Hash займають більше часу під час операції вставки, ніж B-Tree
- Переваги покращеної продуктивності запитів з індексами повинні бути збалансовані з підвищеною вартістю обслуговування цих індексів під час операцій вставки. Цей компроміс має вирішальне значення для оптимізації загальної продуктивності бази даних.

Оптимізація запитів

Index only scans

Сканування за індексом:

SELECT * FROM rooms WHERE id = ...



Index Lookup

Heap Lookup

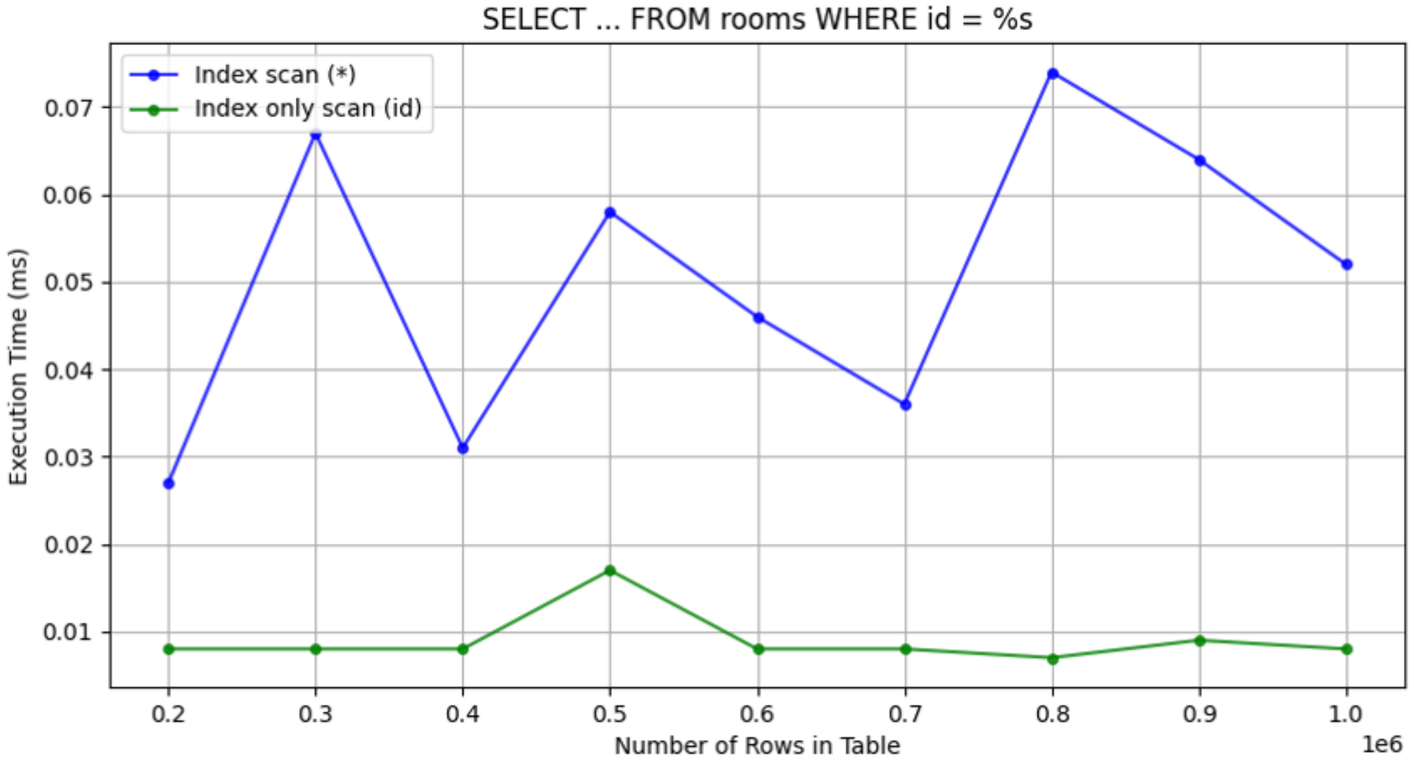
Сканування лише за індексом:

SELECT id FROM rooms WHERE id = ...

Index Lookup

Опт имізація запит ів

Index only scans



- Сканування лише за індексом дає значні переваги в продуктивності, коли запит можна задовольнити повністю з індексу, як видно зі стабільного та низького часу виконання.
- Індексне сканування, яке вимагає отримання всіх стовпців, несе додаткові витрати через доступ до купи, що призводить до більшого і більш варіативного часу виконання.

Висновки

Збільшення `shared_buffers` та `work_mem` значно підвищує продуктивність за рахунок зменшення дискового виводу та покращення операцій сортування відповідно. Збільшення `max_parallel_workers_per_gather` для паралельної обробки може покращити виконання дуже складних запитів, однак витрати на підтримку високої паралельності можуть знижувати продуктивність. Значення за замовчуванням часто ефективно балансує продуктивність і використання ресурсів.

Використання індексів значно зменшує час виконання запитів. Однак введення декількох індексів може сповільнити операції запису, тому важливо збалансувати продуктивність читання і запису. Використання сканування тільки за індексом і явних об'єднань JOIN може ще більше підвищити ефективність запитів.

Загалом, розумне поєднання налаштування сервера та оптимізації запитів є ключем до досягнення оптимальної продуктивності PostgreSQL, гарантуючи, що система зможе ефективно впоратися зі зростаючими вимогами.

Дякую за увагу!