**Тестовое задание на позицию стажера Performance Engineer в Postgres Pro**

Был установлен Postgres в докер, созданы таблицы с помощью команд в задании.

**Задание 1**

Запрос без оптимизаций:

select name from t1 where id = 50000;

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Запрос выполняется медленно, потому что база данных вынуждена проводить full table scan, то есть последовательно читать все записи таблицы t1 и сверять их на соответствие условию в where. Самый очевидный способ оптимизировать это – использовать индексы.

Для этого запроса подойдет 2 типа индексов: B-Tree (по умолчанию) или Hash. Так как Hash имеет более узкую область применения (применяется только если индексированные столбцы участвуют в сравнении с использованием оператора «=») и подходит под наш запрос, используем его:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

*Создали Hash index*

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Контент, сгенерированный ИИ, может содержать ошибки.

Как видим, теперь запрос укладывается в диапазон ~0,37 – 1,5 ms. Задание выполнено.

**Задание 2**

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

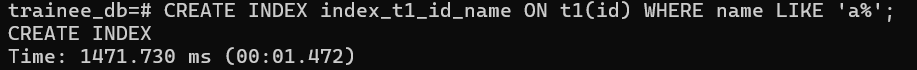
Рассмотрим план запроса, предлагающийся планировщиком без оптимизаций:

1. Seq Scan on t1 – выполняется полное сканирование таблицы t1 с 10M строк с фильтрацией по name LIKE ‘a%’. Возвращает (предположительно) чуть более 600т.
2. Hash – построение хэш-таблицы для быстрого поиска по id из 600т. строк.
3. Seq Scan on t2 – полное сканирование t2 с 5M строк. Для каждой строки в t2 ищет t1.id в хэш-таблице
4. Hash Left Join – Соединение таблиц, перебирает все записи в t2.
5. Aggregate – агрегатная функция max(t2.day), также перебирает все записи в t2 и возвращает 1 строку.

Как мы видим, в 3 и 4 пункте обрабатываются сразу все строки таблицы t2 (что логично, ведь Left join требует результирующей таблицы из всех строк таблицы t2).

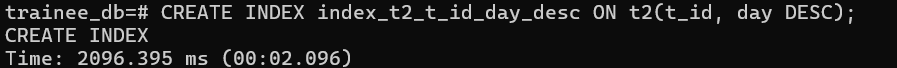
Проблема в том, что даже если бы использовался индекс для t2.t\_id, планировщик все равно отдавал бы предпочтение seq scan, потому что параметр random\_page\_cost = 4, хотя в данном случае используется SSD диск и разница между последовательным и произвольным чтением не такая большая.

Для начала составим специализированный индекс по id для t1:



Этот индекс включает только строки где name LIKE ‘a%’

Создадим индекс для t2:



Он немного оптимизирован для поиска MAX(day), потому что DESC сортировка может чуть-чуть ускорять нахождение максимального значения, хотя и находясь на 2 месте.

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Как видим, текущий план запроса планировщика не использует 2-й индекс, выполняется полный перебор всех строк из t2.

Мы можем попытаться исправить это и задать планировщику более актуальные для нас настройки разницы между последовательным и произвольным чтением, командой:

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Контент, сгенерированный ИИ, может содержать ошибки.

Теперь план будет выглядеть так:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Теперь запрос выполняется менее чем за 1 секунду:

Изображение выглядит как текст, снимок экрана, черный, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

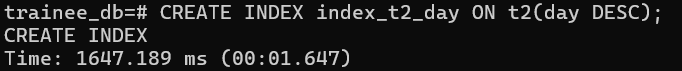
Однако можно заметить, что перебор даже по индексам для таблицы t1 занимает ~70 мс, что уже в семь раз больше значения, до которого надо ускорить. Значит оптимизации мы здесь можем добиться только изменением самого запроса.

*(Я не знаю, позволительно ли менять структуру запроса, который надо оптимизировать, но предположу что да, иначе выполнение задания не кажется мне реальным. Также запрос будет изменен в 3 задании).*

Здесь left join в запросе излишен, ведь поле day в таблице t2, нам нет смысла добавлять поля из t1. Тогда изменим запрос на следующий:

Select max(t2.day) from t2

Для него создадим индекс:



Конечный план запроса:

Изображение выглядит как текст, снимок экрана, черный, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Как мы видим, такой запрос, конечно же, вкладывается в рамки по времени (< 10ms). Задание выполнено.

**Задание 3**

select day from t2 where t\_id not in ( select t1.id from t1 );

Напрямую такой запрос не выполняется даже за 2 минуты ожидания. Предположу что это происходит из-за того, что планировщик сначала выполняет подзапрос, получая таблицу из 10M неотсортированных id, и потом каждое t\_id из 2-й таблицы (которых 5М) проверяет на наличие в t1, что очень ресурсоемко.

Поэтому перепишем наш запрос на аналогичный, но более производительный:

select day from t2 left join t1 on t2.t\_id = t1.id where t1.id is null;

Он уже без индексов укладывается в 10 секунд по времени выполнения. Но без них строится хэш таблица, а для больших данных лучше использовать merge join или merge anti join.

Создадим 2 индекса: по t1.id и по t2.t\_id INCLUDE day. Тогда план запроса будет следующим:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Планировщик пробегается отдельно по t1 и по t2 одновременно, причем, так как индексы уже отсортированы, сразу выполняет merge anti join.

Время выполнения < 2 сек. Задание выполнено.

**Задание 4**

Запрос:

select day from t2 where t\_id in ( select t1.id from t1 where t2.t\_id = t1.id) and day > to\_char(date\_trunc('day',now()- '1 months'::interval),'yyyymmdd');

На самом деле даже без индексов и изменения структуры запроса он уже укладывается в 10 секунд выполнения:

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Но все же добавим пару b-tree индексов для поля id таблицы t1 и составной для полей t\_id и day таблицы t2:

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Контент, сгенерированный ИИ, может содержать ошибки.

Вместо полного сканирования t1 у нас теперь index only scan и хоть по времени он незначительно уступает, хэш таблица для t1 не строится, что серьезно ускоряет запрос. Также индекс для t2 выполняется быстрее из-за фильтрации (у seq scan она значительно медленнее). В итоге запрос стал быстрее в 3 раза.

Время выполнения < 2 сек. Задание выполнено.

**Задание 5**

Очень постараюсь разобраться с 5 заданием и, надеюсь, отправлю вам его😊