Postgres PROFESSIONAL

Оптимизация запросов

Темы

Профилирование

Способы влияния на планы запросов

Примеры

Инструмент

Профилирование

- выделение подзадач
- продолжительность
- количество выполнений

Что оптимизировать?

- чем больше доля подзадачи в общем времени выполнения, тем больше потенциальный выигрыш
- необходимо учитывать затраты на оптимизацию
- полезно взглянуть на задачу шире

Нюансы

Что профилировать

отдельную задачу, вызывающую нарекания

чем точнее, тем лучше: широкий охват «размывает» проблему

Единицы изменения

время —
осмысленность для пользователя

прочитанные или записанные страницы —
стабильность по отношению ко внешним факторам

Профиль приложения

Подзадачи

- клиентская часть
- сервер приложений
- сервер баз данных ← *проблема часто, но не всегда, именно здесь*
- сеть

Как профилировать

технически трудно, нужны разнообразные средства мониторинга
подтвердить предположение обычно не сложно

Профиль СУБД

Журнал сообщений сервера

включается конфигурационными параметрами:

log_min_duration_statements = 0 — время и текст всех запросов

log_line_prefix — идентифицирующая информация

сложно включить для отдельного сеанса

большой объем; при увеличении порога времени теряем информацию

не отслеживаются вложенные запросы
(можно использовать расширение auto_explain)

анализ внешними средствами, такими, как pgBadger

Профиль СУБД

Расширение pg_stat_statements

- подробная информация о запросах в представлении (в том числе о вложенных)
- ограниченный размер хранилища
- запросы считаются одинаковыми «с точностью до констант», даже если имеют разные планы выполнения
- идентификация только по имени пользователя и базе данных

Профиль запроса

Explain analyze

- подзадачи — узлы плана
- продолжительность — actual time
- количество выполнений — loops

Особенности

- помимо наиболее ресурсоемких узлов, кандидаты на оптимизацию — узлы с большой ошибкой прогноза кардинальности
- любое вмешательство может привести к полной перестройке плана
- иногда приходится довольствоваться простым explain

Пути оптимизации

Цель — получить адекватный план

Исправление неэффективностей

- каким-то образом найти и исправить узкое место
- бывает сложно догадаться, в чем проблема
- часто приводит к борьбе с планировщиком

Правильный расчет кардинальности

- добиться правильного расчета кардинальности в каждом узле и положиться на планировщик
- если план все еще неадекватный, настраивать глобальные параметры

Классификация способов

Изменение исходного кода

можно ли изменить текст запроса или приложения

Изменение схемы данных

можно ли вносить изменения в способ хранения данных

Гибкость

ограничивает ли изменение свободу планировщика

Локальность влияния

действует ли изменение локально на один запрос
или глобально на всю систему

Универсальность применения

применяется ли повсеместно или помогает в редких случаях

Для начала

Статистика

без актуальной статистики планировщик бесполезен

статистика не должна быть точной, но должна быть похожей на правду

настройка autovacuum, *default_statistics_target*

Ограничения целостности

помимо всего прочего, могут учитываться и при планировании

Кардинальность

Функциональные индексы

- отдельная статистика

Левое соединение

- показать, что соединение не уменьшает число строк

Устранение коррелированных предикатов

Временные таблицы

- для части запроса, чтобы использовать статистику

Порядок соединения

Перенос условия фильтрации на другую таблицу

- планировщик не всегда может перенести условия от одной таблицы к другой

With-запросы (CTE)

- всегда материализуются

Явные соединения и *join_collapse_limit* = 1

- планировщик не будет менять порядок соединений

Подзапросы и *from_collapse_limit* = 1

- планировщик не будет раскрывать подзапросы

Коррелированные подзапросы

- планировщик может не справиться с раскрытием

Доступы и соединения

Конфигурационные параметры (локально)

методы доступа,	*enable_(seqscan|indexscan|...)*
способы соединения,	*enable_(nestloop|hashjoin|mergejoin)*
операции	*enable_(hashagg|sort|material)*

— грубо, но полезно для экспериментов

размер рабочей памяти	*work_mem*

Подсказки оптимизатору

отсутствуют, но есть конфигурационные параметры и расширения

Денормализация

Индексы

- обновляются автоматически

Поля таблиц

- надо обновлять (триггерами или другим способом)

Материализованные представления

- надо обновлять (по расписанию или другим способом)

Кэширование результатов в приложении

- надо обновлять (при необходимости)

Разное

Секционирование

- для очень больших таблиц, требующих полного сканирования

Изменение запроса

- трансформация охватывает не все эквивалентные формы запроса
  например, операции union, except, intersect не трансформируются

Замена процедурного кода декларативным

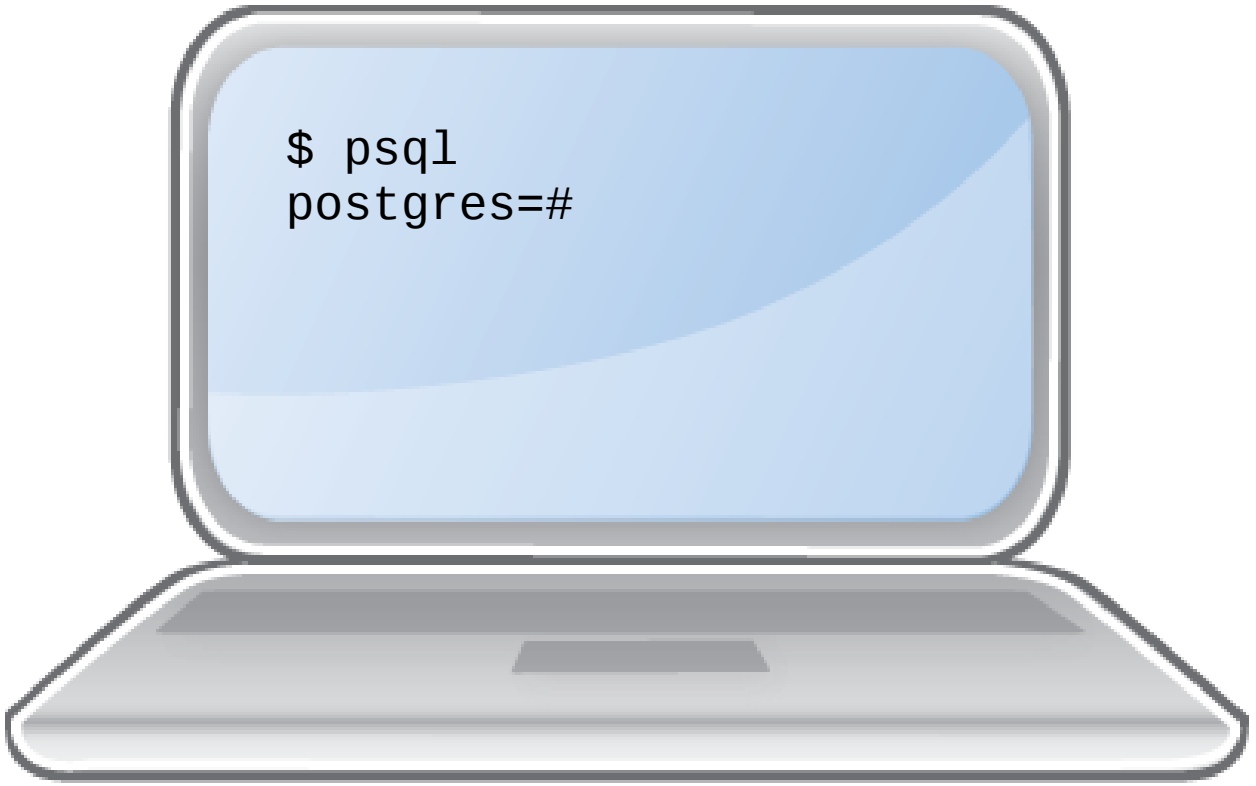
- избавление от циклов и небольших, но многочисленных команд SQL

Глобальные настройки

Глобальные конфигурационные параметры

настройка стоимости отдельных операций	**_cost*
настройка ожидаемого размера кэша	*effective_cache_size*
управление генетическим оптимизатором	*geqo**
размер рабочей памяти	*work_mem*
работа с курсорами	*cursor_tuple_fraction*

Демонстрация



Итоги

Для поиска задач, нуждающихся в оптимизации, используется профилирование

Средства профилирования зависят от задачи:

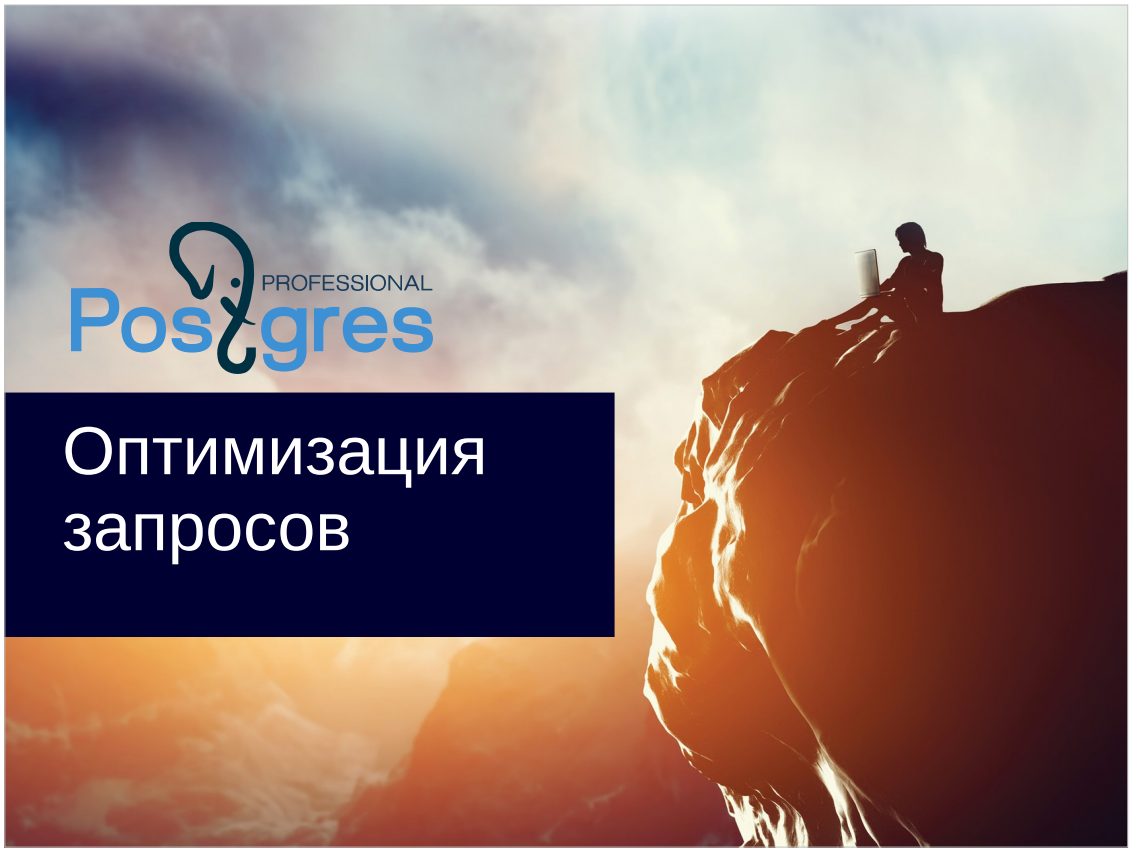
- журнал сообщений сервера и pg_stat_statements
- explain analyze

Доступен широкий спектр методов влияния на план выполнения запросов

Ничто не заменит голову и здравый смысл

Практика

1. Создайте базу DB19.
2. Создайте и наполните данными таблицы так же, как это было сделано в демонстрации. Создайте те же индексы.
   Выполните очистку и соберите статистику.
3. Напишите запрос, выводящий сумму заказов по каждому из последних десяти дней. Если в какой-то из дней заказов не было, должна быть показана нулевая сумма.
4. Постарайтесь максимально оптимизировать этот запрос.
   Сравните время выполнения с первоначальным вариантом.



Авторы: Егор Рогов, Павел Лузанов

Обратная связь
Отзывы, замечания и предложения направляйте по адресу:
edu@postgrespro.ru

Темы

Профилирование

Способы влияния на планы запросов

Примеры

Инструмент

Профилирование

- выделение подзадач
- продолжительность
- количество выполнений

Что оптимизировать?

- чем больше доля подзадачи в общем времени выполнения, тем больше потенциальный выигрыш
- необходимо учитывать затраты на оптимизацию
- полезно взглянуть на задачу шире

В предыдущих темах мы разобрались с тем, как работают запросы, из каких «кирпичиков» строится план выполнения и что влияет на выбор того или иного плана. Это самое сложное и важное. Поняв механизмы, мы с помощью логики и здравого смысла можем разобраться в любой возникающей ситуации и понять, эффективно ли выполняется запрос и что можно сделать, чтобы улучшить показатели.

В этой теме мы рассмотрим несколько типичных примеров, но вначале поговорим о том, как вообще найти тот запрос, который имеет смысл оптимизировать.

В принципе, любая задача оптимизации (не только в контексте СУБД) начинается с профилирования, хоть этот термин и не всегда употребляется явно. Мы должны разбить задачу, вызывающую нарекания, на подзадачи и измерить, какую часть общего времени они занимают. Также полезна информация о числе выполнения каждой из подзадач.

Чем больше доля подзадачи в общем времени, тем больше выигрыш от оптимизации именно этой подзадачи. На практике приходится учитывать и ожидаемые затраты на оптимизацию: получить потенциальный выигрыш может оказаться нелегко.

Часта ситуация, при которой подзадача выполняется быстро, но часто. Может оказаться невозможным ускорить выполнение, но вообще это повод задаться вопросом: а должна ли подзадача выполняться *так* часто? Это может привести к непростой мысли об изменении архитектуры, но в итоге дать существенный выигрыш.

Нюансы

Что профилировать

- отдельную задачу, вызывающую нарекания
- чем точнее, тем лучше: широкий охват «размывает» проблему

Единицы изменения

- время —
  осмысленность для пользователя
- прочитанные или записанные страницы —
  стабильность по отношению ко внешним факторам

Лучше всего строить профиль для одной конкретной задачи, так, чтобы измерения затрагивали только действия, необходимые для выполнения только этой задачи. Если, например, пользователь жалуется на то, что «окно открывается целую минуту», бессмысленно смотреть на всю активность в базе данных за эту минуту: в эти цифры попадут действия, не имеющие никакого отношения к открытию окна.

В каких единицах измерять ресурсы? Самая осмысленная для конечного пользователя характеристика — это время отклика: сколько прошло времени «от нажатия на кнопку» до «получения результата».

Однако с технической точки зрения смотреть на время не всегда удобно. Оно сильно зависит от массы внешних факторов: от наполненности кэша, от текущей нагрузки на сервер. Если проблема решается не на продуктивном, а на другом (тестовом) сервере с иными характеристиками, то добавляется и разница в аппаратуре и настройках.

В этом смысле может оказаться удобнее смотреть, например, на число прочитанных и записанных страниц. Этот показатель более стабилен и как правило отражает объем работы при выполнении запроса, поскольку основное время уходит на чтение и обработку страниц с данными. Хотя — повторимся — такой показатель не имеет смысла для конечного пользователя.

Профиль приложения

Подзадачи

- клиентская часть
- сервер приложений
- сервер баз данных ← *проблема часто, но не всегда, именно здесь*
- сеть

Как профилировать

- технически трудно, нужны разнообразные средства мониторинга
- подтвердить предположение обычно не сложно

Для пользователя имет смысл время отклика. Это означает, что в профиль, вообще говоря, должна входить не только СУБД, но и клиентская часть, и сервер приложений, и передача данных по сети.

Часто проблемы с производительностью кроются именно в СУБД, поскольку один неадекватно построенный план запроса может увеличивать время на порядки. Но это не всегда так. Проблема может оказаться в том, что у клиента медленное соединение с сервером, что клиентская часть долго отрисовывает полученные данные и т. п.

К сожалению, получить такой полный профиль достаточно трудно. Для этого все компоненты информационной системы должны быть снабжены подсистемами мониторинга и трассировки, учитывающими особенности именно этой системы. Но обычно несложно измерить общее время отклика (хотя бы и секундомером) и сравнить с общим временем работы СУБД; убедиться в отсутствии существенных сетевых задержек и т. п. Если же этого не проделать, то вполне может оказаться, что мы будем искать ключи там, где светлее, а не там, где их потеряли. Собственно, в этом и состоит весь смысл профилирования.

Далее мы будем считать установленным, что проблема именно в СУБД.

Профиль СУБД

Журнал сообщений сервера

включается конфигурационными параметрами:

log_min_duration_statements = 0 — время и текст всех запросов

log_line_prefix — идентифицирующая информация

сложно включить для отдельного сеанса

большой объем; при увеличении порога времени теряем информацию

не отслеживаются вложенные запросы
(можно использовать расширение auto_explain)

анализ внешними средствами, такими, как pgBadger

При выполнении пользователем действия, выполняется обычно не один, а несколько запросов. Как определить тот запрос, который имеет смысл оптимизировать? Для этого нужен профиль, детализированный до запросов. Для его получения есть два основных средства: журнал сообщений сервера и статистика.

В журнал сообщений с помощью конфигурационных параметров можно включить вывод информации о запросах и времени их выполнения. Обычно для этого используется параметр *log_min_duration_statement,* хотя есть и другие.

Как локализовать в журнале те запросы, которые относятся к действиям пользователя? Можно было бы включать и выключать параметр для конкретного сеанса, но штатных средств для этого не предусмотрено. Можно фильтровать общий поток сообщений, выделяя только нужные; для этого удобно в параметр *log_line_prefix* вывести дополнительную идентифицирующую информацию.

Все еще более осложняется при использовании пула соединений, особенно на уровне транзакций. В идеале надо предусматривать возможность идентификации сеансов уже при проектировании системы.

Следующая проблема — анализ журнала. Это осуществляется внешними средствами, из которых стандартом де-факто является расширение pgBadger.

Разумеется, можно выводить в журнал и собственные сообщения, если они могут помочь делу.

Профиль СУБД

Расширение pg_stat_statements

- подробная информация о запросах в представлении (в том числе о вложенных)
- ограниченный размер хранилища
- запросы считаются одинаковыми «с точностью до констант», даже если имеют разные планы выполнения
- идентификация только по имени пользователя и базе данных

Второй способ — статистика, а именно расширение pg_stat_statements.

Расширение собирает достаточно подробную информацию о выполняемых запросах (в том числе в терминах ввода-вывода страниц) и отображает ее в представлении pg_stat_statements.

Поскольку число различных запросов может быть очень большим, размер хранилища ограничен конфигурационным параметром; в нем остаются наиболее часто используемые запросы.

При этом «одинаковыми» считаются запросы, имеющие одинаковое (с точностью до констант) дерево разбора. Надо иметь в виду, что такие запросы могли иметь разные планы выполнения и выполняться разное время.

К сожалению, есть сложности и с идентификацией: запросы можно отнести к конкретному пользователю и базе данных, но не к сеансу.

Профиль запроса

Explain analyze

- подзадачи — узлы плана
- продолжительность — actual time
- количество выполнений — loops

Особенности

- помимо наиболее ресурсоемких узлов, кандидаты на оптимизацию — узлы с большой ошибкой прогноза кардинальности
- любое вмешательство может привести к полной перестройке плана
- иногда приходится довольствоваться простым explain

Так или иначе, среди выполненных запросов мы находим тот, оптимизация которого сулит наибольшую выгоду. Как работать с самим запросом? Тут тоже поможет профиль, который выдает команда explain analyze.

Подзадачами такого профиля являются узлы плана (надо учитывать, что это не плоский список, а дерево) и отдельно время планирования. Продолжительность выполнения узла покажет actual time, а число выполнений — loops.

Но здесь имеются особенности. Во-первых, кроме перечисленного, план выполнения содержит дополнительную — и крайне важную — информацию об ожидании оптимизатора относительно кардинальности каждого шага. Как правило, если нет серьезной ошибки в прогнозе кардинальности, то и план будет построен адекватный (если нет, то надо изменять глобальные настройки). Поэтому стоит обращать внимание не только на наиболее ресурсоемкие узлы, но и на узлы с большой (на порядок или выше) ошибкой. Конечно, смотреть надо на наиболее вложенный проблемный узел, поскольку ошибка будет распространяться от него выше по дереву.

Во-вторых, надо быть готовым к тому, что любое вмешательство может привести не к сокращению времени выполнения узла, а к полной перестройке плана.

Наконец, бывают ситуации, когда запрос выполняется так долго, что выполнить explain analyze не представляется возможным. В таком случае придется довольствоваться простым explain и пытаться разобраться в причине неэффективности без полной информации.

Пути оптимизации

Цель — получить адекватный план

Исправление неэффективностей

- каким-то образом найти и исправить узкое место
- бывает сложно догадаться, в чем проблема
- часто приводит к борьбе с планировщиком

Правильный расчет кардинальности

- добиться правильного расчета кардинальности в каждом узле и положиться на планировщик
- если план все еще неадекватный, настраивать глобальные параметры

Цель оптимизации — получить адекватный план выполнения запроса. Но есть разные пути, которыми можно идти к этой цели.

Можно посмотреть в план, понять причину неэффективного выполнения и сделать что-то, исправляющее ситуацию. К сожалению, проблема не всегда бывает очевидной, а исправление часто сводится к борьбе с оптимизатором.

Если идти таким путем, то хочется иметь возможность целиком или частично отключить планировщик и самому создать план выполнения. Это то, что называется подсказками (хинтами) и отсутствует в явном виде в PostgreSQL.

Другой подход состоит в том, чтобы добиться корректного расчета кардинальности в каждом узле плана. Для этого, конечно, нужна корректная статистика, но этого часто бывает недостаточно.

Если идти таким путем, то мы не боремся с планировщиком, а помогаем ему принять верное решение. К сожалению, это часто оказывается слишком сложной задачей.

Если при правильно оцененной кардинальности планировщик все равно строит неэффективный план, это повод заняться настройкой глобальных конфигурационных параметров.

Обычно имеет смысл применять оба способа, смотря по ситуации и сообразуясь со здравым смыслом

Классификация способов

Postgres PROFESSIONAL

Изменение исходного кода

можно ли изменить текст запроса или приложения

Изменение схемы данных

можно ли вносить изменения в способ хранения данных

Гибкость

ограничивает ли изменение свободу планировщика

Локальность влияния

действует ли изменение локально на один запрос
или глобально на всю систему

Универсальность применения

применяется ли повсеместно или помогает в редких случаях

Каким образом мы можем повлиять на расчет кардинальности и на план выполнения запроса? Существуют разные способы, которые можно классифицировать по нескольким признакам.

Можем ли мы изменять текст запроса? Это может быть запрещено условиями лицензионного соглашения или поддержки продукта, или невозможно из-за автоматической генерации запросов ORM-ом и т. п.

Можем ли мы изменять способ хранения данных? Например, создать индекс или применить денормализацию.

Оставляет ли выбранный способ свободу планировщику, помогаем ли мы ему или боремся с ним? В (недостижимом) идеале планировщик должен сам выбирать наилучшие планы. Если же совсем лишить его свободы выбора, то при существенном изменении в распределении данных придется опять заниматься «ручной» оптимизацией.

Влияет ли метод на отдельный запрос, группу запросов или вообще на всю систему? Чем глобальнее воздействие, тем осторожнее надо применять метод, так как могут пострадать другие запросы.

Универсален ли метод, используется ли он постоянно или его применение ограничено отдельными специальными случаями?

Для начала

Статистика

без актуальной статистики планировщик бесполезен

статистика не должна быть точной, но должна быть похожей на правду

настройка autovacuum, *default_statistics_target*

Ограничения целостности

помимо всего прочего, могут учитываться и при планировании

В первую очередь стоит еще раз напомнить, что первый шаг к адекватному плану — актуальная статистика. Признаком неактуальной статистики будет серьезное несоответствие ожидаемого и реального числа строк в листовых узлах плана. (Если несоответствие наблюдается выше по дереву, оно с большой вероятностью вызвано неправильным расчетом селективности соединений или агрегаций.)

Иногда может потребоваться изменить значение *default_statistics_target* (глобально или для отдельных столбцов таблиц).

Также полезно явно указывать ограничения целостности: помимо их проверки, они могут учитываться и при планировании запросов.

Кардинальность

Функциональные индексы

- отдельная статистика

Левое соединение

- показать, что соединение не уменьшает число строк

Устранение коррелированных предикатов

Временные таблицы

- для части запроса, чтобы использовать статистику

Чтобы исправить некорректный расчет кардинальности (при условии наличия аккуратной статистики), можно воспользоваться разными способами.

Если неверно оценивается селективность условия, может помочь функциональный индекс по выражению, поскольку для него будет собрана отдельная статистика. (Не забываем про дополнительный расход места и затраты на обновление индекса.)

Если планировщик думает, что соединение уменьшает число строк, а на самом деле это не так, можно искусственно использовать левое соединение. (Не забываем про то, такая операция в ряде случаев накладывает ограничения на порядок соединений.)

Если в запросе есть коррелированные предикаты, от них лучше избавиться (возможно, изменив структуру данных).

Если ничего не помогает, можно часть запроса записать во временную таблицу. Даже без статистики планировщик может примерно догадаться о числе строк по размеру файла; можно собрать и статистику. (Не забываем о накладных расходах на создание и удаление временной таблицы и на запись и чтение данных, а также о проблеме разрастания таблиц системного каталога.)

Разумеется, список не исчерпывающий, и не все способы годятся в любой ситуации.

Порядок соединения

Перенос условия фильтрации на другую таблицу

планировщик не всегда может перенести условия от одной таблицы к другой

With-запросы (CTE)

всегда материализуются

Явные соединения и *join_collapse_limit* = 1

планировщик не будет менять порядок соединений

Подзапросы и *from_collapse_limit* = 1

планировщик не будет раскрывать подзапросы

Коррелированные подзапросы

планировщик может не справиться с раскрытием

Для исправления порядка соединения таблиц также есть много вариантов.

Иногда условия фильтрации указываются применительно к одной из таблиц, хотя логически их можно было бы перенести и на другую. Планировщик не всегда может сделать это сам.

Чтобы зафиксировать порядок соединений, можно воспользоваться CTE — такие выражения всегда материализуются и не раскрываются в основной запрос. (Не забываем про накладные расходы.)

Можно явно указать порядок соединений (join … on) и установить параметр join_collapse_limit = 1. Тогда планировщик будет строго придерживаться указанного порядка. (Не забываем, что при изменении распределения данных запрос придется снова переписать вручную.)

Похожий параметр есть для подзапросов: при from_collapse_limit = 1 подзапросы не раскрываются в основной запрос.

Планировщик не всегда справляется с раскрытием коррелированых подзапросов; в некоторых случаях запрос имеет смысл переписать.

Итак, на порядок соединений можно довольно гибко влиять, разрешая планировать подзапросы и CTE, но закрепляя порядок их соединения друг с другом.

Напомним, что конфигурационные параметры могут устанавливаться на разных уровнях: для системы (alter system), пользователя (alter role), базы данных (alter database), функции (alter function), сеанса (set) или транзакции (set local). Подробно это рассматривается в курсе DBA1.

Доступы и соединения

Конфигурационные параметры (локально)

методы доступа,	*enable_(seqscan|indexscan|...)*
способы соединения,	*enable_(nestloop|hashjoin|mergejoin)*
операции	*enable_(hashagg|sort|material)*

— грубо, но полезно для экспериментов

размер рабочей памяти	*work_mem*

Подсказки оптимизатору

отсутствуют, но есть конфигурационные параметры и расширения

Сложнее повлиять на методы доступы и способы соединения.

Есть конфигурационные параметры, разрешающие или запрещающие использование определенных методов доступа (enable_seqscan, enable_indexscan и т. п.), способов соединения (enable_nestloop, enable_hashjoin, enable_mergejoin) и некоторые операции (агрегацию хэшированием enable_hashagg, сортировку enable_sort и т. п.).

Установка этих параметров в off не запрещает операции, но устанавливает им очень большую стоимость. Таким образом, планировщик будет пытаться обойтись без них, но все равно может применить в безвыходной ситуации.

Параметр *work_mem,* определяющий размер доступной для операции памяти (см. тему 18. «Использование памяти»), также влияет на выбор планировщика. При дефиците памяти предпочнение отдается сортировке (а не хэшированию), поскольку ее алгоритм менее чувствителен к небольшому размеру памяти.

Еще один (традиционный для других СУБД) способ влияния — подсказки оптимизатору — отсутствует в PostgreSQL (обсуждение). На самом деле часть подсказок существует в виде конфигурационных параметров, а кроме того есть специальные расширения, например https://en.osdn.jp/projects/pghintplan/. (Не забываем, что использование подсказок, сильно ограничивающих свободу планировщика, может навредить в будущем, когда распределение данных изменится.)

Денормализация

Индексы

обновляются автоматически

Поля таблиц

надо обновлять (триггерами или другим способом)

Материализованные представления

надо обновлять (по расписанию или другим способом)

Кэширование результатов в приложении

надо обновлять (при необходимости)

На логическом уровне база данных должна быть нормализованной. Говоря неформально, в хранисых данных не должно быть избыточности. Если это не так, мы имеем дело с ошибкой проектирования.

Однако на уровне хранения некоторое дублирование может оказаться полезным: это может позволить существенно выиграть в производительности ценой того, что избыточные данные необходимо синхронизировать с основными.

Самый простой и частый способ денормализации — индексы (хотя о них обычно не говорят и не думают в таком контексте). Индексы обновляются автоматически.

Можно дублировать некоторые данные (или результаты расчета на основе этих данных) в полях таблиц. Такую информацию необходимо синхронизировать вручную — обычным способом являются триггеры, так что денормализация оказывается локализованной в базе данных и приложению не требуется «думать» об этом.

Другой пример — материализованные представления. Их также надо обновлять, например, по расписнию или другим способом.

В ряде случаев, когда число входных параметров невелико, а получение результата обходится дорого (или происходит часто), полезным оказывается кэширование результата на уровне приложения. Кэш также надо обновлять — когда это делать, зависит от задачи.

Разное

Секционирование

- для очень больших таблиц, требующих полного сканирования

Изменение запроса

- трансформация охватывает не все эквивалентные формы запроса
- например, операции union, except, intersect не трансформируются

Замена процедурного кода декларативным

- избавление от циклов и небольших, но многочисленных команд SQL

Существует множество других способов влияния на производительность. Отметим еще несколько.

Секционирование позволяет организовать работу с данными очень большого объема. Основная выгода для производительности состоит в замене полного сканирования всей таблицы сканированием отдельной секции.

Важный способ — переформулирование запроса в том случае, когда планировщик не может автоматически трансформировать запрос в эквивалентную, но потенциально более выгодную, форму. Например, операции для работы со множествами (union, except, intersect) в настоящее время не трансформируются.

Еще более важный способ — избавление от процедурного кода в приложении (для которого характерно наличие циклов, внутри которых выполняется большое число мелких операторов SQL) и перенос его на сервер БД в виде небольшого числа крупных SQL-команд. Это дает планировщику возможность применить более эффективные способы доступа и соединений и избавляет от многочисленных пересылок данных от клиента к серверу и обратно.

Еще раз повторимся, что приведенный перечень способов воздействия ни в коем случае не является исчерпывающим.

Глобальные настройки

Глобальные конфигурационные параметры

настройка стоимости отдельных операций	**_cost*
настройка ожидаемого размера кэша	*effective_cache_size*
управление генетическим оптимизатором	*geqo**
размер рабочей памяти	*work_mem*
работа с курсорами	*cursor_tuple_fraction*

Если в базе данных собрана аккуратная статистика, для запросов считается адекватная кардинальность, но при этом планировщик выбирает неудачные планы, имеет смысл поработать с глобальными настройками.

Возможно, стоит отрегулировать относительные стоимости элементарных операций, на основе которых оцениваются планы (random_page_cost и пр.).

Увеличение *effective_cache_size* увеличивает предпочтение индексного доступа, так как оптимизатор надеется обнаружить данные в кэше.

Можно управлять перебором вариантов соединения таблиц в запросе и в том числе генетически оптимизатором GEQO.

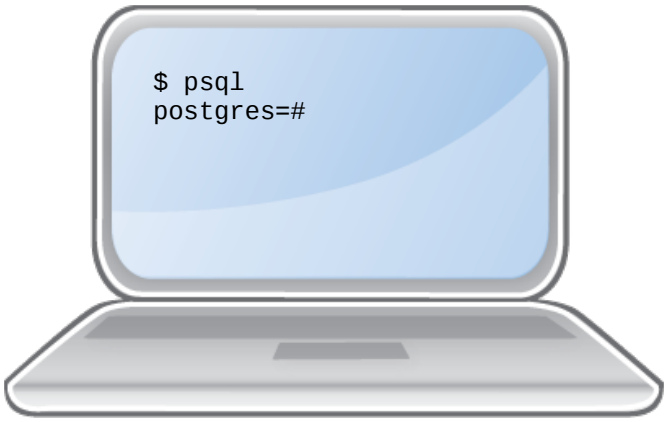
Большое влияние на план оказывает размер доступной рабочей памяти.

При использовании курсоров можно попросить оптимизатор как можно быстрее получать первые данные.

Следует иметь в виду, что эти параметры как правило имеют глобальное воздействие на всю систему, поэтому изменять их значения следует с осторожностью.

http://www.postgresql.org/docs/9.5/static/runtime-config-query.html

Демонстрация



Итоги

Для поиска задач, нуждающихся в оптимизации, используется профилирование

Средства профилирования зависят от задачи:

- журнал сообщений сервера и pg_stat_statements
- explain analyze

Доступен широкий спектр методов влияния на план выполнения запросов

Ничто не заменит голову и здравый смысл

Практика

1. Создайте базу DB19.
2. Создайте и наполните данными таблицы так же, как это было сделано в демонстрации. Создайте те же индексы.
   Выполните очистку и соберите статистику.
3. Напишите запрос, выводящий сумму заказов по каждому из последних десяти дней. Если в какой-то из дней заказов не было, должна быть показана нулевая сумма.
4. Постарайтесь максимально оптимизировать этот запрос.
   Сравните время выполнения с первоначальным вариантом.

В качестве начального варианта можно воспользоваться таким запросом:

```
select   gen.d::date date_ordered,
         sum(i.amount) amount
from     generate_series(
           now(),
           now() - interval '9 day',
           interval '-1 day'
         ) gen(d)
    left join
         orders o
         on (o.date_ordered::date = gen.d::date)
    left join
         items i
         on (i.order_id = o.id)
group by gen.d::date
order by gen.d::date;
```