



# Оптимизация запросов



Способы влияния на планы запросов

Примеры

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

## 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

## 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

## Функциональные индексы

отдельная статистика

## Левое соединение

показать, что соединение не уменьшает число строк

## Устранение коррелированных предикатов

## Временные таблицы

для части запроса, чтобы использовать статистику

## 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

## Конфигурационные параметры (локально)

методы доступа,  
способы соединения,  
операции

*enable\_(seqscan|indexscan|...)*  
*enable\_(nestloop|hashjoin|mergejoin)*  
*enable\_(hashagg|sort|material)*

— грубо, но полезно для экспериментов

размер рабочей памяти

*work\_mem*

## Подсказки оптимизатору

отсутствуют, но есть конфигурационные параметры и расширения

## 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)

## 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 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*

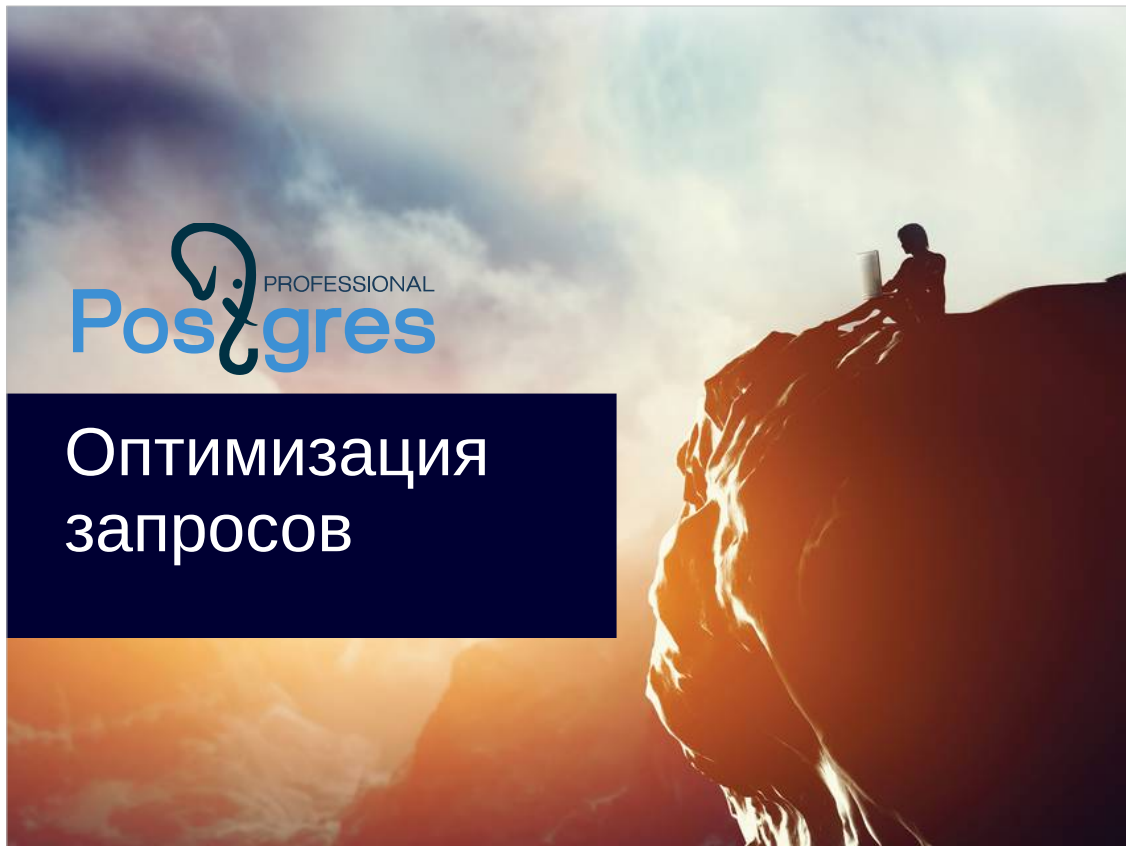
# Демонстрация



Доступен широкий спектр методов влияния  
на план выполнения запросов

Ничто не заменит голову и здравый смысл

1. Создайте базу DB20.
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.



### **Авторские права**

Курс «Администрирование PostgreSQL 9.5. Расширенный курс»

© Postgres Professional, 2016 год.

Авторы: Егор Рогов, Павел Лузанов

### **Использование материалов курса**

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

### **Обратная связь**

Отзывы, замечания и предложения направляйте по адресу:

[edu@postgrespro.ru](mailto:edu@postgrespro.ru)

### **Отказ от ответственности**

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Способы влияния на планы запросов

Примеры

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

## 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?

## 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.

## 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.

## Перенос условия фильтрации на другую таблицу

планировщик не всегда может перенести условия от одной таблицы к другой

## With-запросы (СТЕ)

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 СТЕ — 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 коррелированных подзапросов; в некоторых случаях запрос имеет смысл переписать.

Итак, на порядок соединений можно довольно гибко влиять, разрешая планировать подзапросы и СТЕ, 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.

## Конфигурационные параметры (локально)

методы доступа, способы соединения, операции	<i>enable_(seqscan indexscan ...)</i> <i>enable_(nestloop hashjoin mergejoin)</i> <i>enable_(hashagg sort material)</i>
— грубо, но полезно для экспериментов	
размер рабочей памяти	<i>work_mem</i>

## Подсказки оптимизатору

отсутствуют, но есть конфигурационные параметры и расширения

Сложнее повлиять на методы доступа 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 предпочтение 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.)

## 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 хранимых 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 расписанию 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.

## 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 конфигурационные параметры

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

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>



Доступен широкий спектр методов влияния  
на план выполнения запросов

Ничто не заменит голову и здравый смысл

1. Создайте базу DB20.
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;
```