Postgres PROFESSIONAL

Обработка запроса

Темы

Разбор

Переписывание (трансформация)

Планирование (оптимизация)

Выполнение

Подготовленные операторы

Подходы к оптимизации

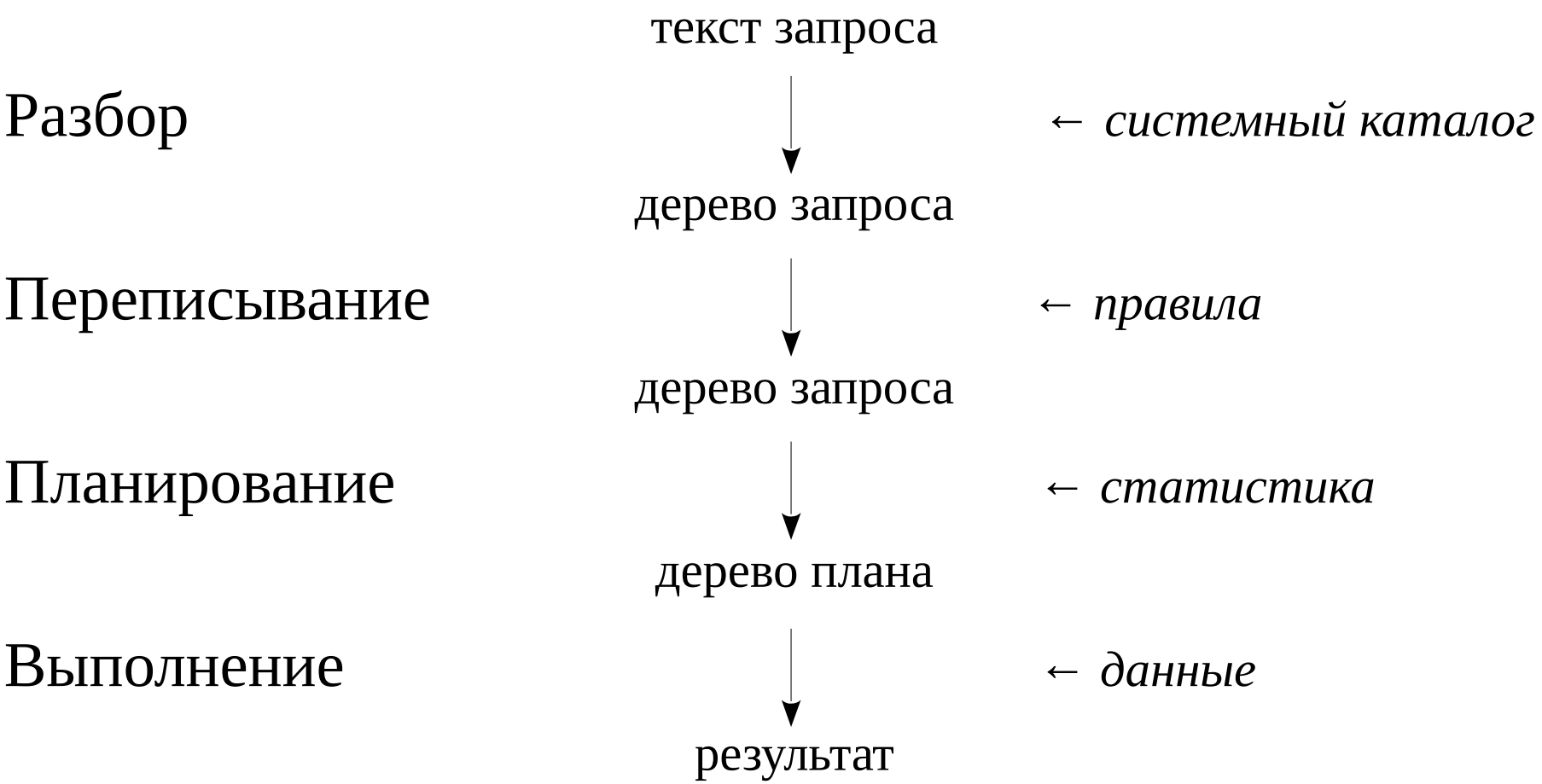
Настройка параметров

- подстройка под имеющуюся нагрузку
- глобальное влияние на всю систему
- мониторинг

Оптимизация запросов

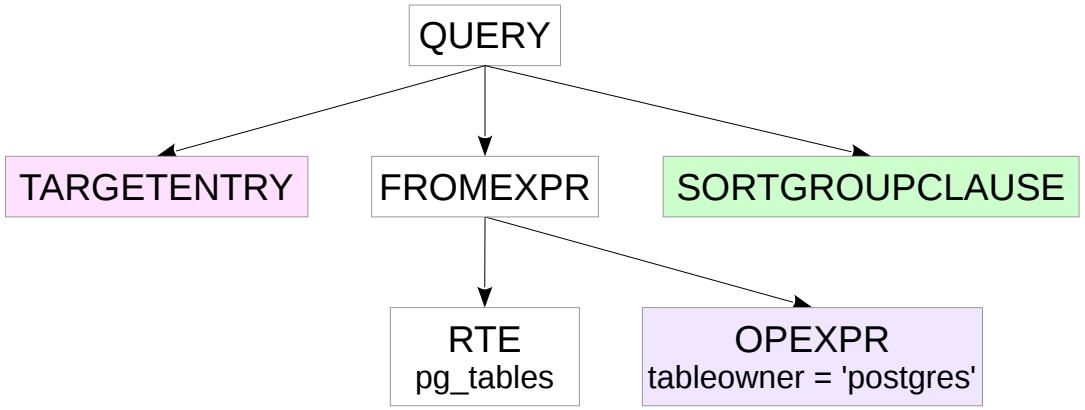
- уменьшение нагрузки
- локальное воздействие (запрос или несколько запросов)
- профилирование

Схема обработки запроса



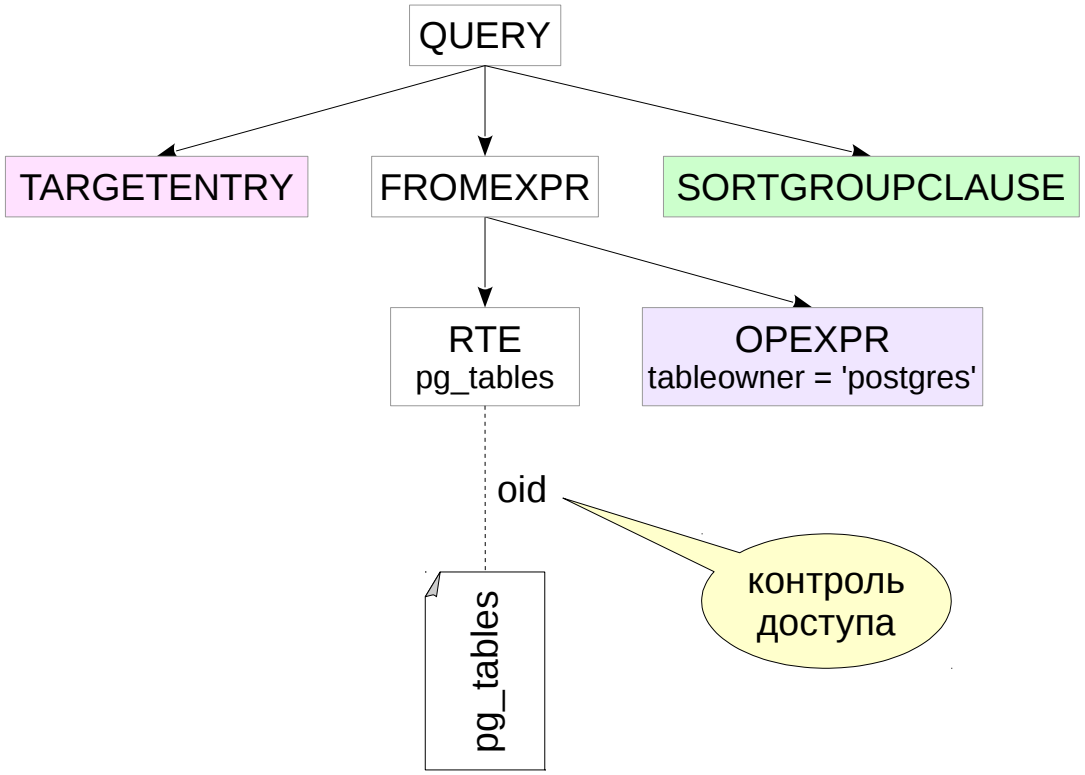
Синтаксический разбор

```
SELECT schemaname, tablename
FROM pg_tables
WHERE tableowner = 'postgres'
ORDER BY tablename;
```

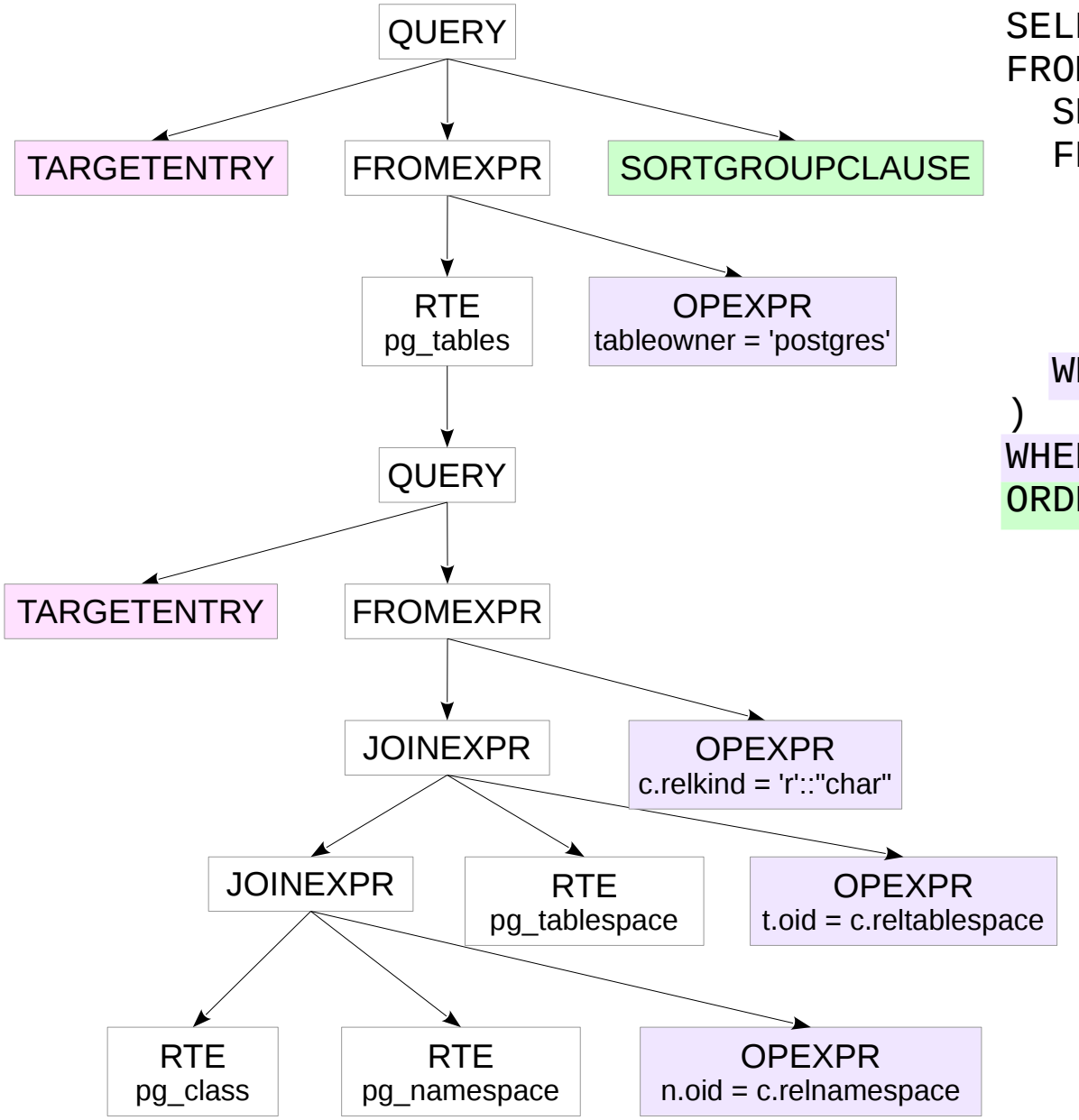


Семантический разбор

```
SELECT schemaname, tablename
FROM pg_tables
WHERE tableowner = 'postgres'
ORDER BY tablename;
```



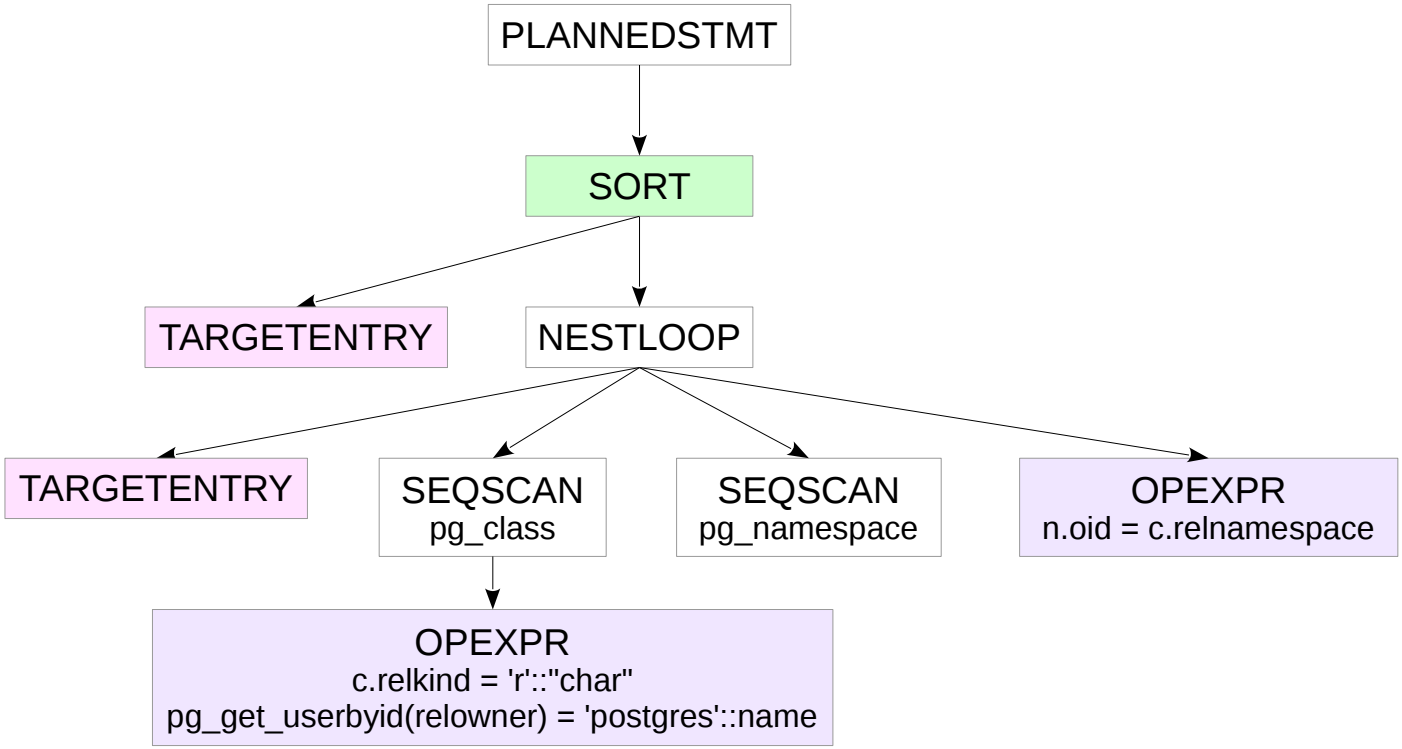
Переписывание




```
SELECT schemaname, tablename
FROM (
  SELECT ...
  FROM pg_class c
    LEFT JOIN pg_namespace n
      ON n.oid = c.relnamespace
    LEFT JOIN pg_tablespace t
      ON t.oid = c.reltablespace
  WHERE c.relkind = 'r'::"char"
)
WHERE tableowner = 'postgres'
ORDER BY tablename;
```

Планирование

```
Sort  (cost=19.59..19.59 rows=1 width=128)
  Sort Key: c.relname
  ->  Nested Loop Left Join  (cost=0.00..19.58 rows=1 width=128)
        Join Filter: (n.oid = c.relnamespace)
        ->  Seq Scan on pg_class c  (cost=0.00..18.44 rows=1 width=72)
              Filter: ((relkind = 'r'::"char") AND
                       (pg_get_userbyid(relowner) = 'postgres'::name))
        ->  Seq Scan on pg_namespace n  (cost=0.00..1.06 rows=6 width=68)
```



Выполнение

Конвейер

- обход дерева от корня вниз
- данные передаются наверх — по мере поступления ли все сразу

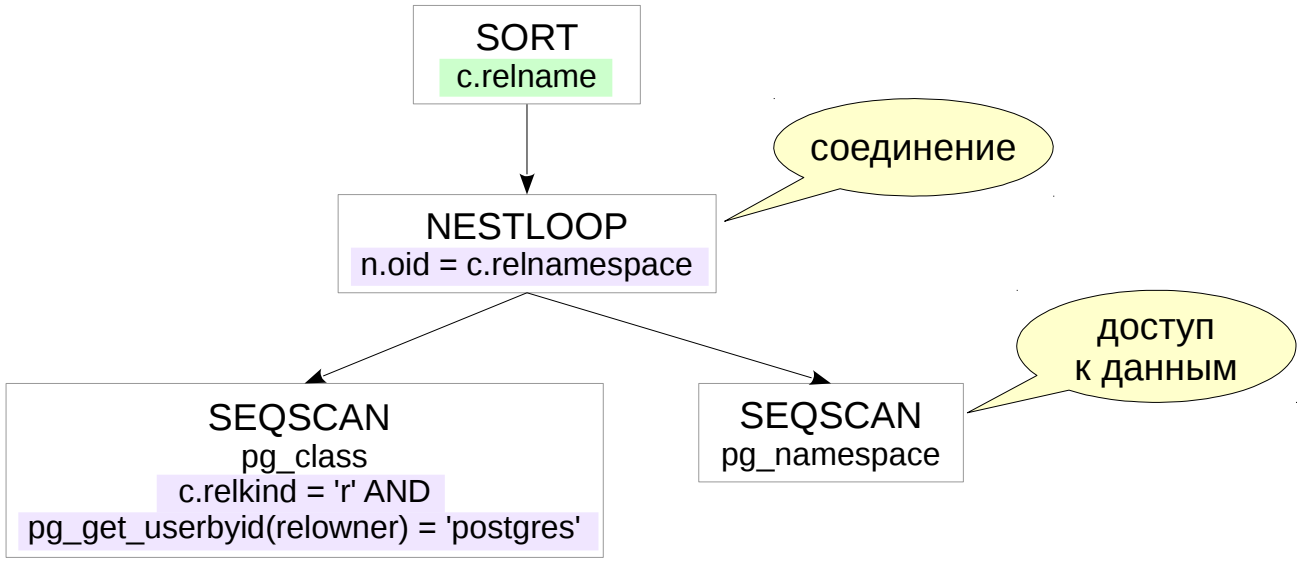
Доступ к данным

- чтение таблиц, индексов

Соединения

- всегда попарно
- важен порядок

Другие операции



Подготовленные операторы

Подготовка

`PREPARE` *имя* `(`*параметры*`) AS` *оператор*

выполняется разбор и переписывание,
а в ряде случаев — и оптимизация

дерево запроса сохраняется в памяти процесса

Выполнение

`EXECUTE` *имя* `(`*параметры*`)`

происходит оптимизация (как правило)
и собственно выполнение

гарантия невозможности внедрения SQL-кода

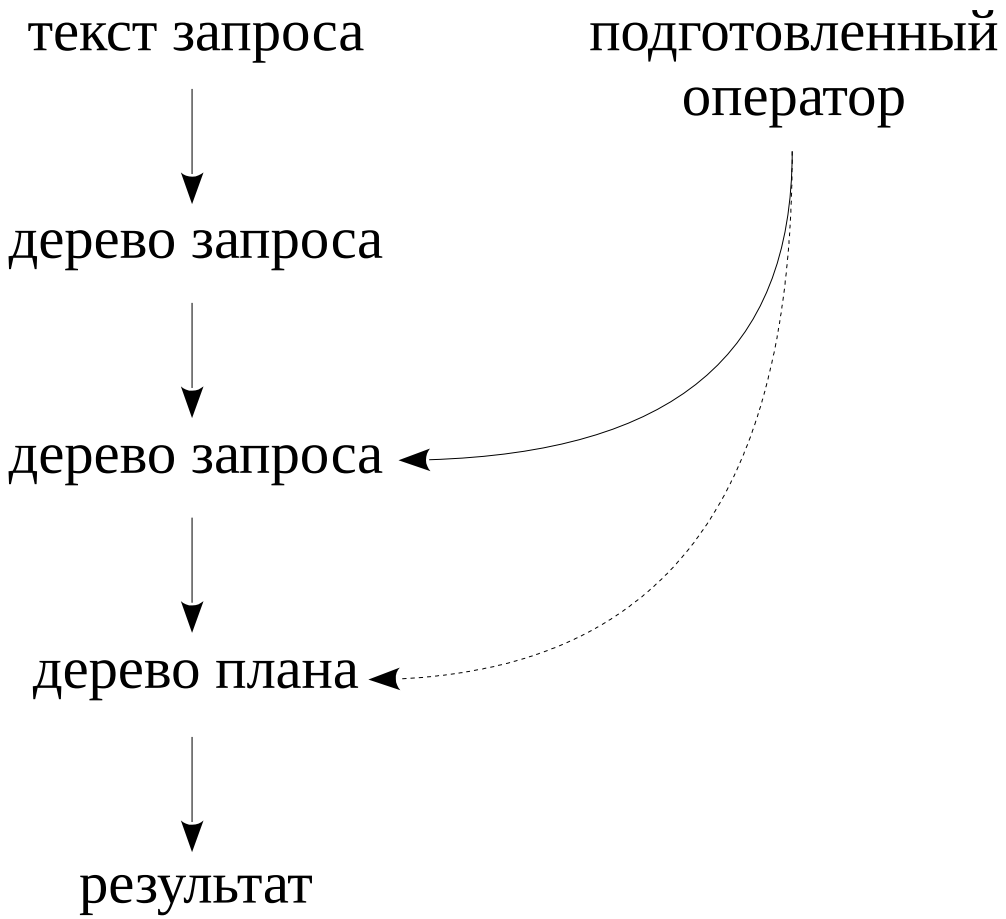
Схема обработки запроса

Разбор

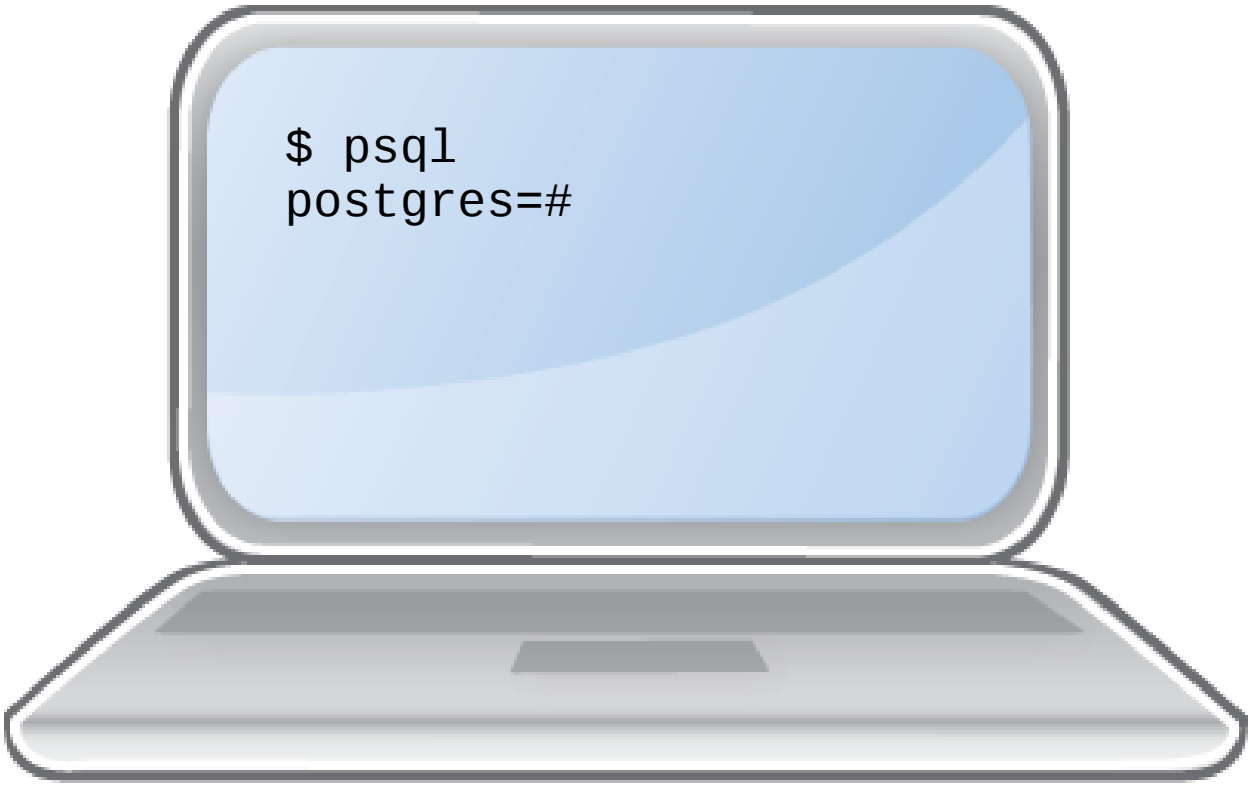
Переписывание

Планирование

Выполнение



Демонстрация



Итоги

Обработка запроса состоит из нескольких шагов:
разбор и переписывание, планирование, выполнение

Стоит подготавливать операторы, если они:

- выполняются в сеансе несколько раз
- используют потенциально опасные данные

Время выполнения зависит от качества планирования

Практика

1. Создайте базу DB10.
2. Напишите запрос, подсчитывающий сумму чисел из таблицы с миллионом строк; посчитайте среднее время выполнения.
3. Подготовьте оператор для этого запроса; посчитайте среднее время выполнения.
4. Во сколько раз ускорилось выполнение?
5. Напишите запрос одной записи из таблицы по значению первичного ключа; посчитайте среднее время выполнения.
6. Подготовьте оператор для этого запроса; посчитайте среднее время выполнения.
7. Во сколько раз ускорилось выполнение в этом случае?

Авторы: Егор Рогов, Павел Лузанов

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

Темы

Разбор

Переписывание (трансформация)

Планирование (оптимизация)

Выполнение

Подготовленные операторы

Подходы к оптимизации

Настройка параметров

- подстройка под имеющуюся нагрузку
- глобальное влияние на всю систему
- мониторинг

Оптимизация запросов

- уменьшение нагрузки
- локальное воздействие (запрос или несколько запросов)
- профилирование

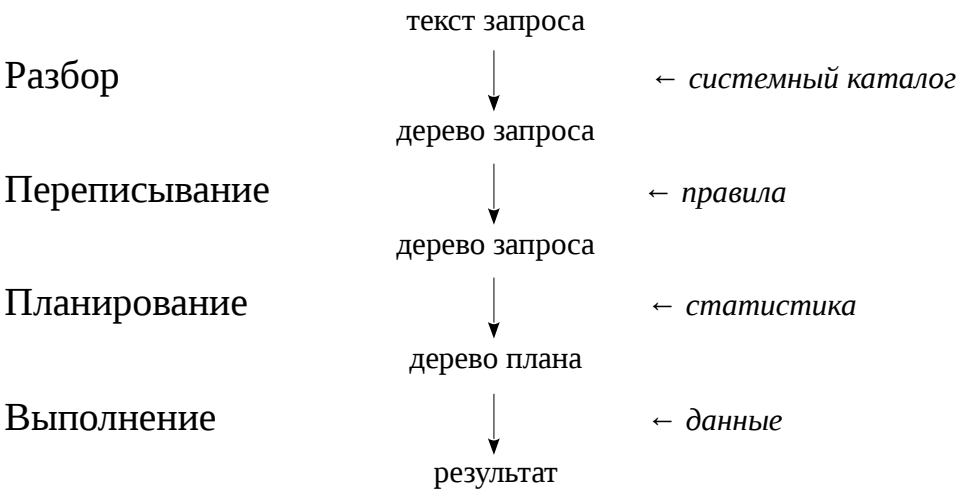
Эта тема начинает раздел, посвященный оптимизации. Вообще «оптимизация» — очень широкое понятие; задумываться об оптимизации необходимо еще на этапе проектирования системы и выбора архитектуры. Мы будем говорить только о тех работах, которые выполняются при эксплуатации уже существующего приложения, причем с точки зрения администратора СУБД.

Можно выделить два основных подхода. Первый состоит в том, чтобы мониторить состояние системы и добиваться того, чтобы она справлялась с имеющейся нагрузкой. Добиваться этого можно разными способами. Можно устанавливать параметры СУБД (основные из которых мы рассматривали в первых девяти темах курса) и настраивать операционную систему. Если настройки не помогают, можно модернизировать аппаратуру — что, разумеется, дороже.

Другой подход, который мы будем рассматривать далее, состоит в том, чтобы не приспосабливаться под нагрузку, а уменьшать ее. «Полезная» нагрузка формируется запросами. Если удается найти узкое место (инструмент для этого — профилирование), то можно попробовать тем или иным способом повлиять на выполнение запроса и получить тот же результат, потратив меньше ресурсов. Такой способ действует более локально (на отдельный запрос или ряд запросов), но уменьшение нагрузки косвенно сказывается и на работе всей системы.

Мы начнем с того, что в деталях разберем механизмы выполнения запросов, а уже после этого поговорим о том, как распознать неэффективную работу и о конкретных способах воздействия.

Схема обработки запроса



Обработка запроса выполняется в несколько этапов.

Во-первых, запрос *разбирается*. Сначала производится синтаксический разбор (parse): текст запроса представляется в виде дерева — так с ним удобнее работать. Затем выполняется семантический анализ (analyze), в ходе которого определяется, на какие объекты БД ссылается запрос и если ли у пользователя доступ к ним (для этого анализатор заглядывает в системный каталог).

Во-вторых, запрос *переписывается* (rewrite) или *трансформируется* с учетом правил. Важный частный случай — подстановка текста запроса вместо имени представления. Заметим, что текст представления опять необходимо разобрать, поэтому мы несколько упрощаем, говоря, что первые два этапа происходят друг за другом.

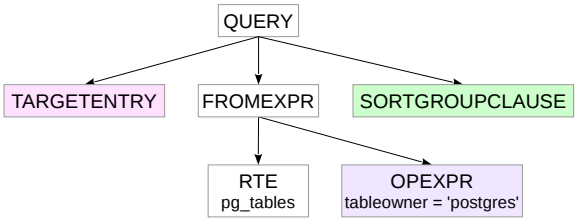
В-третьих, запрос *планируется* (plan). SQL — декларативный язык, и один запрос можно выполнить разными способами. Планировщик (он же оптимизатор) перебирает различные способы выполнения и оценивает их. Оценка дается исходя из некоторой математической модели вычислений и из информации о обрабатываемых данных (статистики). Способ, для которого прогнозируется минимальная стоимость, представляется в виде дерева плана.

В-четвертых, запрос *выполняется* (execute) в соответствии с планом, и результат возвращается клиенту.

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

Синтаксический разбор

```
SELECT schemaname, tablename
FROM pg_tables
WHERE tableowner = 'postgres'
ORDER BY tablename;
```

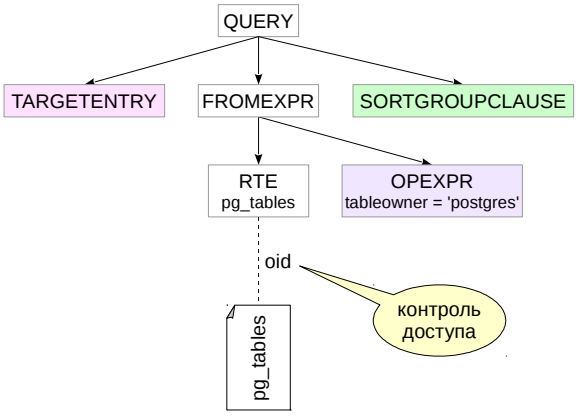


Рассмотрим простой пример: запрос, приведенный на слайде. На этапе синтаксического разбора в памяти серверного процесса из него будет построено дерево, упрощенно показанное на рисунке. Цветом показано примерное соответствие частей текста запроса и узлов дерева.

Если любопытно, то увидеть реальное дерево можно, установив параметр debug_print_parse и заглянув в журнал сообщений сервера (после слов «parse tree»). Практического смысла в этом нет (если, конечно, вы не разработчик ядра PostgreSQL).

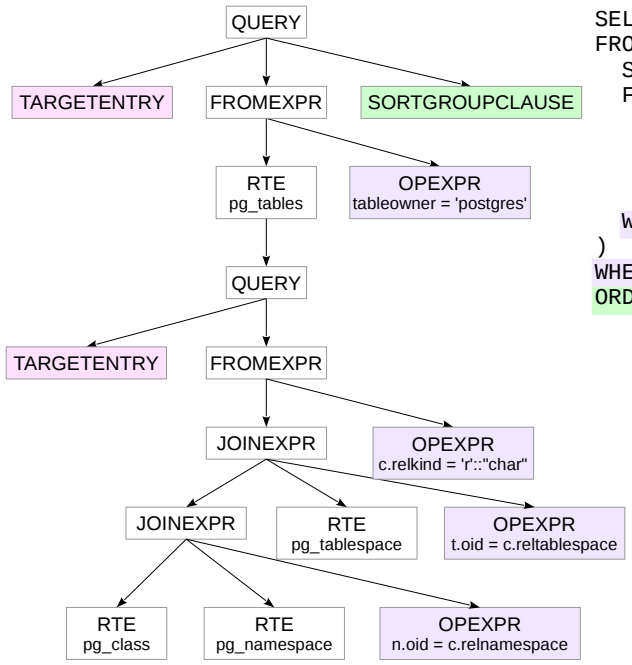
Семантический разбор

```
SELECT schemaname, tablename
FROM pg_tables
WHERE tableowner = 'postgres'
ORDER BY tablename;
```



На этапе семантического разбора анализатор сверится с системным каталогом и свяжет имя «pg_tables» с представлением, имеющем определенный oid в системном каталоге. Также будут проверены права доступа к этому представлению.

Переписывание




```
SELECT schemaname, tablename
FROM (
  SELECT ...
  FROM pg_class c
    LEFT JOIN pg_namespace n
      ON n.oid = c.relnamespace
    LEFT JOIN pg_tablespace t
      ON t.oid = c.reltablespace
  WHERE c.relkind = 'r'::"char"
)
WHERE tableowner = 'postgres'
ORDER BY tablename;
```


На этапе переписывания вместо представления подставляется текст запроса. Текст снова разбирается; в результате снова получается дерево запроса, но уже трансформированное.

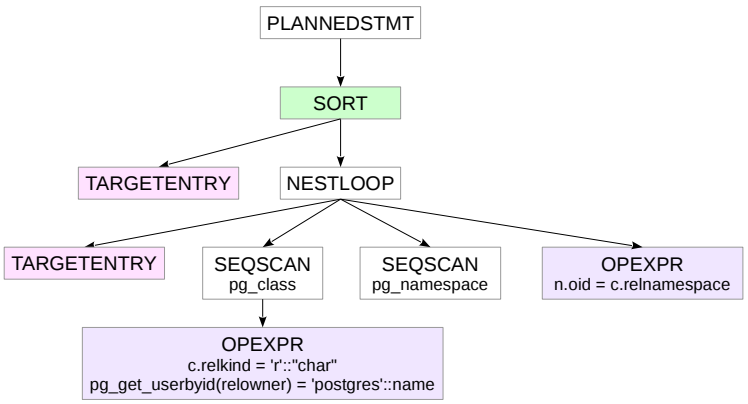
На слайде приведен запрос с подставленным текстом представления (это условность, реально в таком виде запрос не существует — вся работа происходит с деревом запроса).

Поддерево, соответствующее подзапросу, имеет своим родителем узел, ссылающийся на представление. Именно в этом поддереве становится хорошо видна древовидная структура запроса.

Трансформированное дерево можно увидеть в журнале сообщений сервера, установив параметр debug_print_rewritten (после слов «rewritten parse tree»).

Планирование

```
Sort  (cost=19.59..19.59 rows=1 width=128)
  Sort Key: c.relname
  ->  Nested Loop Left Join  (cost=0.00..19.58 rows=1 width=128)
        Join Filter: (n.oid = c.relnamespace)
        ->  Seq Scan on pg_class c  (cost=0.00..18.44 rows=1 width=72)
              Filter: ((relkind = 'r'::"char") AND
                       (pg_get_userbyid(relowner) = 'postgres'::name))
        ->  Seq Scan on pg_namespace n  (cost=0.00..1.06 rows=6 width=68)
```



На этапе планирования из дерева запроса строится дерево, представляющее способ выполнения этого запроса. Здесь операции Seq Scan — это чтение соответствующих таблиц, а Nested Loop — способ соединения двух таблиц. В виде текста на слайде приведен план выполнения — как показывает его команда explain, о которой мы будем говорить позже.

Пока имеет смысл обратить внимание на два момента:

- из трех таблиц осталось только две: планировщик сообразил, что одна из таблиц не нужна для получения результата и ее удаление не изменит результата;

- каждый узел дерева снабжен информацией о стоимости (cost).

Для любопытствующих — увидеть «настоящее» дерево плана можно, установив параметр debug_print_plan (после слова «plan»).

Выполнение

Конвейер

- обход дерева от корня вниз
- данные передаются наверх — по мере поступления ли все сразу

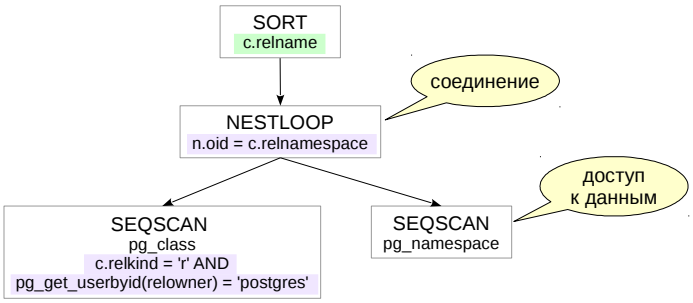
Доступ к данным

- чтение таблиц, индексов

Соединения

- всегда попарно
- важен порядок

Другие операции



На этапе выполнения построенное дерево плана (на слайде оно перерисовано так, чтобы показать основное) работает как конвейер. Выполнение начинается с корня дерева. Корневой узел (в нашем случае это операция сортировки) обращается за данными к нижестоящему узлу; получив данные, он выполняет свою функцию (сортировку) и отдает данные наверх (то есть клиенту).

Сортировка может вернуть данные, только получив всю выборку и полностью отсортировав ее. Другие узлы могут возвращать данные по мере поступления. Например, доступ к данным с помощью чтения таблицы может отдавать наверх данные по мере чтения (это позволяет быстро получить первую часть результата — например, для страничного отображения на веб-странице).

Некоторые узлы соединяют данные, полученные из разных источников. Соединение всегда осуществляется попарно, причем важен порядок соединения. Эти узлы примерно, но не совсем, соответствуют операциям соединения SQL.

Итак, чтобы разобраться с планами выполнения, нужно понять, какие существуют методы доступа к данным, какие есть способы соединения этих данных, и посмотреть некоторые другие операции. Мы будем подробно говорить об этом в следующей теме «План запроса».

Подготовленные операторы

Подготовка

`PREPARE` *имя* `(`*параметры*`) AS` *оператор*

выполняется разбор и переписывание,
а в ряде случаев — и оптимизация

дерево запроса сохраняется в памяти процесса

Выполнение

`EXECUTE` *имя* `(`*параметры*`)`

происходит оптимизация (как правило)
и собственно выполнение

гарантия невозможности внедрения SQL-кода

Если один и тот же запрос выполняется повторно, каждый раз он проходит все этапы обработки заново — все построенные деревья каждый раз выкидываются и строятся вновь. Хотелось бы сохранить подготовительную работу, особенно, если запрос выполняется быстро и, следовательно, этапы разбора, трансформации и планирования занимают существенную часть общего времени выполнения.

Для этого существует механизм подготовленных операторов (prepared statements). Выполняя команду prepare, мы сохраняем дерево запроса в памяти серверного процесса. Затем, при выполнении запроса командой execute, происходит только оптимизация и собственно выполнение.

В ряде случаев планировщик может понять, что нет смысла каждый раз оптимизировать запрос. Тогда будет сохраняться и использоваться дерево плана, что еще эффективнее. Подробнее о том, в каких случаях это происходит, мы будет говорить в теме «Статистика».

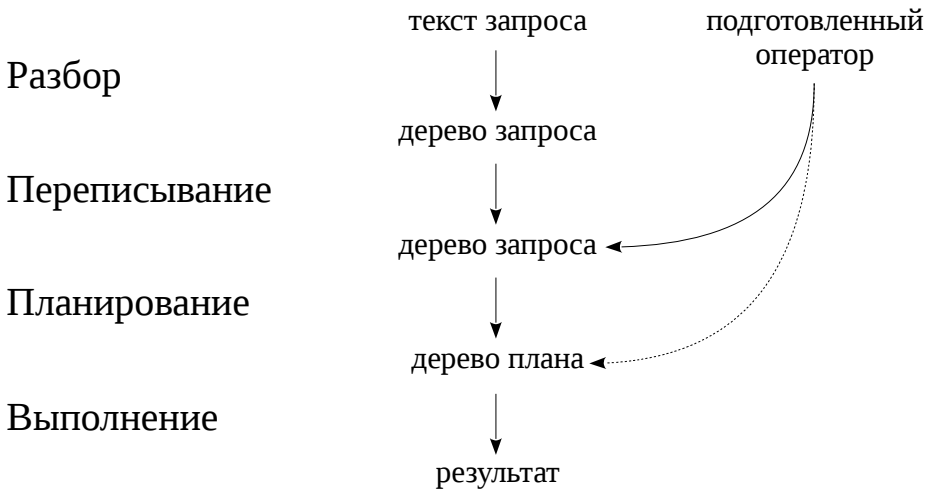
Есть и еще один повод использовать подготовленные операторы: гарантировать безопасность от внедрения SQL-кода, если входные данные для запроса получены из ненадежного источника (например, из поля ввода на веб-форме). Мы посмотрим на это в демонстрации.

http://www.postgresql.org/docs/9.5/static/sql-prepare.html

http://www.postgresql.org/docs/9.5/static/sql-execute.html

Заметим, что разобранный запрос сохраняется в памяти серверного процесса. Глобального хранилища (кэша) запросов в PostgreSQL нет.

Схема обработки запроса



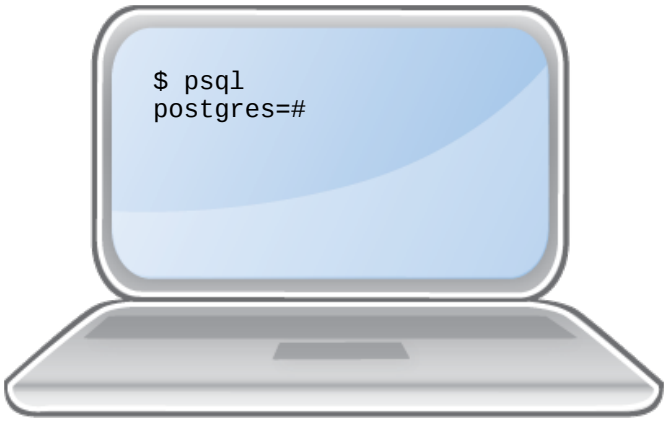
Итак, общая схема обработки запроса:

- текст запроса всегда проходит этапы разбора, переписывания, планирования.

- можно подготовить оператор — тогда запрос будет проходить только планирование (а иногда миновать даже и этот этап).

- в конце концов запрос выполняется и результат возвращается клиенту.

Демонстрация



Итоги

Обработка запроса состоит из нескольких шагов:
разбор и переписывание, планирование, выполнение

Стоит подготавливать операторы, если они:

- выполняются в сеансе несколько раз
- используют потенциально опасные данные

Время выполнения зависит от качества планирования

Практика

1. Создайте базу DB10.
2. Напишите запрос, подсчитывающий сумму чисел из таблицы с миллионом строк; посчитайте среднее время выполнения.
3. Подготовьте оператор для этого запроса; посчитайте среднее время выполнения.
4. Во сколько раз ускорилось выполнение?
5. Напишите запрос одной записи из таблицы по значению первичного ключа; посчитайте среднее время выполнения.
6. Подготовьте оператор для этого запроса; посчитайте среднее время выполнения.
7. Во сколько раз ускорилось выполнение в этом случае?

Для того, чтобы усреднить время, надо выполнить запрос несколько раз. Удобно воспользоваться языком PL/pgSQL, учитывая, что:

- динамический запрос, выполняемый командой PL/pgSQL execute (не путать с командой SQL execute!), каждый раз проходит все этапы;

- запрос SQL, встроенный в PL/pgSQL-код, выполняется с помощью подготовленных операторов.

Пример синтаксиса:

```
do $$
begin
  for i in 1..1000 loop
    execute 'select ... from ...';
  end loop;
end;
$$ language plpgsql;
```

И для подготовленного оператора:

```
do $$
declare
  res integer;
begin
  for i in 1..1000 loop
    select ... into res from ...;
  end loop;
end;
$$ language plpgsql;
```