
Базы Данных

Семинар 9

Описание семинара: План запроса.

PostgreSQL

Унифицированный сервер баз данных, имеющий единый движок – storage engine. Постгрес использует клиент-серверную модель.

При подключении к серверу клиент соединяется с процессом postmaster. В задачи этого процесса входит:

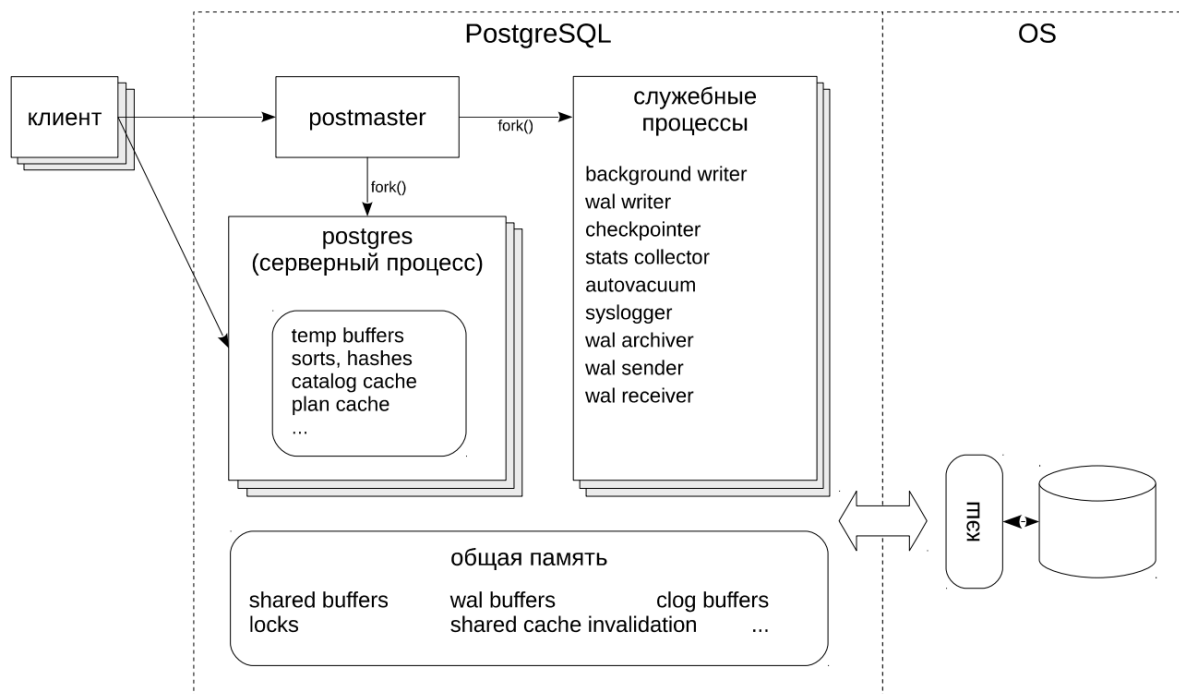
1. Порождение других процессов
2. Присмотр за созданными процессами.

Таким образом, postmaster порождает серверный процесс и дальше клиент работает уже с ним. На каждое соединение создается по серверному процессу, поэтому при большом числе соединений следует использовать пул (например, с помощью расширения pgbouncer). Postmaster также порождает ряд служебных процессов. Дерево процессов можно увидеть с помощью команды `ps fax`.

У экземпляра СУБД имеется общая для всех серверных процессов память. Большую ее часть занимает буферный кэш (shared buffers), необходимый для ускорения работы с данными на диске. Обращение к дискам происходит через операционную систему (которая тоже кэширует данные в оперативной памяти).

PostgreSQL полностью полагается на операционную систему и сам не управляет устройствами. В частности, он считает, что вызов `fsync()` гарантирует попадание данных из памяти на диск.

Кроме буферного кэша в общей памяти находится информация о блокировках и многое другое; через нее же серверные процессы общаются друг с другом. У каждого серверного процесса есть своя локальная память. В ней находится кэш каталога (часто используемая информация о базе данных), планы запросов, рабочее пространство для выполнения запросов и другое. Для работы с такими клиентскими процессами сервер использует семафоры.



Клиентский запрос проходит следующие стадии:

1. Прикладная программа устанавливает **подключение** к серверу PostgreSQL. Эта программа передаёт запрос на сервер и ждёт от него результатов.

2. На *этапе разбора запроса* сервер выполняет синтаксическую проверку запроса, переданного прикладной программой, и создаёт *дерево запроса*. Дерево разбора состоит из узлов двух типов:

- Атомы - лексические элементы следующих типов: ключевые слова (например, SELECT); имена атрибутов или отношений; константы; скобки; операторы (например, + или >); и др.
- Синтаксические категории - имена семейств, представляющих часть запроса. Заключаются в угловые скобки: <SFW>, <Condition>

Граматику языка SQL можно описать с помощью следующих правил:

```

<Query> ::= <SFW>
<Query> ::= (<Query>)
<SFW> ::= SELECT <SelList> FROM <FromList> WHERE <Condition>
<SelList> ::= <Attribute>, <SelList>
<SelList> ::= <Attribute>
<FromList> ::= <Relation>, <FromList>
<FromList> ::= <Relation>
<Condition> ::= <Condition> AND <Condition>
<Condition> ::= <Condition> OR <Condition>
<Condition> ::= NOT <Condition>
<Condition> ::= <Tuple> IN <Query>
<Condition> ::= <Attribute> = <Attribute>
<Condition> ::= <Attribute> > <Attribute>
<Condition> ::= <Attribute> < <Attribute>
<Condition> ::= <Attribute> LIKE <Pattern>
<Condition> ::= EXISTS <Query>
<Tuple> ::= <Attribute>

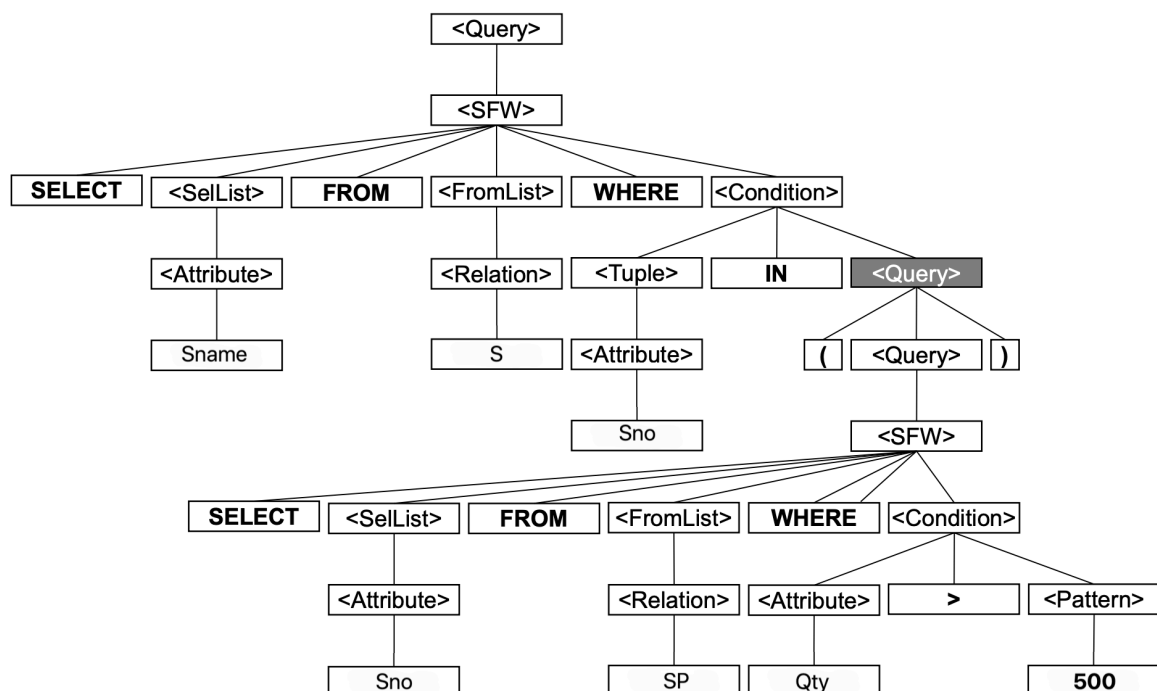
```

Рассмотрим задачу синтаксического анализа на примере запроса. Необходимо найти имена поставщиков, поставляющих хотя бы одну деталь в количестве более 500 штук (в рамках одной поставки).

```

SELECT Sname FROM S
WHERE Sno IN ( SELECT Sno FROM SP
                WHERE Qty > 500 );

```



3. Система правил принимает дерево запроса, созданное на стадии разбора, и ищет в **системных каталогах правила** для применения к этому дереву. Обнаружив подходящие правила, она выполняет преобразования, заданные в теле правил. Одно из применений системы правил заключается в реализации *представлений*. Когда выполняется запрос к представлению (т. е. *виртуальной таблице*), система правил преобразует запрос пользователя в запрос, обращающийся не к представлению, а к базовым таблицам из определения представления.

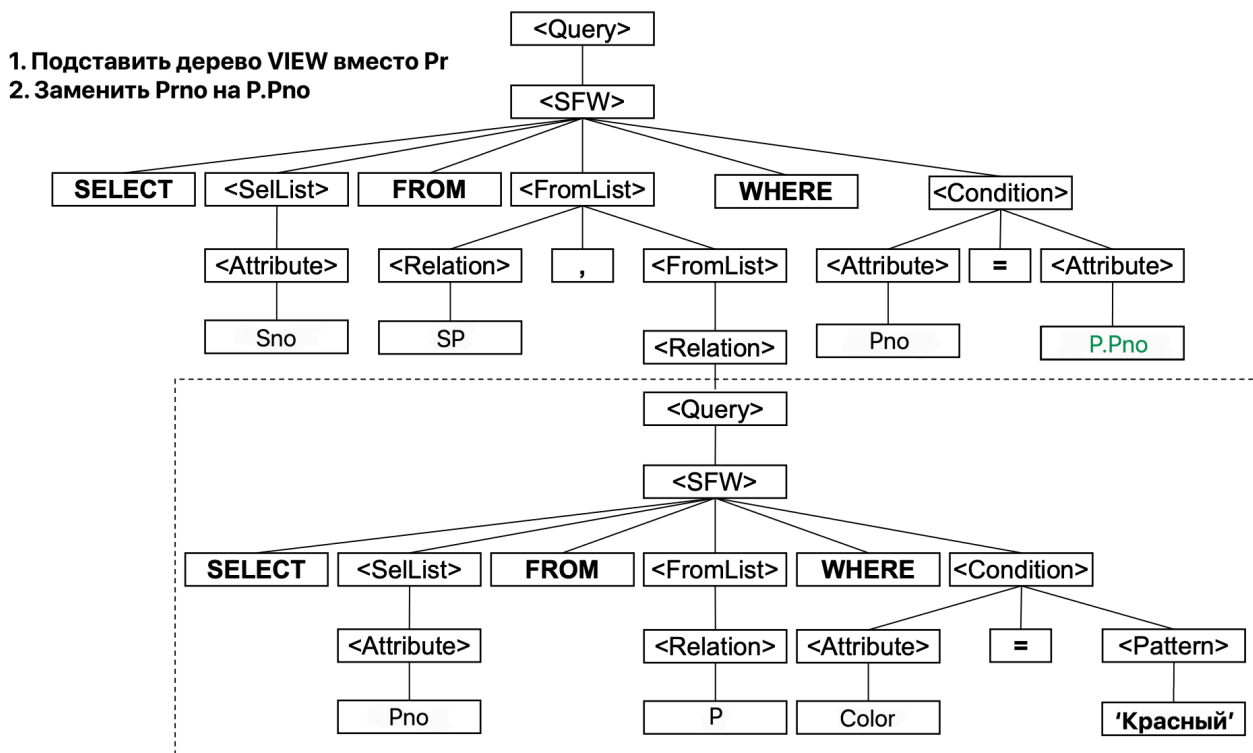
Рассмотрим пример:

Создадим представление, хранящую информацию о красных деталях:

```
CREATE VIEW Pr (Prno) AS
SELECT Rno FROM P
WHERE Color = 'Красный';
```

Теперь найдем все коды поставщиков, поставляющих хотя бы одну красную деталь:

```
SELECT Sno FROM SP, Pr
WHERE Pno= Prno;
```



4. **Планировщик/оптимизатор** принимает дерево запроса (возможно, переписанное) и создаёт *план запроса*, который будет передан *исполнителю*. Он выбирает план, сначала рассматривая все возможные варианты получения одного и того же результата. Например, если для обрабатываемого отношения создан индекс, прочитать отношение можно двумя способами. Во-первых, можно выполнить простое последовательное сканирование, а во-вторых, можно использовать индекс. Затем оценивается стоимость каждого варианта и выбирается самый дешёвый. Затем выбранный вариант разворачивается в полноценный план, который сможет использовать исполнитель.
5. Исполнитель рекурсивно проходит по *дереву плана* и **получает строки** тем способом, который указан в плане. Он сканирует отношения, обращаясь к *системе хранения*,

выполняет *сортировку и соединения*, вычисляет *условия фильтра* и, наконец, возвращает полученные строки.

План запроса

План выполнения показывает, как будут сканироваться таблицы, затрагиваемые оператором — просто последовательно, по индексу и т. д. — а если запрос связывает несколько таблиц, какой алгоритм соединения будет выбран для объединения считанных из них строк.

Наибольший интерес в выводимой информации представляет ожидаемая стоимость выполнения оператора, которая показывает, сколько, по мнению планировщика, будет выполняться этот оператор (это значение измеряется в единицах стоимости, которые не имеют точного определения, но обычно это обращение к странице на диске). Фактически выводятся два числа: стоимость запуска до выдачи первой строки и общая стоимость выдачи всех строк. Для большинства запросов важна общая стоимость, но в таких контекстах, как подзапрос в EXISTS, планировщик будет минимизировать стоимость запуска, а не общую стоимость (так как исполнение запроса всё равно завершится сразу после получения одной строки). Кроме того, если количество возвращаемых строк ограничивается предложением LIMIT, планировщик интерполирует стоимость между двумя этими числами, выбирая наиболее выгодный план.

Как вызвать план запроса в postgres:

EXPLAIN — показать план выполнения оператора

Синтаксис

EXPLAIN [(параметр [, ...])] оператор

EXPLAIN [ANALYZE] [VERBOSE] ... оператор

Здесь допускаются параметры:

- **ANALYZE** [boolean] - Выполнить команду и вывести фактическое время выполнения и другую статистику. По умолчанию этот параметр равен FALSE.
- **VERBOSE** [boolean]
- **COSTS** [boolean] - Вывести рассчитанную стоимость запуска и общую стоимость каждого узла плана, а также рассчитанное число строк и ширину каждой строки. Этот параметр по умолчанию равен TRUE.
- **BUFFERS** [boolean] – Включить информацию об использовании буфера. В частности, вывести число попаданий, блоков прочитанных, загрязненных и записанных в разделяемом и локальном буфере, а также число прочитанных и записанных временных блоков. *Попаданием* считается ситуация, когда требуемый блок уже находится в кеше и чтения с диска удастся избежать. Блоки в общем буфере содержат данные обычных таблиц и индексов, в локальном — данные временных таблиц и индексов, а временные блоки предназначены для краткосрочного использования при выполнении сортировки, хеширования, материализации и подобных узлов плана. Число *загрязнённых* блоков показывает, сколько ранее не модифицированных блоков изменила данная операция; тогда как число *записанных* блоков показывает, сколько ранее загрязнённых блоков данный серверный процесс вынес из кеша при обработке запроса. Значения, указываемые для узла верхнего уровня, включают значения всех его дочерних узлов. В текстовом формате выводятся только ненулевые значения. Этот параметр действует только в режиме ANALYZE. По умолчанию его значение равно FALSE.

- **TIMING** [boolean] - Включить в вывод фактическое время запуска и время, затраченное на каждый узел. Постоянное чтение системных часов может значительно замедлить запрос, так что если достаточно знать фактическое число строк, имеет смысл сделать этот параметр равным FALSE. Время выполнения всего оператора замеряется всегда, даже когда этот параметр выключен и на уровне узлов время не подсчитывается. Этот параметр действует только в режиме ANALYZE. По умолчанию его значение равно TRUE.
- **FORMAT** { TEXT | XML | JSON | YAML }

оператор - Любой оператор SELECT, INSERT, UPDATE, DELETE, VALUES, EXECUTE, DECLARE, CREATE TABLE AS и CREATE MATERIALIZED VIEW AS, план выполнения которого вас интересует.

Структура плана запроса представляет собой дерево *узлов плана*. Узлы на нижнем уровне дерева — это узлы сканирования, которые возвращают необработанные данные таблицы. Разным типам доступа к таблице соответствуют разные узлы: последовательное сканирование, сканирование индекса и сканирование битовой карты. Источниками строк могут быть не только таблицы, но и например, предложения VALUES и функции, возвращающие множества во FROM, и они представляются отдельными типами узлов сканирования. Если запрос требует объединения, агрегатных вычислений, сортировки или других операций с исходными строками, над узлами сканирования появляются узлы, обозначающие эти операции. И так как обычно операции могут выполняться разными способами, на этом уровне тоже могут быть узлы разных типов. В выводе команды EXPLAIN для каждого узла в дереве плана отводится одна строка, где показывается базовый тип узла плюс оценка стоимости выполнения данного узла, которую сделал для него планировщик. Если для узла выводятся дополнительные свойства, в вывод могут добавляться дополнительные строки, с отступом от основной информации узла. В самой первой строке (основной строке самого верхнего узла) выводится общая стоимость выполнения для всего плана; именно это значение планировщик старается минимизировать.

Взгляните на следующий простейший пример, просто иллюстрирующий формат вывода:

```
EXPLAIN SELECT * FROM tenk1;
```

QUERY PLAN

```
-----
Seq Scan on tenk1  (cost=0.00..458.00 rows=10000 width=244)
```

Этот запрос не содержит предложения WHERE, поэтому он должен просканировать все строки таблицы, так что планировщик выбрал план простого последовательного сканирования. Числа, перечисленные в скобках (слева направо), имеют следующий смысл:

- Приблизительная стоимость запуска. Это время, которое проходит, прежде чем начнётся этап вывода данных, например для сортирующего узла это время сортировки.
- Приблизительная общая стоимость. Она вычисляется в предположении, что узел плана выполняется до конца, то есть возвращает все доступные строки. На практике родительский узел может досрочно прекратить чтение строк дочернего (см. приведённый ниже пример с LIMIT).
- Ожидаемое число строк, которое должен вывести этот узел плана. При этом так же предполагается, что узел выполняется до конца.
- Ожидаемый средний размер строк, выводимых этим узлом плана (в байтах).

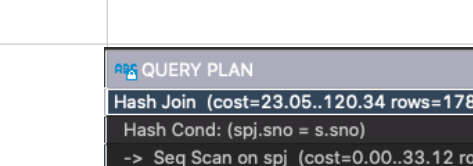
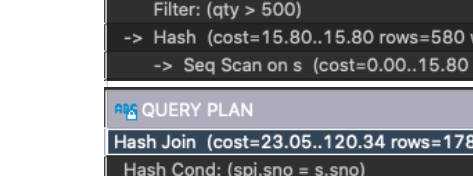
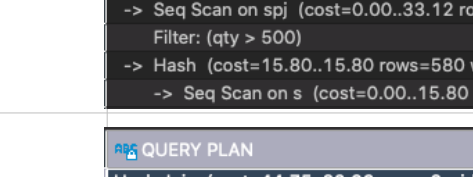
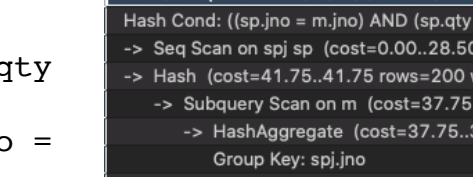
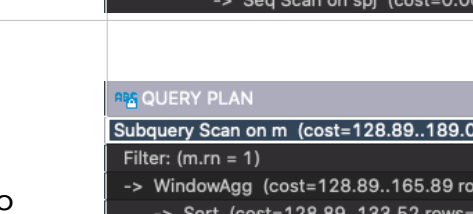
Стоимость может измеряться в произвольных единицах, определяемых параметрами планировщика. Традиционно единицей стоимости считается операция чтения страницы с

диска; то есть `seq_page_cost` обычно равен 1.0, а другие параметры задаётся относительно него. Примеры в этом разделе выполняются со стандартными параметрами стоимости.

Важно понимать, что стоимость узла верхнего уровня включает стоимость всех его потомков. Также важно осознавать, что эта стоимость отражает только те факторы, которые учитывает планировщик. В частности, она не зависит от времени, необходимого для передачи результирующих строк клиенту, хотя оно может составлять значительную часть общего времени выполнения запроса. Тем не менее планировщик игнорирует эту величину, так как он всё равно не сможет изменить её, выбрав другой план. (Мы верим в то, что любой правильный план запроса выдаёт один и тот же набор строк.)

Значение `rows` здесь имеет особенность — оно выражает не число строк, обработанных или просканированных узлом плана, а число строк, выданных этим узлом. Часто оно окажется меньше числа просканированных строк в результате применённой к узлу фильтрации по условиям `WHERE`. В идеале, на верхнем уровне это значение будет приблизительно равно числу строк, которое фактически возвращает, изменяет или удаляет запрос.

Примеры для демонстрации плана запроса:

<pre> explain select * from dbspj.spj s </pre>	 <p>QUERY PLAN</p> <p>Seq Scan on spj s (cost=0.00..28.50 rows=1850 width=16)</p>
<pre> explain SELECT Sname FROM dbSPJ.S s JOIN dbSPJ.SPJ spj ON s.Sno = spj.Sno WHERE Qty > 500; </pre>	 <p>QUERY PLAN</p> <p>Hash Join (cost=23.05..120.34 rows=1789 width=58)</p> <p>Hash Cond: (spj.sno = s.sno)</p> <p>-> Seq Scan on spj (cost=0.00..33.12 rows=617 width=4)</p> <p>Filter: (qty > 500)</p> <p>-> Hash (cost=15.80..15.80 rows=580 width=62)</p> <p>-> Seq Scan on s (cost=0.00..15.80 rows=580 width=6)</p>
<pre> explain SELECT Sname FROM dbSPJ.S s JOIN dbSPJ.SPJ spj ON s.Sno = spj.Sno AND Qty > 500; </pre>	 <p>QUERY PLAN</p> <p>Hash Join (cost=23.05..120.34 rows=1789 width=58)</p> <p>Hash Cond: (spj.sno = s.sno)</p> <p>-> Seq Scan on spj (cost=0.00..33.12 rows=617 width=4)</p> <p>Filter: (qty > 500)</p> <p>-> Hash (cost=15.80..15.80 rows=580 width=62)</p> <p>-> Seq Scan on s (cost=0.00..15.80 rows=580 width=6)</p>
<pre> explain SELECT sp.pno, sp.jno, m.qty FROM dbspj.spj sp JOIN (SELECT spj.jno, MAX(qty) AS qty FROM dbspj.spj spj GROUP BY spj.jno) m ON sp.jno = m.jno AND sp.qty = m.qty </pre>	 <p>QUERY PLAN</p> <p>Hash Join (cost=44.75..82.96 rows=9 width=12)</p> <p>Hash Cond: ((sp.jno = m.jno) AND (sp.qty = m.qty))</p> <p>-> Seq Scan on spj sp (cost=0.00..28.50 rows=1850 width=4)</p> <p>-> Hash (cost=41.75..41.75 rows=200 width=8)</p> <p>-> Subquery Scan on m (cost=37.75..41.75 rows=200 width=4)</p> <p>-> HashAggregate (cost=37.75..39.75 rows=200 width=4)</p> <p>Group Key: spj.jno</p> <p>-> Seq Scan on spj (cost=0.00..28.50 rows=1850 width=4)</p>
<pre> explain SELECT pno, jno, qty FROM (SELECT pno, jno, qty, ROW_NUMBER() OVER(PARTITION BY jno ORDER BY qty DESC) AS rn FROM dbspj.spj spj) m WHERE m.rn = 1 </pre>	 <p>QUERY PLAN</p> <p>Subquery Scan on m (cost=128.89..189.02 rows=9 width=12)</p> <p>Filter: (m.rn = 1)</p> <p>-> WindowAgg (cost=128.89..165.89 rows=1850 width=24)</p> <p>-> Sort (cost=128.89..133.52 rows=1850 width=12)</p> <p>Sort Key: spj.jno, spj.qty DESC</p> <p>-> Seq Scan on spj (cost=0.00..28.50 rows=1850 width=4)</p>