ТЕОРЕТИЧЕСКАЯ БАЗА

Одной из ключевых задач системы управления базами данных является эффективное выполнение SQL-запросов, полученных от клиента. В достижении этой цели важную роль играет работа планировщика запросов — механизма, отвечающего за анализ структуры запроса и выбор наиболее оптимального плана его выполнения.

Выделяют 4 основных этапа обработки запроса в PostgreSQL:

- разбор (parse) — представление текста запроса в виде дерева (синтаксических разбор и семантический анализ);
- переписывание (rewrite) с учётом правил — подстановка текста представления вместо его названия в дереве запроса;
- планирование (plan) — формирование множества планов выполнения запроса и выбор наиболее оптимального;
- выполнение (execute) запроса в соответствии с выбранным планом.

Самым важным этапом обработки запроса является планирование. Планировщик перебирает методы доступа к данным, порядок и способы соединения таблиц для получения множества различных вариантов планов. Чтобы выбрать наилучший, планировщик должен знать о размере таблиц и распределении данных по столбцам или, другими словами, о статистике — информации, собираемой процессом автоанализа (ANALYZE). Именно с ее помощью вычисляется стоимость (cost), на основе которой и оцениваются планы.

Оценка стоимости запроса включает стоимость чтения страниц и стоимость обработки строк. Единица стоимости — стоимость чтения одной страницы с диска (условные единицы). При вычислении стоимости оптимизатор PostgreSQL учитывает дисковый ввод-вывод (вытеснение файлов из диска в кэш и обратно) и ресурсы процессора (кроме того, что нужно загрузить страницу из диска, ее нужно прочитать и проанализировать расположение версий строк, проверить их видимость и т.д.). Стоимость плана зависит от стоимости каждого его узла, а она, в свою очередь,

напрямую зависит не только от того, что узел делает, но и от количества строк, которое им обрабатывается. Следовательно, для планировщика очень важно знать статистику, чтобы посчитать стоимость и выбрать наилучший план. Но для подсчета стоимости, необходимо знать кардинальность (rows) — итоговое количество строк, которое «отобрал» узел плана. Кардинальность узла рассчитывается с помощью селективности — доли строк, полученной из выборки (числа от 0 до 1). Вычисление кардинальности — рекурсивный процесс, то есть чтобы оценить кардинальность узла, необходимо оценить кардинальность дочерних узлов. Стоит отметить, что большинство ошибок оптимизатора связаны именно с неправильной оценкой кардинальности, поэтому важно следить за возможностью использования статистики и её обновлением .

Чтобы анализировать планы выполнения запросов, необходимо понимать информацию, которую они содержат.

Планы выполнения могут состоять из следующих основных узлов:

- метода доступа к данным (обращение к данным и их выборка):
    - Seq Scan (последовательное сканирование);
    - Index Scan (индексное сканирование);
    - Index Only Scan (сканирование только по индексу);
    - Bitmap Scan (сканирование по битовой карте);
- Соединения (механизм реализации операций JOIN, IN, EXIST):
    - Nested Loop (соединение вложенными циклами);
    - Hash Join (соединение хешированием);
    - Merge Join (соединение слиянием);
- сортировки (Sort);
- группировки (Group);
- агрегации (Aggregate).

Для каждого из узлов есть модификации для параллельного выполнения, при работе которых задачи разделяются между рабочими процессами. Однако такие планы рассмотрены не будут.

Изучим принцип работы, эффективность и особенности каждого из узлов.

Говоря об индексном сканировании, стоит отметить, что затрагиваться будут B-tree (сбалансированное ветвистое дерево) индексы, так как являются самыми распространенными на практике. При создании B-tree индекса строится упорядоченное дерево, хранящее ключи для сопоставления с идентификаторами строк таблицы. Таблица не сканируется полностью, а с помощью индекса выбираются только нужные значения. Метод Index Scan используется при высокой селективности (выборка небольшой части из таблицы). Индексное дерево поддерживает операции больше и меньше. Оно используется для ускорения доступа к данным, поддержания ограничений целостности (индексы автоматически создаются для первичных и уникальных ключей) и в некоторых случаях для сортировки данных (если в предложении ORDER BY используются проиндексированные столбцы). Недостатком такого метода доступа является возможность обращения к одной и той же табличной странице несколько раз, ведь на каждое тратятся ресурсы.

Если в запросе используются только проиндексированные данные, то применяется метод Index Only Scan. Индексы же в этом случае называют покрывающими — они «покрывают» все потребности запроса, читаются только индексные страницы без обращения к табличным. Но при таком сканировании используется карта видимости, так как в индексах не хранится информация о видимости строк. Если карта видимости не актуальна, то планировщик может решить, что все же нужно прочитать табличную страницу для проверки видимости строки. Тогда, в худшем случае, сканирование выродится до обычного Index Scan. Однако, при актуальной карте видимости, алгоритм Index Only Scan является самым эффективным методом доступа при любой селективности, так как индекс имеет меньше размер, чем вся таблица, следовательно, чтение только индекса может быть

предпочтительнее. Также как и при последовательном сканировании, при индексном сканировании данные возвращаются по мере чтения.

Следует сказать про include-индексы. Они не используются при поиске по индексу и не учитываются в ограничениях целостности. Такие индексы применяются, чтобы сделать индекс покрывающим. В них добавляются неключевые столбцы, значения которых хранятся в виде дополнительной информации в индексных записях (по ним не строится дерево индекса).

Метод Bitmap Scan состоит из двух этапов: построение битовой карты и сканирование по битовой карте. Сначала сканируется индекс (Bitmap Index Scan) и строится битовая карта, состоящая из фрагментов (если оперативной памяти не хватает, то фрагменты строятся с потерей точности, что сказывается на производительности) — в ней отмечаются те версии строк, которые удовлетворяют условию и должны быть прочитаны. Далее карта сканируется (Bitmap Heap Scan) и возвращаются результирующие строки, читая табличные страницы всего один раз. Именно поэтому Bitmap Scan становится эффективнее Index Scan при средней селективности. Узел Bitmap Heap Scan начинает работу только при полностью построенной битовой карте.

Метод Bitmap Scan так же позволяет объединять в одном запросе условия по нескольким индексам (BitmapOr и BitmapAnd). Битовая карта строится по каждому из условий (могут содержать столбцы, по которым построены разные индексы), а сканирование и обращение к табличным страницам выполняется по объединенной битовой карте сканирования.

Если данные в таблице физически упорядочены, так же как и индекс, то построение битовой карты становится бессмысленным. В этом случае даже обычное индексное сканирование будет читать страницы только один раз, так как данные уже отсортированы, что, следовательно, в некоторых случаях может являться оптимизацией. Команда CLUSTER используется для физической сортировки данных в таблице по индексу. Она работает аналогично команде VACUUM FULL, так как полностью перестраивает

таблицу и блокирует ее на время работы команды. Однако после выполнения команды CLUSTER, изменения в таблице могут ухудшить кластеризацию.

Последовательное сканирование — это чтение всех страниц файла или нескольких файлов данных таблицы. Данные при применении этого метода возвращаются клиенту по мере чтения. При понижении селективности в определенный момент Seq Scan становится более производительным, чем предварительное построение битовой карты и затем чтение таблиц, так как постоянное обращение к индексным страницам только увеличивает накладные расходы. Тем не менее, из-за того, что при индексном доступе результат возвращается отсортированным, это в некоторых случаях может увеличить его шансы даже при низкой селективности. Хотя, в целом, выбор метода доступа становится более значимым, когда количество страниц в таблице больше одной, так как эффективнее последовательное чтение страницы, чем предварительно чтение индекса.

Рассмотрим алгоритмы соединения. Самый простой метод — Nested Loop. Он используется, когда нужно соединить попарно несколько наборов строк. Соединение происходит сразу, путем перебора строк внутреннего набора во внутреннем цикле и поиска соответствия строкам внешнего набора, перебираемого внешнем циклом. Характеристики метода:

- не требует подготовительных действий;
- эффективен для небольших объемах данных, так как накладные расходы растут пропорционально росту выборки;
- важен порядок соединения, так как метод применяется, когда внешний набор данных имеет небольшой размер, и во внутреннем наборе есть эффективный доступ к строкам по индексу (либо небольшой объем строк для внутреннего набора);
- вычислительная сложность и стоимость соединения пропорциональны произведению количества строк внешнего набора на среднее количество строк одной итерации внутреннего набора;

- эффективен для запросов, где результат должен быть получен быстро;
- поддерживает соединения, как по равенству, так и по неравенству;
- модификации включают Left (Right), Semi (возвращает строки одного набора, если для них нашлось хотя бы одно соответствие в другом наборе) и Anti (возвращает строки одного набора, если для них не нашлось соответствия в другом наборе).

При соединении хешированием строится хеш-таблица (вычисляется хеш-функция для каждой строки первого набора и определяется номер корзины) (Hash), далее для каждой строки второго набора вычисляется хеш-функция и проводится сопоставление с хеш-таблицей (Hash Join) — однопроходное соединение. Двухпроходное соединение применяется, когда хеш-таблица не помещается в оперативную память, из-за чего имеет больше дополнительных расходов. Характеристики метода:

- требует подготовки — построение хеш-таблицы;
- эффективен для больших наборов строк;
- чтобы хеш-таблица поместилась в оперативную, важен порядок соединения таблиц (внутренний набор должен быть меньше внешнего, чтобы минимизировать хеш-таблицу), также в предложении SELECT следует указывать не «*», а конкретные столбцы, так как все столбцы, участвующие в запросе, попадают в хеш-таблицу;
- вычислительная сложность и стоимость пропорциональны сумме числа строк в наборах;
- характерен для запросов, в которых надо обработать большое число строк, а общая пропускная способность важнее отклика системы;
- поддерживает только эквисоединения (условия равенства);
- модификации Hash Join включают Left (Right), Semi, Anti и Full Join.

Для группировки (GROUP BY) и устранения дубликатов (DISTINCT) используются методы, схожие с методами соединения. Один из способов

заключается в построение хеш-таблицы по нужным полям и в получении из нее уникальных значений — узел HashAggregate. Если группировка происходила другим методом, то в плане выполнения запроса будет использоваться узел GroupAggregate, а для устранения дубликатов — узел Unique.

Соединение слиянием используется, когда данные уже отсортированы или могут быть получены отсортированными из нижестоящего узла плана. Метод удобен, когда по полям соединения есть индекс — сортировка не должна выполняться отдельно, так как из индекса можно получать уже отсортированные данные. В отличие от соединения хешированием, слияние без сортировки хорошо подходит для случая, когда надо быстро получить первые строки. Характеристики метода:

- не требует подготовки;
- эффективен для большого числа строк, а если не требуется начальная подготовка, то применяется и для коротких быстрых запросов;
- не важен порядок соединения таблиц;
- если предварительная сортировка не требуется, то сложность — количествоо строк, а если требуется, то сложность — NlogN + MlogM;
- поддерживает только эквисоединения.

Рассмотрим узел сортировки данных. Когда данные могут быть отсортированы в оперативной памяти, используется алгоритм быстрой сортировки (quicksort). Также может быть использована частичная пирамидальная сортировка (top-N heapsort), когда нужна только часть значений (LIMIT) или инкрементальная сортировка (incremental sort), когда данные уже отсортированы, но не по всем ключам. Если оперативной памяти недостаточно, применяется внешняя сортировка (external merge), при которой упорядоченные данные хранятся во временных сортируемых файлах. Чтобы начать возвращать данные вышестоящему узлу, сортировка должна полностью завершиться.

Если данные сортируются по индексируемому столбцу, то сортировка происходит с помощью индексов, без использования узла Sort, что эффективнее в большинстве случаев. Если сортировка происходит по нескольким столбцам, по которым есть общий индекс, то важен порядок, в котором расположены столбцы в нем. Если порядок столбцов в предложении ORDER BY нарушен, то вместо индексов будет использован узел Sort. В этом случае необходимо создание нового индекса с другим порядком столбцов. Однако, если таблица маленькая, то планировщик может принять решение не использовать индексы, а выполнять сортировку в памяти.

Узел агрегации появляется в плане выполнения, когда в запросе была использована агрегатная функция. Начальная и полная стоимость узла Aggregate практически совпадают, так как он не может начать возвращать данные раньше, чем посчитает все строки.