Авторы: Алексей Береснев, Илья Баштанов, Павел Толмачев

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

Темы

Перенос статистики

Мониторинг выполнения

Указания оптимизатору

Перенос статистики

Расширение dump_stat

Расширение dump_stat

Позволяет сохранить в файле статистику планировщика

- SQL-скрипт с командами INSERT

Восстановление статистики также использует dump_stat

- можно восстановить на другом сервере (той же основной версии)
- не требуется выполнение ANALYZE

Модуль dump_stat позволяет выгрузить и восстановить содержимое таблицы pg_statistic.

С помощью этого модуля собранную статистику планировщика можно перенести на другой сервер. При этом не придется выполнять команды ANALYZE, что сократит время простоя для больших баз данных. Еще один сценарий использования — перенос статистики с производственного сервера на тестовый стенд с меньшим набором данных, чтобы сохранить планы выполнения запросов.

Функция dump_statistic модуля выдает SQL-команды INSERT, которые можно записать в скрипт, а затем применить к совместимой базе данных.

Для успешного восстановления статистики расширение должно быть установлено и на исходном, и на целевом сервере, так как в сгенерированных командах используются вызовы функций расширения.

При обновлении версии СУБД структура таблицы pg_statistic может поменяться, поэтому скрипт может оказаться несовместимым.

Подробная информация о модуле:

https://postgrespro.ru/docs/enterprise/13/dump-stat

Перенос статистики модулем dump_stat

Создадим таблицу по образцу bookings.flights и соберем статистику.

```
=> \c demo
```

```
You are now connected to database "demo" as user "student".
```

```
=> CREATE TABLE public.flights
AS SELECT * FROM bookings.flights;
```

```
SELECT 33121
```

```
=> ANALYZE public.flights;
```

```
ANALYZE
```

```
=> SELECT attname, null_frac, avg_width, n_distinct
FROM pg_stats
WHERE tablename = 'flights' AND schemaname = 'public';
```

```
       attname       | null_frac | avg_width | n_distinct
---------------------+-----------+-----------+-------------
 flight_id           |         0 |         4 |          -1
 flight_no           |         0 |         7 |         710
 scheduled_departure |         0 |         8 |  -0.3076598
 scheduled_arrival   |         0 |         8 | -0.28685728
 departure_airport   |         0 |         4 |         104
 arrival_airport     |         0 |         4 |         104
 status              |         0 |         8 |           6
 aircraft_code       |         0 |         4 |           8
 actual_departure    |     0.493 |         8 | -0.35889617
 actual_arrival      |    0.4948 |         8 | -0.35183117
(10 rows)
```

Подключим расширение dump_stat

```
=> CREATE EXTENSION dump_stat;
```

```
CREATE EXTENSION
```

Перегруженная функция dump_statistic позволяет выгрузить статистику по всей базе данных, по строкам отношений в заданной схеме или по конкретной таблице.

Выгрузим статистику по таблице public.flights:

```
=> COPY (SELECT dump_statistic('public','flights')) TO '/tmp/flights.stat';
```

```
COPY 10
```

Теперь удалим таблицу и создадим ее заново, но с выключенной автоочисткой, чтобы исключить автоматический сбор статистики.

```
=> DROP TABLE public.flights;
```

```
DROP TABLE
```

```
=> CREATE TABLE public.flights
WITH (autovacuum_enabled = off)
AS SELECT * FROM bookings.flights;
```

```
SELECT 33121
```

Статистики пока нет:

```
=> SELECT attname, null_frac, avg_width, n_distinct
FROM pg_stats
WHERE tablename = 'flights' AND schemaname = 'public';
```

```
 attname | null_frac | avg_width | n_distinct
---------+-----------+-----------+------------
(0 rows)
```

Загрузим сохраненную ранее статистику:

```
=> \i /tmp/flights.stat
```

```
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
```

```
=> SELECT attname, null_frac, avg_width, n_distinct
FROM pg_stats WHERE tablename = 'flights' and schemaname = 'public';
```

```
       attname       | null_frac | avg_width | n_distinct
---------------------+-----------+-----------+-------------
 flight_id           |         0 |         4 |          -1
 flight_no           |         0 |         7 |         710
 scheduled_departure |         0 |         8 |  -0.3076598
 scheduled_arrival   |         0 |         8 | -0.28685728
 departure_airport   |         0 |         4 |         104
 arrival_airport     |         0 |         4 |         104
 status              |         0 |         8 |           6
 aircraft_code       |         0 |         4 |           8
 actual_departure    |     0.493 |         8 | -0.35889617
 actual_arrival      |    0.4948 |         8 | -0.35183117
(10 rows)
```

Статистика успешно загрузилась. Расширение далее не требуется.

```
=> DROP EXTENSION dump_stat;
```

```
DROP EXTENSION
```

Мониторинг выполнения

Расширение pg_query_state

Расширение pg_query_state

Информация о выполняющемся запросе

- текущее состояние выполнения
- актуальная статистика

Формат вывода как у EXPLAIN ANALYZE

Модуль pg_query_state выводит текущее состояние выполнения запроса в работающем обслуживающем процессе. В отличие от команды EXPLAIN ANALYZE, предоставляющей данные об уже выполненном запросе, модуль pg_query_state информирует о ходе выполнения запроса, который работает в настоящий момент. Формат вывода практически идентичен выводу обычной команды EXPLAIN ANALYZE.

Модуль позволяет исследовать любой обслуживающий процесс и определять его фактическое состояние. Это полезно, когда обслуживающий процесс выполняет длительный запрос и необходимо оценить оставшееся время.

Модуль показывает состояние как основного запроса, так и запросов, вызванных из функций. Для распараллеленных запросов модуль позволяет наблюдать за операциями, которые выполняет ведущий и дополнительные рабочие процессы.

Подробная информация о модуле:

https://postgrespro.ru/docs/enterprise/13/pg-query-state

Отслеживание планов запросов активных сеансов

Подключим расширение и соответствующую разделяемую библиотеку.

```
=> \c demo
```

```
You are now connected to database "demo" as user "student".
```

```
=> CREATE EXTENSION pg_query_state;
```

```
CREATE EXTENSION
```

```
=> ALTER SYSTEM SET shared_preload_libraries = 'pg_query_state';
```

```
ALTER SYSTEM
```

Необходима перезагрузка экземпляра.

```
student$ sudo systemctl restart postgrespro-ent-13.service
```

```
student$ psql demo
```

Откроем еще один сеанс:

```
student$ psql demo
```

Номер обслуживающего процесса второго сеанса:

```
=> SELECT * FROM pg_backend_pid();
```

```
 pg_backend_pid 
----------------
         258741
(1 row)
```

Запустим во втором сеансе запрос. Запрос написан таким образом, что перебирает в цикле все перелеты из таблицы ticket_flights и для каждого из них выбирает номер бронирования из таблицы tickets. В данном случае это очень неэффективный способ получения результата, но большое время выполнения запроса позволит отследить в первом сеансе ход его выполнения.

Мы выведем оценку стоимости и кардинальности (аналогичную той, что выдает команда EXPLAIN) и актуальную информацию о выполнении узлов плана.

```
=> SELECT tf.*, (SELECT t.book_ref FROM tickets t WHERE t.ticket_no = tf.ticket_no)
FROM ticket_flights tf;
```

```
=> SELECT plan FROM pg_query_state(258741, costs => true);
```

```
                                                                                 plan                                                                                 
----------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on ticket_flights tf  (cost=0.00..8845172.70 rows=1045726 width=60) (Current loop: actual rows=2515, loop number=1)                                        +
   SubPlan 1                                                                                                                                                          +
     ->  Index Scan using tickets_pkey on tickets t  (cost=0.42..8.44 rows=1 width=7) (actual rows=1 loops=2514) (Current loop: actual rows=1, loop number=2515)+
           Index Cond: (ticket_no = tf.ticket_no)
(1 row)
```

Небольшая пауза...

```
=> SELECT plan FROM pg_query_state(258741, costs => true);
```

```
                                                                                   plan                                                                                   
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on ticket_flights tf  (cost=0.00..8845172.70 rows=1045726 width=60) (Current loop: actual rows=332978, loop number=1)                                          +
   SubPlan 1                                                                                                                                                              +
     ->  Index Scan using tickets_pkey on tickets t  (cost=0.42..8.44 rows=1 width=7) (actual rows=1 loops=332977) (Current loop: actual rows=1, loop number=332978)+
           Index Cond: (ticket_no = tf.ticket_no)
(1 row)
```

Сравните для узла Seq Scan прогноз числа строк (rows) с актуальной информацией (Current loop: rows). Таким образом можно обоснованно предположить, какая часть работы уже проделана.

Вложенный узел Index Scan выполняется в цикле. Для него выводится среднее значение по прошедшим циклам (actual, по аналогии с выводом команды EXPLAIN ANALYZE) и информация о текущем, еще не завершенном цикле (Current loop).

Как и команда EXPLAIN ANALYZE, расширение позволяет выводить информацию о времени выполнения узлов и о количестве использованных буферов, а также отслеживать параллельные запросы и запросы, выполняющиеся в вызываемых функциях.

Удалим расширение и перезапустим экземпляр.

```
=> DROP EXTENSION pg_query_state;
```

```
DROP EXTENSION
```

```
=> ALTER SYSTEM RESET shared_preload_libraries;
```

```
ALTER SYSTEM
```

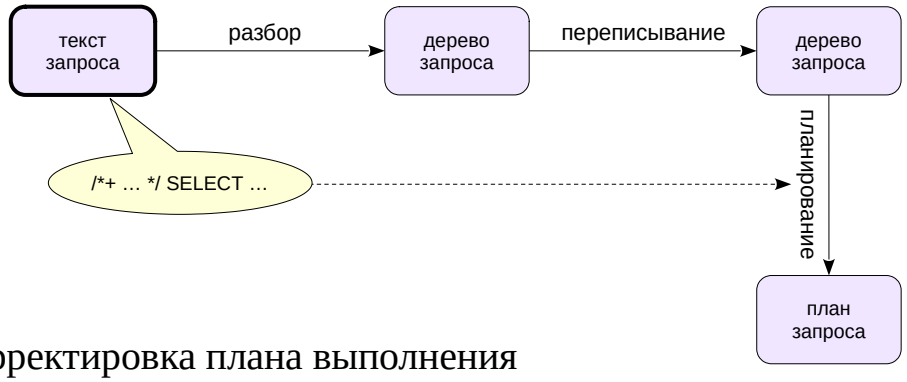
```
student$ sudo systemctl restart postgrespro-ent-13.service
```

Указания оптимизатору

Расширение pg_hint_plan

Расширение plantuner

Расширение pg_hint_plan



Корректировка плана выполнения с помощью указаний

- в комментарии специального вида в коде запроса
- в служебной таблице расширения

Модуль pg_hint_plan разработан японской компанией NTT и включен в состав Postgres Pro Enterprise. Репозиторий проекта:

https://github.com/ossc-db/pg_hint_plan

Расширение предназначено для принудительной корректировки планов выполнения с помощью указаний (hints), помещаемых непосредственно в код SQL в виде комментариев специального вида /*+ … */. В случаях, когда принудительно выбранный план выполнить нельзя, планировщик игнорирует указания.

Обработчик pg_hint_plan считывает указания только из первого блочного комментария и немедленно прекращает разбор, обнаруживая недопустимый символ. Допустимыми символами являются буквы, цифры, пробелы, подчеркивания, запятые и скобки. Комментарий с указаниями не может содержать в себе другой блочный комментарий. Указания можно размещать в нескольких строках для улучшения читаемости.

Ошибки в записи или неправильные имена указаний считаются ошибками синтаксиса и выводятся в журнал сообщений сервера. Указания с неправильными обращениями к объектам игнорируются, а в журнале помечаются как неиспользованные. При дублировании или противоречиях в указаниях будет применяться последнее указание, а в журнал выведется сообщение о дублировании.

Подробности в документации:

https://postgrespro.ru/docs/enterprise/13/pg-hint-plan

Типы указаний

Метод доступа

Способ соединения

Порядок соединений

Корректировка числа строк соединения

Параллельная обработка таблицы

Установка параметра на время планирования

Фразы указаний состоят из имени указания и последующих параметров, заключенных в скобки и разделенных пробелами.

Модуль поддерживает шесть типов указаний:

https://postgrespro.ru/docs/enterprise/13/pg-hint-plan#PG-HINT-PLAN-HINT-LIST

Методы доступа, способы соединения и другие вопросы, относящиеся к планам выполнения запросов, подробно рассматриваются в курсе QPT «Оптимизация запросов».

Таблица указаний

Таблица hint_plan.hints

- id — автоматически заполняемый номер
- norm_query_string — шаблон для выбора запросов
- application_name — имя приложения для выбора сеанса
- hint — код указания

Указания в таблице приоритетнее, чем указания в коде

- *pg_hint_plan.enable_hint_table* = on

Часто бывает затруднительно изменить код SQL в приложении для явной вставки указаний. В таких случаях указания можно внести в специальную таблицу hint_plan.hints, разрешив ее использование с помощью параметра *pg_hint_plan.enable_hint_table*.

Идентификатор указания в этой таблице (поле id) заполняется автоматически.

Столбец norm_query_string устанавливает шаблон для выбора запросов, к которым будет применяться указание. Константы в шаблоне должны быть заменены символом знака вопроса. Пробельные символы значимы.

Столбец application_name определяет, к каким сеансам будет применяться указание. Если указана пустая строка, будут выбираться все сеансы.

Столбец hint задает указания (без обрамляющей разметки комментариев).

Указания, заданные в таблице, имеют бóльший приоритет, чем указания в комментариях.

Модуль pg_hint_plan

Расширение pg_hint_plan предоставляется вместе с Postgres Pro Enterprise в виде отдельного пакета, который устанавливается средствами ОС из репозитория. В виртуальной машине, предназначенной для курса, этот пакет уже установлен.

Так как функционал расширения реализован в разделяемой библиотеке, она должна быть предварительно загружена в сеансе с помощью команды LOAD или настроена глобально в конфигурационном параметре shared_preload_libraries.

```
student$ psql demo
```

```
=> LOAD 'pg_hint_plan';
```

```
LOAD
```

Для данного запроса используется параллельный план с двумя рабочими процессами:

```
=> EXPLAIN (costs off) SELECT count(*) FROM ticket_flights;
```

```
                      QUERY PLAN
------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 2
         ->  Partial Aggregate
               ->  Parallel Seq Scan on ticket_flights
(5 rows)
```

Для управления параллелизмом применяется указание Parallel. С параметром hard оно принудительно задает количество процессов.

В этом примере мы отключаем параллельное выполнение, указывая ноль:

```
=> /*+ Parallel(ticket_flights 0 hard) */
EXPLAIN (costs off) SELECT count(*) FROM ticket_flights;
```

```
            QUERY PLAN
---------------------------------
 Aggregate
   ->  Seq Scan on ticket_flights
(2 rows)
```

Принудительно установим три рабочих процесса:

```
=> /*+ Parallel(ticket_flights 3 hard) */
EXPLAIN (costs off) SELECT count(*) FROM ticket_flights;
```

```
                      QUERY PLAN
------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 3
         ->  Partial Aggregate
               ->  Parallel Seq Scan on ticket_flights
(5 rows)
```

Параметр soft лишь меняет значение параметра max_parallel_workers_per_gather, а в остальном планировщику остается свобода выбора. В данном случае планировщик ограничился двумя процессами из-за относительного небольшого размера таблицы:

```
=> /*+ Parallel(ticket_flights 3 soft) */
EXPLAIN (costs off) SELECT count(*) FROM ticket_flights;
```

```
                      QUERY PLAN
------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 2
         ->  Partial Aggregate
               ->  Parallel Seq Scan on ticket_flights
(5 rows)
```

Указание Leading позволяет влиять на порядок соединений.

Теперь поменяем порядок соединений. Если указаны двойные скобки, то задается не только порядок, но и направление (внешняя таблица и внутренняя таблица) соединений:

```
=> /*+ Leading((t (tf f))) */
EXPLAIN (costs off)
SELECT t.passenger_name, tf.flight_id, f.flight_no
FROM tickets t, ticket_flights tf, flights f
WHERE t.ticket_no = tf.ticket_no AND tf.flight_id = f.flight_id ;
```

```
                      QUERY PLAN
------------------------------------------------------
 Hash Join
   Hash Cond: (t.ticket_no = tf.ticket_no)
   ->  Seq Scan on tickets t
   ->  Hash
         ->  Hash Join
               Hash Cond: (tf.flight_id = f.flight_id)
               ->  Seq Scan on ticket_flights tf
               ->  Hash
                     ->  Seq Scan on flights f
(9 rows)
```

Попробуем другой порядок:

```
=> /*+ Leading(((tf f) t)) */
EXPLAIN (costs off)
SELECT t.passenger_name, tf.flight_id, f.flight_no
FROM tickets t, ticket_flights tf, flights f
WHERE t.ticket_no = tf.ticket_no AND tf.flight_id = f.flight_id ;
```

```
                   QUERY PLAN
-------------------------------------------------
 Hash Join
   Hash Cond: (tf.ticket_no = t.ticket_no)
   ->  Hash Join
         Hash Cond: (tf.flight_id = f.flight_id)
         ->  Seq Scan on ticket_flights tf
         ->  Hash
               ->  Seq Scan on flights f
   ->  Hash
         ->  Seq Scan on tickets t
(9 rows)
```

Часто достаточно указать лишь несколько начальных таблиц, оставив планировщику свободу выбора для остальных соединений. В этом примере планировщик начнет с соединения таблиц f и tf, и поскольку пара помещена в одинарные скобки, сможет сам выбрать направление соединения:

```
=> /*+ Leading(f tf) */
EXPLAIN (costs off)
SELECT t.passenger_name, tf.flight_id, f.flight_no
FROM tickets t, ticket_flights tf, flights f
WHERE t.ticket_no = tf.ticket_no AND tf.flight_id = f.flight_id ;
```

```
                    QUERY PLAN
--------------------------------------------------
 Nested Loop
   ->  Hash Join
         Hash Cond: (tf.flight_id = f.flight_id)
         ->  Seq Scan on ticket_flights tf
         ->  Hash
               ->  Seq Scan on flights f
   ->  Index Scan using tickets_pkey on tickets t
         Index Cond: (ticket_no = tf.ticket_no)
(8 rows)
```

В предыдущих примерах для добавления указаний пришлось редактировать тексты запросов. Если это невозможно, удобно использовать таблицу hint_plan.hints, она создается при добавлении расширения в базу данных.

```
=> CREATE EXTENSION pg_hint_plan;
```

```
CREATE EXTENSION
```

Возьмем запрос с индексным доступом:

```
=> EXPLAIN (costs off)
SELECT * FROM bookings WHERE book_ref = 'CDE08B';
```

```
                QUERY PLAN
-------------------------------------------
 Index Scan using bookings_pkey on bookings
   Index Cond: (book_ref = 'CDE08B'::bpchar)
(2 rows)
```

Поместим в таблицу указаний требование использовать последовательное сканирование.

Константы, фигурирующие в целевом запросе, в столбце norm_query_string таблицы hint_plan.hints должны заменяться знаками «?». В остальном текст запроса должен посимвольно совпадать, включая переводы строк:

```
=> INSERT INTO hint_plan.hints (norm_query_string, application_name, hints)
VALUES (
  E'EXPLAIN (costs off)\nSELECT * FROM bookings WHERE book_ref = ?;',
  'psql',
  'SeqScan(bookings)'
);
```

```
INSERT 0 1
```

Включим использование таблицы указаний:

```
=> SET pg_hint_plan.enable_hint_table TO on;
```

```
SET
```

Теперь план запроса содержит последовательное сканирование:

```
=> EXPLAIN (costs off)
SELECT * FROM bookings WHERE book_ref = 'CDE08B';
```

```
                 QUERY PLAN
---------------------------------------------
 Gather
   Workers Planned: 1
   ->  Parallel Seq Scan on bookings
         Filter: (book_ref = 'CDE08B'::bpchar)
(4 rows)
```

Отключим расширение.

```
=> DROP EXTENSION pg_hint_plan;
```

```
DROP EXTENSION
```

Расширение plantuner

Исключает индексы из рассмотрения оптимизатором

plantuner.disable_index
plantuner.enable_index

Улучшает статистику для таблиц, имеющих страницы только в буферном кеше

plantuner.fix_empty_table

Расширение plantuner позволяет исключить из рассмотрения оптимизатора определенные индексы (pg_hint_plan не дает такой возможности). Оптимизатор не будет рассматривать планы, которые используют индексы, указанные в параметре *plantuner.disable_index*. Однако если имя индекса указать в параметре *plantuner.enable_index*, он все же будет доступен оптимизатору.

Если таблица создана недавно, ее страницы находятся только в буферном кеше. В таком случае оптимизатор, оценивая число страниц и строк в таблице, иногда ошибается в бóльшую сторону. Указание *plantuner.fix_empty_table* = on обнуляет эти оценки.

https://postgrespro.ru/docs/enterprise/13/plantuner

Модуль plantuner

Пусть таблица ticket_flights имеет индексы по стоимости билета и по классу обслуживания:

```
student$ psql demo
```

```
=> CREATE INDEX ticket_flights_amount_idx ON ticket_flights(amount);
```

```
CREATE INDEX
```

```
=> CREATE INDEX ticket_flights_fare_conditions_idx ON ticket_flights(fare_conditions);
```

```
CREATE INDEX
```

Следующий запрос будет использовать индекс по стоимости для любого класса обслуживания:

```
=> \set fare Business
```

```
=> EXPLAIN (costs off)
SELECT avg(amount)
FROM ticket_flights
WHERE fare_conditions=:'fare' and amount>50000;
```

```
                          QUERY PLAN
-------------------------------------------------------------
 Aggregate
   ->  Bitmap Heap Scan on ticket_flights
         Recheck Cond: (amount > '50000'::numeric)
         Filter: ((fare_conditions)::text = 'Business'::text)
         ->  Bitmap Index Scan on ticket_flights_amount_idx
               Index Cond: (amount > '50000'::numeric)
(6 rows)
```

```
=> \set fare Economy
```

```
=> EXPLAIN (costs off)
SELECT avg(amount)
FROM ticket_flights
WHERE fare_conditions=:'fare' and amount>50000;
```

```
                         QUERY PLAN
------------------------------------------------------------
 Aggregate
   ->  Bitmap Heap Scan on ticket_flights
         Recheck Cond: (amount > '50000'::numeric)
         Filter: ((fare_conditions)::text = 'Economy'::text)
         ->  Bitmap Index Scan on ticket_flights_amount_idx
               Index Cond: (amount > '50000'::numeric)
(6 rows)
```

Расширение plantuner позволяет исключить из рассмотрения индекс по стоимости, сохранив при этом возможность выбора метода доступа:

```
=> LOAD 'plantuner';
```

```
LOAD
```

```
=> SET plantuner.disable_index='ticket_flights_amount_idx';
```

```
SET
```

Если в запросе есть предикат с высокой селективностью, буден выбран другой индекс:

```
=> \set fare Business
```

```
=> EXPLAIN (costs off)
SELECT avg(amount)
FROM ticket_flights
WHERE fare_conditions=:'fare' and amount>50000;
```

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Aggregate
   ->  Bitmap Heap Scan on ticket_flights
         Recheck Cond: ((fare_conditions)::text = 'Business'::text)
         Filter: (amount > '50000'::numeric)
         ->  Bitmap Index Scan on ticket_flights_fare_conditions_idx
               Index Cond: ((fare_conditions)::text = 'Business'::text)
(6 rows)
```

А если селективность низкая, планировщик выбирает полное сканирование:

```
=> \set fare Economy
```

```
=> EXPLAIN (costs off)
SELECT avg(amount)
FROM ticket_flights
WHERE fare_conditions=:'fare' and amount>50000;
```

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Finalize Aggregate
   ->  Gather
         Workers Planned: 2
         ->  Partial Aggregate
               ->  Parallel Seq Scan on ticket_flights
                     Filter: ((amount > '50000'::numeric) AND ((fare_conditions)::text = 'Economy'::text))
(6 rows)
```

Следует понимать, что указания оптимизатору, как правило, дают эффект лишь в частных случаях: при определенных значениях констант и определенном состоянии данных в таблицах.

Итоги

Модуль dump_stat позволяет выгружать и загружать статистику без необходимости выполнять ANALYZE

Расширение pg_query_state помогает отслеживать ход выполнения долгих запросов

Модули pg_hint_plan и plantuner позволяют управлять планами запросов

Практика

1. В базе данных demo выполните запрос с соединением таблиц tickets и ticket_flights по критерию равенства значений столбцов ticket_no. Проверьте, что соединение выполняется с помощью слияния merge join.

2. Используя расширение pg_hint_plan, заставьте планировщик выбрать соединение хешированием hash join.

3. С помощью этого же расширения добейтесь, чтобы для доступа к таблице ticket_flights использовалось сканирование по битовой карте.

4. Подключите расширение plantuner и запретите использовать индекс tickets_pkey при выполнении запроса из пункта 3.

1. Используйте следующий запрос:

```
EXPLAIN (analyze, summary off, timing off, costs off)
SELECT t.ticket_no, t.passenger_name, tf.flight_id, tf.amount
FROM tickets t JOIN ticket_flights tf ON t.ticket_no =
tf.ticket_no
ORDER BY t.ticket_no
```

2. Загрузите библиотеку командой `LOAD 'pg_hint_plan'`.

Используйте указание `/*+ HashJoin(t tf) */`.

3. Используйте указание `/*+ BitmapScan(t) */`.

4. Загрузите библиотеку командой `LOAD 'plantuner'`. Запретите использовать индекс с помощью команды:

```
SET plantuner.disable_index='tickets_pkey';
```

1. Запрос с соединением слиянием

```
=> \c demo
```

```
You are now connected to database "demo" as user "student".
```

Получим план запроса с соединением таблиц tickets и ticket_flights.

```
=> EXPLAIN (analyze, summary off, timing off, costs off)
SELECT t.ticket_no, t.passenger_name, tf.flight_id, tf.amount
FROM tickets t JOIN ticket_flights tf ON t.ticket_no = tf.ticket_no
ORDER BY t.ticket_no;
```

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------------
 Merge Join (actual rows=1045726 loops=1)
   Merge Cond: (t.ticket_no = tf.ticket_no)
   ->  Index Scan using tickets_pkey on tickets t (actual rows=366733 loops=1)
   ->  Index Scan using ticket_flights_pkey on ticket_flights tf (actual rows=1045726 loops=1)
(4 rows)
```

Соединение выполнено слиянием - merge join.

---

2. Указание выполнить слияние хешированием

Загрузим динамическую библиотеку pg_hint_plan:

```
=> LOAD 'pg_hint_plan';
```

```
LOAD
```

Используем указание HashJoin:

```
=> EXPLAIN (analyze, summary off, timing off, costs off)
/*+ HashJoin(t tf) */
SELECT t.ticket_no, t.passenger_name, tf.flight_id, tf.amount
FROM tickets t JOIN ticket_flights tf ON t.ticket_no = tf.ticket_no
ORDER BY t.ticket_no;
```

```
                                     QUERY PLAN
-------------------------------------------------------------------------------------
 Gather Merge (actual rows=1045726 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Sort (actual rows=348575 loops=3)
         Sort Key: t.ticket_no
         Sort Method: external merge  Disk: 17816kB
         Worker 0:  Sort Method: external merge  Disk: 17832kB
         Worker 1:  Sort Method: external merge  Disk: 17592kB
         ->  Parallel Hash Join (actual rows=348575 loops=3)
               Hash Cond: (tf.ticket_no = t.ticket_no)
               ->  Parallel Seq Scan on ticket_flights tf (actual rows=348575 loops=3)
               ->  Parallel Hash (actual rows=122244 loops=3)
                     Buckets: 65536  Batches: 8  Memory Usage: 3488kB
                     ->  Parallel Seq Scan on tickets t (actual rows=122244 loops=3)
(14 rows)
```

---

3. Указание использовать сканирование по битовой карте

Используем указание BitmapScan:

```
=> EXPLAIN (analyze, summary off, timing off, costs off)
/*+ BitmapScan(t) */
SELECT t.ticket_no, t.passenger_name, tf.flight_id, tf.amount
FROM tickets t JOIN ticket_flights tf ON t.ticket_no = tf.ticket_no
ORDER BY t.ticket_no;
```

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Gather Merge (actual rows=1045726 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Sort (actual rows=348575 loops=3)
         Sort Key: t.ticket_no
         Sort Method: external merge  Disk: 17808kB
         Worker 0:  Sort Method: external merge  Disk: 17632kB
         Worker 1:  Sort Method: external merge  Disk: 17808kB
         ->  Nested Loop (actual rows=348575 loops=3)
               ->  Parallel Seq Scan on ticket_flights tf (actual rows=348575 loops=3)
               ->  Bitmap Heap Scan on tickets t (actual rows=1 loops=1045726)
                     Recheck Cond: (ticket_no = tf.ticket_no)
                     Heap Blocks: exact=349622
                     ->  Bitmap Index Scan on tickets_pkey (actual rows=1 loops=1045726)
                           Index Cond: (ticket_no = tf.ticket_no)
(15 rows)
```

---

4. Запрет использования индекса

Загрузим динамическую библиотеку plantuner:

```
=> LOAD 'plantuner';
```

```
LOAD
```

Запретим использовать индекс tickets_pkey:

```
=> SET plantuner.disable_index='tickets_pkey';
```

```
SET
```

Получим план:

```
=> EXPLAIN (analyze, summary off, timing off, costs off)
SELECT t.ticket_no, t.passenger_name, tf.flight_id, tf.amount
FROM tickets t JOIN ticket_flights tf ON t.ticket_no = tf.ticket_no
ORDER BY t.ticket_no;
```

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Merge Join (actual rows=1045726 loops=1)
   Merge Cond: (tf.ticket_no = t.ticket_no)
   ->  Index Scan using ticket_flights_pkey on ticket_flights tf (actual rows=1045726 loops=1)
   ->  Materialize (actual rows=1045725 loops=1)
         ->  Sort (actual rows=366733 loops=1)
               Sort Key: t.ticket_no
               Sort Method: external merge  Disk: 14408kB
               ->  Seq Scan on tickets t (actual rows=366733 loops=1)
(8 rows)
```