

# Статистика

## Расширенная статистика



16

### Авторские права

© Postgres Professional, 2019–2024

Авторы: Егор Рогов, Павел Лузанов, Павел Толмачев, Илья Баштанов

Фото: Олег Бартунов (монастырь Пху и пик Бхрикути, Непал)

### 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.

Функциональная зависимость

Наиболее частые комбинации значений

Число уникальных комбинаций значений

Статистика по выражениям

## Включает

- многовариантную статистику (по нескольким столбцам)
- статистику по выражениям

## Объект базы данных, создается вручную

- хранится в `pg_statistic_ext` и `pg_statistic_ext_data`
- представления `pg_stats_ext` и `pg_stats_ext_exprs`

После создания статистика собирается автоматически

Базовой статистики, которая собирается автоматически, может не хватать для точных оценок кардинальности и селективности.

PostgreSQL позволяет администратору вручную определить, какая дополнительная — расширенная — статистика требуется. Можно собрать статистику, охватывающую не один столбец, а сразу несколько (многовариантная статистика), или статистику по произвольному выражению.

Заметьте, что базовая статистика собирается для таблиц и их столбцов, но не для индексов (за исключением индексов по выражениям). Поэтому и индекс, построенный по нескольким столбцам, не приводит сам по себе к появлению многовариантной статистики.

Расширенная статистика создается командой `CREATE STATISTICS`. После того, как объект создан, соответствующая статистика будет собираться автоматически в фоновом режиме или командой `ANALYZE`.

<https://postgrespro.ru/docs/postgresql/16/planner-stats#PLANNER-STATS-EXTENDED>

Собранная информация хранится в таблицах `pg_statistic_ext` и `pg_statistic_ext_data`; доступная пользователю статистика отображается в представлениях `pg_stats_ext` и `pg_stats_ext_exprs`.

## Функциональная зависимость (dependencies)

значение одного столбца определяет значение другого столбца  
статистика улучшает оценку селективности условий

city	index
Самара	443000
Казань	420000
Нижний Новгород	603000
Великий Новгород	173000

Существует несколько видов многовариантной статистики (то есть статистики по нескольким столбцам таблицы), которые можно указать при создании объекта расширенной статистики.

*Функциональная зависимость между столбцами* показывает, насколько данные в одном столбце определяются значением другого столбца.

В примере на слайде почтовый индекс однозначно определяет город, поэтому селективность условия `city = 'Самара' and index = '443000'` определяется селективностью предиката `index = '443000'`. Такие предикаты, селективность которых нельзя рассчитывать отдельно друг от друга, называются *коррелированными*.

<https://postgrespro.ru/docs/postgresql/16/planner-stats#PLANNER-STATS-EXTENDED-FUNCTIONAL-DEPS>

## Функциональные зависимости

Рассмотрим запрос с двумя условиями:

```
=> SELECT count(*)
FROM flights
WHERE flight_no = 'PG0007' AND departure_airport = 'VK0';

count
-----
    396
(1 row)
```

Оценка оказывается сильно заниженной:

```
=> EXPLAIN
SELECT * FROM flights
WHERE flight_no = 'PG0007' AND departure_airport = 'VK0';

QUERY PLAN
```

```
-----
Bitmap Heap Scan on flights (cost=10.49..816.84 rows=14 width=63)
  Recheck Cond: (flight_no = 'PG0007'::bpchar)
  Filter: (departure_airport = 'VK0'::bpchar)
  -> Bitmap Index Scan on flights_flight_no_scheduled_departure_key (cost=0.00..10.49
rows=276 width=0)
    Index Cond: (flight_no = 'PG0007'::bpchar)
(5 rows)
```

Причина в том, что планировщик полагается на независимость предикатов и считает итоговую селективность как произведение селективностей предикатов. Это хорошо видно в приведенном плане: оценка в узле Bitmap Index Scan (условие на flight\_no) верная, а после фильтрации в узле Bitmap Heap Scan (условие на departure\_airport) — заниженная.

Однако мы понимаем, что номер рейса однозначно определяет аэропорт отправления: фактически, второе условие избыточно (конечно, считая, что аэропорт указан правильно).

Это можно объяснить планировщику с помощью статистики по функциональной зависимости:

```
=> CREATE STATISTICS (dependencies)
ON flight_no, departure_airport FROM flights;
```

```
CREATE STATISTICS
```

```
=> ANALYZE flights;
```

```
ANALYZE
```

Собранная статистика хранится в следующем виде:

```
=> SELECT dependencies
FROM pg_stats_ext
WHERE statistics_name = 'flights_flight_no_departure_airport_stat';

dependencies
-----
{"2 => 5": 1.000000, "5 => 2": 0.011200}
(1 row)
```

Сначала идут порядковые номера атрибутов, а после двоеточия — коэффициент зависимости.

```
=> EXPLAIN
SELECT * FROM flights
WHERE flight_no = 'PG0007' AND departure_airport = 'VK0';
```

# QUERY PLAN

```

-----
Bitmap Heap Scan on flights (cost=10.56..816.91 rows=276 width=63)
  Recheck Cond: (flight_no = 'PG0007'::bpchar)
  Filter: (departure_airport = 'VK0'::bpchar)
-> Bitmap Index Scan on flights_flight_no_scheduled_departure_key (cost=0.00..10.49
rows=276 width=0)
    Index Cond: (flight_no = 'PG0007'::bpchar)
(5 rows)

```

Теперь оценка улучшилась.

Команда \d показывает объекты расширенной статистики для конкретной таблицы:

=> \d flights

```

                                Table "bookings.flights"
    Column          |          Type          | Collation | Nullable |
-----+-----+-----+-----+-----
Default
-----+-----+-----+-----+-----
 flight_id         | integer                |           | not null |
nextval('flights_flight_id_seq'::regclass)
 flight_no         | character(6)           |           | not null |
 scheduled_departure | timestamp with time zone |           | not null |
 scheduled_arrival  | timestamp with time zone |           | not null |
 departure_airport  | character(3)           |           | not null |
 arrival_airport    | character(3)           |           | not null |
 status            | character varying(20)   |           | not null |
 aircraft_code      | character(3)           |           | not null |
 actual_departure    | timestamp with time zone |           |          |
 actual_arrival     | timestamp with time zone |           |          |
Indexes:
    "flights_pkey" PRIMARY KEY, btree (flight_id)
    "flights_flight_no_scheduled_departure_key" UNIQUE CONSTRAINT, btree (flight_no,
scheduled_departure)
Check constraints:
    "flights_check" CHECK (scheduled_arrival > scheduled_departure)
    "flights_check1" CHECK (actual_arrival IS NULL OR actual_departure IS NOT NULL AND
actual_arrival IS NOT NULL AND actual_arrival > actual_departure)
    "flights_status_check" CHECK (status::text = ANY (ARRAY['On Time'::character
varying::text, 'Delayed'::character varying::text, 'Departed'::character varying::text,
'Arrived'::character varying::text, 'Scheduled'::character varying::text,
'Cancelled'::character varying::text]))
Foreign-key constraints:
    "flights_aircraft_code_fkey" FOREIGN KEY (aircraft_code) REFERENCES
aircrafts_data(aircraft_code)
    "flights_arrival_airport_fkey" FOREIGN KEY (arrival_airport) REFERENCES
airports_data(airport_code)
    "flights_departure_airport_fkey" FOREIGN KEY (departure_airport) REFERENCES
airports_data(airport_code)
Referenced by:
    TABLE "ticket_flights" CONSTRAINT "ticket_flights_flight_id_fkey" FOREIGN KEY
(flight_id) REFERENCES flights(flight_id)
Statistics objects:
    "bookings.flights_flight_no_departure_airport_stat" (dependencies) ON flight_no,
departure_airport FROM flights

```

В разделе Statistics objects показаны имена, столбцы и нестандартные целевые значения объектов статистики.

## Наиболее частые комбинации значений (mcv)

как `pg_stats.most_common_vals/freqs`, но для нескольких столбцов статистика улучшает оценку селективности условий

city	river
Самара	Волга
Казань	Волга
Нижний Новгород	Ока
Нижний Новгород	Волга
Великий Новгород	Волхов

*Список наиболее частых комбинаций значений* позволяет сохранить несколько комбинаций значений и их частоту.

На слайде показаны возможные пары город — река. Очевидно, что предикаты со столбцами `city` и `river` коррелированы, но, в отличие от предыдущего примера, ни один из этих столбцов не определяет другой (между городами и реками отношение «многие ко многим»). В этом случае статистика по функциональным зависимостям не поможет улучшить оценки.

<https://postgrespro.ru/docs/postgresql/16/planner-stats#PLANNER-STATS-EXTENDED-N-DISTINCT-COUNTS>

## Частые комбинации

Столбцы могут быть коррелированы, но не всегда между ними есть прямая функциональная зависимость. Выполним такой запрос:

```
=> EXPLAIN (analyze, timing off, summary off)
SELECT * FROM flights
WHERE departure_airport = 'LED' AND aircraft_code = '321';
```

QUERY PLAN

```
-----
Gather  (cost=1000.00..5593.89 rows=740 width=63) (actual rows=5148 loops=1)
  Workers Planned: 1
  Workers Launched: 1
  -> Parallel Seq Scan on flights  (cost=0.00..4519.89 rows=435 width=63) (actual
rows=2574 loops=2)
    Filter: ((departure_airport = 'LED'::bpchar) AND (aircraft_code = '321'::bpchar))
    Rows Removed by Filter: 104860
(6 rows)
```

Планировщик ошибается в несколько раз. Учет функциональной зависимости недостаточно исправит ситуацию:

```
=> CREATE STATISTICS (dependencies)
ON departure_airport, aircraft_code FROM flights;

CREATE STATISTICS

=> ANALYZE flights;

ANALYZE

=> EXPLAIN
SELECT * FROM flights
WHERE departure_airport = 'LED' AND aircraft_code = '321';
```

QUERY PLAN

```
-----
Gather  (cost=1000.00..5702.69 rows=1828 width=63)
  Workers Planned: 1
  -> Parallel Seq Scan on flights  (cost=0.00..4519.89 rows=1075 width=63)
    Filter: ((departure_airport = 'LED'::bpchar) AND (aircraft_code = '321'::bpchar))
(4 rows)
```

В этом случае можно добавить расширенную статистику по частым комбинациям значений нескольких столбцов:

```
=> DROP STATISTICS flights_departure_airport_aircraft_code_stat;
```

DROP STATISTICS

```
=> CREATE STATISTICS (mcv)
ON departure_airport, aircraft_code FROM flights;
```

CREATE STATISTICS

```
=> ANALYZE flights;
```

ANALYZE

```
=> EXPLAIN
SELECT * FROM flights
WHERE departure_airport = 'LED' AND aircraft_code = '321';
```

QUERY PLAN

```
-----
Seq Scan on flights  (cost=0.00..5847.00 rows=5436 width=63)
  Filter: ((departure_airport = 'LED'::bpchar) AND (aircraft_code = '321'::bpchar))
(2 rows)
```

Теперь оценка улучшилась.

Количество собираемых наиболее частых значений определяется параметром `default_statistics_target`. Его можно



задать для конкретной статистики:

```
=> ALTER STATISTICS flights_departure_airport_aircraft_code_stat
SET STATISTICS 300;
```

ALTER STATISTICS

```
=> ANALYZE flights;
```

ANALYZE

```
=> EXPLAIN (analyze, timing off, summary off)
SELECT * FROM flights
WHERE departure_airport = 'LED' AND aircraft_code = '321';
```

#### QUERY PLAN

```
-----
Seq Scan on flights (cost=0.00..5847.00 rows=5171 width=63) (actual rows=5148 loops=1)
  Filter: ((departure_airport = 'LED'::bpchar) AND (aircraft_code = '321'::bpchar))
  Rows Removed by Filter: 209719
(3 rows)
```

Оценка кардинальности еще немного улучшилась.

Однако это не привело к дальнейшему улучшению плана, поэтому увеличение ориентира статистики в данном случае вряд ли оправдано.

Статистику по наиболее частым комбинациям можно посмотреть так:

```
=> SELECT values, frequency
FROM pg_statistic_ext
  JOIN pg_statistic_ext_data ON oid = stxoid,
  pg_mcv_list_items(stxdmccv) m
WHERE stxname = 'flights_departure_airport_aircraft_code_stat'
LIMIT 10;
```

values	frequency
{SV0,SU9}	0.029755555555555556
{DME,SU9}	0.029388888888888888
{DME,CR2}	0.024544444444444445
{LED,321}	0.024066666666666667
{VK0,CR2}	0.021844444444444444
{SV0,CR2}	0.018555555555555554
{BZK,SU9}	0.018266666666666667
{KJA,CN1}	0.014422222222222222
{VK0,SU9}	0.013866666666666666
{DME,CN1}	0.013344444444444445

(10 rows)

## Число уникальных комбинаций значений (ndistinct)

как `pg_stats.ndistinct`, но для нескольких столбцов

статистика улучшает оценку кардинальности для группировки

city	region	index
Самара	Самарская обл.	443000
Казань	Респ. Татарстан	420000
Нижний Новгород	Нижегородская обл.	603000
Великий Новгород	Новгородская обл.	173000

*Число уникальных комбинаций значений* позволяет улучшить оценку кардинальности при группировке по нескольким столбцам.

В примере на слайде число возможных комбинаций всех полей, очевидно, нельзя определить, просто перемножив количество уникальных значений для каждого столбца по отдельности.

<https://postgrespro.ru/docs/postgresql/16/planner-stats#PLANNER-STATS-EXTENDED-N-DISTINCT-COUNTS>

## Уникальные комбинации

Другая ситуация, в которой планировщик ошибается с оценкой, связана с группировкой. Количество пар аэропортов, связанных прямыми рейсами, ограничено:

```
=> SELECT count(*) FROM (
      SELECT DISTINCT departure_airport, arrival_airport FROM flights
    ) t;

 count
-----
      618
(1 row)
```

Но планировщик не знает об этом:

```
=> EXPLAIN
SELECT DISTINCT departure_airport, arrival_airport FROM flights;

              QUERY PLAN
-----
HashAggregate  (cost=5847.01..5955.16 rows=10816 width=8)
  Group Key: departure_airport, arrival_airport
    -> Seq Scan on flights  (cost=0.00..4772.67 rows=214867 width=8)
(3 rows)
```

Расширенная статистика позволяет исправить и эту оценку (если не указать вид статистики, в создаваемый объект будут включены все поддерживаемые виды):

```
=> CREATE STATISTICS
ON departure_airport, arrival_airport FROM flights;

CREATE STATISTICS

=> ANALYZE flights;

ANALYZE

=> EXPLAIN
SELECT DISTINCT departure_airport, arrival_airport FROM flights;

              QUERY PLAN
```

```
-----
Unique  (cost=5616.51..5621.15 rows=618 width=8)
  -> Sort  (cost=5616.51..5618.06 rows=618 width=8)
      Sort Key: departure_airport, arrival_airport
        -> Gather  (cost=5519.88..5587.86 rows=618 width=8)
            Workers Planned: 1
              -> HashAggregate  (cost=4519.88..4526.06 rows=618 width=8)
                  Group Key: departure_airport, arrival_airport
                    -> Parallel Seq Scan on flights  (cost=0.00..3887.92 rows=126392
width=8)
(8 rows)
```

Статистику по уникальным комбинациям можно увидеть так:

```
=> SELECT n_distinct
FROM pg_stats_ext
WHERE statistics_name = 'flights_departure_airport_arrival_airport_stat';

 n_distinct
-----
 {"5, 6": 618}
(1 row)
```

Посмотреть список всех объектов расширенной статистики можно командой \dX:

```
=> \x \dX \x
```

Expanded display is on.

List of extended statistics

```
-[ RECORD 1 ]+-----
Schema      | bookings
Name        | flights_departure_airport_aircraft_code_stat
Definition  | departure_airport, aircraft_code FROM flights
Ndistinct   |
Dependencies|
MCV         | defined
-[ RECORD 2 ]+-----
Schema      | bookings
Name        | flights_departure_airport_arrival_airport_stat
Definition  | departure_airport, arrival_airport FROM flights
Ndistinct   | defined
Dependencies| defined
MCV         | defined
-[ RECORD 3 ]+-----
Schema      | bookings
Name        | flights_flight_no_departure_airport_stat
Definition  | flight_no, departure_airport FROM flights
Ndistinct   |
Dependencies| defined
MCV         |
```

Expanded display is off.

Отображается наличие статистики по типам (Ndistinct, Dependencies, MCV), а значения нужно смотреть в таблице pg\_statistic\_ext\_data.

## Расширенная статистика по выражению

как если бы в таблице был определен генерируемый столбец  
статистика улучшает оценку селективности условий с выражениями

city	index	address
Самара	443000	443000, г. Самара
Казань	420000	420000, г. Казань
Нижний Новгород	603000	603000, г. Нижний Новгород
Великий Новгород	173000	173000, г. Великий Новгород

*Расширенная статистика по выражению* позволяет собрать всю базовую статистику, которая была бы собрана, если бы в таблице был определен столбец, вычисляемый по этому выражению.

Если в предикате слева или справа от оператора стоит не имя столбца, а выражение, планировщик использует фиксированную оценку селективности. С помощью статистики по выражению это можно исправить.

Заметим, что выражения также можно использовать вместо имен столбцов в многовариантной статистике любого типа.

## Статистика по выражению

Как мы видели в теме «Базовая статистика», если в условии используются выражение, планировщик, не имея информации о селективности, использует константу и часто ошибается:

```
=> SELECT count(*) FROM flights
WHERE extract(month FROM scheduled_departure AT TIME ZONE 'Europe/Moscow') = 1;
```

```
count
-----
16831
(1 row)
```

```
=> EXPLAIN
SELECT * FROM flights
WHERE extract(month FROM scheduled_departure AT TIME ZONE 'Europe/Moscow') = 1;
```

QUERY PLAN

```
-----
Gather  (cost=1000.00..5943.27 rows=1074 width=63)
  Workers Planned: 1
  -> Parallel Seq Scan on flights  (cost=0.00..4835.87 rows=632 width=63)
       Filter: (EXTRACT(month FROM (scheduled_departure AT TIME ZONE
'Europe/Moscow'::text)) = '1'::numeric)
(4 rows)
```

В теме «Базовая статистика» мы исправляли ситуацию, построив индекс по выражению, но это возможно не во всех случаях. К тому же индексы требуют ресурсов для хранения и постоянной синхронизации. Можно поступить иначе — добавить расширенную статистику по выражению:

```
=> CREATE STATISTICS
ON extract(month FROM scheduled_departure AT TIME ZONE 'Europe/Moscow')
FROM flights;
```

CREATE STATISTICS

```
=> ANALYZE flights;
```

ANALYZE

```
=> EXPLAIN
SELECT * FROM flights
WHERE extract(month FROM scheduled_departure AT TIME ZONE 'Europe/Moscow') = 1;
```

QUERY PLAN

```
-----
Seq Scan on flights  (cost=0.00..6384.17 rows=16552 width=63)
  Filter: (EXTRACT(month FROM (scheduled_departure AT TIME ZONE 'Europe/Moscow'::text))
= '1'::numeric)
(2 rows)
```

Оценка стала корректной.

Расширенная статистика по выражениям хранится отдельно. Вот несколько столбцов:

```
=> SELECT statistics_name, expr, n_distinct, most_common_vals
FROM pg_stats_ext_exprs \gx
```

```
-[ RECORD 1
]-+-----
statistics_name | flights_expr_stat
expr            | EXTRACT(month FROM (scheduled_departure AT TIME ZONE
'Europe/Moscow'::text))
n_distinct      | 12
most_common_vals | {8,9,3,5,10,12,7,1,11,4,6,2}
```

Расширенная статистика помогает в сложных случаях

Создается вручную, поддерживается автоматически

Может увеличить затраты на анализ и планирование

1. Используя команды из демонстрации, создайте статистики типа `dependencies`, `mcv` и `ndistinct` для таблицы `flights`. Измерьте время выполнения команды `ANALYZE`. Удалите расширенные статистики, снова измерьте время выполнения команды `ANALYZE` и сравните с предыдущим результатом.
2. Напишите запрос, выбирающий все перелеты в бизнес-классе стоимостью более 100 тыс. рублей. Поможет ли расширенная статистика улучшить оценку кардинальности результата? Если да, то какой тип статистики лучше использовать?

1. Выполняйте измерение времени несколько раз и усредняйте результаты, чтобы сгладить неравномерности.



## 1. Затраты на анализ

Создадим расширенные статистики аналогично тому, как это было сделано в демонстрации.

Статистика по функциональной зависимости:

```
=> CREATE STATISTICS flights_dep(dependencies)
ON flight_no, departure_airport FROM flights;
```

CREATE STATISTICS

Списки наиболее частых комбинаций значений:

```
=> CREATE STATISTICS flights_mcv(mcv)
ON departure_airport, aircraft_code FROM flights;
```

CREATE STATISTICS

Статистика по уникальным комбинациям значений:

```
=> CREATE STATISTICS flights_nd(nddistinct)
ON departure_airport, arrival_airport FROM flights;
```

CREATE STATISTICS

Измерим время анализа.

```
=> \timing on
```

Timing is on.

```
=> ANALYZE flights;
```

ANALYZE

Time: 273,254 ms

В первый раз анализ может занимать существенно больше времени, чем обычно.

```
=> ANALYZE flights;
```

ANALYZE

Time: 249,536 ms

```
=> ANALYZE flights;
```

ANALYZE

Time: 239,203 ms

```
=> \timing off
```

Timing is off.

Удалим созданные расширенные статистики:

```
=> DROP STATISTICS flights_dep;
```

DROP STATISTICS

```
=> DROP STATISTICS flights_mcv;
```

DROP STATISTICS

```
=> DROP STATISTICS flights_nd;
```

DROP STATISTICS

Повторно измеряем время:

```
=> \timing on
```

Timing is on.

```
=> ANALYZE flights;
```

ANALYZE

Time: 158,210 ms

```
=> ANALYZE flights;
```

ANALYZE

Time: 156,144 ms

```
=> \timing off
```

Timing is off.

Было создано всего три расширенные статистики, но время анализа до и после удаления заметно различается. С увеличением количества собираемых расширенных статистик будет расти и время на анализ, что может привести к увеличению нагрузки на сервер.

Поэтому использовать расширенную статистику нужно осмысленно и только в тех случаях, когда это необходимо.

## 2. Применение расширенной статистики

Перелеты в бизнес-классе стоимостью свыше 100 тысяч Р:

```
=> EXPLAIN (analyze, timing off, summary off)
SELECT *
FROM ticket_flights
WHERE fare_conditions = 'Business' and amount > 100_000;
```

QUERY PLAN

```
-----
Gather  (cost=1000.00..124410.16 rows=10064 width=32) (actual rows=111203 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  -> Parallel Seq Scan on ticket_flights  (cost=0.00..122403.76 rows=4193 width=32)
      (actual rows=37068 loops=3)
      Filter: ((amount > '100000'::numeric) AND ((fare_conditions)::text =
'Business'::text))
      Rows Removed by Filter: 2760216
(6 rows)
```

Оптимизатор ошибается на порядок. Причина в том, что стоимость билета и класс обслуживания коррелируют между собой, поэтому расширенная статистика должна помочь исправить оценку. Поскольку между столбцами нет прямой функциональной зависимости, добавим статистику по наиболее частым значениям:

```
=> CREATE STATISTICS (mcv) ON fare_conditions, amount FROM ticket_flights;
```

CREATE STATISTICS

```
=> ANALYZE ticket_flights;
```

ANALYZE

```
=> EXPLAIN (timing off, summary off)
SELECT *
FROM ticket_flights
WHERE fare_conditions = 'Business' and amount > 100_000;
```

QUERY PLAN

```
-----
Gather  (cost=1000.00..126440.07 rows=30398 width=32)
  Workers Planned: 2
  -> Parallel Seq Scan on ticket_flights  (cost=0.00..122400.27 rows=12666 width=32)
      Filter: ((amount > '100000'::numeric) AND ((fare_conditions)::text =
'Business'::text))
(4 rows)
```

Оценка немного улучшилась, но все еще сильно отличается от точного значения. Можно заметить, что доля строк, удовлетворяющих условию, немногим более 1%:

```
=> SELECT count(*) FILTER (WHERE fare_conditions = 'Business' and amount > 100_000)
/ count(*)::float
FROM ticket_flights;
```

?column?

```
-----
0.01325130614791586
(1 row)
```

Поскольку по умолчанию хранится 100 наиболее частых пар, среди них с большой вероятностью встретится не более 1-2 удовлетворяющих условию, и из-за этого оценка оптимизатора не будет адекватной. Попробуем повысить точность, увеличив объем статистики:

```
=> ALTER STATISTICS ticket_flights_fare_conditions_amount_stat SET STATISTICS 500;
```

ALTER STATISTICS

```
=> ANALYZE ticket_flights;
```

ANALYZE

=> EXPLAIN (timing off, summary off)

SELECT \*

FROM ticket\_flights

WHERE fare\_conditions = 'Business' and amount > 100\_000;

QUERY PLAN

```
-----  
-----  
Gather  (cost=1000.00..134201.18 rows=107921 width=32)  
  Workers Planned: 2  
    -> Parallel Seq Scan on ticket_flights  (cost=0.00..122409.08 rows=44967 width=32)  
        Filter: ((amount > '100000'::numeric) AND ((fare_conditions)::text =  
'Business'::text))  
(4 rows)
```

Оценка стала практически точной.