

# Многоверсионность АВТООЧИСТКА



## **Авторские права**

© Postgres Professional, 2016–2022.

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

## **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.

Автоматическая очистка (autovacuum)

Автоанализ

Настройка процесса автоочистки

Работает аналогично обычной очистке

Выполняется периодически

для таблиц с определенным количеством изменений  
в том числе для toast-таблиц

Процесс `autovacuum launcher`

постоянно запущен  
планирует запуск рабочих процессов

Процессы `autovacuum worker`

запускаются процессом `postmaster` по просьбе `autovacuum launcher`  
подключаются к заданной БД, перебирают и очищают таблицы

Автоматическая очистка — механизм, позволяющий запускать обычную очистку в определенные моменты времени, в зависимости от количества изменений в таблицах. Это удобнее и правильнее, чем запуск по расписанию (`cron`), поскольку учитывает динамику системы.

При включенной автоочистке в системе всегда присутствует процесс `autovacuum launcher`, который занимается планированием работы. Реальную очистку выполняют процессы `autovacuum worker`, несколько экземпляров которых могут работать параллельно.

Процессы `autovacuum worker` запускаются процессом `postmaster` по просьбе `autovacuum launcher` (поскольку именно `postmaster` отвечает за порождение всех новых процессов).

<https://postgrespro.ru/docs/postgresql/13/routine-vacuuming#AUTOVACUUM>

Периодически запускает рабочий процесс  
в каждую активную базу данных

раз в *autovacuum\_naptime*

над одной базой могут работать несколько процессов

общее количество рабочих процессов  $\leq$  *autovacuum\_max\_workers*

## Настройки

*autovacuum* = on

*track\_counts* = on

} должны быть включены оба

*autovacuum\_naptime* = 60 s

*autovacuum\_max\_workers* = 3

?

Процесс *autovacuum launcher* составляет список баз данных, в которых есть какая-либо активность (точнее, для которых собирается статистика использования).

Новый рабочий процесс запускается раз в *autovacuum\_naptime* для каждой базы данных из списка (то есть при наличии *N* баз процессы будут порождаться в *N* раз чаще). Если за время *autovacuum\_naptime* рабочий процесс не успел очистить всю базу данных, в ту же базу будет направлен другой рабочий процесс, и они будут работать вместе. Общее количество рабочих процессов ограничено параметром *autovacuum\_max\_workers*.

Чтобы автоматическая очистка в принципе работала, должны быть установлены параметры *autovacuum* и *track\_counts*. Последний включает сбор статистики использования.

Составляет список объектов базы данных для очистки

число ненужных версий строк превышает

$autovacuum\_vacuum\_threshold +$

$+ autovacuum\_vacuum\_scale\_factor \times \text{число строк в таблице}$

число строк, вставленных с момента последней очистки, превышает

$autovacuum\_vacuum\_insert\_threshold +$

$+ autovacuum\_vacuum\_insert\_scale\_factor \times \text{число строк в таблице}$

включаются обычные таблицы и toast-таблицы,

материализованные представления

не включаются временные таблицы

Рабочий процесс подключается к указанной базе данных и строит список всех таблиц, материализованных представлений и toast-таблиц, требующих очистки.

Очистки требуют объекты, в которых накопилось значительное количество ненужных («мертвых») версий строк.

Начиная с версии 13, очищаются также объекты, в которые с момента прошлой очистки было вставлено значительное количество новых строк. Это важно для таблиц, в которые данные только вставляются: их тоже необходимо очищать, чтобы обновить карту видимости и заморозить старые версии (рассматривается в теме «Заморозка»).

Точные формулы приведены на слайде. Они используют два параметра, один из которых (threshold) определяет абсолютное значение, а другой (scale\_factor) — относительную долю строк.

Число строк определяется приблизительно, по статистике:

- общее число — `pg_class.reltuples`,
- число ненужных — `pg_stat_all_tables.n_dead_tup`,
- число вставленных — `pg_stat_all_tables.n_ins_since_vacuum`.

Составляет список объектов базы данных для анализа

число изменившихся версий строк превышает  
 $autovacuum\_analyze\_threshold +$   
 $+ autovacuum\_analyze\_scale\_factor \times \text{число строк в таблице}$

включаются обычные таблицы и материализованные представления  
не включаются toast-таблицы и временные таблицы

По очереди очищает и/или анализирует выбранные объекты

в одной базе может одновременно работать несколько процессов  
на уровне одной таблицы параллелизма нет

Также строится список всех таблиц и материализованных представлений, требующих анализа. Toast-таблицы не анализируются.

Анализа требуют объекты, в которых с момента предыдущего анализа изменилось значительное число строк. Оно определяется по статистике: `pg_stat_all_tables.n_mod_since_analyze`.

Построив два списка, рабочий процесс по очереди очищает и/или анализирует отобранные объекты и по окончании очистки завершается.

В одной БД может одновременно работать несколько процессов, параллельно обрабатывая разные таблицы. На уровне одной таблицы параллелизма нет.

## Настройки автоочистки

*autovacuum\_vacuum\_threshold* = 50

*autovacuum\_vacuum\_scale\_factor* = 0.2



*autovacuum\_vacuum\_insert\_threshold* = 1000

*autovacuum\_vacuum\_insert\_scale\_factor* = 0.2



## Параметры хранения таблиц

*autovacuum\_enabled*

*toast.autovacuum\_enabled*

и одноименные настроечные параметры

Настройки по умолчанию предполагают вызов автоочистки при изменении таблицы на 20 % или вставке 20 % новых строк. Это достаточно высокое значение: автоочистка будет вызываться редко, но работать будет долго, особенно для больших таблиц. В большинстве случаев этот процент следует уменьшить.

В случае необходимости можно настроить эти параметры на уровне отдельных таблиц с помощью параметров хранения:

CREATE TABLE ... WITH (параметр=значение)

Причем параметры можно отдельно настраивать для toast-таблиц.

Кроме того, на уровне отдельных таблиц автоочистку можно отключить.

<https://postgrespro.ru/docs/postgresql/13/runtime-config-autovacuum>

<https://postgrespro.ru/docs/postgresql/13/sql-createtable>

## Настройки автоанализа

*autovacuum\_analyze\_threshold* = 50

*autovacuum\_analyze\_scale\_factor* = 0.1

?

## Параметры хранения таблиц

*autovacuum\_analyze\_threshold*

*autovacuum\_analyze\_scale\_factor*

для toast-таблиц анализ не выполняется

Настройки по умолчанию предполагают вызов автоанализа при изменении таблицы на 10%.

В случае необходимости, можно настроить эти параметры на уровне отдельных таблиц с помощью параметров хранения.

Toast-таблицы не анализируются, поэтому соответствующих параметров для них нет.



## Настройки для регулирования нагрузки

*autovacuum\_vacuum\_cost\_limit* = -1  
(при -1 используется *vacuum\_cost\_limit* = 200)

общий предел  
для всех рабочих  
процессов

*autovacuum\_vacuum\_cost\_delay* = 2ms  
(при -1 используется *vacuum\_cost\_delay* = 0)

*vacuum\_cost\_page\_hit*, *vacuum\_cost\_page\_miss*, *vacuum\_cost\_page\_dirty*

## Параметры хранения таблиц

*autovacuum\_vacuum\_cost\_limit*  
*toast.autovacuum\_vacuum\_cost\_limit*

*autovacuum\_vacuum\_cost\_delay*  
*toast.autovacuum\_vacuum\_cost\_delay*

Регулирование нагрузки при автоматической очистке работает так же, как и при обычной. Однако существуют дополнительные параметры *autovacuum\_vacuum\_cost\_limit* и *autovacuum\_vacuum\_cost\_delay*, которые, если не равны -1, перекрывают параметры *vacuum\_cost\_limit* и *vacuum\_cost\_delay*.

До версии 12 значение параметра *autovacuum\_vacuum\_cost\_delay* составляло 20 мс, что в совокупности с небольшим значением *autovacuum\_vacuum\_cost\_limit* приводило к очень медленной работе автоочистки. С новым значением 2 мс ситуация улучшилась. Для дополнительного ускорения можно увеличить значение *autovacuum\_vacuum\_cost\_limit*.

Кроме того следует учитывать, что предел, устанавливаемый этим параметром, общий для всех рабочих процессов. Поэтому при увеличении *autovacuum\_max\_workers* следует увеличивать и *autovacuum\_vacuum\_cost\_limit*.

Также эти параметры могут указываться и на уровне отдельных таблиц.  
<https://postgrespro.ru/docs/postgresql/13/runtime-config-resource#RUNTIME-CONFIG-RESOURCE-VACUUM-COST>

## Настройки памяти

`autovacuum_work_mem = -1`  
(при `-1` используется `maintenance_work_mem = 64MB`)

настройки действуют для каждого рабочего процесса,  
память выделяется сразу в полном объеме

большой объем памяти уменьшает избыточную обработку  
индексных страниц

## Мониторинг

`log_autovacuum_min_duration`  
`pg_stat_progress_vacuum`

Кроме настроек, влияющих на то, когда и как работать процессу автоочистки, есть возможность отрегулировать выделяемую память для рабочих процессов (autovacuum worker).

По умолчанию размер памяти ограничен параметром `maintenance_work_mem`, который действует не только на автоочистку, но и на все остальные служебные фоновые процессы. Обычно этот параметр можно установить в достаточно большое значение, поскольку фоновых процессов не так много. Однако число рабочих процессов автоочистки (регулируемое параметром `autovacuum_max_workers`) может быть большим, а память выделяется сразу полностью (а не по необходимости). Поэтому для процессов автоочистки можно настроить отдельное ограничение с помощью параметра `autovacuum_work_mem`.

Как говорилось в теме «Очистка», процесс очистки может работать и с минимальным объемом памяти. Но если на таблице созданы индексы, то небольшое значение может привести к повторным сканированиям одних и тех же индексных страниц — автоочистка будет работать медленнее. В идеале следует подобрать такое минимальное значение, при котором нет повторных сканирований.

Для мониторинга есть параметр `log_autovacuum_min_duration`, который выводит информацию об очистке в журнал сообщений сервера.

Напомним, что зачастую следует не увеличивать размер памяти, а уменьшать порог срабатывания очистки, чтобы за один раз обрабатывалось меньше данных.

## Автоочистка

Создадим таблицу с 1000 строками:

```
=> CREATE DATABASE mvcc_autovacuum;
```

CREATE DATABASE

```
=> \c mvcc_autovacuum
```

You are now connected to database "mvcc\_autovacuum" as user "student".

```
=> CREATE TABLE tvac(  
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,  
    n numeric  
);
```

CREATE TABLE

```
=> INSERT INTO tvac(n) SELECT 1 FROM generate_series(1,1000);
```

INSERT 0 1000

Выставим настройки автоочистки.

Небольшое время ожидания, чтобы сразу видеть результат:

```
=> ALTER SYSTEM SET autovacuum_naptime = 1;
```

ALTER SYSTEM

Один процент строк:

```
=> ALTER SYSTEM SET autovacuum_vacuum_scale_factor = 0.01;
```

ALTER SYSTEM

Нулевой порог:

```
=> ALTER SYSTEM SET autovacuum_vacuum_threshold = 0;
```

ALTER SYSTEM

Выставим настройки автоанализа.

Два процента строк:

```
=> ALTER SYSTEM SET autovacuum_analyze_scale_factor = 0.02;
```

ALTER SYSTEM

Нулевой порог:

```
=> ALTER SYSTEM SET autovacuum_analyze_threshold = 0;
```

ALTER SYSTEM

Перечитаем настройки:

```
=> SELECT pg_reload_conf();
```

```
pg_reload_conf  
-----  
t  
(1 row)
```

И подождем немного, чтобы сработал автоанализ.

---

Создадим представление, показывающее, нуждается ли наша таблица в очистке. Здесь мы учитываем только мертвые версии, но аналогично можно добавить и условие для вставленных строк.

```
=> CREATE VIEW vacuum_v AS
WITH params AS (
    SELECT (SELECT setting::integer
            FROM pg_settings
            WHERE name = 'autovacuum_vacuum_threshold') AS vacuum_threshold,
           (SELECT setting::float
            FROM pg_settings
            WHERE name = 'autovacuum_vacuum_scale_factor') AS vacuum_scale_factor
)
SELECT st.relname,
       st.n_dead_tup dead_tup,
       (p.vacuum_threshold + p.vacuum_scale_factor*c.reltuples)::integer max_dead_tup,
       st.n_dead_tup > (p.vacuum_threshold + p.vacuum_scale_factor*c.reltuples)::integer need_vacuum,
       st.last_autovacuum
FROM   pg_stat_all_tables st,
       pg_class c,
       params p
WHERE  c.oid = st.relid
AND    c.relname = 'tvac';
```

CREATE VIEW

Сейчас таблица не требует очистки (в ней нет ненужных версий) и она ни разу не очищалась:

```
=> SELECT * FROM vacuum_v;
```

relname	dead_tup	max_dead_tup	need_vacuum	last_autovacuum
tvac	0	10	f	

(1 row)

Можно создать аналогичное представление и для анализа:

```
=> CREATE VIEW analyze_v AS
WITH params AS (
    SELECT (SELECT setting::integer
            FROM pg_settings
            WHERE name = 'autovacuum_analyze_threshold') as analyze_threshold,
           (SELECT setting::float
            FROM pg_settings
            WHERE name = 'autovacuum_analyze_scale_factor') as analyze_scale_factor
)
SELECT st.relname,
       st.n_mod_since_analyze mod_tup,
       (p.analyze_threshold + p.analyze_scale_factor*c.reltuples)::integer max_mod_tup,
       st.n_mod_since_analyze > (p.analyze_threshold + p.analyze_scale_factor*c.reltuples)::integer need_analyze,
       st.last_autoanalyze
FROM   pg_stat_all_tables st,
       pg_class c,
       params p
WHERE  c.oid = st.relid
AND    c.relname = 'tvac';
```

CREATE VIEW

Представление показывает, что таблица не требует анализа; автоанализ уже был выполнен:

```
=> SELECT * FROM analyze_v;
```

relname	mod_tup	max_mod_tup	need_analyze	last_autoanalyze
tvac	0	20	f	2022-12-16 19:44:07.540008+03

(1 row)

Отключим автоочистку на уровне таблицы и изменим 11 строк (больше 1%):

```
=> ALTER TABLE tvac SET (autovacuum_enabled = off);
```

ALTER TABLE

```
=> UPDATE tvac SET n = n + 1 WHERE id <= 11;
```

UPDATE 11

Проверим представления:

```
=> SELECT * FROM vacuum_v;
```

relname	dead_tup	max_dead_tup	need_vacuum	last_autovacuum
tvac	11	10	t	

(1 row)

=> **SELECT \* FROM analyze\_v;**

relname	mod_tup	max_mod_tup	need_analyze	last_autoanalyze
tvac	11	20	f	2022-12-16 19:44:07.540008+03

(1 row)

Как видно, таблице требуется автоочистка.

Включим автоочистку для таблицы и подождем несколько секунд...

=> **ALTER TABLE tvac SET (autovacuum\_enabled = on);**

ALTER TABLE

=> **SELECT \* FROM vacuum\_v;**

relname	dead_tup	max_dead_tup	need_vacuum	last_autovacuum
tvac	0	10	f	2022-12-16 19:44:13.636191+03

(1 row)

=> **SELECT \* FROM analyze\_v;**

relname	mod_tup	max_mod_tup	need_analyze	last_autoanalyze
tvac	11	20	f	2022-12-16 19:44:07.540008+03

(1 row)

Автоочистка пришла и обработала таблицу. Число ненужных версий снова равно нулю. При этом автоанализ не выполнялся.

Изменим еще 11 строк:

=> **ALTER TABLE tvac SET (autovacuum\_enabled = off);**

ALTER TABLE

=> **UPDATE tvac SET n = n + 1 WHERE id <= 11;**

UPDATE 11

=> **SELECT \* FROM vacuum\_v;**

relname	dead_tup	max_dead_tup	need_vacuum	last_autovacuum
tvac	11	10	t	2022-12-16 19:44:13.636191+03

(1 row)

=> **SELECT \* FROM analyze\_v;**

relname	mod_tup	max_mod_tup	need_analyze	last_autoanalyze
tvac	22	20	t	2022-12-16 19:44:07.540008+03

(1 row)

Теперь должна отработать и автоочистка, и автоанализ.

Проверим это.

=> **ALTER TABLE tvac SET (autovacuum\_enabled = on);**

ALTER TABLE

Несколько секунд ожидания...

=> **SELECT \* FROM vacuum\_v;**

relname	dead_tup	max_dead_tup	need_vacuum	last_autovacuum
tvac	0	10	f	2022-12-16 19:44:18.637668+03

(1 row)

```
=> SELECT * FROM analyze_v;
```

relname	mod_tup	max_mod_tup	need_analyze	last_autoanalyze
tvac	0	20	f	2022-12-16 19:44:18.641373+03

(1 row)

Все правильно, отработали оба процесса.

Показанные представления можно использовать для мониторинга очереди таблиц, ожидающих очистку и анализ, убрав условие на имя таблицы. Для полноты картины в них требуется учесть параметры хранения на уровне отдельных таблиц.

```
=> ALTER TABLE tvac SET (autovacuum_vacuum_scale_factor = 0.01);
```

```
ALTER TABLE
```

```
=> SELECT unnest(reloptions) FROM pg_class WHERE relname = 'tvac';
```

unnest
autovacuum_enabled=on
autovacuum_vacuum_scale_factor=0.01

(2 rows)

## Баланс между разрастанием и накладными расходами

итеративный процесс

### Основные параметры

<i>autovacuum_vacuum_scale_factor</i>	— частота обработки
<i>autovacuum_max_workers</i>	— параллелизм
<i>autovacuum_vacuum_cost_limit</i>	— скорость работы
индивидуальная настройка важных таблиц параметрами хранения	

### Мониторинг

разрастание таблиц  
очередь таблиц, ожидающих очистки  
нагрузка на диски при работе очистки

12

Автоочистка управляется довольно большим числом взаимосвязанных параметров. В хорошо настроенной системе автоочистка не дает таблицам неадекватно разрастаться, и при этом не создает лишних накладных расходов.

Поиск баланса — итеративный процесс. Нужно выставить разумные начальные значения (исходя из размера и характера использования таблиц), проводить постоянный мониторинг и вносить коррективы.

Увеличение *threshold/scale\_factor* приводит к большему разрастанию таблиц, но очистка выполняется реже и большими порциями (возможны пиковые нагрузки), суммарные накладные расходы ниже. Уменьшение параметров приводит к более частой обработке небольшими порциями, накладные расходы в этом случае выше, а таблицы разрастаются меньше.

Пиковые нагрузки можно попытаться сгладить параметрами *cost\_limit/cost\_delay*, уменьшающими скорость работы очистки.

Значение, заданное параметрами *threshold/scale\_factor*, определяет лишь желаемый момент срабатывания очистки. При большом числе сильно измененных таблиц процесс очистки может долго выполнять итерацию и приступит к обработке очередной таблицы далеко не сразу. В этом случае надо либо увеличивать скорость работы очистки (если она была уменьшена параметрами *cost\_limit/cost\_delay*), либо увеличивать число параллельно работающих процессов параметром *max\_workers* (что приведет к увеличению накладных расходов).

Автоматическая очистка запускает очистку и анализ таблиц, динамически реагируя на изменения данных

Автоочистка допускает тонкую настройку на уровне как системы, так и отдельных таблиц

Настройка автоочистки — итеративный поиск баланса



1. Настройте автоочистку на запуск при изменении 10% строк, время «сна» — одна секунда.
2. Создайте таблицу с большим количеством строк.
3. Двадцать раз с интервалом в несколько секунд изменяйте по 5–6% случайных строк. Каждое изменение выполняйте в отдельной транзакции.
4. Сколько раз отработала автоочистка? Насколько разрослась таблица? Совпадают ли результаты с ожидаемыми и как их объяснить?

## 1. Настройка автоочистки

Настраиваем автоочистку на 10% строк.

```
=> ALTER SYSTEM SET autovacuum_vacuum_threshold = 0;
```

ALTER SYSTEM

```
=> ALTER SYSTEM SET autovacuum_vacuum_scale_factor = 0.1;
```

ALTER SYSTEM

Устанавливаем запуск рабочих процессов на один раз в секунду:

```
=> ALTER SYSTEM SET autovacuum_naptime = '1s';
```

ALTER SYSTEM

```
=> SELECT pg_reload_conf();
```

```
pg_reload_conf
-----
t
(1 row)
```

## 2. Таблица

```
=> CREATE DATABASE mvcc_autovacuum;
```

CREATE DATABASE

```
=> \c mvcc_autovacuum
```

You are now connected to database "mvcc\_autovacuum" as user "student".

```
=> CREATE TABLE t(n integer);
```

CREATE TABLE

```
=> INSERT INTO t(n) SELECT 1 FROM generate_series(1,100000);
```

INSERT 0 100000

## 3. Изменение данных

Начальный размер таблицы:

```
=> SELECT pg_size_pretty(pg_table_size('t'));
```

```
pg_size_pretty
-----
3568 kB
(1 row)
```

Выполняем обновление таблицы с задержками в несколько секунд между командами.

```
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
```

UPDATE 5530

```
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
```

UPDATE 5544

```
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
```

UPDATE 5560

```
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
```

UPDATE 5490

```
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
```

UPDATE 5517

```
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
```

UPDATE 5441

```

=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5529
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5477
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5493
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5538
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5516
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5479
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5598
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5546
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5511
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5566
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5406
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5512
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5446
=> UPDATE t SET n = n + 1 WHERE random() < 0.055;
UPDATE 5501

```

#### 4. Оценка разрастания

Статистика по таблице (autovacuum\_count — сколько раз выполнялась автоочистка):

```

=> SELECT * FROM pg_stat_all_tables WHERE relname='t' \gx

```

```

-[ RECORD 1 ]-----+-----
reloid          | 60008
schemaname      | public
relname         | t
seq_scan        | 20
seq_tup_read     | 2000000
idx_scan        |
idx_tup_fetch   |
n_tup_ins       | 100000
n_tup_upd       | 110200
n_tup_del       | 0
n_tup_hot_upd   | 78929
n_live_tup      | 100000
n_dead_tup      | 5501
n_mod_since_analyze | 0
n_ins_since_vacuum | 0
last_vacuum     |
last_autovacuum | 2022-12-16 19:51:36.730627+03
last_analyze    |
last_autoanalyze | 2022-12-16 19:51:38.822922+03
vacuum_count     | 0
autovacuum_count | 4
analyze_count    | 0
autoanalyze_count | 10

```

Вот как увеличилась таблица:

```
=> SELECT pg_size_pretty(pg_table_size('t'));
```

```

pg_size_pretty
-----
4416 kB
(1 row)

```

Суммарно изменилось порядка 110000 строк, что составляет 110% от размера таблицы. Можно было бы предположить, что автоочистка сработает 10-11 раз.

Однако автоочистка выполнялась примерно в два раза реже, поскольку существенная часть обновлений была оптимизирована внутривстраничной очисткой (в нашем случае, HOT-очисткой: поле `n_tup_hot_upd`).

Таблица увеличилась не в два раза, а меньше — примерно на треть. В идеале достаточно было бы 5-6%, но:

- карта свободного пространства не обновляется при внутривстраничной очистке, а автоочистка срабатывает только на второй раз;
- в результате HOT-обновлений на страницах появляются «лишние» `redirect`-указатели — при небольшой длине строк, как в нашем примере, этот эффект становится заметен;
- карта свободного пространства хранит не точную, а округленную информацию.