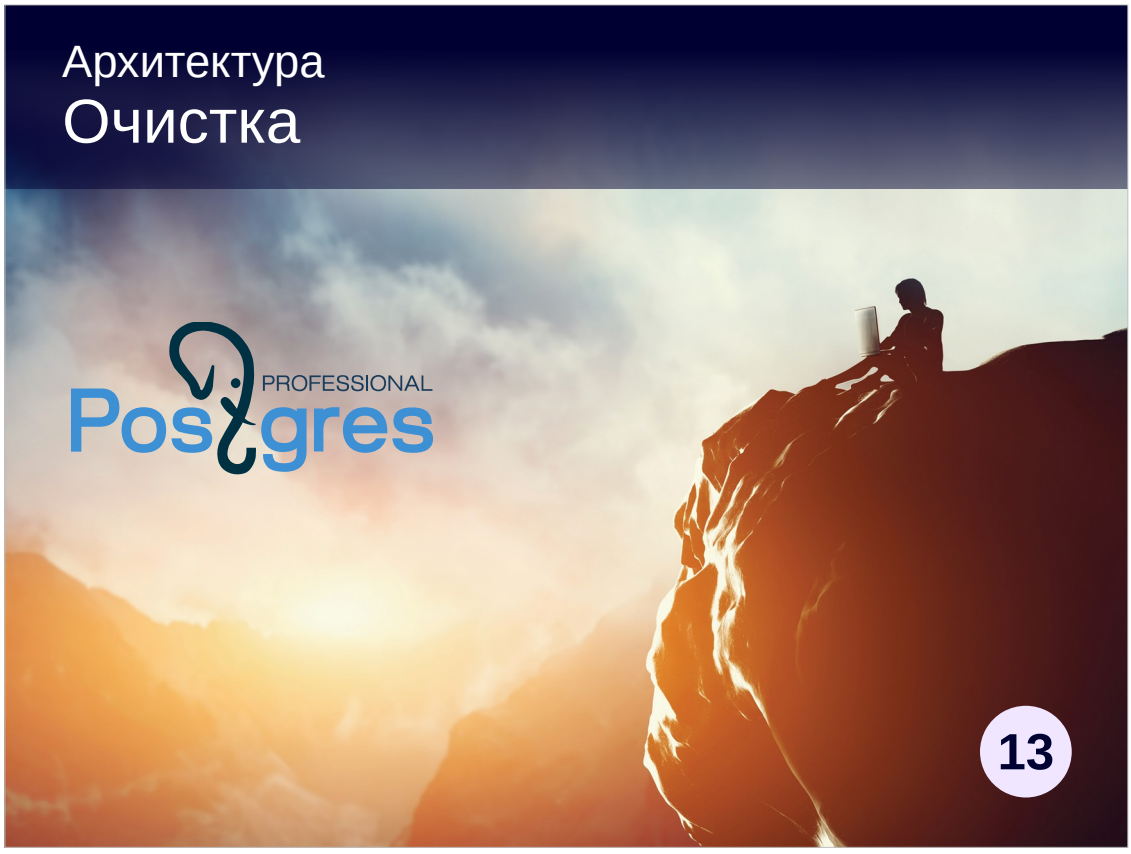


Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

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

Темы

Задачи, требующие периодического выполнения

Очистка и анализ

Разрастание таблиц и индексов

Полная очистка и перестроение индексов

Периодические задачи

Очистка страниц от исторических данных,
которые образуются из-за многоверсионности

- из таблиц вычищаются мертвые версии строк
- из индексов вычищаются записи, ссылающиеся на мертвые версии

Механизм многоверсионности позволяет эффективно реализовать изоляцию на основе снимков, но в результате в табличных страницах накапливаются старые версии строк, а в страницах индексов — ссылки на эти версии. Какое-то время исторические версии нужны, чтобы транзакции могли работать со своими снимками данных. Но со временем не остается ни одного снимка данных, которому требовалась бы старая версия строки; такая версия называется «мертвой».

Процедура очистки вычищает мертвые версии строк из табличных страниц и ненужные индексные записи, которые ссылались на такие версии.

Если своевременно не вычищать исторические данные, таблицы и индексы будут неконтролируемо разрастаться и поиск в них актуальных версий строк будет замедляться.

https://postgrespro.ru/docs/postgresql/13/routine-vacuuming

Периодические задачи

Обновление карты видимости

отмечает страницы, на которых все версии строк видны во всех снимках

используется для оптимизации работы процесса очистки
и ускорения индексного доступа

существует только для таблиц

Кроме этой главной задачи, процедура очистка берет на себя и другие задачи по поддержанию работоспособности экземпляра. Очистка обновляет карту видимости и карту свободного пространства. Это служебная информация, которая хранится вместе с основными данными.

В карте видимости отмечены страницы, которые содержат только актуальные версии строк, видимые во всех снимках данных. Иными словами, это страницы, которые давно не изменялись и успели полностью очиститься от неактуальных версий.

Карта видимости применяется:

- Для оптимизации очистки.
  В отмеченные страницы очистке не надо заглядывать — в них не может быть мертвых версий.
- Для ускорения доступа только по индексу.
  Информация о версионности хранится только для таблиц, но не для индексов (поэтому у индексов не бывает карты видимости). Получив из индекса ссылку на версию строки, нужно прочитать табличную страницу, чтобы проверить ее видимость. Но если в самом индексе уже есть все нужные столбцы, и при этом страница отмечена в карте видимости, то обращения к таблице можно избежать.

Если не обновлять карту видимости, индексный доступ будет работать менее эффективно. Более подробно об этом рассказывается в курсе QPT «Оптимизация запросов».

Периодические задачи

Обновление карты свободного пространства

- отмечает свободное пространство в страницах после очистки
- используется при вставке новых версий строк
- существует и для таблиц, и для индексов

В карте свободного пространства отмечено наличие пустого места внутри страниц. Это место постоянно меняется: при добавлении новых версий строк оно уменьшается, при очистке — увеличивается.

Карта используется при вставке новых версий строк, чтобы быстро найти подходящую страницу, на которую поместятся добавляемые данные. Для ускорения поиска карта свободного пространства имеет сложную древовидную структуру.

Карта свободного пространства может существовать и для индексов. Но, поскольку индексная запись вставляется в строго определенное место индекса, в карте отмечаются только пустые страницы, образовавшиеся при удалении из них всех записей. Такие страницы исключаются из индексной структуры и при необходимости подключаются затем в подходящее место индекса.

Периодические задачи

Обновление статистики

- используется оптимизатором запросов
- вычисляется на основе случайной выборки

Для работы оптимизатора запросов необходима статистическая информация («статистика») о данных, такая как количество строк в таблицах и распределение данных в столбцах. Сбор статистики называется анализом.

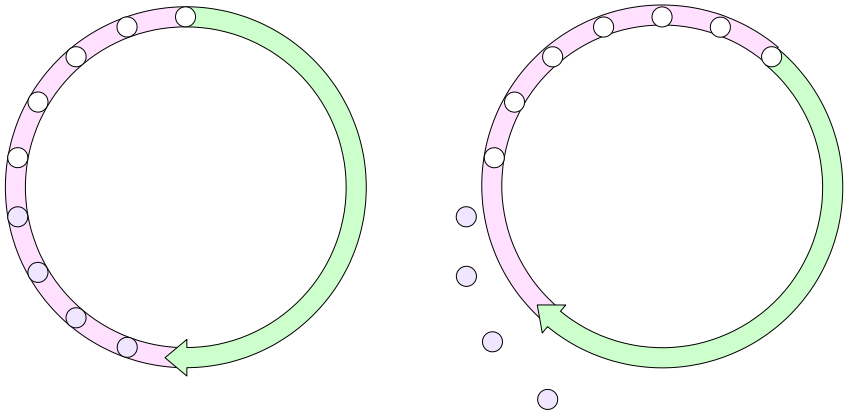
Для анализа из таблицы читается случайная выборка данных определенного размера. Это позволяет быстро собрать информацию даже по очень большим таблицам. Результат получается не точный, но этого и не требуется. В любом случае данные все время изменяются и постоянно поддерживать абсолютно точную статистику невозможно. Достаточно, чтобы она периодически обновлялась и не слишком сильно отличалась от действительности.

Если не обновлять статистику, она перестанет соответствовать реальным данным и оптимизатор станет строить плохие планы выполнения. Из-за этого запросы могут начать выполняться на порядки медленнее, чем могли бы.

Периодические задачи

Заморозка

предотвращение последствий переполнения 32-битного счетчика транзакций



Как уже говорилось, PostgreSQL упорядочивает события с помощью номеров транзакций. Под счетчик отведено 32 бита и рано или поздно он переполнится. Но если счетчик сбросится в ноль, то упорядоченность транзакций нарушится.

Поэтому пространство номеров закольцовано: для любой транзакции половина номеров против часовой стрелки находится в прошлом, а другая половина — в будущем.

Достаточно старые версии строк должны помечаться как «замороженные». Такой признак говорит о том, что версия строки появилась так давно, что номер создавшей ее транзакции больше не имеет значения и его можно использовать повторно.

Для того чтобы не просматривать лишние страницы, в карту видимости добавлен бит, отмечающий те страницы, на которых все версии строк уже заморожены.

Если не выполнять заморозку своевременно, сервер не сможет выделить очередной номер транзакции. Это аварийная ситуация: сервер остановится, все незавершенные транзакции оборвутся. После этого администратор должен будет вручную стартовать сервер и выполнить заморозку.

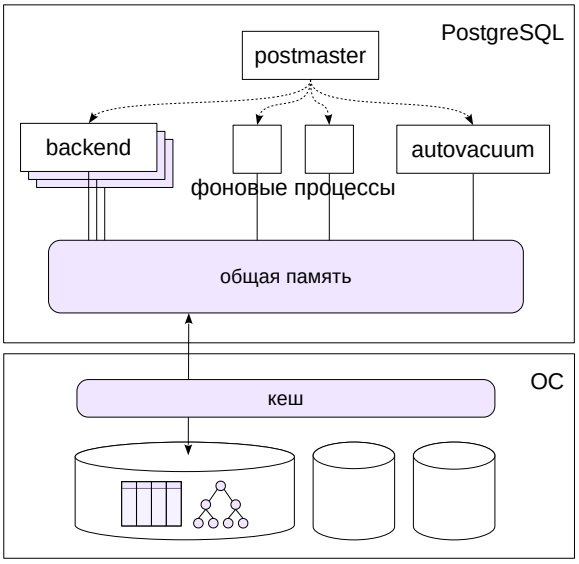
Автоматическая очистка

Autovacuum launcher

- фоновый процесс
- периодически запускает рабочие процессы

Autovacuum worker

- очищает таблицы отдельной базы данных, требующие обработки



Выполнение всех описанных выше периодических задач обслуживания берет на себя фоновый процесс автоочистки (autovacuum). Он динамически реагирует на частоту обновления таблиц: чем активней изменения, тем чаще таблица будет обрабатываться.

В системе постоянно присутствует процесс autovacuum launcher, который планирует работу очистки и запускает необходимое число рабочих процессов autovacuum worker, работающих параллельно.

Очистка работает постранично, не приводя к блокировкам других транзакций, хотя и создает, конечно, нагрузку на подсистему ввода-вывода.

Автоматическая очистка перестанет работать при отключении любого из двух параметров *autovacuum* или *track_counts*. Может ошибочно показаться, что отключение способно увеличить производительность системы за счет исключения «лишних» операций ввода-вывода.
На самом деле отказ от очистки влечет за собой последствия, описанные выше: неконтролируемое разрастание файлов, замедление запросов и риск аварийной остановки сервера. В конечном итоге это приведет к полному параличу системы.

Автоочистка должна работать. Она настраивается большим количеством конфигурационных параметров; эта настройка детально обсуждается в курсе DBA2 «Настройка и мониторинг».

Запуск вручную

Очистка

`VACUUM [`*`таблица, ...`*`]`	очистка отдельных таблиц
`VACUUM`	очистка всей БД
`$ vacuumdb`	обертка для использования в ОС

Анализ

```
ANALYZE
$ vacuumdb --analyze-only
```

Очистка и анализ

```
VACUUM ANALYZE
$ vacuumdb --analyze
```


При необходимости очистку и анализ можно запускать вручную.

Для этого служат команды VACUUM (только очистка), ANALYZE (только анализ), а также VACUUM ANALYZE (и очистка, и анализ).

Автоочистка отличается от запуска очистки и анализа по расписанию тем, что реагирует на активность изменения данных. Слишком частый запуск по расписанию создаст ненужную нагрузку на систему. Если же запускать очистку редко, при большом объеме изменений файлы могут успеть существенно вырасти в размерах.

https://postgrespro.ru/docs/postgresql/13/sql-vacuum

https://postgrespro.ru/docs/postgresql/13/sql-analyze

Очистка

Создадим таблицу, в целях эксперимента отключив для нее автоматическую очистку, чтобы контролировать время срабатывания:

```
=> CREATE TABLE bloat(
  id integer GENERATED ALWAYS AS IDENTITY,
  d timestamptz
) WITH (autovacuum_enabled = off);
```

```
CREATE TABLE
```

Заполним таблицу данными и создадим индекс:

```
=> INSERT INTO bloat(d)
  SELECT current_timestamp FROM generate_series(1,100000);
```

```
INSERT 0 100000
```

```
=> CREATE INDEX ON bloat(d);
```

```
CREATE INDEX
```

Сейчас все строки таблицы имеют ровно одну, актуальную, версию.

Теперь обновим одну строку:

```
=> UPDATE bloat SET d = d + interval '1 day' WHERE id = 1;
```

```
UPDATE 1
```

Запустим очистку вручную и попросим ее рассказать о том, что происходит:

```
=> VACUUM (verbose) bloat;
```

```
INFO:  vacuuming "public.bloat"
INFO:  scanned index "bloat_d_idx" to remove 1 row versions
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.01 s
INFO:  "bloat": removed 1 row versions in 1 pages
DETAIL:  CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
INFO:  index "bloat_d_idx" now contains 100000 row versions in 87 pages
DETAIL:  1 index row versions were removed.
0 index pages have been deleted, 0 are currently reusable.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s.
INFO:  "bloat": found 1 removable, 100000 nonremovable row versions in 541 out of 541 pages
DETAIL:  0 dead row versions cannot be removed yet, oldest xmin: 511
There were 0 unused item identifiers.
Skipped 0 pages due to buffer pins, 0 frozen pages.
0 pages are entirely empty.
CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.03 s.
VACUUM
```

Из вывода команды можно заключить: что:

- из таблицы вычищена одна версия (removed 1 row versions);
- из индекса удалена одна запись (1 index row versions were removed).

Проблема разрастания

Postgres PROFESSIONAL

Очистка не уменьшает размер таблиц и индексов

- «дыры» в страницах используются для новых данных, но место не возвращается операционной системе

Причины разрастания

- неправильная настройка автоочистки
- массовое изменение данных
- долгие транзакции

Негативное влияние

- перерасход места на диске
- замедление последовательного просмотра таблиц
- уменьшение эффективности индексного доступа

Очистка вычищает неактуальные версии строк из страниц. В страницах образуется свободное пространство, которое затем используется для размещения новых данных. Но освободившееся место не возвращается операционной системе, то есть с точки зрения ОС размер файлов данных не уменьшается.

В случае индексов (B-деревьев) дело осложняется тем, что если на странице не хватает места для размещения индексной записи, страница расщепляется на две. Получившиеся страницы уже никогда не объединяются, даже если из них будут удалены все индексные записи.

При правильной настройке процесса автоочистки файлы данных вырастают на некоторую постоянную величину за счет обновлений между запусками очистки. Но если выполняется одномоментное изменение большого объема данных или в системе присутствуют долгие транзакции (удерживающие снимки данных и не позволяющие вычищать неактуальные версии строк), очистка не сможет своевременно освобождать место. В результате размер таблиц и индексов может продолжать увеличиваться.

Разрастание файлов данных ведет не только к перерасходу места на диске (в том числе и для резервных копий), но и к ухудшению производительности.

https://wiki.postgresql.org/wiki/Show_database_bloat

https://postgrespro.ru/docs/postgresql/13/pgstattuple

Оценка разрастания таблиц и индексов

Понять, насколько критично разрослись объекты и пора ли предпринимать радикальные меры, можно разными способами:

- запросами к системному каталогу;
- используя расширение pgstattuple.

```
=> CREATE EXTENSION pgstattuple;
```

```
CREATE EXTENSION
```

С помощью расширения можно проверить состояние таблицы:

```
=> SELECT * FROM pgstattuple('bloat') \gx
```

```
-[ RECORD 1 ]------+--------
table_len          | 4431872
tuple_count        | 100000
tuple_len          | 4000000
tuple_percent      | 90.26
dead_tuple_count   | 0
dead_tuple_len     | 0
dead_tuple_percent | 0
free_space         | 16720
free_percent       | 0.38
```

- tuple_percent — доля полезной информации (не 100% из-за накладных расходов).

И индекса:

```
=> SELECT * FROM pgstatindex('bloat_d_idx') \gx
```

```
-[ RECORD 1 ]------+-------
version            | 4
tree_level         | 1
index_size         | 712704
root_block_no      | 3
internal_pages     | 1
leaf_pages         | 85
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 89.17
leaf_fragmentation | 0
```

- leaf_pages — количество листовых страниц индекса;
- avg_leaf_density — заполненность листовых страниц;
- leaf_fragmentation — характеристика физической упорядоченности страниц.

Теперь обновим сразу половину строк:

```
=> UPDATE bloat SET d = d + interval '1 day' WHERE id % 2 = 0;
```

```
UPDATE 50000
```

Посмотрим на таблицу снова:

```
=> SELECT * FROM pgstattuple('bloat') \gx
```

```
-[ RECORD 1 ]------+--------
table_len          | 6643712
tuple_count        | 100000
tuple_len          | 4000000
tuple_percent      | 60.21
dead_tuple_count   | 50000
dead_tuple_len     | 2000000
dead_tuple_percent | 30.1
free_space         | 21004
free_percent       | 0.32
```

Плотность уменьшилась.

Чтобы не читать всю таблицу целиком, можно попросить pgstattuple показать приблизительную информацию:

```
=> SELECT * FROM pgstattuple_approx('bloat') \gx
```

```
-[ RECORD 1 ]--------+--------------------
table_len            | 6643712
scanned_percent      | 100
approx_tuple_count   | 100000
approx_tuple_len     | 4000000
approx_tuple_percent | 60.207305795314426
dead_tuple_count     | 50000
dead_tuple_len       | 2000000
dead_tuple_percent   | 30.103652897657213
approx_free_space    | 21004
approx_free_percent  | 0.31614856273119607
```

И посмотрим на индекс:

```
=> SELECT * FROM pgstatindex('bloat_d_idx') \gx
```

```
-[ RECORD 1 ]------+--------
version            | 4
tree_level         | 1
index_size         | 1040384
root_block_no      | 3
internal_pages     | 1
leaf_pages         | 125
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 90.95
leaf_fragmentation | 0
```

Заполненность листовых страниц осталась на прежнем уровне, но количество страниц увеличилось.

Перестроение объектов

Полная очистка

```
VACUUM FULL
$ vacuumdb --full
```

полностью перестраивает содержимое таблиц и индексов

полностью блокирует работу с таблицей

Перестроение индексов

```
REINDEX
```

перестраивает индексы

полностью блокирует работу с индексом
и блокирует изменение таблицы

Для того чтобы уменьшить физический размер разросшихся таблиц и индексов, требуется *полная очистка*.

Команда VACUUM FULL полностью перезаписывает содержимое таблицы и ее индексов, минимизируя занимаемое место. Однако этот процесс требует исключительной блокировки таблицы и поэтому не может выполняться параллельно с другими транзакциями.

https://postgrespro.ru/docs/postgresql/13/sql-vacuum

Можно перестроить только индекс или несколько индексов, не трогая таблицу. Это выполняется командой REINDEX. Она устанавливает блокировку на запись в таблицу (чтение доступно), поэтому транзакции, пытающиеся изменить таблицу, будут заблокированы.

https://postgrespro.ru/docs/postgresql/13/sql-reindex

Если продолжительная исключительная блокировка нежелательна, можно рассмотреть стороннее расширение pg_repack https://github.com/reorg/pg_repack, позволяющее выполнить перестроение таблицы и ее индексов «на лету».

Перестроение объектов

Неблокирующее перестроение индексов

`REINDEX ... CONCURRENTLY`

перестраивает индексы, не блокируя изменение таблицы

выполняется дольше и может завершиться неудачно

не транзакционно

не работает для системных индексов

не работает для индексов, связанных с ограничениями-исключениями

Начиная с 12-й версии, команда REINDEX с указанием CONCURRENTLY работает без блокировки таблицы на запись. Однако неблокирующее перестроение выполняется дольше и может завершиться неудачно (из-за взаимоблокировок) — в таком случае индекс потребуется еще раз перестроить.

У неблокирующего перестроения индексов есть ряд ограничений. Такая операция не может выполняться внутри транзакции. В неблокирующем режиме нельзя перестроить системные индексы и индексы, связанные с ограничениями-исключениями (EXCLUDE).

Перестроение объектов

Для перестроения индексов удобно использовать команду REINDEX с указанием CONCURRENTLY. Это позволяет не останавливать работу системы на время перестроения.

```
=> REINDEX TABLE CONCURRENTLY bloat;
```

```
REINDEX
```

Теперь посмотрим на индекс:

```
=> SELECT * FROM pgstatindex('bloat_d_idx') \gx
```

```
-[ RECORD 1 ]------+-------
version            | 4
tree_level         | 1
index_size         | 712704
root_block_no      | 3
internal_pages     | 1
leaf_pages         | 85
empty_pages        | 0
deleted_pages      | 0
avg_leaf_density   | 89.17
leaf_fragmentation | 0
```

Плотность вернулась к начальным значениям.

Для перестроения таблицы вместе с ее индексами можно воспользоваться командой VACUUM FULL. Однако, в отличие от REINDEX .. CONCURRENTLY, она полностью блокирует работу с таблицей.

```
=> VACUUM FULL bloat;
```

```
VACUUM
```

```
=> SELECT * FROM pgstattuple('bloat') \gx
```

```
-[ RECORD 1 ]------+--------
table_len          | 4431872
tuple_count        | 100000
tuple_len          | 4000000
tuple_percent      | 90.26
dead_tuple_count   | 0
dead_tuple_len     | 0
dead_tuple_percent | 0
free_space         | 16724
free_percent       | 0.38
```

Плотность увеличилась, освобожденное место отдано операционной системе.

Итоги

Версии строк накапливаются, поэтому необходима периодическая очистка

Процедура очистки решает много других задач:

- обновление карт видимости и свободного пространства
- сбор статистики для планировщика
- заморозка старых версий строк

Автоочистка должна работать, но требует настройки

При разрастании может потребоваться полная очистка

Практика

1. Отключите процесс автоочистки и убедитесь, что он не работает.
2. В новой базе данных создайте таблицу с одним числовым столбцом и индекс по этой таблице. Вставьте в таблицу 100 000 случайных чисел.
3. Несколько раз измените половину строк таблицы, контролируя на каждом шаге размер таблицы и индекса.
4. Выполните полную очистку.
5. Повторите пункт 4, вызывая после каждого изменения обычную очистку. Сравните результаты.
6. Включите процесс автоочистки.

1. Установите параметр *autovacuum* в значение off и попросите сервер перечитать файлы конфигурации.

3. Используйте функции `pg_table_size(`*`имя-таблицы`*`)` и `pg_indexes_size(`*`имя-таблицы`*`)`. Подробнее о функциях для вычисления размеров различных объектов говорится в модуле «Организация данных».

6. Установите параметр *autovacuum* в значение on (или сбросьте значение этого параметра командой RESET), затем попросите сервер перечитать файлы конфигурации.

1. Отключение автоочистки

```
=> SELECT pid, backend_start, backend_type
FROM pg_stat_activity
WHERE backend_type = 'autovacuum launcher';

  pid  |         backend_start         |    backend_type
-------+-------------------------------+--------------------
 26845 | 2022-05-06 14:45:01.581355+03 | autovacuum launcher
(1 row)


=> ALTER SYSTEM SET autovacuum = off;

ALTER SYSTEM

=> SELECT pg_reload_conf();

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


=> SELECT pid, backend_start, backend_type
FROM pg_stat_activity
WHERE backend_type = 'autovacuum launcher';

 pid | backend_start | backend_type
-----+---------------+--------------
(0 rows)
```

2. База данных, таблица и индекс

Создаем базу данных, таблицу и индекс:

```
=> CREATE DATABASE admin_maintenance;

CREATE DATABASE

=> \c admin_maintenance

You are now connected to database "admin_maintenance" as user "student".

=> CREATE TABLE t(n numeric);

CREATE TABLE

=> CREATE INDEX t_n on t(n);

CREATE INDEX
```

Вставляем строки:

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

INSERT 0 100000
```

Для удобства записываем запрос, вычисляющий размер таблицы и индекса, в переменную psql:

```
=> \set SIZE 'SELECT pg_size_pretty(pg_table_size(''t'')) table_size, pg_size_pretty(pg_indexes_size(''t'')) index_size\\g (footer=off)'

=> :SIZE

 table_size | index_size
------------+------------
 4360 kB    | 4432 kB
```

3. Изменение строк без очистки

```
=> UPDATE t SET n=n WHERE n < 0.5;

UPDATE 49936

=> :SIZE

 table_size | index_size
------------+------------
 6520 kB    | 6640 kB


=> UPDATE t SET n=n WHERE n < 0.5;

UPDATE 49936

=> :SIZE

 table_size | index_size
------------+------------
 8680 kB    | 7648 kB


=> UPDATE t SET n=n WHERE n < 0.5;

UPDATE 49936
```

```
=> :SIZE

 table_size | index_size
------------+------------
 11 MB      | 11 MB
```

Размер таблицы и индекса постоянно растет.

4. Полная очистка

```
=> VACUUM FULL t;

VACUUM

=> :SIZE

 table_size | index_size
------------+------------
 4336 kB    | 3104 kB
```

Размер таблицы практически вернулся к начальному, индекс стал компактнее (построить индекс по большому объему данных эффективнее, чем добавлять эти данные к индексу построчно).

5. Изменение строк с очисткой

```
=> UPDATE t SET n=n WHERE n < 0.5;

UPDATE 49936

=> VACUUM t;

VACUUM

=> :SIZE

 table_size | index_size
------------+------------
 6528 kB    | 4648 kB


=> UPDATE t SET n=n WHERE n < 0.5;

UPDATE 49936

=> VACUUM t;

VACUUM

=> :SIZE

 table_size | index_size
------------+------------
 6528 kB    | 4648 kB


=> UPDATE t SET n=n WHERE n < 0.5;

UPDATE 49936

=> VACUUM t;

VACUUM

=> :SIZE

 table_size | index_size
------------+------------
 6528 kB    | 4648 kB
```

Размер увеличился один раз и затем стабилизировался.

Пример показывает, что удаление (и изменение) большого объема данных по возможности следует разделить на несколько транзакций. Это позволит автоматической очистке своевременно удалять ненужные версии строк, что позволит избежать чрезмерного разрастания таблицы.

6. Восстанавливаем автоочистку

```
=> ALTER SYSTEM RESET autovacuum;

ALTER SYSTEM

=> SELECT pg_reload_conf();

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