



# Автоочистка и заморозка



Автоматическая очистка (autovacuum)

Автоанализ

Настройка процесса автоочистки

Переполнение номера транзакции и заморозка

Работает аналогично ручной 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

## Алгоритм

для каждой базы данных (в которой есть активность)  
запускать рабочий процесс раз в *autovacuum\_naptime*

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

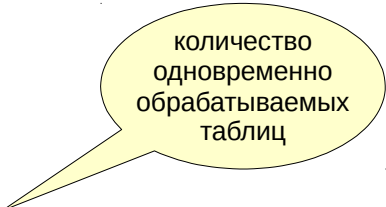
## Настройки

`autovacuum = on`

`track_counts = on`

`autovacuum_naptime = 60 s`

`autovacuum_max_workers = 3`

A yellow speech bubble callout pointing to the value '3' in the `autovacuum_max_workers` setting. It contains the text: "количество одновременно обрабатываемых таблиц".

количество  
одновременно  
обрабатываемых  
таблиц

## Алгоритм

выполнять по очереди очистку всех таблиц (включая TOAST),  
в которых число ненужных версий строк превышает

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

а также выполнять анализ всех таблиц,  
в которых число изменившихся версий строк превышает

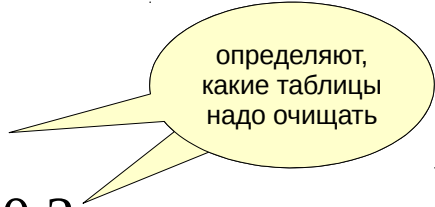
$autovacuum\_analyze\_threshold +$   
 $+ autovacuum\_analyze\_scale\_factor * \text{число строк в таблице}$

в одной БД могут параллельно работать несколько процессов

## Настройки очистки

autovacuum\_vacuum\_threshold = 50

autovacuum\_vacuum\_scale\_factor = 0.2



определяют,  
какие таблицы  
надо очищать

## Параметры хранения таблиц

autovacuum\_enabled

toast.autovacuum\_enabled

autovacuum\_vacuum\_threshold

toast.autovacuum\_vacuum\_threshold

autovacuum\_vacuum\_scale\_factor

toast.autovacuum\_vacuum\_scale\_factor

*разрешение автоочистки  
на уровне отдельной таблицы*

## Настройки анализа

`autovacuum_analyze_threshold = 50`

`autovacuum_analyze_scale_factor = 0.1`

## Параметры хранения таблиц

`autovacuum_analyze_threshold`

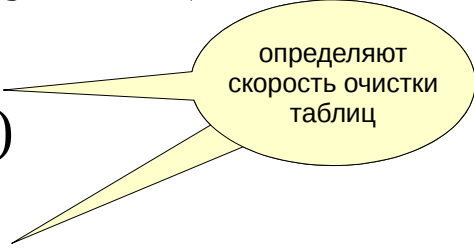
`autovacuum_analyze_scale_factor`

## Настройки для регулирования нагрузки

`autovacuum_vacuum_cost_limit = -1`  
(при -1 используется `vacuum_cost_limit`)

`autovacuum_vacuum_cost_delay = 20 ms`  
(при -1 используется `vacuum_cost_delay`)

`vacuum_cost_page_hit, vacuum_cost_page_miss, vacuum_cost_page_dirty`



определяют  
скорость очистки  
таблиц

## Параметры хранения таблиц

`autovacuum_vacuum_cost_limit`  
`toast.autovacuum_vacuum_cost_limit`

`autovacuum_vacuum_cost_delay`  
`toast.autovacuum_vacuum_cost_delay`

## Настройки памяти

`autovacuum_work_mem = -1`

(при `-1` используется `maintenance_work_mem`)

`maintenance_work_mem = 64MB`

настройки действуют для каждого рабочего процесса

большой объем памяти уменьшает избыточную обработку

индексных страниц

## Поиск баланса

между разрастанием таблиц и накладными расходами

## Основные параметры

`autovacuum_vacuum_threshold`, `autovacuum_vacuum_scale_factor`

`autovacuum_max_workers`

`autovacuum_vacuum_cost_limit`, `autovacuum_vacuum_cost_delay`

индивидуальная настройка важных таблиц параметрами хранения

## Итеративный процесс

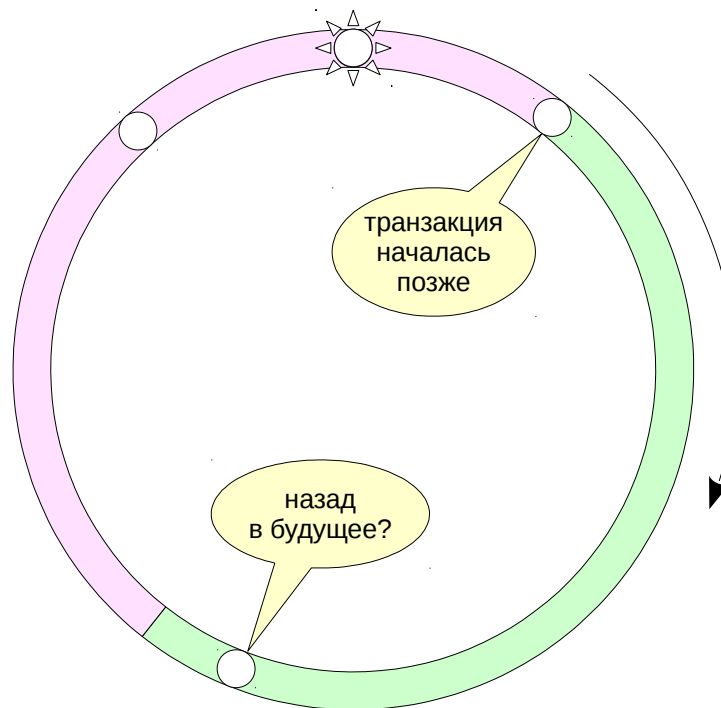
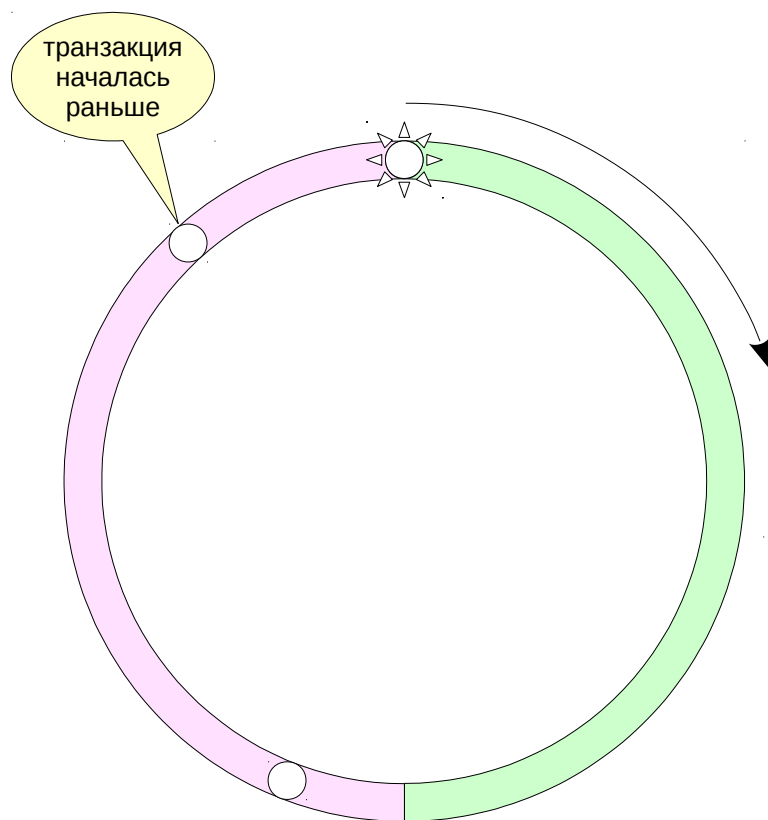
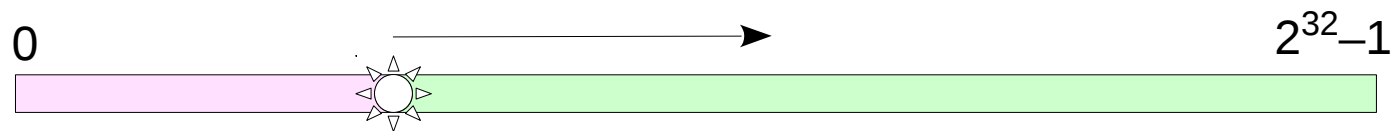
мониторинг автоочистки: `log_autovacuum_min_duration`

мониторинг разрастания таблиц

мониторинг очереди таблиц, ожидающих очистки

мониторинг нагрузки на диски при работе очистки

# Счетчик транзакций



Транзакции попадают из прошлого в будущее?

из-за цикличности номеров транзакций

если это произойдет, данные потеряют согласованность

Еще одна задача процесса очистки

транзакции, изменения которых видны во всех снимках, помечаются специальным образом и считаются «замороженными»

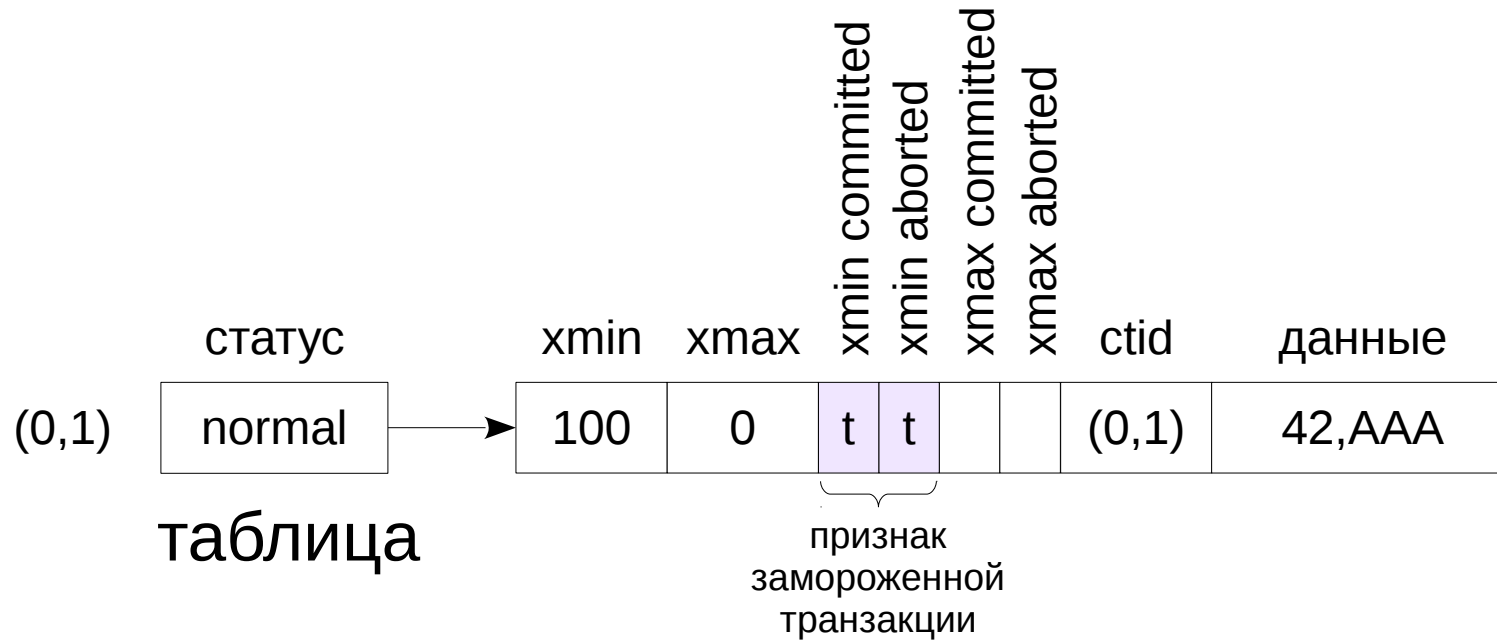
исключение из правил видимости версий строк:

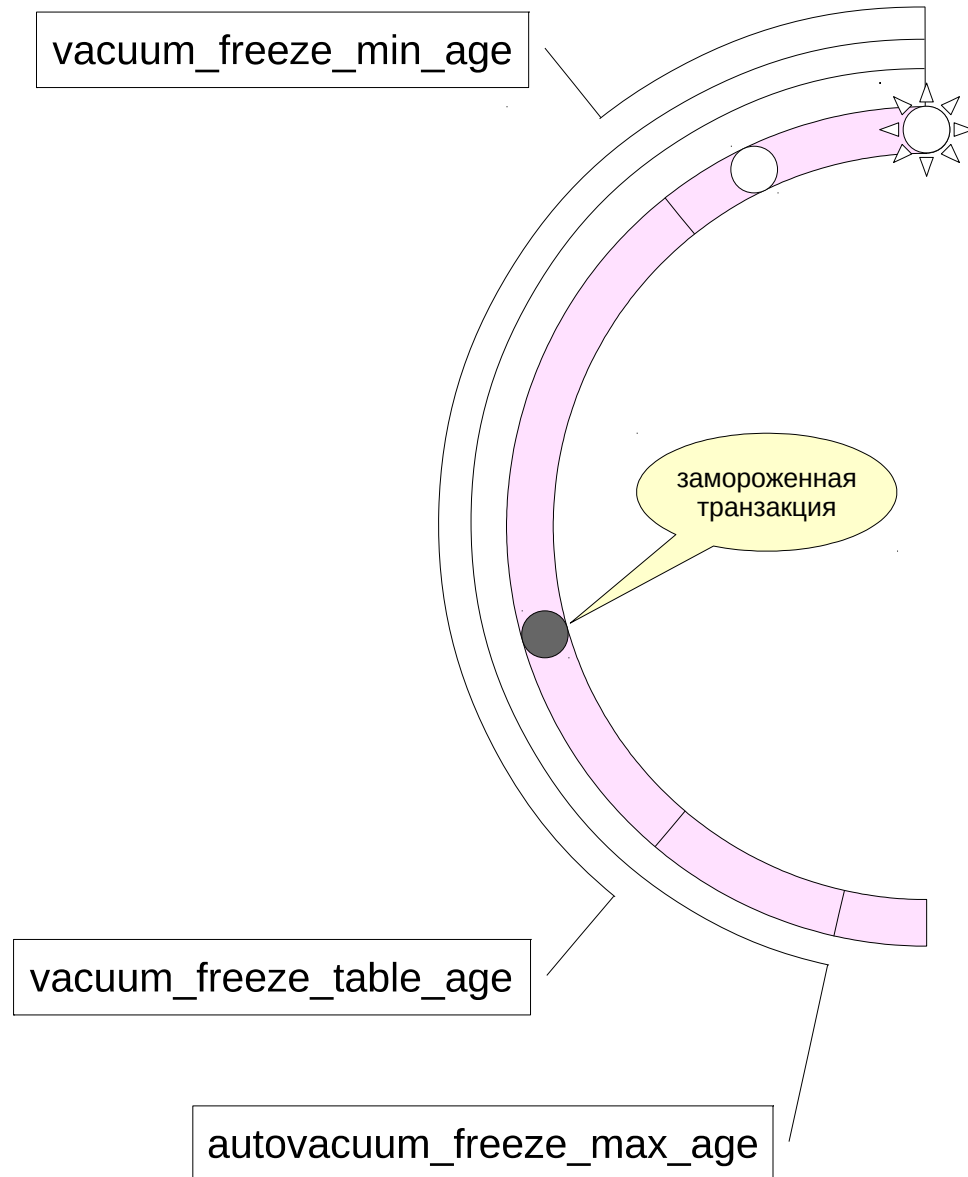
замороженные транзакции считаются старше любых обычных

если в будущем оказывается незамороженная транзакция,

PostgreSQL останавливается для предотвращения несогласованности

# Заморозка





`vacuum_freeze_min_age`

минимальный возраст,  
с которого обычная очистка  
начинает заморозку

`vacuum_freeze_table_age`

возраст, при котором выполняется  
полное сканирование

`autovacuum_freeze_max_age`

возраст, при котором заморозка  
запускается принудительно  
определяет размер CLOG

## Настройки

`vacuum_freeze_min_age` = 50 000 000

`vacuum_freeze_table_age` = 150 000 000

`autovacuum_freeze_max_age` = 200 000 000

## Параметры хранения таблиц

`autovacuum_freeze_min_age`

`toast.autovacuum_freeze_min_age`

`vacuum_freeze_table_age`

`toast.vacuum_freeze_table_age`

`autovacuum_freeze_max_age`

`toast.autovacuum_freeze_max_age`

## Настройки для multixact

```
vacuum_multixact_freeze_min_age      = 5 000 000
vacuum_multixact_freeze_table_age    = 150 000 000
autovacuum_multixact_freeze_max_age  = 400 000 000
```

## Параметры хранения таблиц

```
autovacuum_multixact_freeze_min_age
toast.autovacuum_multixact_freeze_min_age

vacuum_multixact_freeze_table_age
toast.vacuum_multixact_freeze_table_age

autovacuum_multixact_freeze_max_age
toast.autovacuum_multixact_freeze_max_age
```

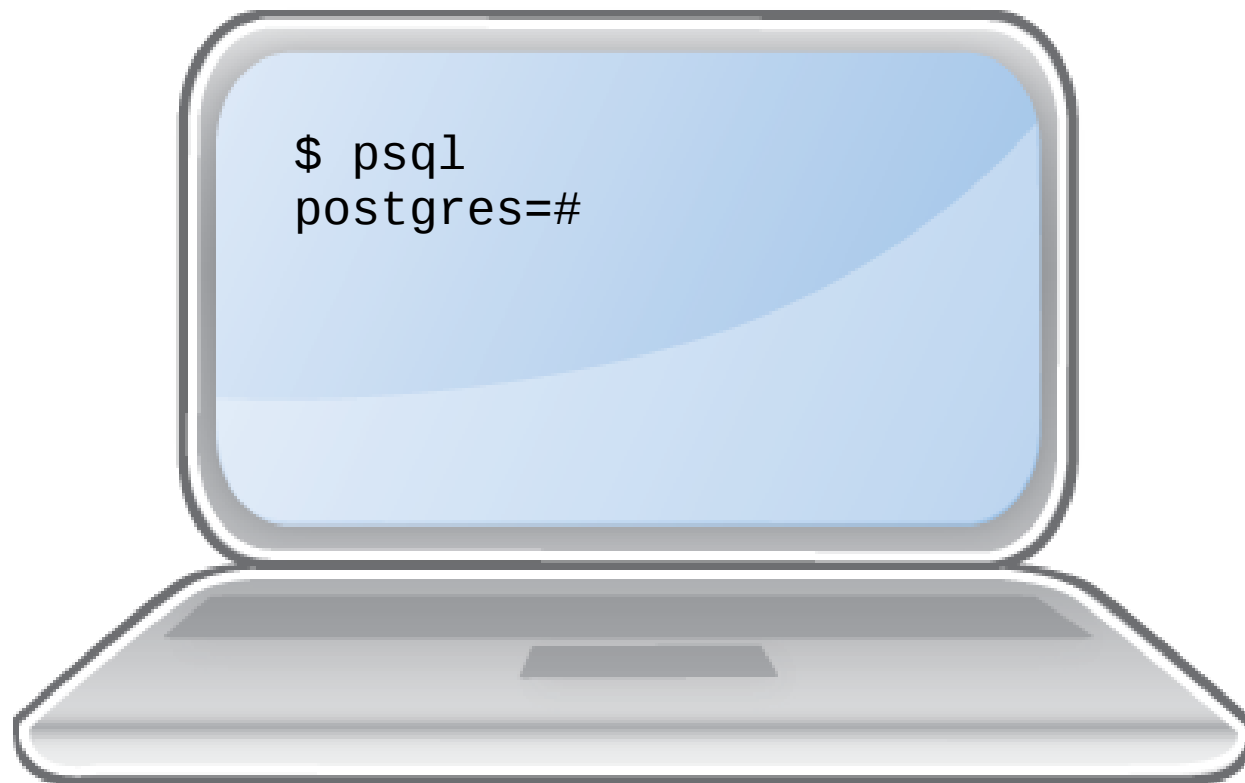
## VACUUM FREEZE

принудительная заморозка транзакций любого возраста  
тот же эффект и при VACUUM FULL

## COPY ... WITH FREEZE

принудительная заморозка сразу после загрузки  
таблица должна быть создана или опустошена в той же транзакции  
замороженные данные становятся видны сразу —  
нарушаются правила изоляции транзакции

# Демонстрация



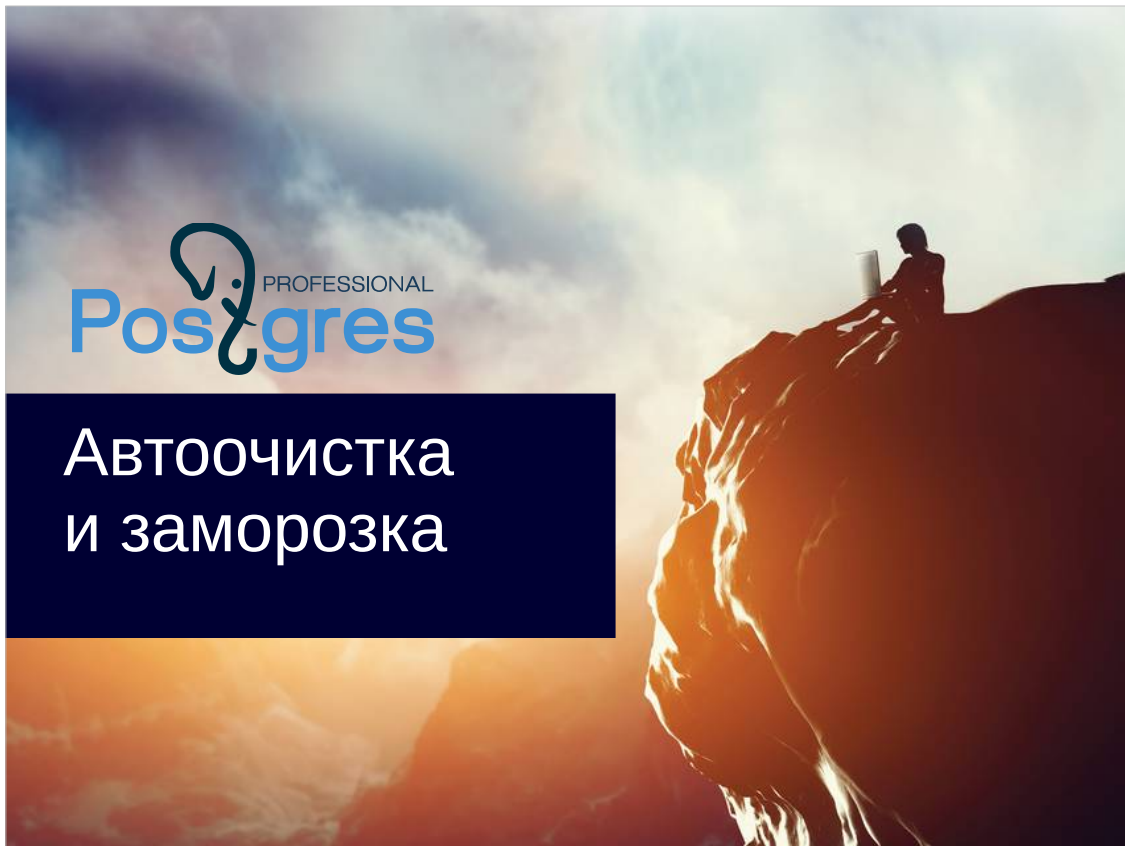
Автоматическая очистка запускает очистку и анализ таблиц, динамически реагируя на изменения данных

Кроме освобождения места на странице процесс очистки выполняет важную задачу заморозки

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

Настройка автоочистки — итеративный поиск баланса

1. Настроить автоочистку на запуск при изменении 10% строк, время «сна» установить в одну секунду.
2. Создать базу данных DB6, в ней создать и проанализировать таблицу с 100 000 строк.
3. Десять раз с интервалом в несколько секунд изменять таблицу порциями по 5 000 строк.
4. Сколько раз отработала автоочистка? На сколько разрослась таблица? Совпадают ли результаты с ожидаемыми и как их объяснить?
5. Повторить пункт 2 с другой таблицей; затем во втором сеансе создать снимок данных, включающий текущие версии строк таблицы; затем повторить пункт 3; после этого завершить второй сеанс.
6. Сравнить новые результаты с предыдущими.



### **Авторские права**

Курс «Администрирование PostgreSQL 9.5. Расширенный курс»

© Postgres Professional, 2016 год.

Авторы: Егор Рогов, Павел Лузанов

### **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

Переполнение номера транзакции и заморозка

Работает аналогично ручной 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, чем запуск по расписанию (`cron`), поскольку учитывает динамику системы.

При включенной автоочистке в системе всегда присутствует один 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).

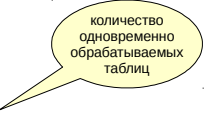
<http://www.postgresql.org/docs/9.5/static/routine-vacuuming.html>

## Алгоритм

для каждой базы данных (в которой есть активность)  
запускать рабочий процесс раз в *autovacuum\_naptime*  
количество рабочих процессов  $\leq$  *autovacuum\_max\_workers*

## Настройки

*autovacuum* = on  
*track\_counts* = on  
*autovacuum\_naptime* = 60 s  
*autovacuum\_max\_workers* = 3



количество  
одновременно  
обрабатываемых  
таблиц

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

Новый рабочий процесс запускается раз в *autovacuum\_naptime* для каждой БД (то есть при наличии N баз процессы будут порождаться в N раз чаще). Общее количество рабочих процессов ограничено параметром *autovacuum\_max\_workers*.

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

## Алгоритм

выполнять по очереди очистку всех таблиц (включая TOAST),  
в которых число ненужных версий строк превышает  
 $autovacuum\_vacuum\_threshold +$   
 $+ autovacuum\_vacuum\_scale\_factor * \text{число строк в таблице}$

а также выполнять анализ всех таблиц,  
в которых число изменившихся версий строк превышает  
 $autovacuum\_analyze\_threshold +$   
 $+ autovacuum\_analyze\_scale\_factor * \text{число строк в таблице}$

в одной БД могут параллельно работать несколько процессов

Рабочий процесс подключается к указанной базе данных и строит список всех таблиц, материализованных представлений и toast-таблиц, требующих очистки — у которых число ненужных («мертвых») версий строк превышает пороговое значение, заданное двумя параметрами:

- `autovacuum_vacuum_threshold` определяет абсолютный минимум,
- `autovacuum_vacuum_scale_factor` определяет относительную долю изменившихся строк.

А также строит список объектов, требующих анализа — у которых число измененных версий строк превышает пороговое значение, заданное двумя аналогичными параметрами:

- `autovacuum_analyze_threshold`,
- `autovacuum_analyze_scale_factor`.

При этом число строк определяется приблизительно, по данным статистики:

- общее число — `pg_class.reltuples`,
- число ненужных — `pg_stat_all_tables.n_dead_tup`,
- число измененных — `pg_stat_all_tables.n_mod_since_analyze`.

Дальше процесс по очереди очищает и/или анализирует отобранные объекты и по окончании очистки завершается.

В одной БД могут одновременно работать несколько процессов. В этом случае один процесс будет пропускать объект, над которым работает другой процесс.

## Настройки очистки

`autovacuum_vacuum_threshold = 50`

`autovacuum_vacuum_scale_factor = 0.2`

определяют,  
какие таблицы  
надо очищать

## Параметры хранения таблиц

`autovacuum_enabled`

`toast.autovacuum_enabled`

*разрешение автоочистки  
на уровне отдельной таблицы*

`autovacuum_vacuum_threshold`

`toast.autovacuum_vacuum_threshold`

`autovacuum_vacuum_scale_factor`

`toast.autovacuum_vacuum_scale_factor`

Настройки по умолчанию предполагают вызов автоочистки при изменении таблицы на 20%. Это достаточно большое значение: автоочистка будет вызываться редко, но работать будет долго, особенно для больших таблиц.

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.

<http://www.postgresql.org/docs/9.5/static/runtime-config-autovacuum.html>

<http://www.postgresql.org/docs/9.5/static/sql-createtable.html>

## Настройки анализа

```
autovacuum_analyze_threshold = 50  
autovacuum_analyze_scale_factor = 0.1
```

## Параметры хранения таблиц

```
autovacuum_analyze_threshold  
autovacuum_analyze_scale_factor
```

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.

<http://www.postgresql.org/docs/9.5/static/runtime-config-autovacuum.html>

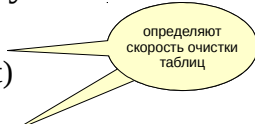
<http://www.postgresql.org/docs/9.5/static/sql-createtable.html>

## Настройки для регулирования нагрузки

`autovacuum_vacuum_cost_limit = -1`  
(при -1 используется `vacuum_cost_limit`)

`autovacuum_vacuum_cost_delay = 20 ms`  
(при -1 используется `vacuum_cost_delay`)

`vacuum_cost_page_hit`, `vacuum_cost_page_miss`, `vacuum_cost_page_dirty`



определяют  
скорость очистки  
таблиц

## 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`.

Также эти параметры могут указываться и на уровне отдельных таблиц.

<http://www.postgresql.org/docs/9.5/static/runtime-config-resource.html>

## Настройки памяти

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

`maintenance_work_mem = 64MB`

настройки действуют для каждого рабочего процесса  
большой объем памяти уменьшает избыточную обработку  
индексных страниц

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 фоновые процессы (но не на процессы, обслуживающие клиентов — про них будет идти отдельный разговор в теме «Использование памяти»).

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, поэтому для них можно настроить отдельное ограничение с помощью параметра `autovacuum_work_mem`.

В принципе, автоочистка может работать и с минимальным объемом памяти. Но если на таблице созданы индексы, то чем меньше выделено оперативной памяти, тем выше вероятность повторной обработки одних и тех же индексных страниц — следовательно, автоочистка будет работать медленнее.

<http://www.postgresql.org/docs/9.5/static/runtime-config-resource.html>

## Поиск баланса

между разрастанием таблиц и накладными расходами

## Основные параметры

`autovacuum_vacuum_threshold`, `autovacuum_vacuum_scale_factor`

`autovacuum_max_workers`

`autovacuum_vacuum_cost_limit`, `autovacuum_vacuum_cost_delay`

индивидуальная настройка важных таблиц параметрами хранения

## Итеративный процесс

мониторинг автоочистки: `log_autovacuum_min_duration`

мониторинг разрастания таблиц

мониторинг очереди таблиц, ожидающих очистки

мониторинг нагрузки на диски при работе очистки

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.

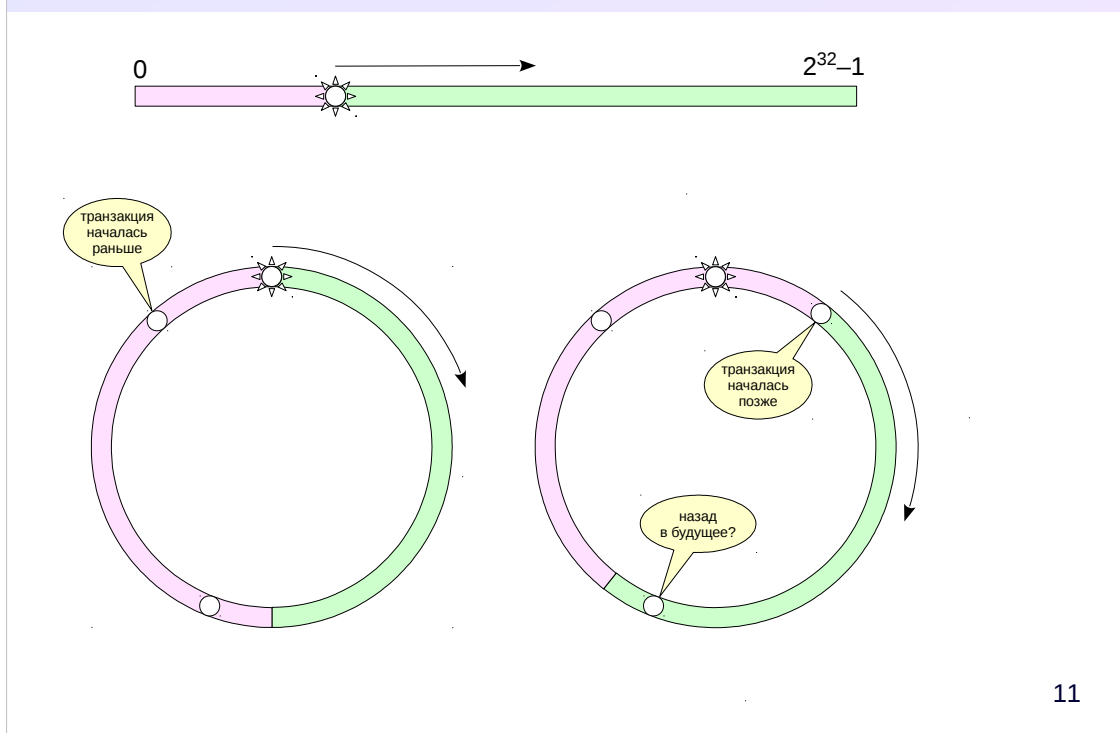
Поиск баланса — итеративный процесс. Нужно выставить разумные начальные значения (исходя из размера и характера использования таблиц), проводить постоянный мониторинг и вносить коррективы.

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).

# Счетчик транзакций



Кроме освобождения места в страницах, очистка выполняет также задачу по предотвращению проблем, связанных с переполнением счетчика транзакций.

Под номер транзакции в PostgreSQL выделено 32 бита. Это довольно большое число (около 4 млрд), но при активной работе сервера это число может быть исчерпано (при нагрузке 1000 транзакций в секунду это произойдет через полтора месяца непрерывной работы).

Но механизм многоверсионности полагается на то, что транзакции нумеруются последовательно — из двух транзакций начавшейся раньше считается транзакция с меньшим номером. Так что обнуление номера транзакции при переполнении приведет к катастрофе.

Поэтому номера транзакций «закольцованы» — для любой транзакции половина транзакций находится в будущем, а другая половина — в прошлом.

Что же происходит с транзакциями, которые находились в далеком прошлом, но в результате работы оказались в будущем?

## Транзакции попадают из прошлого в будущее?

из-за цикличности номеров транзакций  
если это произойдет, данные потеряют согласованность

## Еще одна задача процесса очистки

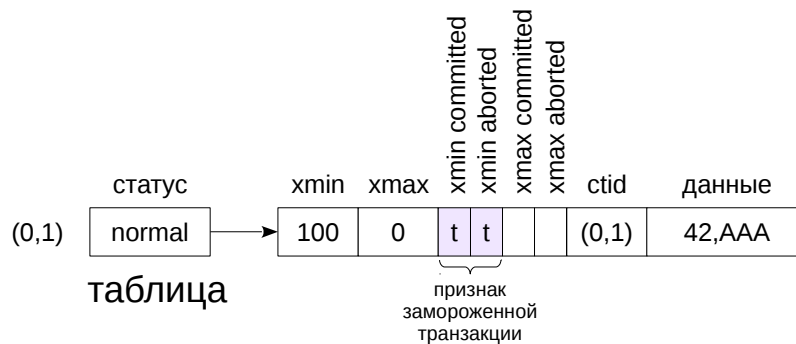
транзакции, изменения которых видны во всех снимках,  
помечаются специальным образом и считаются «замороженными»  
исключение из правил видимости версий строк:  
замороженные транзакции считаются старше любых обычных  
если в будущем оказывается незамороженная транзакция,  
PostgreSQL останавливается для предотвращения несогласованности

Чтобы не допустить путешествий из прошлого в будущее, процесс очистки выполняет еще одну задачу. Он находит «весьма старые» версии строк, которые не просто видны во всех снимках, но и изменение которых маловероятно, и специальным образом помечает («замораживает») их. Замороженная транзакция считается старше любых обычных транзакций — для них в правилах видимости версий сделано специальное исключение. Таким образом, попадая в будущее, версия строки все равно остается в прошлом.

Если возникает ситуация, при которой попасть в будущее рискует еще не замороженная транзакция, PostgreSQL аварийно остановится. Это возможно в двух случаях: либо транзакция не завершена и, следовательно, не может быть заморожена, либо не сработала очистка.

При запуске сервера транзакция будет автоматически откачена; дальше администратор должен выполнить очистку и после этого система может продолжать работать дальше.

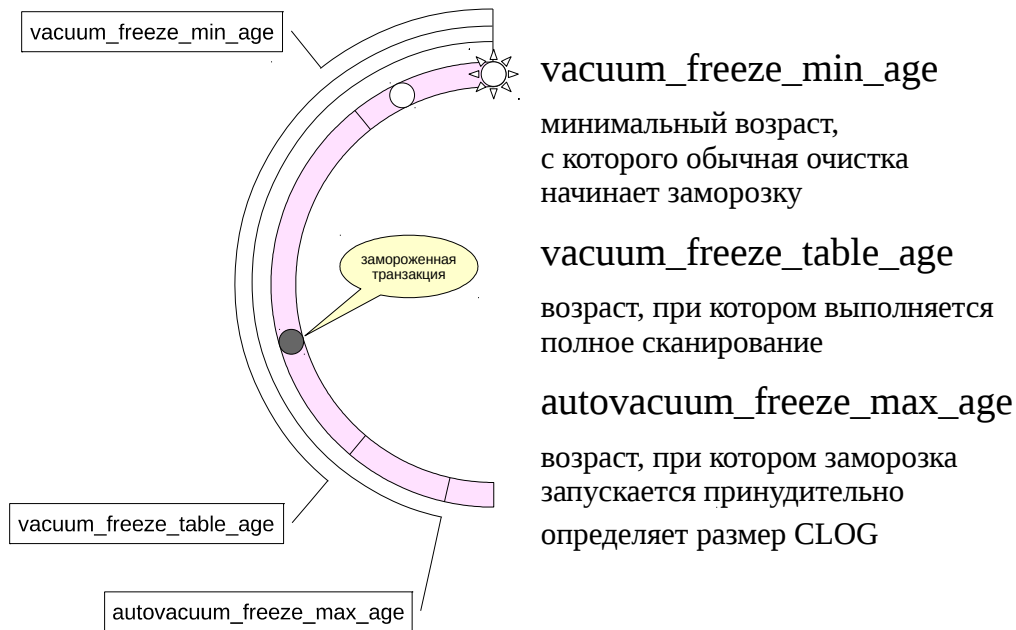
<http://www.postgresql.org/docs/9.5/static/routine-vacuuming.html>



Для того, чтобы пометить номер транзакции `xmin` как замороженный, выставляются одновременно оба бита-подсказки — бит фиксации и бит отката.

Заметим, что транзакцию `xmax` замораживать не нужно. Ее наличие означает, что данная версия строки больше не актуальна. После того, как она перестанет быть видимой во всех снимках, версия строки будет очищена.

Многие источники (включая документацию) упоминают специальный номер `FrozenTransactionId = 2`, которым помечаются замороженные транзакции. Такая система действовала до версии 9.4, но сейчас заменена на биты-подсказки — это позволяет сохранить в версии строки исходный номер транзакции, что удобно для целей поддержки и отладки.



При обычной очистке заморозка выполняется только частично, так как при этом не просматриваются давно не изменявшиеся страницы. Поэтому время от времени выполняет полное сканирование таблицы, что может вызвать большую нагрузку на дисковую подсистему.

Заморозкой управляют три основных параметра.

- `Vacuum_freeze_min_age` определяет минимальный возраст транзакции (количество транзакций, прошедших с начала данной), при котором ее можно замораживать.

Чем меньше будет это значение, тем больше будет накладных расходов (замороженные версии строк вероятно изменятся и их придется замораживать снова). Однако большое значение приведет к более частому выполнению полного сканирования.

- `Vacuum_freeze_table_age` определяет возраст транзакции, при котором пора выполнять заморозку с помощью полного сканирования. Для этого каждая таблица хранит номер транзакции, который был заморожен в прошлый раз (`pg_class.relfrozexid`). Таким образом, полное сканирование будет выполняться раз в  $(vacuum\_freeze\_table\_age - vacuum\_freeze\_min\_age)$  транзакций.

- `Autovacuum_freeze_max_age` определяет возраст транзакции, при котором заморозка будет выполняться принудительно — автоочистка запустится, даже если она выключена. Этот параметр также определяет размер CLOG. Между `vacuum_freeze_table_age` и `autovacuum_freeze_max_age` должно быть некоторое окно, чтобы очистка успела отработать — иначе система будет аварийно остановлена.

## Настройки

```
vacuum_freeze_min_age      = 50 000 000  
vacuum_freeze_table_age    = 150 000 000  
autovacuum_freeze_max_age = 200 000 000
```

## Параметры хранения таблиц

```
autovacuum_freeze_min_age  
toast.autovacuum_freeze_min_age  
  
vacuum_freeze_table_age  
toast.vacuum_freeze_table_age  
  
autovacuum_freeze_max_age  
toast.autovacuum_freeze_max_age
```

Значения по умолчанию довольно консервативны. Предел для `autovacuum_freeze_max_age` составляет порядка 2 млрд транзакций, а используется значение в 10 раз меньшее. Это сделано для ограничения размера CLOG (два байта на транзакцию, т. е. примерно 50 МБ при значении по умолчанию).

Параметры также можно устанавливать на уровне отдельных таблиц с помощью параметров хранения.

## Настройки для multixact

```
vacuum_multixact_freeze_min_age      = 5 000 000
vacuum_multixact_freeze_table_age     = 150 000 000
autovacuum_multixact_freeze_max_age  = 400 000 000
```

## Параметры хранения таблиц

```
autovacuum_multixact_freeze_min_age
toast.autovacuum_multixact_freeze_min_age

vacuum_multixact_freeze_table_age
toast.vacuum_multixact_freeze_table_age

autovacuum_multixact_freeze_max_age
toast.autovacuum_multixact_freeze_max_age
```

Еще один тонкий момент составляют «мультитранзакции», рассмотренные в теме «Снимки и блокировки». В случаях, когда одна строка блокируется несколькими транзакциями, выделяется специальный номер `multixactid`, который используется для той же цели, что и обычный номер транзакции. Следовательно, для них тоже необходимо выполнять аналог «заморозки» — старые номера `multixactid` заменяются на новые (или на обычный номер транзакции, если это допустимо).

За нее отвечают параметры, аналогичные параметрам обычной заморозки.

## VACUUM FREEZE

принудительная заморозка транзакций любого возраста  
тот же эффект и при VACUUM FULL

## COPY ... WITH FREEZE

принудительная заморозка сразу после загрузки  
таблица должна быть создана или опустошена в той же транзакции  
замороженные данные становятся видны сразу —  
нарушаются правила изоляции транзакции

Иногда бывает удобно управлять заморозкой вручную, а не дожидаться автоочистки.

Заморозку можно вызвать вручную командой `vacuum freeze` — при этом будут заморожены все транзакции без оглядки на их возраст. При перестройке таблицы командой `vacuum full` все строки также замораживаются.

Данные можно заморозить и при начальной загрузке с помощью команды `copy`, указав параметр `freeze`. Для этого таблица должна быть создана (или опустошена командой `truncate`) в той же транзакции, что и `copy`. Поскольку для замороженных строк действуют отдельные правила видимости, такие строки будут сразу видны другим транзакциям, даже если текущая транзакция еще не зафиксирована. Это нарушает обычные правила изоляции, так что с такой командой надо быть осторожным.



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

Кроме освобождения места на странице процесс очистки выполняет важную задачу заморозки

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

Настройка автоочистки — итеративный поиск баланса

1. Настроить автоочистку на запуск при изменении 10% строк, время «сна» установить в одну секунду.
2. Создать базу данных DB6, в ней создать и проанализировать таблицу с 100 000 строк.
3. Десять раз с интервалом в несколько секунд изменять таблицу порциями по 5 000 строк.
4. Сколько раз отработала автоочистка? На сколько разрослась таблица? Совпадают ли результаты с ожидаемыми и как их объяснить?
5. Повторить пункт 2 с другой таблицей; затем во втором сеансе создать снимок данных, включающий текущие версии строк таблицы; затем повторить пункт 3; после этого завершить второй сеанс.
6. Сравнить новые результаты с предыдущими.

Оценить разрастание таблицы можно разными способами. Например, запомнить размер таблицы до изменений и сравнить с размером после изменений. На практике, это, конечно, неудобно. Поэтому можно воспользоваться расширением `pgstattuple` (показывает точную информацию, но полностью читает таблицу), или сделать приблизительную оценку на основе системного каталога и статистики (примеры в вики).

Для создания снимка данных проще всего начать транзакцию с уровнем изоляции `repeatable read`, и в ней выполнить запрос к таблице.