14-MAY-26

Tantor: DBA1-18

Администрирование PostgreSQL 18

Слайды

tantor

Автор: Олег Иванов

I

Введение

Администрирование PostgreSQL

Предварительная подготовка

- Знакомство с SQL
- Навыки работы в операционной системе Linux
- Рекомендуется опыт работы с любой реляционной БД

Целевая аудитория

- Администраторы баз данных PostgreSQL
- Разработчики приложений для баз данных
- Сотрудники технической поддержки

Материалы курса

- Учебное пособие в форме книги в формате pdf, которое содержит теоретическую часть курса
- Практические задания
- виртуальная машина для выполнения практических заданий

Разделы курса

0. Введение
1. Установка и управление СУБД
2. Архитектура (5 частей)
3. Конфигурация
4. Базы данных (2 части)
5. Журналирование
6. Безопасность
7. Резервное копирование (2 части)
8. Репликация (2 части)
9. Платформа Tantor, обзор возможностей
10. Новые возможности Tantor Postgres 17.5

О курсе

- очное или дистанционное обучение с инструктором:
  - продолжительность 5 дней
  - начало в 10:00
  - перерыв на обед 13:00-14:00
  - окончание до 17:00 (последний день до 15:00)

О компании Тантор Лабс

- с 2016 года на международном рынке
- с 2021 года на российском рынке
- разработка СУБД Tantor Postgres
- разработка Платформы Tantor для мониторинга и управления СУБД семейства PostgreSQL, а также кластеров Patroni
- многолетний опыт эксплуатации высоконагруженных систем
- входит в Группу Астра

СУБД Tantor Postgres

Tantor BE

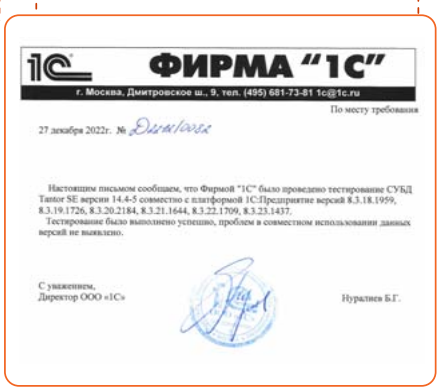
Новые возможности и доработки по сравнению с PostgreSQL, техническая поддержка

Tantor SE

СУБД Enterprise-уровня, подходит для наиболее нагруженных OLTP-систем или КХД размером до 100ТБ

Tantor SE 1C

СУБД для высоких нагрузок, оптимизированная и одобренная для работы с приложениями 1С



ФИРМА "1С"

г. Москва, Дмитровское ш., 9, тел. (495) 681-73-81 1c@1c.ru

По месту требования

27 декабря 2022г. № Disk/0002

Настоящим письмом сообщаем, что Фирмой "1С" было проведено тестирование СУБД Tantor SE версии 14.4-5 совместно с платформой 1С:Предприятие версий 8.3.18.1959, 8.3.19.1726, 8.3.20.2184, 8.3.21.1644, 8.3.22.1709, 8.3.23.1437.
Тестирование было выполнено успешно, проблем в совместном использовании данных версий не выявлено.

С уважением,
Директор ООО «1С»

Нуралиев Б.Г.

Tantor Polar

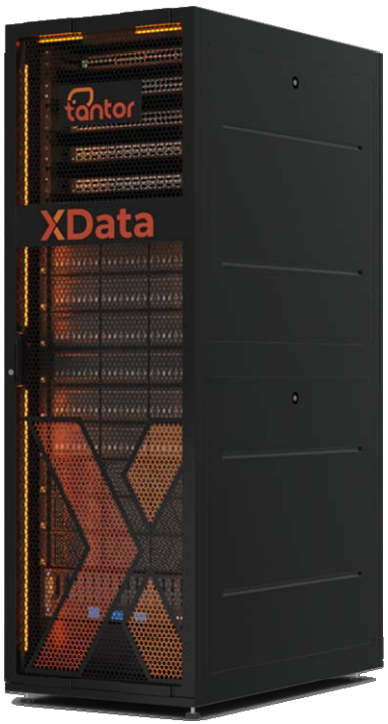
СУБД в составе Tantor xData Gen3 совместима с приложениями 1С

В составе Tantor xData

Максимальные сборки СУБД, оптимизированнаые для работы с 1С

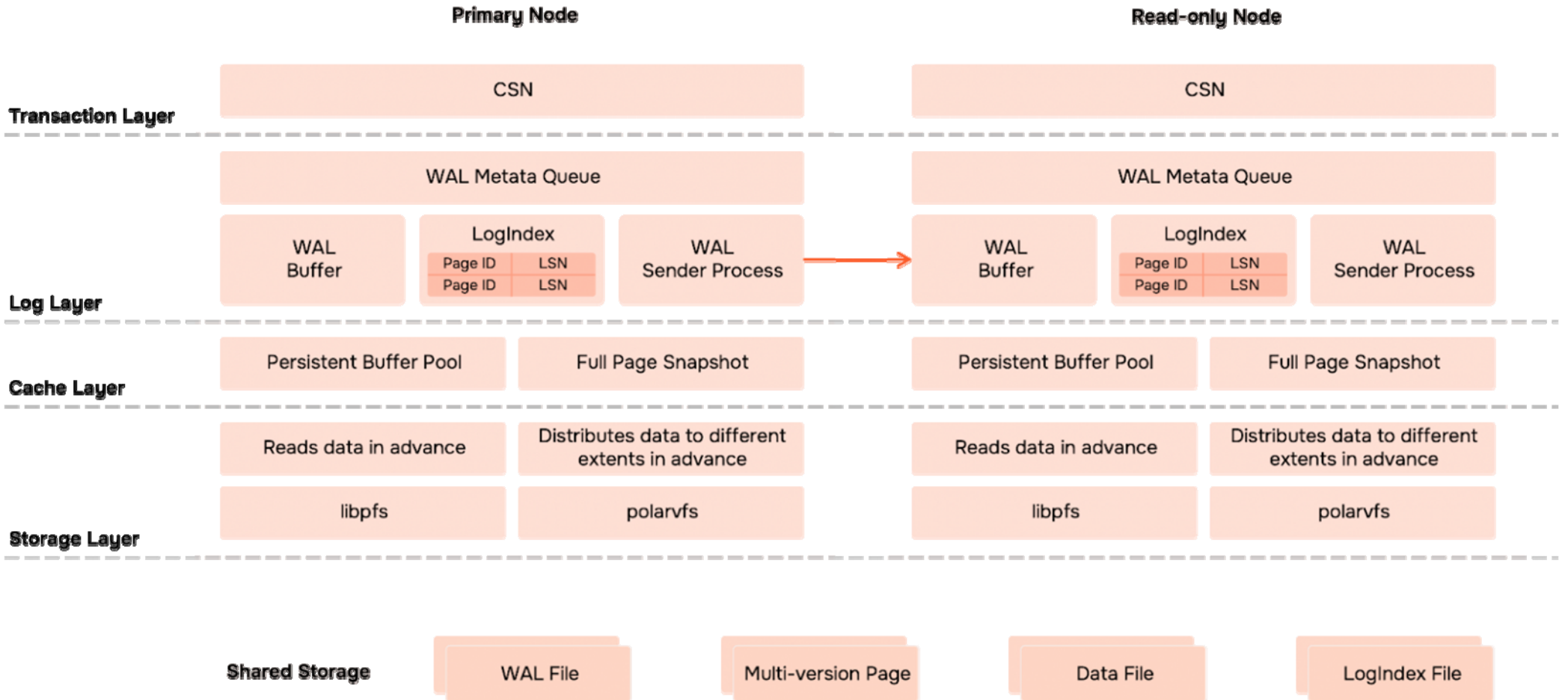
Tantor XData

- Программно-Аппаратный Комплекс, с высокой производительностью, отказоустойчивостью, безопасностью
- версия 2A, 2Y - процессоры AMD EPYC и x86-64
- версия 2B - процессоры Baikal-S
- Gen3 - включает Tantor Polar
- высокая производительность и масштабируемость
- снижение затрат на инфраструктуру и администрирование
- в состав входит СУБД Tantor Postgres и Платформа Tantor



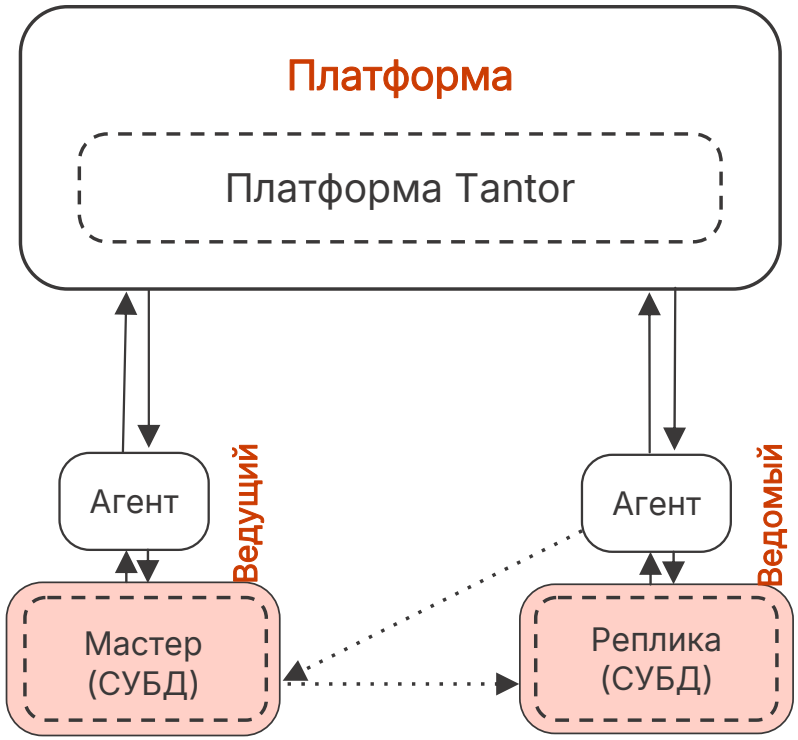
Tantor Polar

- несколько экземпляров обслуживают одну СУБД



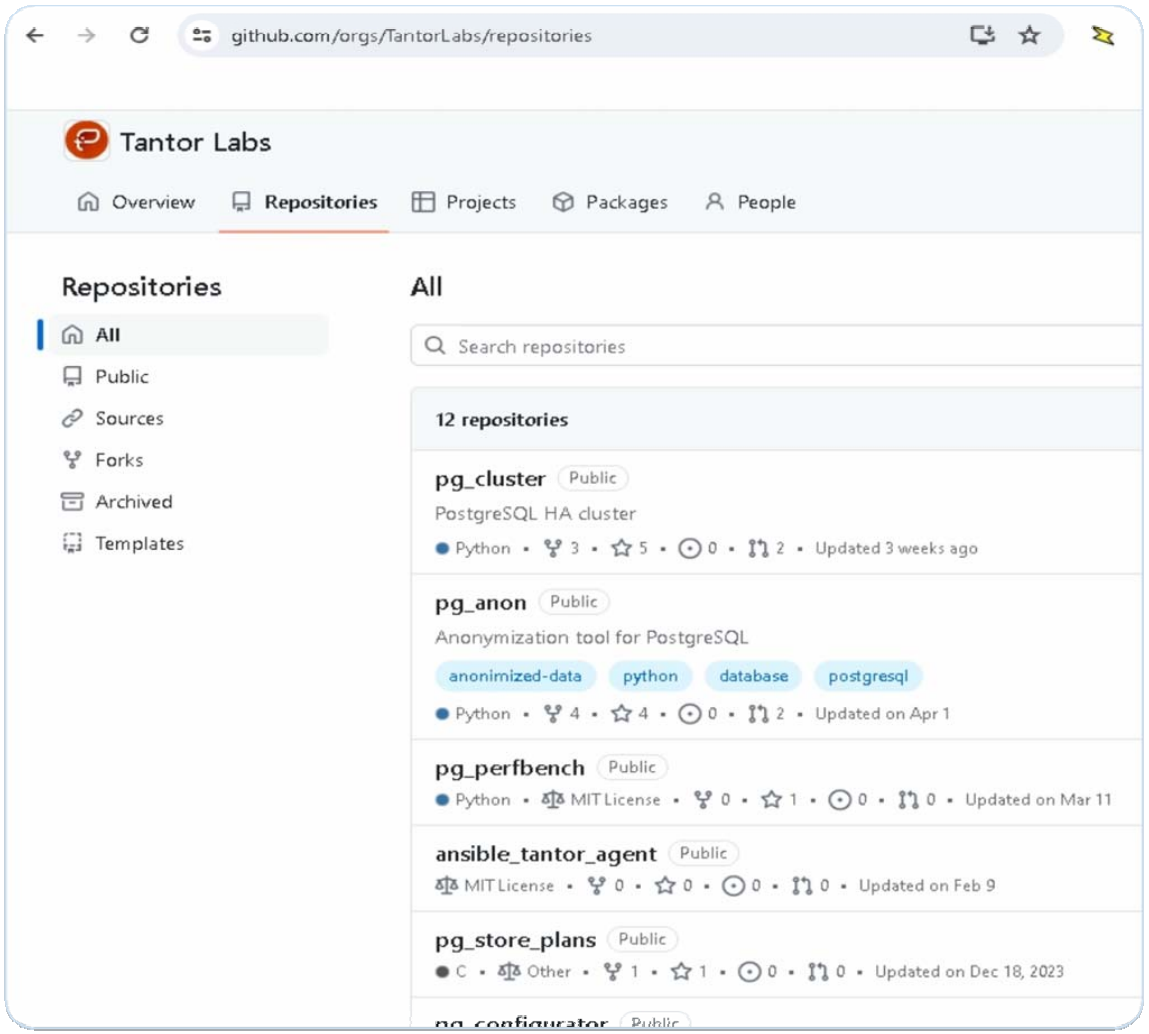
Платформа Tantor

- программное обеспечение для управления большим количеством СУБД и кластеров Patroni
- управляет СУБД Tantor Postgres и форками PostgreSQL
- сбор показателей работы экземпляров PostgreSQL, хранение и обработка показателей, рекомендации по настройке производительности
- интеграция с почтовыми системами, службами каталогов, мессенджерами



Доработка расширений PostgreSQL

- 1. pg_cluster
- 2. pg_anon
- 3. pg_perfbench
- 4. ansible_tantor_agent
- 5. pg_configurator
- 6. pg_store_plans
- 7. ldap2pg
- 8. citus
- 9. wal-g
- 10. odyssey
- 11. plantuner
- 12. pg_orchestrator
- 13. pgtools
- 14. pipelinedb



Конференции PGBootCamp

- Тантор Лабс принимает участие в организации конференций
- Конференция PGBootCamp проводилась:
  - Москва 19 марта 2026
  - Екатеринбург 10 апреля 2025
  - Казань 17 сентября 2024
  - Минск 16 апреля 2024
  - Москва 5 октября 2023

1

1a

Установка

Предварительные требования

- PostgreSQL может работать под:
  - Linux, macOS, Windows, BSD, Solaris
- Tantor Postgres выпускается для Linux:
  - **RPM**: Redos 7.3, 8; AltLinux p10, p11; MSVSphere; Oracle Linux 8; Rocky 8, 9
  - **DEB**: Astra Linux Special Edition 4.7, 1.7, 1.8; Ubuntu 20, 22; Debian 10, 11, 12, 13
- с Astra Linux поставляется пакет tantor-free-server-18
- минимальные требования к оборудованию:
  - 4 ядра CPU, 4Гб RAM, 40Гб SSD

Проверка возможности установки

- Дистрибутивы распространяются в виде файла rpm и deb
- В дистрибутивах указаны зависимости
- список нужных пакетов можно получить командами apt, dpkg, rpm

```
apt satisfy postgresql-18
postgresql-18 : Depends: postgresql-client-18 (= 18.3-1.pgdg11+1) but it is
not going to be installed
 Depends: postgresql-common (>= 275~) but 246astra6+ci1 is to be installed
 Depends: libicu67 (>= 67.1-1~) but it is not installable
 Depends: libldap-2.4-2 (>= 2.4.7) but it is not installable
 Depends: libpq5 (>= 17~~) but 15.14-astra.se3.1 is to be installed
 Depends: libssl1.1 (>= 1.1.1) but it is not installable
 Depends: liburing1 (>= 0.7) but it is not installable
```

Инсталлятор

- проверяет возможные конфликты библиотек и предлагает команду для их устранения
- добавляет адрес репозитория в список apt и yum
- скачивает подходящий дистрибутив и выполняет установку, создание кластера, создание службы
- представляет собой текстовый скрипт:
  - можно ознакомиться с тем, какие действия выполняет инсталлятор
  - легко выяснить в чем ошибки и исправить их

Локальная установка

- Параметры инсталлятора `./db_installer.sh --help`
- при установке можно создать кластер
- дистрибутивы (пакеты rpm и deb) имеют стандартный формат, их можно разархивировать, выяснить какие изменения вносятся в операционную систему в процессе установки

```
wget https://public.tantorlabs.ru/db_installer.sh
chmod +x db_installer.sh
export NEXUS_URL="nexus-public.tantorlabs.ru"
apt update
./db_installer.sh --edition=be --major-version=18 --do-initdb
```

Процесс установки

- создается пользователь с именем postgres
- создается служба запуска кластера
- создаются директории:
  - `/opt/tantor/db/18`
  - `/var/lib/postgresql`
  - `/var/lib/postgresql/tantor-se-18/data`
  - `/var/run/postgresql`
  - `/usr/lib/systemd/system/tantor-se-server-18.service`

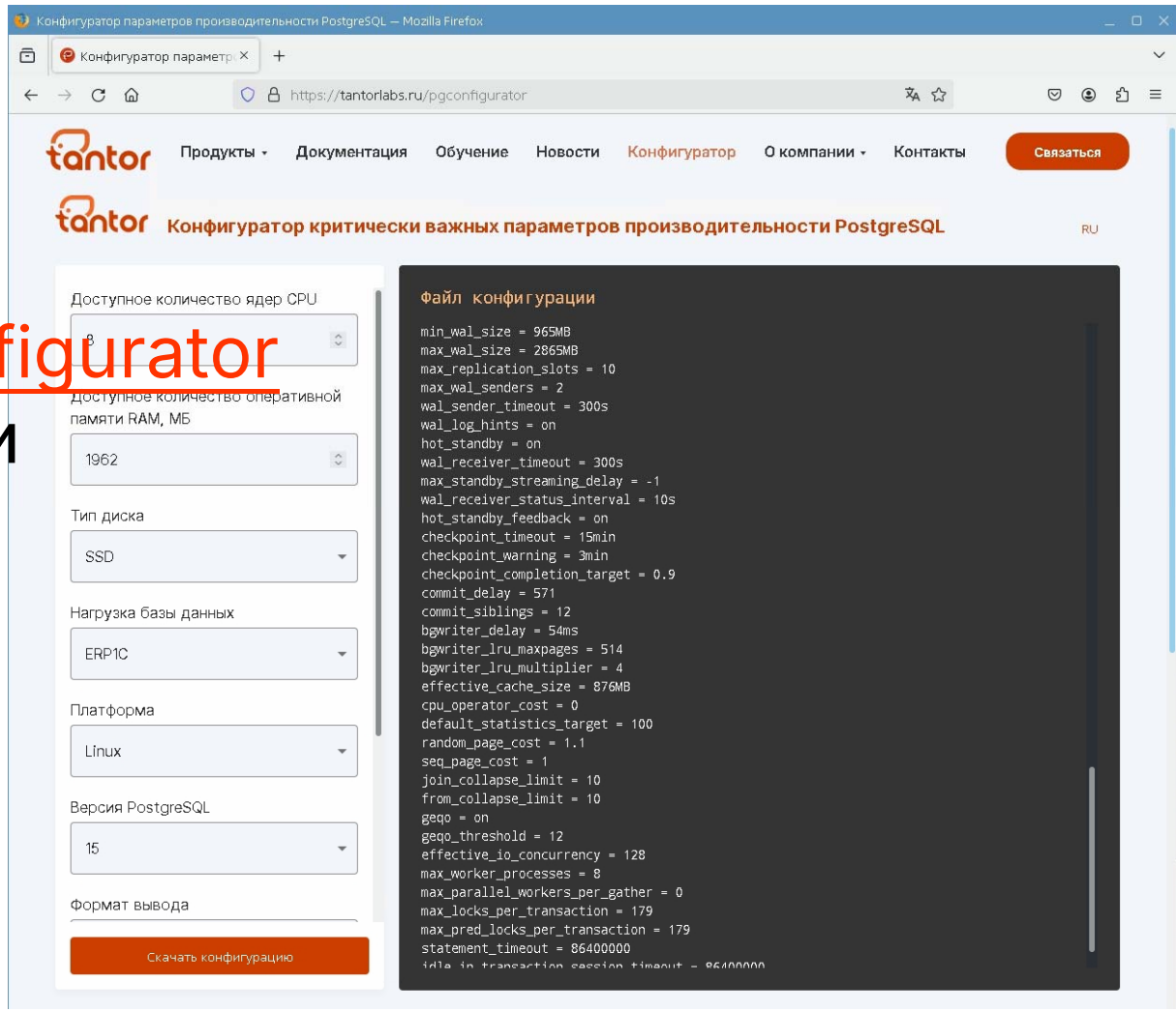
После установки

- Установить переменную окружения PGDATA в файл профиля пользователя postgres (`/var/lib/postgresql/.bash_profile`)
- установить начальные значения параметров конфигурации
- сконфигурировать программы управления и мониторинга (Платформа Тантор)

```
cat /var/lib/postgresql/.bash_profile
export PGDATA=/var/lib/postgresql/tantor-se-18/data
#export LC_MESSAGES=ru_RU.utf8
export PATH=/opt/tantor/db/18/bin:$PATH
```

Конфигураторы

- утилита pg_configurator
- веб-версия: http://tantorlabs.ru/pgconfigurator
- вводятся характеристики хоста и планируемой нагрузки
- выдает параметры конфигурации



Создание кластера утилитой initdb

- Кластер создается утилитой initdb
- Перед запуском утилиты нужно создать директорию для файлов кластера
- Выбрать настройки локализации
  - `--lc-collate` (LC_COLLATE) - правила сортировки текста
  - `--lc-ctype` (LC_CTYPE) - классификация символов (заглавные, прописные, цифры)
  - `--encoding` (из LOCALE) - схема кодирования символов

```
initdb -k -g --locale-provider=libc --encoding=UTF8 --locale=en_US.UTF8
--lc-collate=en_US.UTF8 --lc-ctype=en_US.UTF8 -c cluster_name='replica'
-c port=5433
```

Провайдеры локализации

- Для создания кластера и баз данных лучше выбирать кодировку UTF8
- Три провайдера: builtin, libc, icu
  - **libc** - провайдер по умолчанию, **icu** появился в 10 версии, **builtin** появился в 17 версии
- builtin некорректно сортирует буквы ё, Ё

```
 create database lab01builtin LOCALE_PROVIDER=builtin
BUILTIN_LOCALE='PG_UNICODE_FAST' TEMPLATE=template0;
 \c lab01builtin \\
 select string_agg(n, ' ') from (SELECT n FROM unnest(ARRAY['a', 'e',
'ё', 'Ж', 'я', 'Ё','E']) n ORDER BY n);
   string_agg
---------------
 Ё E Ж a e я ё
(1 row)
```

1

1b

Управление

Утилита управления экземпляром pg_ctl

- Основные действия:
  - start - запуск экземпляра
  - stop -m smart | fast | immediate - остановка
  - restart - перезапуск
  - reload - перечитывает файлы конфигурации без остановки экземпляра
  - status - показывает статус экземпляра
  - promote - преобразовать реплику в мастер
- запускается под пользователем postgres

Относительные пути и pg_ctl

- При запуске или перезапуске экземпляра не стоит использовать относительные пути в параметрах и переменных окружения:

```
postgres@tantor:~/tantor-se-18/data$ pg_ctl start -D .
server started
postgres@tantor:~/tantor-se-18/data$ cd ..
postgres@tantor:~/tantor-se-18$ pg_ctl stop
pg_ctl: directory "." is not a database cluster directory
postgres@tantor:~/tantor-se-18$ pg_ctl status
pg_ctl: directory "." is not a database cluster directory
postgres@tantor:~/tantor-se-18$ cd data
postgres@tantor:~/tantor-se-18/data$ pg_ctl status
pg_ctl: server is running (PID: 20205)
/opt/tantor/db/18/bin/postgres "-D" "."
```

Процесс postgres

- `pg_ctl` запускает процесс postgres, который порождает (fork) остальные процессы экземпляра
- параметром `-o` утилиты `pg_ctl` можно передавать процессу postgres параметры командной строки и параметры конфигурации кластера

```
pg_ctl start -o "--config_file=./postgresql.conf --work_mem=8MB"
pg_ctl start -o "-c config_file=./postgresql.conf -c work_mem=8MB"
```

- `postgres --single` однопользовательский и однопроцессный режим. Режим используют для исправления содержимого кластера в сложных случаях повреждений
- выхода из однопользовательского режима: `ctrl+d`

Управление экземпляром через systemctl

- управляет только экземплярами запущенными службой, экземпляр, запущенный `pg_ctl`, утилитой `systemctl` не останавливается
- `systemctl` использует `pg_ctl` для запуска экземпляра
- ожидает запуска экземпляра 300 секунд (параметр `TimeoutSec=300` в файле-описателе службы)

```
systemctl enable tantor-se-server-18
Created symlink /etc/systemd/system/multi-user.target.wants/tantor-se-server-18.service → /lib/systemd/system/tantor-se-server-18.service.
systemctl is-enabled tantor-se-server-18
enabled
systemctl start tantor-se-server-18
cat /lib/systemd/system/tantor-se-server-18.service | grep c=
TimeoutSec=300
systemctl status tantor-se-server-18
```

systemctl и pg_ctl

- управляет только экземплярами запущенными ей, экземпляр, запущенный `pg_ctl` утилитой `systemctl` не управляется:

```
postgres@tantor:~$ pg_ctl start
waiting for server to start.... done
server started
postgres@tantor:~$ sudo systemctl stop tantor-se-server-18
postgres@tantor:~$ pg_ctl status
pg_ctl: server is running (PID: 33223)
```

- `systemctl` использует `pg_ctl` для запуска экземпляра:

```
cat /lib/systemd/system/tantor-se-server-18.service | grep /pg_ctl
ExecStart=/opt/tantor/db/18/bin/pg_ctl start -D ${PGDATA} -s -w -t ${PGSTARTTIMEOUT}
ExecStop=/opt/tantor/db/18/bin/pg_ctl stop -D ${PGDATA} -s -m fast
ExecReload=/opt/tantor/db/18/bin/pg_ctl reload -D ${PGDATA} -s
```

Работа в контейнере docker

- изменяемые файлы, в частности PGDATA, должны лежать на томах (volumes)
- работа в контейнере не добавляет высокой доступности
- процесс postgres **не** должен иметь **PID=1**
- при создании и запуске контейнера нужно использовать параметр `docker run -d --init`

```
root@tantor:~# docker exec имя_контейнера ps
  PID USER      TIME COMMAND
    1 postgres  0:38 postgres
   31 postgres  0:09 postgres: logger
   32 postgres  0:45 postgres: checkpointer
   33 postgres  0:38 postgres: background writer
...
root@tantor:~# docker rm -f имя_контейнера
root@tantor:~# docker run --init -d -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=postgres -e
POSTGRES_INITDB_ARGS="--data-checksums" -e POSTGRES_HOST_AUTH_METHOD=trust -p 5434:5434 -e
PGDATA=/var/lib/postgresql/data -d -v /root/data:/var/lib/postgresql/data --name имя_контейнера postgres
```

Три режима остановки экземпляра

- `smart` - запрещает новые подключения и ждёт добровольного отсоединения существующих сессий. Этот режим не практичен
- `fast` - запрещаются новые подключения, всем серверным процессам отправляется сигнал прервать транзакции и завершиться. Это лучший выбор. Используется по умолчанию
- `immediate` - режим немедленного выключения

Остановка экземпляра

- Экземпляр можно остановить командой `pg_ctl stop`, независимо от того, каким способом был запущен экземпляр
- Использование `pg_ctl` - наиболее удобный и гарантированный способ погасить экземпляр
- Можно послать сигнал процессу postgres напрямую:
  `kill -INT  `head -1 $PGDATA/postmaster.pid``
- `kill -INT $(head -1 $PGDATA/postmaster.pid)`
- Не рекомендуется посылать сигнал `SIGKILL (9)` ни с одному процессу экземпляра
- Для остановки процесса используется функция `pg_terminate_backend(PID)`
- **`systemctl stop` не гарантирует остановки экземпляра**

Сообщения об остановке экземпляра

- В журнале кластера, при выполнении контрольных точек (параметр `log_checkpoints=on`), будут присутствовать сообщения типа:

  ```
  СООБЩЕНИЕ: начата контрольная точка: shutdown immediate
  ```

  или

  ```
  LOG: checkpoint starting: shutdown immediate
  ```

- Текст в сообщении "`shutdown immediate`" относится к свойствам контрольной точки, а не к режиму остановки экземпляра. При остановке экземпляра в режиме `immediate` (команда `pg_ctl stop -m immediate`) контрольная точка не выполняется
- в PostgreSQL нет команды shutdown immediate

Утилиты управления (обёртки команд SQL)

- находятся в директории `/opt/tantor/db/18/bin`
- путь к директории включен в переменную окружения `PATH` пользователя postgres в Linux
- Часть утилит командной строки - обёртки для команд SQL

```
postgres@tantor:~$ createdb db01 -e
SELECT pg_catalog.set_config('search_path', '', false);
CREATE DATABASE db01;
postgres@tantor:~$ createuser alice -e
SELECT pg_catalog.set_config('search_path', '', false);
CREATE ROLE alice NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT LOGIN
NOREPLICATION NOBYPASSRLS;
postgres@tantor:~$ vacuumdb --all --no-truncate --no-index-cleanup
--parallel=3 --freeze --disable-page-skipping
vacuumdb: vacuuming database "db01"
vacuumdb: vacuuming database "postgres"
vacuumdb: vacuuming database "template1"
```

Утилиты резервирования

- **pg_archivecleanup** - используется на репликах (резервных кластерах)
- **pg_basebackup** - создаёт физические бэкапы
- **pg_combinebackup** - накладывает инкрементальные бэкапы на полные
- **pg_dump**, **pg_dumpall**, **pg_restore** для логического резервирования (создания дампов)
- **pg_receivewal** - для создания поточных архивов WAL
- **pg_resetwal** для изменения размера WAL-сегментов
- **pgcopydb** - утилита для логического резервирования

Утилиты управления (другие)

- **pg_checksums** - включение/отключение подсчета контрольных сумм блоков данных и проверка блоков данных
- **pg_rewind** - синхронизация PGDATA кластеров
- **pg_upgrade** - обновление на новую версию PostgreSQL
- **pg_test_fsync** - измеряет скорость записи в WAL сегменты в разных режимах
- **pg_test_timing** - измеряет скорость и стабильность получения меток времени
- **pg_config** - информация о параметрах сборки PostgreSQL
- **pg_controldata** - показывает содержимое управляющего файла `$PGDATA/global/pg_control`
- **pgbench** - утилита нагрузочного тестирования

Утилиты управления (продолжение)

- **pg_isready** - проверка что кластер принимает соединения
- **oid2name** - удобная утилита для поиска к какому объекту относится файл или директория
- **pgcompacttable** - утилита для уменьшения размеров файлов таблиц
- **pg_repack** - относится к одноимённому расширению, реализующему аналог VACUUM FULL, только без монопольной блокировки
- **pg_ctl** - управляет экземпляром кластера, была рассмотрена ранее
- **initdb** - создает кластера, была рассмотрена ранее

1

1c

Утилита psql

Терминальный клиент psql

- позволяет:
  - интерактивно вводить команды SQL
  - из файла `psql -f скрипт.sql`
  - из командной строки `psql -c "select user;select now()"`
- файлы конфигурации
  - глобальный `/opt/tantor/db/18/etc/postgresql/psqlrc`
  - локальный `~/.psqlrc`
- описание параметров командной строки `psql --help`

```
postgres@tantor:~$ cat ~/.psqlrc
\set AUTOCOMMIT off
\set ON_ERROR_ROLLBACK interactive
\setenv PAGER 'less -XS'
\setenv PSQL_EDITOR '/usr/bin/mcedit'
\pset pager off
```

psql: подключение к базе данных

- `psql` подключается к одной базе данных из кластера баз данных
- Для подсоединения к базе нужно пройти аутентификацию
  - способы аутентификации рассматриваются в следующих главах
- Роль и пользователь синонимы и абсолютно одинаковы
- CREATE ROLE по умолчанию устанавливает атрибут NOLOGIN, команда CREATE USER устанавливает LOGIN
- использование пайпа (поточной передачи):

```
postgres@tantor:~$ createdb db02
postgres@tantor:~$ pg_dump -F c db01 | pg_restore -d db02
postgres@tantor:~$ pg_dump db01 | psql -q -d db02
```

Соединение с базой данных

- Параметры командной строки для соединения с базой данных:
- -U имя, по умолчанию имя пользователя операционной системы
- -d имя_базы, по умолчанию имя пользователя
- -h хост, по умолчанию /var/run/postgresql
- -p порт, по умолчанию 5432
- Переподсоединение:
  `\c имя_базы имя_пользователя хост порт`
- Параметры текущего соединения передаются символом "-"
  `\c - - localhost`
- Клиентская балансировка

```
psql --host=tantor,localhost load_balance_hosts=random --port=5432,5432
postgres=# \c - - localhost
You are now connected to database "postgres" as user "postgres" on host
"localhost" (address "127.0.0.1") at port "5432".
postgres=#
```

Параметры соединения

- Команда вывода деталей текущего соединения:

```
postgres=# \conninfo
           Connection Information
      Parameter       |        Value
----------------------+---------------------
 Database             | postgres
 Client User          | postgres
 Socket Directory     | /var/run/postgresql
 Server Port          | 5432
```

```
postgres@tantor:~$ psql -q postgres://user1:pass@localhost:5432/db01
db01=> \c - postgres /var/run/postgresql 5432
You are now connected to database "db01" as user "postgres".
db01=# \c - user1 - 5432
You are now connected to database "db01" as user "user1".
db01=> \c postgres://localhost:5432/db01?user=user1&password=pass
You are now connected to database "db01" as user "user1".
```

Получение справки по командам psql

- команды psql начинаются на обратный слэш ****
- параметры командной строки **psql --help**
- справка по командам psql **\?**
- список команд SQL **\h**
- текст команды, формируемых psql:

```
postgres=# \set ECHO_HIDDEN on
postgres=# \db
/******** QUERY *********/
SELECT spcname AS "Name",
  pg_catalog.pg_get_userbyid(spcowner) AS "Owner",
  pg_catalog.pg_tablespace_location(oid) AS "Location"
FROM pg_catalog.pg_tablespace
ORDER BY 1;
/************************/
       List of tablespaces
    Name    |  Owner   | Location
------------+----------+----------
 pg_default | postgres |
 pg_global  | postgres |
```

История команд и постраничный вывод

- если результат не помещается в окне терминала используется постраничный вывод (pageing)
  - используются утилиты операционной системы `more` или `less`
- утилита задаётся командой **`\setenv PAGER 'less -XS'`**
- постраничный вывод отключается командой **`\pset pager off`**
- в режиме постраничного вывода для less (после двоеточия): q - выход, z - вперёд, b - назад, h - помощь
- история команд и настройки хранится в файлах:

```
postgres@tantor:~$ psql -q
postgres=# \! ls .psql*
.psql_history  .psqlrc
postgres=# \! cat .psqlrc
\setenv PAGER 'less -XS'
\setenv PSQL_EDITOR '/usr/bin/mcedit'
\pset pager off
```

Форматирование вывода в psql

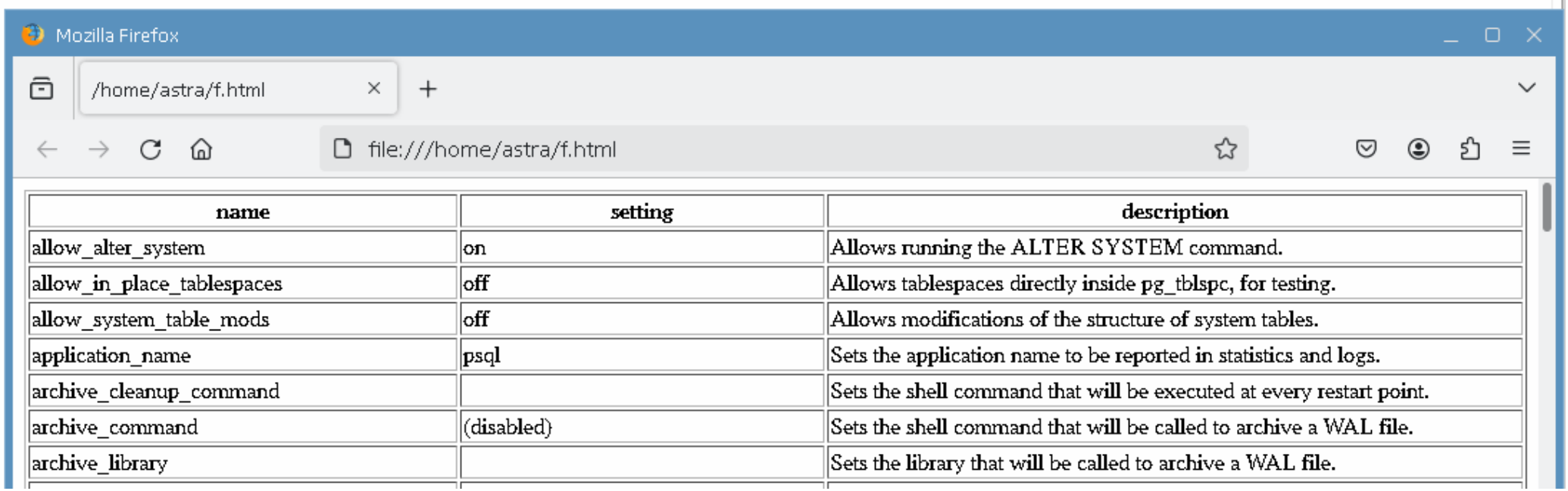
- Можно настроить формат вывода параметрами `\pset` и переключателями `\a \t \x`
- посмотреть текущие настройки форматирования: `\pset`
- отображение времени выполнения команды `\timing on`

```
postgres=# select * from pg_am limit 1;
 oid | amname |       amhandler        | amtype
-----+--------+------------------------+--------
   2 | heap   | heap_tableam_handler | t
(1 row)
postgres=# \pset linestyle unicode
Line style is unicode.
postgres=# select * from pg_am limit 1;
 oid │ amname │       amhandler        │ amtype
─────┼────────┼────────────────────────┼────────
   2 │ heap   │ heap_tableam_handler │ t
(1 row)
```

Вывод результата запроса в формате HTML

- можно выводить результаты запросов в формате HTML
  - **`psql -html`** или **`psql -H`**
  - или **`\pset format html`**

```
astra@tantor:~$ psql -c "show all;" -H -o f.html | xdg-open f.html
astra@tantor:~$
```



name	setting	description
allow_alter_system	on	Allows running the ALTER SYSTEM command.
allow_in_place_tablespaces	off	Allows tablespaces directly inside pg_tblspc, for testing.
allow_system_table_mods	off	Allows modifications of the structure of system tables.
application_name	psql	Sets the application name to be reported in statistics and logs.
archive_cleanup_command		Sets the shell command that will be executed at every restart point.
archive_command	(disabled)	Sets the shell command that will be called to archive a WAL file.
archive_library		Sets the library that will be called to archive a WAL file.

Приглашение к вводу команд (промпт) psql

- Приглашение (промпт) можно менять командами **\set PROMPT1**
- "*****" открыта и не зафиксирована транзакция
- "**'**" символ, парный которому нужно ввести, чтобы прекратить ввод
- "**-**" вторая и последующие строки (PROMPT2)
- "**!**" транзакция в состоянии сбоя и может только откатиться, даже если набрать **commit;**

```
postgres@postgres =# begin;
BEGIN
postgres@postgres *=# select '
postgres@postgres *'# abcd'
postgres@postgres *-# name;
 name
------
     +
 abcd
(1 row)
postgres@postgres *=# sele;
ERROR:  syntax error at or near
LINE 1: sele;
        ^
postgres@postgres !=# commit;
ROLLBACK
```

Переменные окружения, на которые реагирует psql

- Переменные окружения операционной системы, на которые реагирует psql
- Популярные переменные: `PGUSER`, `PGDATABASE`, `PGHOST`, `PGPORT`, `PSQL_EDITOR`
- Можно устанавливать интерактивно или в файлах параметров (`~/.psqlrc`)

```
postgres@tantor:~$ export PGUSER=user1
postgres@tantor:~$ export PGDATABASE=db01
postgres@tantor:~$ psql -q
db01=> select user;
 user
-------
 user1
(1 row)
```

Переменные psql \set

- устанавливаются командой **\set имя значение**
- Срок жизни до выхода из psql или до выполнения команды **\unset имя**

```
postgres=# \set test1 'select user'
postgres=# :test1;
   user
----------
 postgres
(1 row)
```

Выполнение команд в psql

- посмотреть переменные psql: команда **\set**
- очистка буфера набранных команд **\r**
- просмотр буфера или последней команды если буфер пуст **\p**
- **;** завершает команду SQL
- **\gx** завершает команду SQL, результат будет выведен в расширенном формате

```
postgres=# select 'select 123 c' abc\gset
postgres=# :abc\gx
-[ RECORD 1 ]
c | 123
postgres=# set my.abcd = 123;
postgres=# show my.abcd\gx
-[ RECORD 1 ]
my.abcd | 123
```

```
postgres=# select '
postgres'# \r
postgres'# '
postgres-# \r
postgres=# select (
postgres(# \r
postgres=# select "
postgres"# \r
postgres"# "
postgres-# \r
```

Параметр ON_ERROR_ROLLBACK

- Делает интерактивныйю работу с транзакциями в psql удобной
- Ошибка в команде не переводит транзакцию в состояние сбоя, а откатывает ошибочную команду и транзакция продолжается

```
postgres=#
\set ON_ERROR_ROLLBACK off
postgres=# begin transaction;
BEGIN
postgres=*# slect 1;
ERROR:  syntax error at or near
"slect"
postgres=!# commit;
ROLLBACK
```

```
postgres=#
\set ON_ERROR_ROLLBACK interactive
postgres=# begin transaction;
BEGIN
postgres=*# slect 1;
ERROR:  syntax error at or near
"slect"
postgres=*# commit;
COMMIT
```

Автоматическая фиксация транзакций

- по умолчанию psql работает в режиме автоматической фиксации транзакции (AUTOCOMMIT):

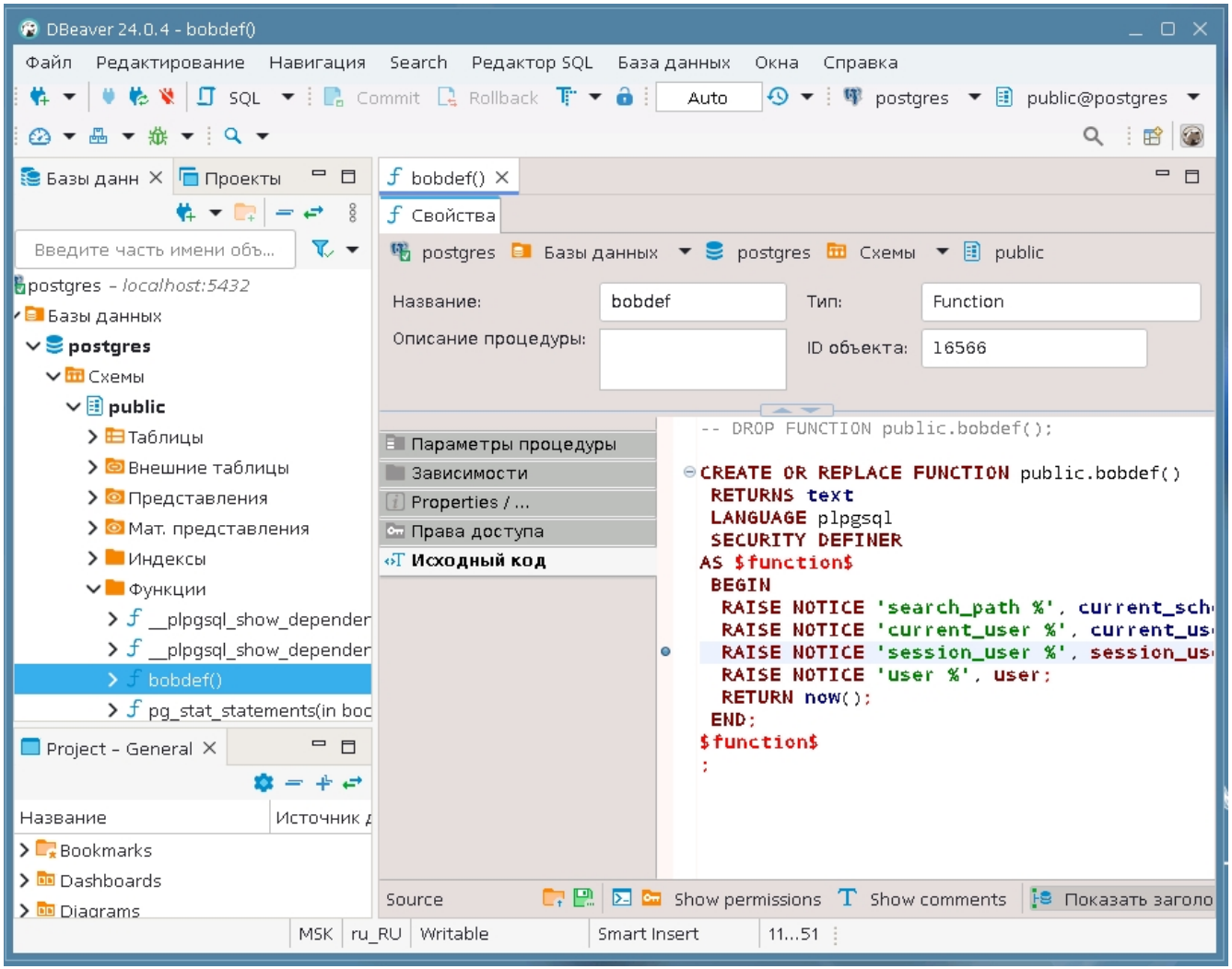
```
postgres=# \set ON_ERROR_ROLLBACK interactive
postgres=# \set AUTOCOMMIT off
postgres=# create table t (c text);
CREATE TABLE
postgres=*# vacuum t;
ERROR:  VACUUM cannot run inside a transaction block
postgres=*# select * from t\gx
(0 rows)
postgres=*# rollback;
ROLLBACK
postgres=*# \d t
Did not find any relation named "t".
```

Выполнение командных файлов в psql

- `\! linux_command` - выполнить команду операционной системы
- `\o file.sql` - перенаправить вывод в файл
- `\o` - вернуть вывод на экран
- `\i file.sql` - выполнить команды из файла
- Еще примеры как выполнить команды из файла:
  ```
  psql < file.sql
  psql -f file.sql
  ```
- Выполнить каждую строку, формируемую запросом SELECT как команду в psql:
  ```
  SELECT 'checkpoint;' \gexec
  ```

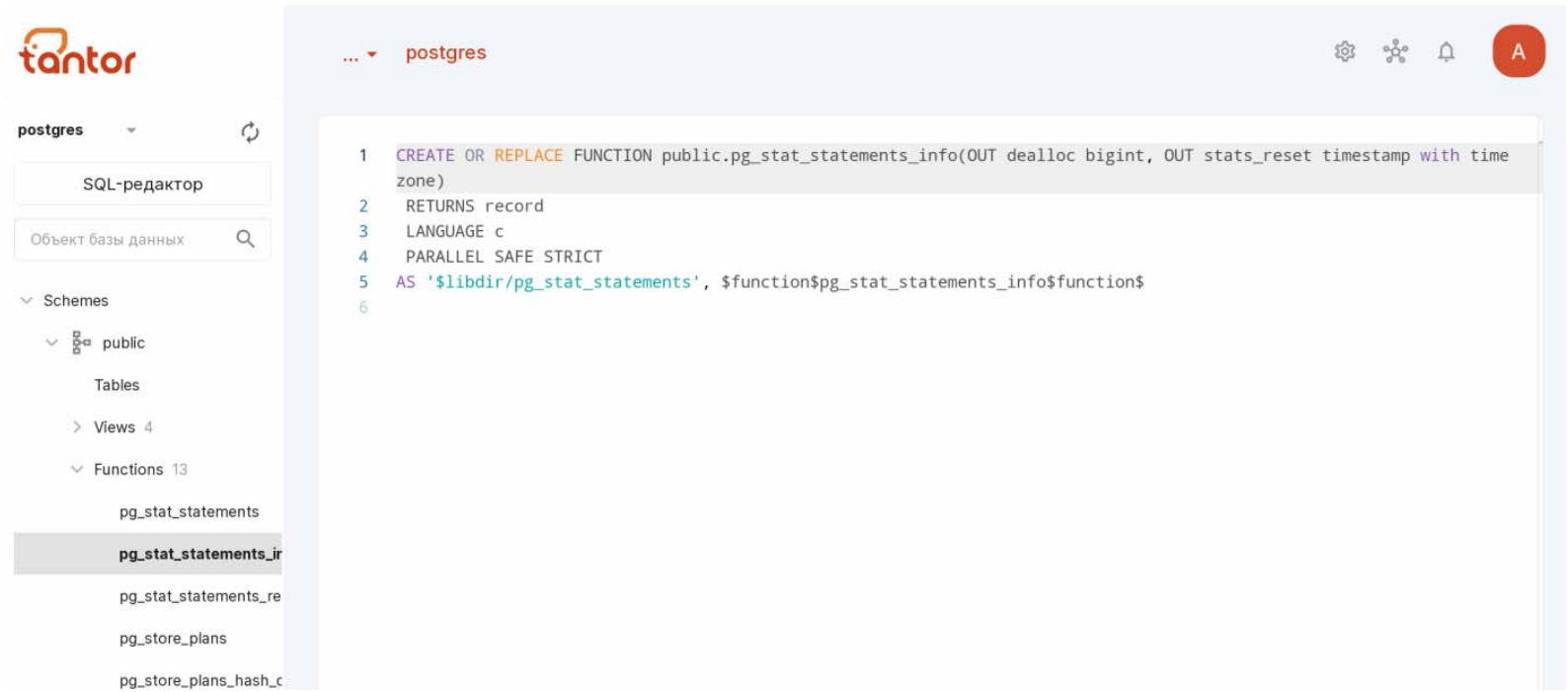
Графические приложения: DBeaver

- имеет свободно распространяемую версию
- написан на java



Графические приложения: Платформа Tantor

- приложение для администрирования и мониторинга большого числа кластеров PostgreSQL и кластеров Patroni
- приложение класса Enterprise Manager



Демонстрация

- Скачивание инсталлятора
- Установка разрешения на исполнение инсталлятора
- Установка адреса расположения дистрибутивов
- Установка с созданием базы данных
- Проверка, что кластер работает
- Остановка служб
- Деинсталляция

Практика

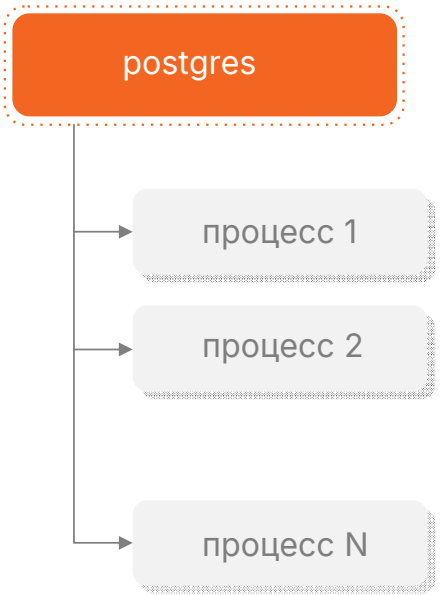
1. Создание кластера
2. Создание кластера утилитой initdb
3. Режим одного пользователя
4. Передача параметров экземпляру в командной строке
5. Локализация
6. Однобайтные кодировки
7. Использование утилит управления
8. Настройка терминального клиента psql
9. Использование терминального клиента psql
10. Восстановление сохраненного кластера

2

2a

Архитектура PostgreSQL

Экземпляр PostgreSQL

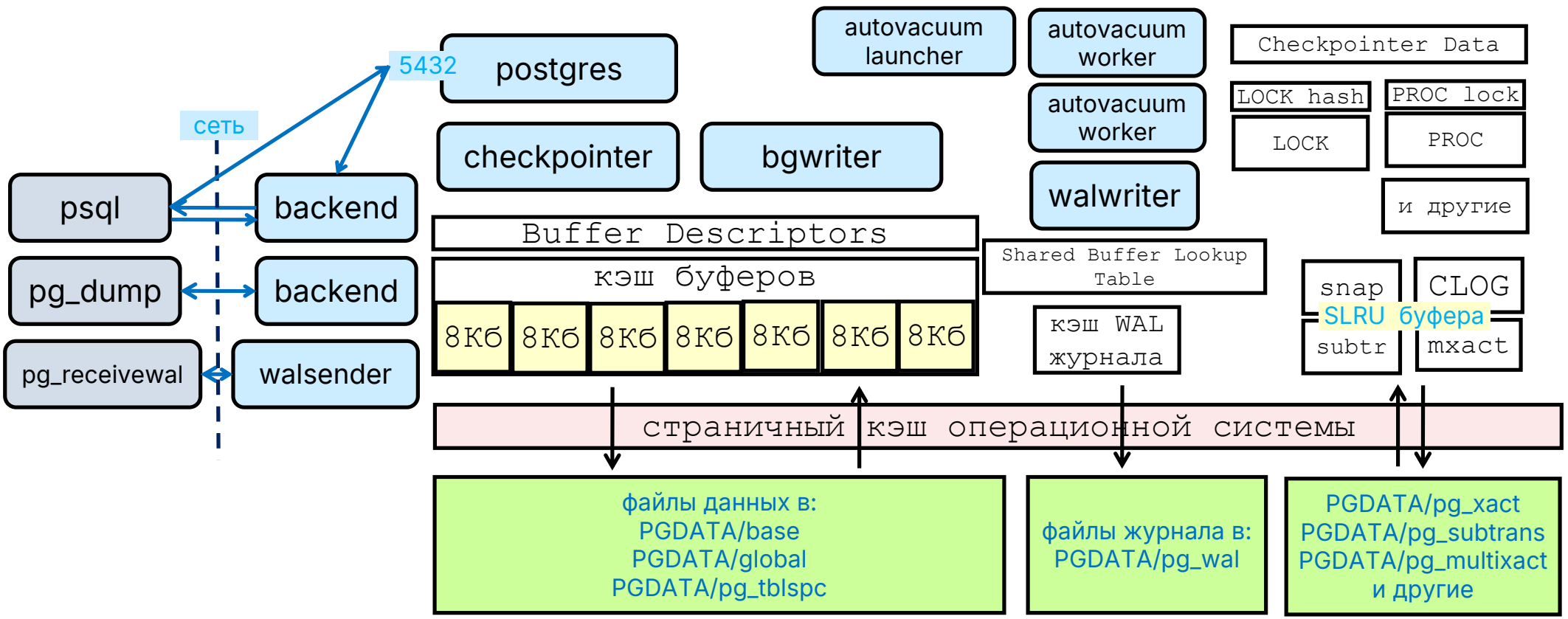


- экземпляр PostgreSQL - процесс postgres, порожденные им процессы операционной системы, память, которую используют эти процессы
- пример процессов:

```
postgres@tantor:~$ ps -eLo ppid,pid,cmd | egrep 'PPID|postgres'
   PPID     PID CMD
      1  743184 /opt/tantor/db/18/bin/postgres                  основной процесс
 743184  743185 postgres: logger                                процесс записи в журнал logging collector
 743184  743186 postgres: checkpointer                         фоновый процесс контрольной точки
 743184  743187 postgres: background writer                    процесс фоновой записи
 743184  743189 postgres: walwriter                            процесс фоновой записи в журнал
 743184  743190 postgres: autovacuum launcher                  процесс запуска автовакуума
 743184  743191 postgres: pg_stat_advisor BackgroundTaskManager процесс расширения
 743184  743192 postgres: autoprewarm leader                   процесс расширения pg_prewarm
 743184  743193 postgres: logical replication launcher         процесс запуска логической репликации
 644740  795748 psql -d demo -U alice -h /var/run/postgresql   клиент, утилита psql
 743184  795749 postgres: alice demo [local] idle              процесс, обслуживающий psql
```

Процессы экземпляра PostgreSQL

- процессы, память, кластер



Запуск экземпляра, процесс postgres

- запускается процесс postgres
- считываются файлы параметров и комбинируются с параметрами, переданными в командной строке
- проверяются разрешения на директорию PGDATA
- проверяется наличие управляющего файла `pg_control`
- выделяется память, загружаются разделяемые библиотеки
- в PGDATA создается файл postmaster.pid наличие которого и правильность номера процесса проверяется раз в минуту
- процесс postgres регистрирует серверные сокеты, создается файл UNIX-сокета
- читается файл с параметрами аутентификации `pg_hba.conf`
- запускается процесс startup и фоновые процессы

Запуск серверного процесса

- для каждой сессии создается серверный процесс
- Инициализируются (выделяются и заполняются) три кэша в локальной памяти серверного процесса:
  - Кэш для быстрого доступа к таблицам (RelationCache)
  - Кэш таблиц системного каталога (CatalogCache)
  - Кэш планов выполнения команд (PlanCache)
- выделяется память под менеджер "порталов" TopPortalContext
- клиент аутентифицируется
- догружаются разделяемые библиотеки, указанные в параметрах `session_preload_libraries` и `local_preload_libraries`
- выделяется память MessageContext для текста команд и процесс готов к приему команд

Общая память процессов экземпляра

- большее 70 структур памяти
- размеры и список - в представлении `pg_shmem_allocations`
- размер части структур задаётся параметрами конфигурации

```
 select * from (select *, lead(off) over(order by off) - off as true_size from
pg_shmem_allocations) as a order by 1;
          name          |    off    |   size    | allocated_size | true_size
------------------------+-----------+-----------+----------------+-----------
 <anonymous>            |           |   4946048 |        4946048 |
 Archiver Data          | 147726208 |         8 |            128 |       128
...
 XLOG Recovery Ctl      |   4377728 |       104 |            128 |       128
                        | 148145024 |   2849920 |        2849920 |
(77 rows)
```

Кэш таблиц системного каталога

- выделяется в локальной памяти каждого процесса в контексте CacheMemoryContext
- при создании или удалении объекта, процесс зафиксировавший транзакцию, посылает сообщение в кольцевой буфер shmInvalBuffer разделяемой памяти
- буфер хранит до 4096 сообщений
- процессы потребляют сообщения и обновляют свои локальные кэши
- если процесс пропустит сообщения, то полностью очистит свой локальный кэш таблиц системного каталога (CatalogCache) и будет его заново заполнять

```
 select * from (select *, lead(off) over(order by off) - off as true_size
from pg_shmem_allocations) as a where name='shmInvalBuffer' order by 1;
       name       |    off    | size  | allocated_size | true_size
------------------+-----------+-------+----------------+-----------
 shmInvalBuffer | 219072256 | 86816 |          86912 |     86912
(1 row)
```

Представление pg_stat_slru

- в PGDATA есть поддиректории, в которых сохраняются служебные данные кластера
- для ускорения доступа на чтение-запись в файлы этих директорий используются кэши в разделяемой памяти экземпляра
- статистика используется для установки параметров конфигурации, задающих размеры SLRU-кэшей

```
select name, blks_hit, blks_read, blks_written, blks_exists, flushes, truncates from pg_stat_slru;
       name        | blks_hit | blks_read | blks_written | blks_exists | flushes | truncates
-------------------+----------+-----------+--------------+-------------+---------+-----------
 commit_timestamp  |        0 |         0 |            0 |           0 |     103 |         0
 multixact_member  |        0 |         0 |            0 |           0 |     103 |         0
 multixact_offset  |        0 |         3 |            2 |           0 |     103 |         0
 notify            |        0 |         0 |            0 |           0 |       0 |         0
 serializable      |        0 |         0 |            0 |           0 |       0 |         0
 subtransaction    |        0 |         0 |           26 |           0 |     103 |       102
 transaction       |   349634 |         4 |           87 |           0 |     103 |         0
 other             |        0 |         0 |            0 |           0 |       0 |         0
(8 rows)
```

Локальная память процесса

- доступна только одному процессу, поэтому блокировки для доступа к ней не нужны
- большая часть структур не занимает много памяти и интересны только для понимания алгоритмов работы процессов
- параметры, наиболее сильно влияющие на выделение локальной памяти процесса:
  - `work_mem` - выделяется для обслуживание узлов (шагов) плана выполнения (если шаги способны выполняться одновременно), в том числе каждым параллельным процессом. Вместе с параметром `hash_mem_multiplier` влияет на память, выделяемую каждым серверным и параллельным процессом.
  - `maintenance_work_mem` значение по умолчанию 64MB. Задает объем памяти, выделяемый каждым процессом (серверным, параллельным), участвующем в выполнении команд `VACUUM`, `ANALYZE`, `CREATE INDEX`, `ALTER TABLE ADD FOREIGN KEY`

Представление pg_backend_memory_contexts

- показывает память, выделенную серверным процессом, обслуживающим **текущую** сессию
- контекст памяти (memory contexts) - набор частей памяти (chunks), которые выделяются процессом для выполнения какой-то задачи
- для выполнения подзадачи может выделяться дочерний контекст
- Контексты образуют дерево (иерархию)
- в представлении иерархию отображают столбцы: name (название контекста памяти), parent (название родительского контекста памяти, level
- в столбце ident содержится детализация того, что хранится в контексте

```
select sum(total_bytes), sum(used_bytes), sum(free_bytes) from pg_backend_memory_contexts;
   sum    |   sum    |  sum
---------+----------+--------
 2223144 | 1560928 | 662216
```

Функция pg_log_backend_memory_contexts(PID)

- начиная с 17 версии у команды EXPLAIN есть опция memory (по умолчанию отключена), которая выдает сколько памяти использовал планировщик и общую память серверного процесса
- память чужих сессий можно вывести в диагностический лог кластера функцией:

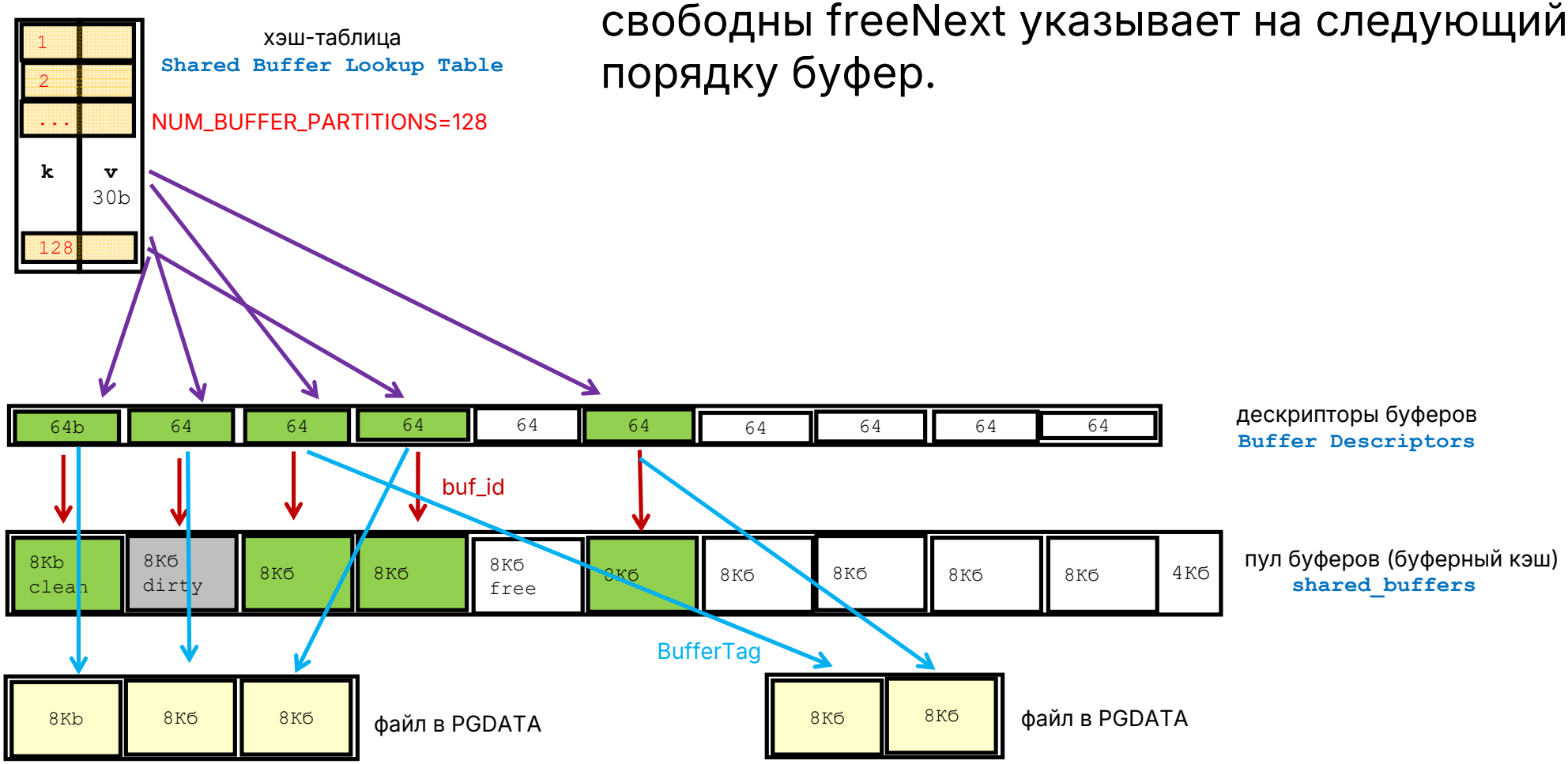
```
postgres=# SELECT pg_log_backend_memory_contexts(pg_backend_pid());

LOG: logging memory contexts of PID 111
LOG: level: 1; TopMemoryContext: 99456 total in 5 blocks; 3072 free (8
chunks); 96384 used
LOG: level: 2; search_path processing cache: 8192 total in 1 blocks;
5656 free (8 chunks); 2536 used
...
LOG: Grand total: 1301048 bytes in 229 blocks; 346960 free (286
chunks); 954088 used
```

Структуры памяти, обслуживающие буферный кэш

- **`Buffer Blocks`** - сам буферный кэш
  - выделяется память по числу буферов*8192 байта плюс 4096 байт
  - параметр конфигурации `shared_buffers` задаёт число буферов
- **`Buffer Descriptors`** - описатели (заголовки) буферов
  - выделяется память по числу буферов * размер описателя (64 байта)
- в каждом описателе хранится:
  - адрес блока на диске в виде метки блока (BufferTag)
    - адрес блока содержит идентификаторы: табличное простанство, базы данных, файл, форк, номер блока от начала первого файла
  - адрес буфера в виде порядкового номера буфера в буферном кэше
- описатель связан 1:1 (один к одному) с буфером
- данных в 20 байтах которые занимает BufferTag достаточно (не нужно никуда обращаться), чтобы считать блок с диска

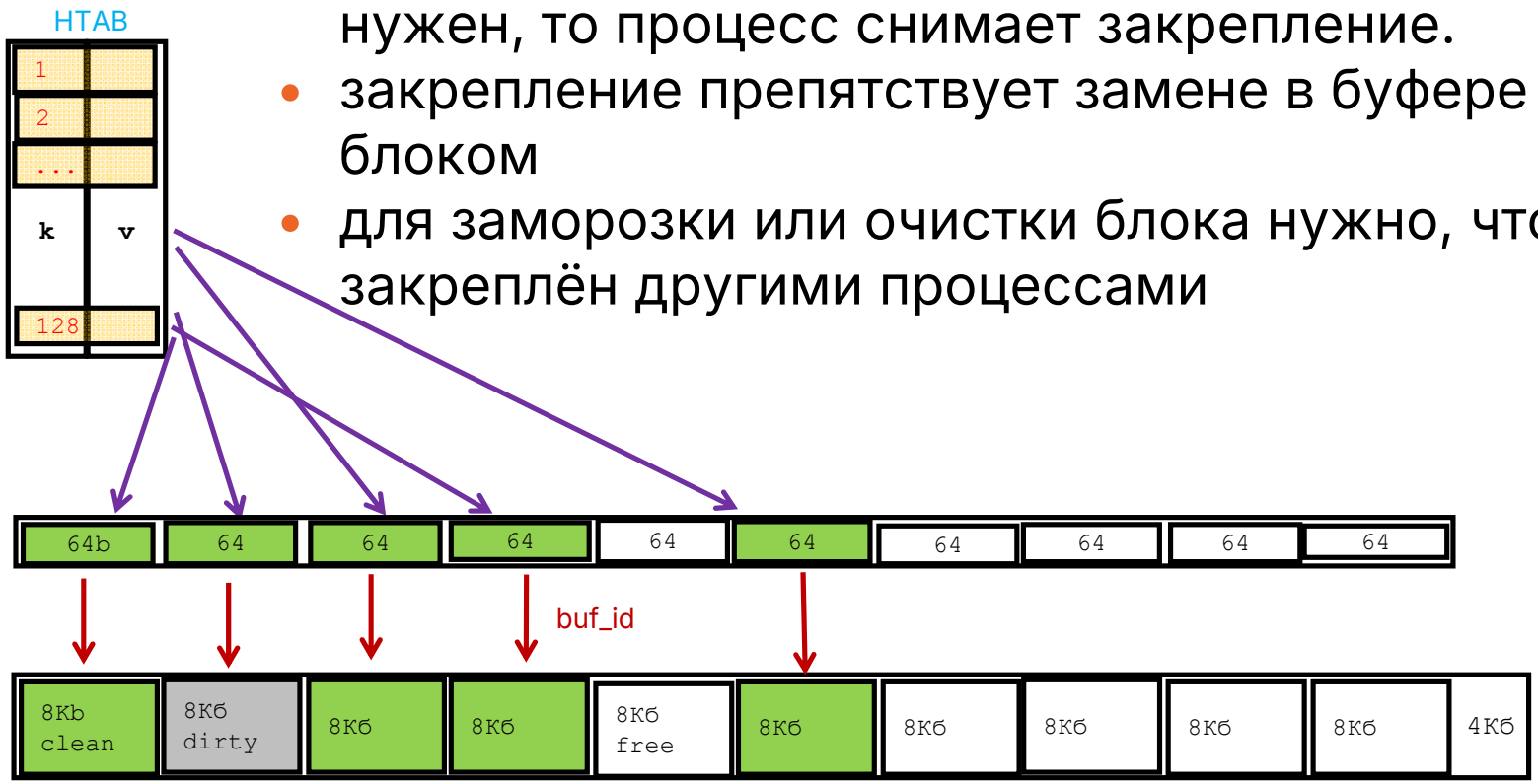
Структуры памяти, обслуживающие буферный кэш

- после запуска экземпляра, пока все буфера свободны freeNext указывает на следующий по порядку буфер.



Закрепление блока в буфере

- если процесс хочет работать с блоком, то он его ищет в буферном кэше. Если находит, то закрепляет. Множество процессов может закрепить буфер. Если процессу буфер не нужен, то процесс снимает закрепление.
- закрепление препятствует замене в буфере блока другим блоком
- для заморозки или очистки блока нужно, чтобы блок не был закреплён другими процессами



Закрепление буфера (pin) и блокировка content_lock

- закрепление (pin) используется для того, чтобы блок в буфере не был заменен на другой
- для чтения или изменения содержимого блока в буфере нужна легковесная блокировка content_lock, ссылка на которую сохраняется в описателе блока `Buffer Descriptors`
- блокировка должна удерживаться короткое время, в отличие от pin
- для удаления места занимаемого строкой (HOT cleanup или vacuum) после pin и Exclusive процесс дожидается, чтобы другие процессы незакрепляли блок в буфере
- для добавления в блок новой строки или изменения xmin, xmax существующих строк процесс должен получить блокировку content_lock типа Exclusive
- если процесс имеет pin и Shared content_lock, то он может менять некоторые биты в t_infomask, в частности статус фиксации/отката

Стратегии замены буферов

- методы замены блоков в буферном кольце (buffer cache replacement strategy):
  - BULKREAD. Для последовательного чтения блоков таблиц (Seq Scan) размер которых **не меньше 1/4 кэша** буферов используется набор буферов в буферном кэше размером 256Кб
  - VACUUM. Грязные страницы не убираются из кольца, а посылаются на запись. Размер кольца задается параметром конфигурации `vacuum_buffer_usage_limit`. По умолчанию 256Кб.
  - BULKWRITE. Используется командами COPY и CREATE TABLE AS SELECT. Размер кольца 16Мб.
- в буферном кэше блок может находиться только в одном буфере
- если буфер стал грязным, он исключается из буферного кольца

Поиск блока в буферном кэше

Процесс экземпляра:

- создает в своей локальной памяти экземпляр структуры BufferTag
- вычисляет 4-байтный хэш от BufferTag
- по значению хэша определяет номер партиции в хэш-таблице **`Shared Buffer Lookup Table`**
- запрашивает легковесную (LWLock) блокировку типа BufMappingLock партиции таблицы в которую попало значение хэша
- в хэш-таблице находит порядковый номер блока в кэше буферов или -1 если блока нет в кэше

Освобождение буферов при удалении файлов

- при удалении базы данных выполняется **полное сканирование всех дескрипторов буферов**
- полное сканирование дескрипторов выполняется, если размер удаляемого relation больше 1/32 пула буферов
- в остальных случаях поиск дескрипторов идет через хэш-таблицу
- при увеличении кэша буферов с 1Гб до 16Гб время на поиск буферов при удалении таблицы увеличивается на порядок
- поиск выполняется при удалении файлов в результате DROP, TRUNCATE и вакуума (если не отключено усечение файлов)

```
pgbench --file=CreateAndDrop.sql -j 1 -c 1 -T 10
TPS для HP 128MB 417
TPS для HP  1GB  375
TPS для HP  4GB  227
TPS для HP  8GB  127
TPS для HP 16GB   55
TPS для HP 18GB   33
```

CreateAdndDrop.sql:

```
create table x(id int);
insert into x values (1);
drop table x;
```

Процесс фоновой записи bgwriter

- процесс bgwriter записывает (writeback) грязные буфера и помечает их как чистые
- работа bgwriter снижает вероятность того, что процессы натолкнутся на грязные блоки при поиске буфера-кандидата (victim) на вытеснение (eviction) для замены другим блоком
- "поиск свободного блока" это обычно буфера-кандидата на вытеснение из буфера блока, так как все буфера обычно заняты и список свободных блоков пуст
- при вытеснении грязного блока из буфера обращения к шине ввода-вывода нет, это "writeback": копирование из памяти (буфер) в память (страничный кэш linux)

```
select name, setting, context, max_val, min_val from pg_settings where name ~ 'bgwr';
          name           | setting | context |  max_val   | min_val
-------------------------+---------+---------+------------+---------
 bgwriter_delay          | 200     | sighup  | 10000      | 10
 bgwriter_flush_after    | 64      | sighup  | 256        | 0
 bgwriter_lru_maxpages   | 100     | sighup  | 1073741823 | 0
 bgwriter_lru_multiplier | 2       | sighup  | 10         | 0
```

Очистка кэша буферов процессом bgwriter

- буфер не попадает в список свободных буферов, грязный буфер становится чистым
- на диск записываются только грязные, незакрепленные блоки с `usage_count=0`
- **bgwriter не меняет `usage_count`**
- при включенном подсчете контрольных сумм блок копируется из кэша буферов в локальную память процесса bgwriter и в локальной памяти подсчитывается и сохраняется в заголовке блока контрольная сумма

Контрольная точка

- выполняет процесс checkpointer:
  - периодически по истечении `checkpoint_timeout`
  - по разрастанию журнала `max_wal_size`
  - в конце процедуры остановки или запуска экземпляра
  - продвижении реплики
  - командам checkpoint, create database
  - при резервировании

```
postgres=# checkpoint;
CHECKPOINT
LOG:  checkpoint starting: immediate force wait
LOG:  checkpoint complete: wrote 5 buffers (0.0%);
```

Шаги выполнения контрольной точки

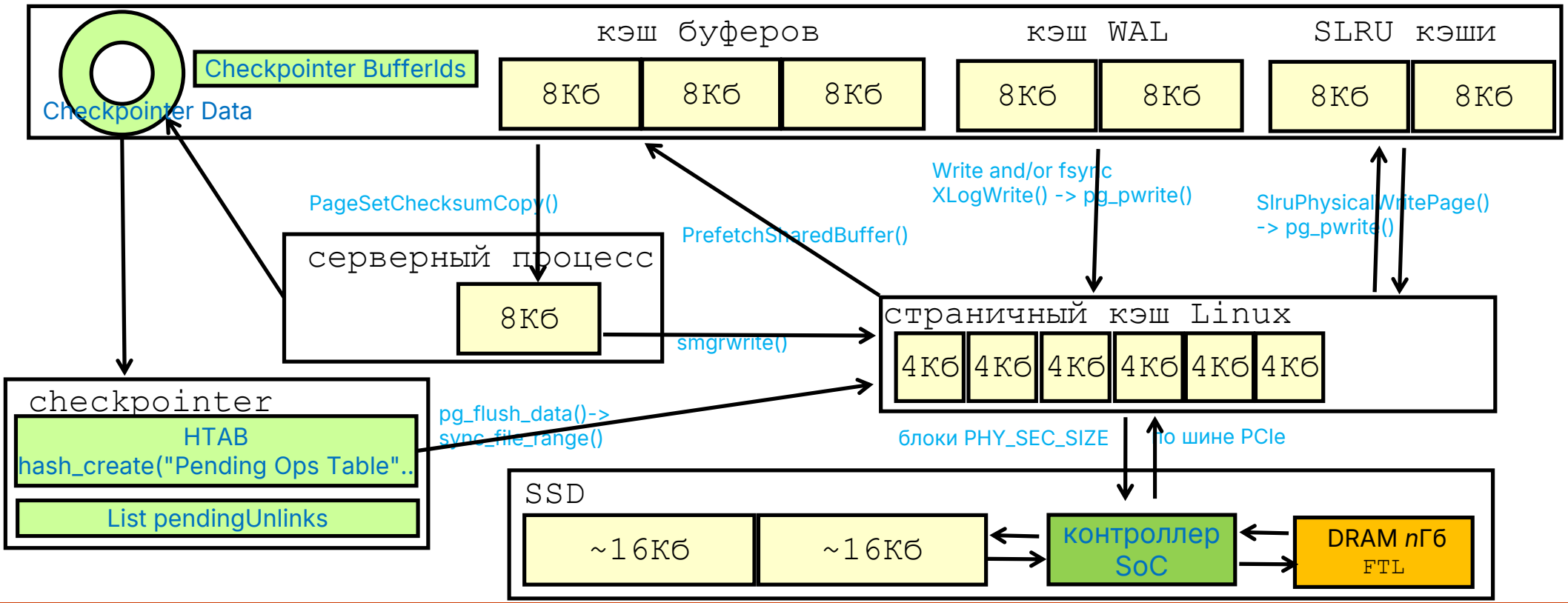
- если экземпляр останавливается, в файл `pg_control` записывается статус о начале гашения экземпляра
- вычисляется LSN следующей журнальной записи. Это будет LSN начала контрольной точки
- ждёт снятия процессами признака DELAY_CHKPT_START
- на диск сбрасываются slru буфера и другие структуры разделяемой памяти
- checkpointer в цикле пробегает все описатели буферов и для грязных блоков устанавливает флаг BM_CHECKPOINT_NEEDED, сохраняет адрес блока (5 чисел) в структуре памяти Checkpoint BufferIds для последующей сортировки
- после установки флага checkpointer не блокирует буфер и блок в буфере может быть сброшен на диск и заменен другим

Шаги выполнения контрольной точки

- сохраненные идентификаторы блоков сортируются в порядке: tblspc, relation, fork, block
- блоки посылаются по одному в страничный кэш linux
- Если checkpoint_flush_after не равен нулю, то выполняется синхронизация по уже отсортированным диапазонам блоков по каждому файлу
- в WAL сохраняется моментальный снимок со списком активных транзакций
- формируется журнальная запись окончания контрольной точки, содержащая LSN журнальной записи, которая была сформирована на момент начала контрольной точки
- в файле `pg_control` сохраняется LSN сформированной журнальной записи
- удаляются WAL-сегменты, которые не должны удерживаться

Взаимодействие процессов экземпляра с диском

- блоки по 8Кб считываются в разделяемую память через страничный кэш (блоки по 4Кб)
- блоки записываются на диск через страничный кэш
- для записи используется оптимизированный алгоритм синхронизации



Практика

1. Транзакция в psql
2. Список фоновых процессов
3. Буферный кэш, команда EXPLAIN
4. Журнал предварительной записи
5. Контрольная точка
6. Восстановление после сбоя

2

2b Архитектура PostgreSQL

Многоверсионность

Версии строк

- В блоках таблицы хранятся версии строк (rows), которые называют кортежами (tuple)
- Запросы (SELECT) должны выдавать данные на один момент времени ("консистентные"), что называют "целостность по чтению" (read consistency)
- Все версии строк физически хранятся в файлах таблицы по возможности рядом друг с другом (в тех же блоках файлов данных)

id	col1	col2	col3
1	a	b	c
2	b	c	a

tuple, строка, версия строки (синонимы)

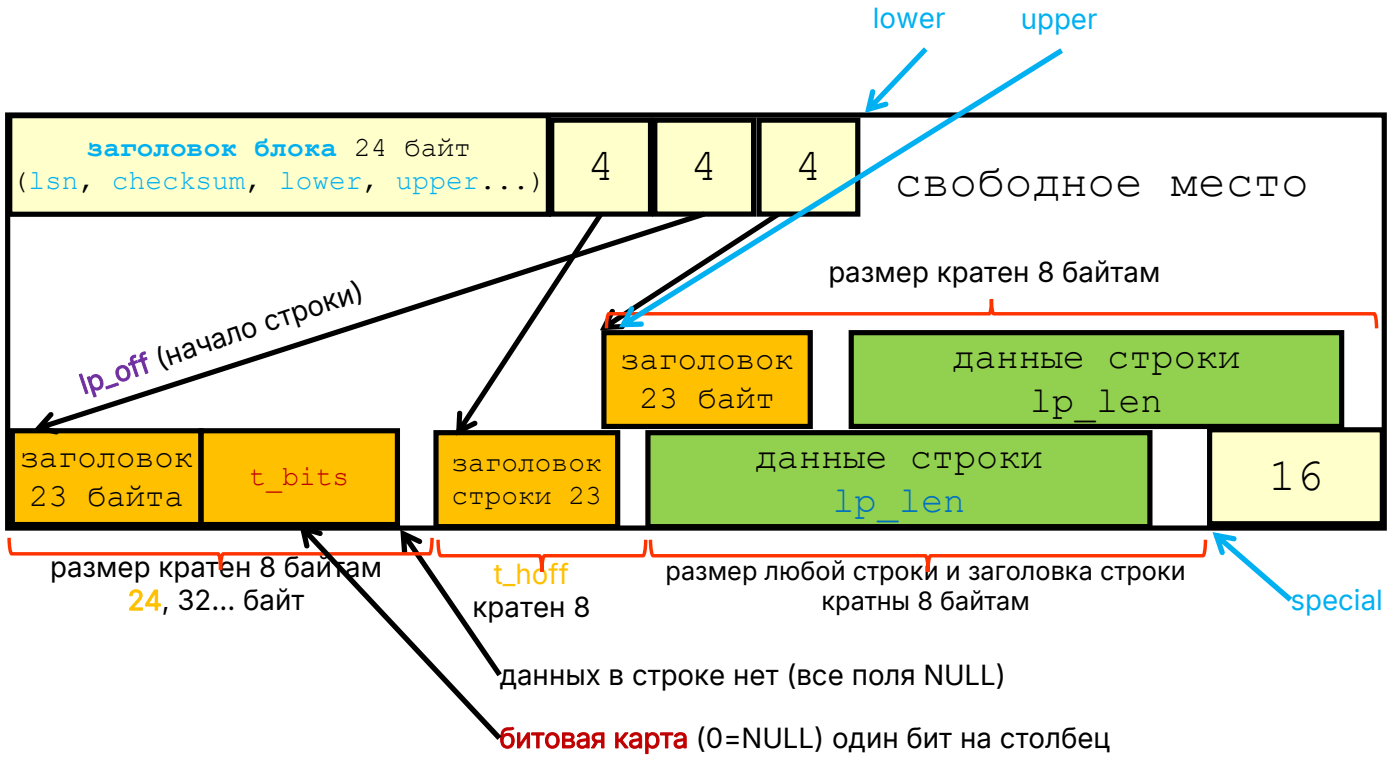
Таблицы

- объект в котором хранятся данные
- несколько видов: обычные таблицы (heap tables, строки хранятся неупорядоченно), нежурналируемые, временные, секционированные
- расширения могут создавать новые способы хранения данных и методы доступа к ним
- число и порядок следования столбцов задаются при создании таблицы
- после создания таблицы можно добавлять и удалять столбцы. При добавлении столбца он добавляется последним - после всех существующих столбцов
- можно поменять тип столбца

Служебные столбцы

- `xmin` - номер транзакции (xid), создавшей версию строки
- `xmax` - номер транзакции (xid), удаляющей или пытавшейся (транзакция не была зафиксирована по любой причине: вызван rollback, серверный процесс прерван) удалить строку или ноль
- `ctid` адрес физического расположения строки
- `tableoid` - oid таблицы, в которой физически содержится строка. Значения имеют смысл для секционированных и унаследованных таблиц
- `cmin` порядковый номер команды внутри транзакции начиная с нуля, создавшей версию строки
- `cmax` порядковый номер команды внутри транзакции начиная с нуля, удаляющей или пытавшейся удалить строку

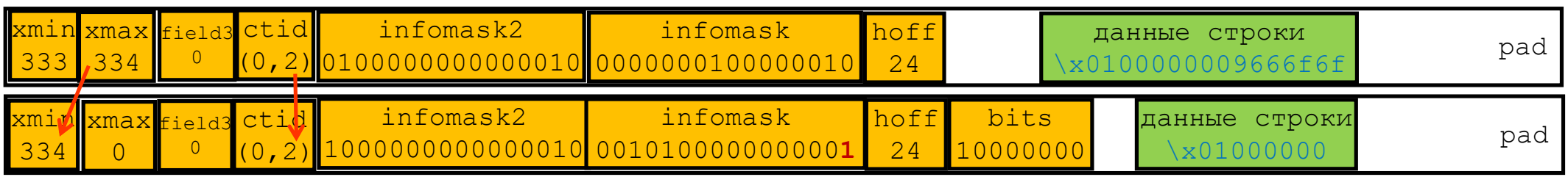
Структура блока данных




```
select * from page_header
(get_raw_page('t','main',0));
-[ RECORD 1 ]---------
lsn       | 0/110DCF10
checksum  | 0
flags     | 0
lower     | 928
upper     | 944
special   | 8176
pagesize  | 8192
version   | 5
prune_xid | 0
```

Заголовок версии строки

- xmin - xid транзакции, создавшей версию строки
- xmax - xid транзакции, удалившей версию строки
- ctid - ссылка на следующую версию этой строки
- bits - битовая карта пустых значений
- hoff - смещение до начала данных строки
- lp_off смещение до начала строки




```
select * from heap_page_items(get_raw_page('t','main',0));
lp|lp_off|lp_flags|lp_len|t_xmin|t_xmax|t_field3|t_ctid|t_infomask2|t_infomask|t_hoff| t_bits |t_oid
--+------+--------+------+------+------+--------+------+-----------+----------+------+--------+-----
 1|  8144|       1|    32|   333|   334|       0 | (0,2)|      16386|       258 |   24 |        |
 2|  8112|       1|    28|   334|     0|       0 | (0,2)|      32770|     10241 |   24 |10000000|
```

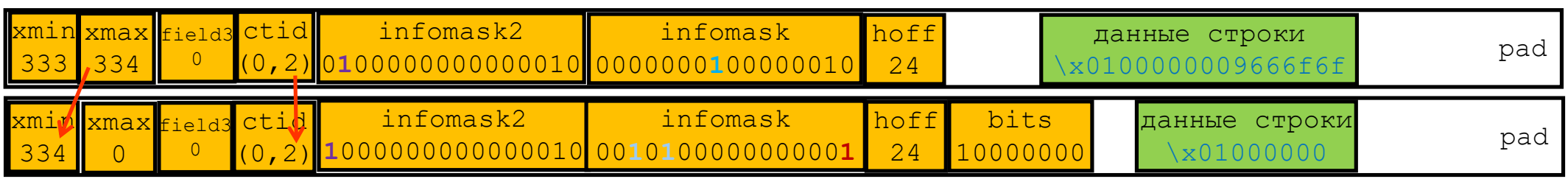
Вставка строки

- xmin - xid транзакции, вставившей строку
- xmax -ноль
- ctid - ссылка на себя
- транзакция, вставившая строку не помечает в infomask то, что она была зафиксирована. Это будет сделано при последующем обращении к строке в другой транзакции или другим запросом


```
select * from heap_page_items(get_raw_page('t','main',0));
lp|lp_off|lp_flags|lp_len|t_xmin|t_xmax|t_field3|t_ctid|t_infomask2|t_infomask|t_hoff| t_bits |t_oid
--+------+--------+------+------+------+--------+------+-----------+----------+------+--------+-----
 1|  8144|       1|    32|   333|     0|       0 | (0,1)|          2 |     2050 |   24 |        |
```

Обновление строки

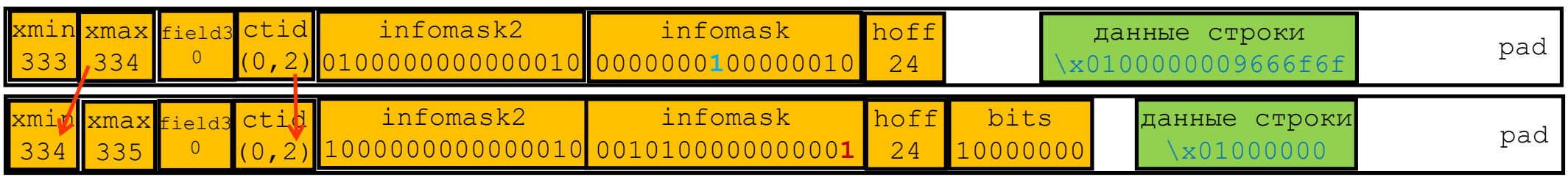
- ctid прежней версии указывает на адрес новой версии строки
- ctid актуальной версии строки указывает сам на себя
- xmax прежней версии меняется с нуля на номер транзакции, которая создала новую версию строки




```
select * from heap_page_items(get_raw_page('t','main',0));
lp|lp_off|lp_flags|lp_len|t_xmin|t_xmax|t_field3|t_ctid|t_infomask2|t_infomask|t_hoff| t_bits |t_oid
--+------+--------+------+------+------+--------+------+-----------+----------+------+--------+-----
 1|  8144|       1|    32|   333|   334|       0 | (0,2)|      16386|       258 |   24 |        |
 2|  8112|       1|    28|   334|     0|       0 | (0,2)|      32770|     10241 |   24 |10000000|
```

Удаление строки

- xmax актуальной версии строки (xmax=0) устанавливается в xid удаляющей транзакции




```
select * from heap_page_items(get_raw_page('t','main',0));
lp|lp_off|lp_flags|lp_len|t_xmin|t_xmax|t_field3|t_ctid|t_infomask2|t_infomask|t_hoff| t_bits |t_oid
--+------+--------+------+------+------+--------+------+-----------+----------+------+--------+-----
 1|  8144|       1|    32|   333|   334|       0 | (0,2)|      16386|      1282 |   24 |        |
 2|  8112|       1|    28|   334|   335|       0 | (0,2)|      40962|      8449 |   24 |10000000|
```

Типы данных наименьшего размера: `boolean`, `"char"`, `char`, `smallint`

- список типов данных и их характеристики есть в таблице `pg_type`
- если столбец будет использоваться для поиска, стоит оценить эффективность индексирования столбцов, составных индексов, эффективность сканирования индекса доступными способами (Bitmap Index Scan, Index Scan, Index Only Scan)
- Типы данных, занимающие наименьшее место:
  - `boolean` занимает 1 байт
  - `"char"` занимает 1 байт, хранит символы ASCII
  - `char` занимает 2 байта,
    - хранит символы в кодировке базы данных
  - `smallint`, занимает 2 байта
    - хранит целые числа от -32768 до 32767

Типы данных переменной длины

- для строк переменной длины стоит использовать тип text
- размерность для text не указывается
- занимаемое место:
  - один байт, если длина поля меньше 127 байт и строка пустая ''
  - если кодировка UTF8, то ASCII символы занимают 1 байт. Поэтому значение '1' займет 2 байта: \x0531. Значение '11' займёт 3 байта: \x073131. поле состоящее из буквы 'э' займет 3 байта: \x07d18d
  - если длина поля больше 126 символов, то заголовок поля станет 4 байта и поля будут выравниваться по 4 байта
- поля могут сжиматься и оставаться в блоке
- поля могут выноситься в TOAST, оставляя при этом в блоке 18 байт (не выравниваются)
- двоичные данные стоит хранить в типе данных bytea. Это тип данных переменной длины и его поведение такое же, как у типа text

Целочисленные типы данных

- целые числа можно хранить в типах `int(integer, int4), bigint (int8), smallint (int2)`
- обычно используются для столбцов PRIMARY KEY
- `bigint` выравнивается по 8 байт
- `int` для первичного или уникального ключа ограничит число строк в таблице 4млрд (2^32)
- для генерации значений для типов `smallint, int` и `bigint` используются последовательности и есть синонимы `smallserial(serial2), serial (serial4), bigserial (serial8)`
- для хранения чисел может использоваться тип переменной длины `numeric` (синоним `decimal`), **накладные расходы 4 байта на хранение длины поля**

Хранение дат, времени, их интервалов

- для хранения дат, времени, интервалов используются типы:
  - date (4 байта, с точностью до дня)
  - `timestamp, timestamptz, time` точность до микросекунды, размер одинаковый 8 байт, содержимое одинаковое
  - `timetz` - длина 12 байт, `interval` - длина 16 байт
- типы данных `timestamp, timestamptz` не хранят часовой пояс
- `timestamptz` приводит хранимое время к временной зоне клиента
- `timestamptz` физически хранит значения в UTC

```
create table t(t TIMESTAMP, ttz TIMESTAMPTZ);
insert into t values (CURRENT_TIMESTAMP, CURRENT_TIMESTAMP);
set timezone='UTC';
select t, ttz from t;
 2024-11-25 23:19:47.833968 | 2024-11-25 20:19:47.833968+00
update t set ttz=t;
select lp_off, lp_len, t_hoff, t_data from heap_page_items(get_raw_page('t','main',0)) order by lp_off;
 lp_off | lp_len | t_hoff |                t_data
--------+--------+--------+-----------------------------------
   8096 |     40 |     24 | \x70580939c1ca020070580939c1ca0200
   8136 |     40 |     24 | \x7044c4bcc3ca020070580939c1ca0200
select t, ttz from t;
 2024-11-25 20:19:47.833968 | 2024-11-25 20:19:47.833968+00
```

Типы данных для вещественных чисел

- фиксированной длины, с плавающей точкой, с округлением до 6 или 15 разрядов (значащих чисел в десятичном формате):
  - `float4 , real , float(1..24)` - в 4 байтах хранит не менее 6 разрядов
  - `float8 , float , double precision , float(25..53)` в 8 байтах хранит не менее 15 разрядов

```
select 12345678901234567890123456789.1234567890123456789::float4::numeric;
       12345700000000000000000000000
select 12345678901234567890123456789.1234567890123456789::float8::numeric;
       12345678901234600000000000000
```

- переменной длины, **без потери точности** при вычислениях:
  - `numeric , decimal`
  - точность можно задать параметрами: `numeric(presision,scale)`

```
select 1234567890123456789.123456789::numeric + 0.00000000000000000000123456789::numeric;
       1234567890123456789.12345678900000000000123456789
```

- пример экспоненциальной записи одинаковых чисел с разной мантиссой и порядком:

```
select 12345.6::float4, '12.3456e+03'::float4, '123.456e+02'::float4,'1234.56e+01'::float4;
1.235e+04 | 1.235e+04 | 1.235e+04 | 1.235e+04
```

Моментальный снимок

- представляет собой согласованное состояние базы данных на момент времени
- В снимок данных входит:
  - xmin - номер самой старой активной транзакции
  - xmax - значение, на единицу больше номера последней завершенной транзакции
  - xip_list - список активных транзакций
- Функция, возвращающая содержимое моментального снимка и функция экспортирующая его для другой сессии:

```
postgres=# BEGIN TRANSACTION;
postgres=*# select pg_current_snapshot();
 pg_current_snapshot
---------------------
 362:362:
postgres=*# select pg_export_snapshot();
     pg_export_snapshot
----------------------------
 00000024-0000000000000000A-1
```

Транзакция

- Транзакция - набор команд
- Начинается явно или неявно
- Завершается одним из двух действий: фиксация (команды COMMIT, END) или откатом (команда ROLLBACK)
- Результат прерванной (aborted) транзакции такой же, как явно отменённой командой ROLLBACK
- Пример неявного начала транзакции:

```
postgres=# do $$
begin
 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
 perform 1;
end $$;
ERROR: SET TRANSACTION ISOLATION LEVEL must be called before any query
CONTEXT: SQL statement "SET TRANSACTION ISOLATION LEVEL REPEATABLE READ"
PL/pgSQL function inline_code_block line 3 at SQL statement
```

```
postgres=# do $$
begin
 ROLLBACK;
 SET TRANSACTION
ISOLATION LEVEL REPEATABLE
READ;
 perform 1;
 ROLLBACK AND CHAIN;
 perform 1;
 COMMIT AND NO CHAIN;
end $$;
DO
```

Свойства транзакций

- Атомарность (Atomicity) - при фиксации выполнены все команды без исключений, при откате - ни одна команда не выполнена
  - после фиксации, изменения моментально становятся видны другим сессиям
- Целостность (consistency) - отсутствие нарушения декларативных ограничений целостности
- Изоляция (isolation) транзакций друг от друга
  - Реализуется одним из уровней изоляции
- Отказоустойчивость (Durability) - если клиент получил подтверждение об успешности фиксации транзакции (COMMIT COMPLETE), то может быть уверен, что результат транзакции не пропадет

Уровни изоляции транзакций

- READ UNCOMMITTED - чтение незафиксированных данных. В PostgreSQL не используется
- READ COMMITTED - чтение зафиксированных данных. Используется по умолчанию
- REPEATABLE READ - повторяемость по чтению. Стоит использовать только для чтения, а не изменения данных (READ ONLY)
- SERIALIZABLE - упорядоченное выполнение
  - на уровнях REPEATABLE READ и REPEATABLE READ, если менялись данные, возможна ошибка сериализации (serialization failure): "не могу сериализовать доступ" (can't serialize access), транзакция переходит в состояние сбоя и не может зафиксироваться

Феномены изоляции транзакций

- в стандарте ISO SQL-92 и последующих определено три феномена конкурентного доступа, которые должны отсутствовать на уровнях изоляции
- на всех уровнях не могут нарушаться ограничения целостности
- грязное чтение в PostgreSQL не допускается ни на одном из уровней изоляции, поэтому уровень Read uncommitted не отличается от Read committed

Уровень изоляции	Грязное чтение (P1)	Неповторяемое чтение (P2)	Фантомное чтение (P3)	Нарушение сериализации
Read uncommitted	Допускается, но не в PG	Возможно	Возможно	Возможно
Read committed	нет	Возможно	Возможно	Возможно
Repeatable read	нет	нет	Допускается, но не в PG	Возможно
Serializable	нет	нет	нет	нет

Пример ошибки сериализации

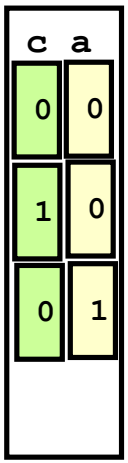
- Пример ошибки сериализации:

```
  postgres=# drop table if exists a;
DROP TABLE
postgres=# create table a (x int);
CREATE TABLE
1 postgres=# begin transaction isolation
level serializable;
BEGIN
3 postgres=*# insert into b select count(*)
from a;
INSERT 0 1
5 postgres=*# commit;
COMMIT
```

```
 postgres=# drop table if exists b;
DROP TABLE
postgres=# create table b (x int);
CREATE TABLE


 2 postgres=# begin transaction isolation level
serializable;
BEGIN
4 postgres=*# insert into a select count(*) from b;
INSERT 0 1

 6 postgres=*# commit;
ERROR:  could not serialize access due to read/write
dependencies among transactions
DETAIL:  Reason code: Canceled on identification as
a pivot, during commit attempt.
HINT:  The transaction might succeed if retried.
```

- В Oracle Database на уровне SERIALIZABLE ошибки нет

Статусы транзакций (CLOG)



- Журнал представляет собой битовый массив, в котором для каждой транзакции отведено два бита
- Значения битов:
  - 00 - транзакция в процессе
  - 01 - зафиксирована
  - 10- прервана (aborted, откачена)
  - 11 - подтранзакция зафиксирована, но родительская транзакция не завершена
- К CLOG обращаются процессы, в том числе вакуум, выполняющий заморозку версий строк, чтобы узнать статус транзакций
- Максимальный размер файлов CLOG зависит от параметра конфигурации autovacuum_freeze_max_age

Фиксация транзакции

- При фиксации транзакции выполняется запись в журнал транзакций (WAL) о фиксации транзакции
- Выполняется запись бита в буфер журнала CLOG
- Освобождаются ресурсы, которые использовались в процессе транзакции: блокировки, курсоры (кроме курсоров WITH HOLD), контексты (частей) локальной памяти процесса
- Файлы CLOG сохраняются в WAL в начале контрольной точки и изменения в них до следующей контрольной точки не журналируются. При восстановлении после сбоя содержимое CLOG восстанавливается по записям WAL

Подтранзакции

- подтранзакции это точки сохранения
  - используются чтобы откатиться, а не переводить транзакцию в состояние сбоя
- создаются
  - командой SAVEPOINT
  - секцией EXCEPTION в блоке на языке pl/pgsql
  - в psql при открытии транзакции при установке параметра `\set ON_ERROR_ROLLBACK interactive`
- в структуре PGPROC сохраняется до 64 номеров подтранзакций
- подтранзакциям, которые только читают данные присваивается виртуальный номер
- если встречается команда изменения данных, то подтранзакциям вплоть до основной транзакции присваиваются реальные номера

Типы блокировок

- spinlock (циклическая проверка)
  - Используются для очень краткосрочных действий - не дольше нескольких десятков инструкций процессора
  - Средств мониторинга нет
- легковесные (LWLocks)
  - Используются для доступа к структурам в разделяемой памяти
  - Имеют монопольный (на чтение и изменение) и разделяемый режим (чтение)
  - не больше 200 одновременно
- обычные (тяжеловесные)
  - Автоматически освобождаются по окончании транзакции
  - Есть несколько уровней блокировок
  - Обслуживают блокировки **12 типов**, в том числе advisory locks.
- предикатные блокировки (SIReadLock) используются транзакциями с уровнем изоляции SERIALIZABLE

Блокировки объектов

- При выполнении команд запрашивается блокировка на объекты, затрагиваемые командой
  - SELECT автоматически запрашивает блокировки ACCESS SHARE на таблицы, индексы, представления
- Пока блокировки не будут получены, команда не начнет выполняться
- Блокировки на уровне объектов используют "честную" очередь блокировок. Это означает, что блокировки будут обслуживаться в порядке их запроса независимо от уровней запрашиваемых блокировок и нет приоритетов
- параметром конфигурации lock_timeout можно установить максимальное время ожидания получения блокировки на объекты или строки

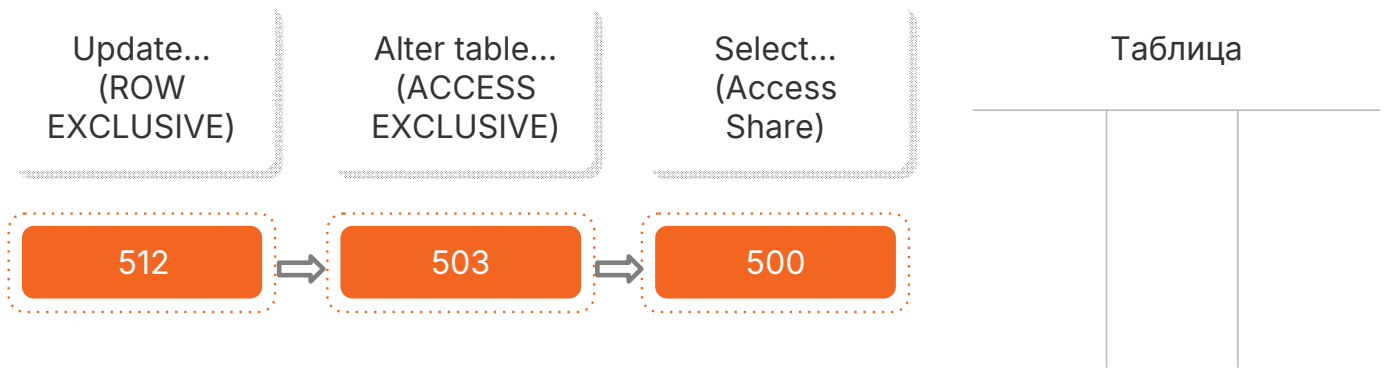
Совместимость блокировок

- слабые блокировки могут быть получены по быстрому пути
- сильные блокировки таблиц препятствуют установку слабых блокировок по быстрому пути
- автовакуум и автоанализ не мешают использовать быстрый путь
- автовакуум и автоанализ не блокируют команды, в случае конфликта блокировок рабочий процесс автовакуума прерывает обработку таблицы

	ACCESS SHARE	ROW SHARE	ROW EXCL.	SHARE UPDATE EXCL.	**SHARE**	**SHARE ROW EXCL.**	**EXCL.**	**ACCESS EXCL.**
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCL.				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

Блокировки объектов

- Блокировки на уровне объектов формируют очереди
- Уровень блокировок может быть совместим
- Очередь "честная" и транзакцию с высоким уровнем блокирования нельзя обогнать:



Блокировки строк

- FOR UPDATE - устанавливает DELETE и UPDATE, который меняет значение в столбце, входящим в уникальный индекс (не по выражению и не частичный), а также SELECT FOR UPDATE
- FOR NO KEY UPDATE устанавливают все остальные UPDATE
- Если не планируете удалить строку или менять значение в ключевом столбце, всегда используйте SELECT FOR NO KEY UPDATE
- Таблица совместимости:

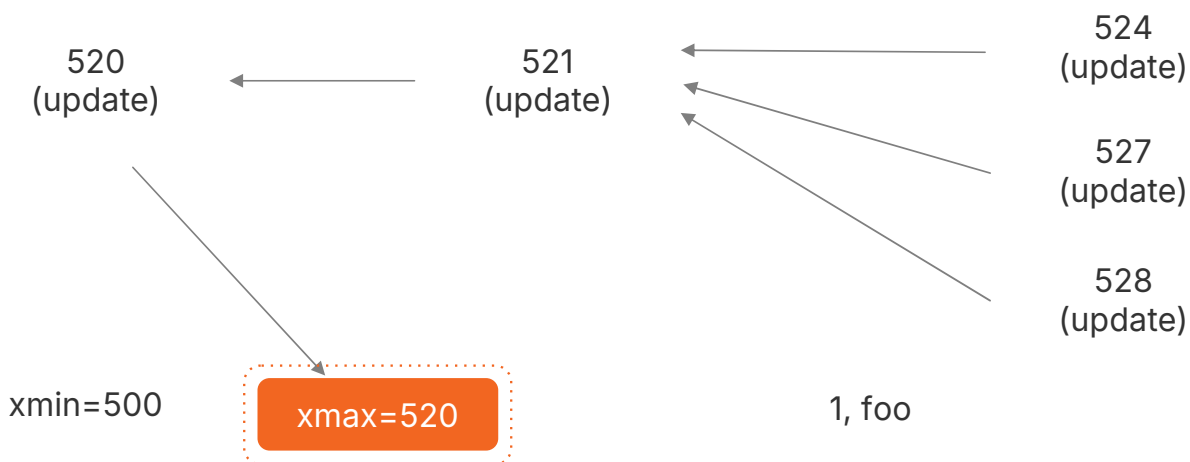
запрашиваемый режим	строка заблокирована в режиме			
	FOR KEY SHARE	FOR SHARE	FOR NO KEY UPDATE	FOR UPDATE
FOR KEY SHARE				X
FOR SHARE			X	X
FOR NO KEY UPDATE		X	X	X
FOR UPDATE	X	X	X	X

Мультитранзакции

- блокировка FOR NO KEY UPDATE устанавливается командой UPDATE, которая не вносит изменения в ключевые столбцы
- блокировка FOR KEY SHARE устанавливается командой DELETE и UPDATE которая обновляет значения ключевых столбцов
- также разделяемые блокировки устанавливаются командами SELECT .. FOR SHARE, FOR NO KEY UPDATE, FOR KEY SHARE
- разделяемые блокировки позволяют одновременно работать со строкой нескольким транзакциям
- одновременная работа реализуется "мультитранзакциями", которые имеют свой счетчик `xid`, свои файлы и кэши
- соответствие между `xid` транзакций и мультитранзакций хранятся в директории `PGDATA/pg_multixact`
- advisory locks не являются заменой блокировок строк, так как их количество ограничено

Очередь при блокировке строки

- есть первый в очереди и остальные
- первого в очереди может обогнать транзакция, чей уровень блокирования строки совместим с уровнем транзакции, уже заблокировавшей строку
- если первый в очереди получает блокировку, то на его место выбирается транзакция случайным образом независимо от длительности ожидания



Практика

1. Вставка, обновление и удаление строки
2. Видимость версии строки на различных уровнях изоляции
3. Состояние транзакции по CLOG
4. Блокировка таблицы
5. Блокировка строки

2

2c Архитектура PostgreSQL

Регламентные работы

Автовакуум

- выполняется рабочими процессами автовакуума
- выбираются таблицы, в которых было обновлено или вставлено больше 20% от размера таблицы
- если давно не было заморозки версий строк, то выполняется заморозка
- после вакуумирования таблицы выполняет автоанализ
- Автовакуум запрашивает блокировку уровня SHARE UPDATE EXCLUSIVE и если не может получить блокировку на таблицу, то эта таблица не вакуумируется в этом цикле автовакуума
- временные таблицы автовакуум не обрабатывает

Процессы автовакуума

- В 18 версии появился параметр autovacuum_worker_slots (по умолчанию 16), который ограничивает число рабочих процессов автовакуума. Значение этого параметра не меняется без перезапуска экземпляра
- Начиная с 18 версии можно без перезапуска экземпляра менять значение параметра autovacuum_max_workers (по умолчанию 3)
- Если одна таблица вакуумируется дольше log_autovacuum_min_duration (по умолчанию, 10 минут), то в диагностический лог кластера выводится сообщение

Энергичная заморозка

- Если автовакуум долго не обрабатывал таблицу в режиме заморозки, цикл автовакуума он запустится в агрессивном режиме
- В агрессивном режиме автовакуум ждёт получения блокировки и не пропускает закреплённые другими процессами блоки
- Энергичная заморозка сканирует 20% блоков из карты видимости с битом all_visible и агрессивному режиму заморозки остаётся заморозить меньше блоков
- Карта заморозки была встроена в карту видимости в версии PostgreSQL 9.6

Представление pg_stat_progress_vacuum

- содержит по одной строке для каждого серверного процесса, выполняющего команду VACUUM и каждого autovacuum worker выполняющих вакуумирование в момент обращения к представлению
- в столбце phase отражается текущая фаза вакуума: `initializing` (подготовительная, проходит быстро), `scanning heap`, `vacuuming indexes`, `vacuuming heap`, `cleaning up indexes`, `truncating heap`, `performing final cleanup` (финальная)
- по столбцам `heap_blks_total` , `heap_blks_scanned` , `heap_blks_vacuumed` оценить ход выполнения очистки
- `num_dead_tuples` - число TID, которые сейчас помещены в структуру памяти. Если достигнет `max_dead_tuples` увеличится значение в **`index_vacuum_count`**
- `VACUUM` **`FULL`** отслеживается через `pg_stat_progress_`**`cluster`**
- `ANALYZE` отслеживается через `pg_stat_progress_analyze`

Параметры команды VACUUM

- `DISABLE_PAGE_SKIPPING` обрабатывает все блоки таблиц без исключения
- `SKIP_LOCKED false` - не дает пропускать заблокированные объекты, секции таблиц, блоки
- `INDEX_CLEANUP auto/on/off` указывает нужно ли обрабатывать индексы. OFF используется если нужно быстрее убрать мертвые строки из блоков таблиц
- `PROCESS_TOAST false` - отключает обработку таблиц TOAST
- `TRUNCATE false` - отключает пятую фазу
- `PARALLEL` *n*. Число *n* ограничивает число фоновых процессов
- `FULL` полная очистка, использует монопольные блокировки, последовательно устанавливаемые на каждую обрабатываемую таблицу

Параметры команды VACUUM

- `SKIP_DATABASE_STATS` отключает обновление `pg_database.datfrozenxid`
  - позволяет не выполнять полное сканирование таблицы `pg_class`
- `VERBOSE` - выводит статистику выполнения команды
- `FREEZE` - выполняет заморозку строк во всех блоках, кроме тех в которых все строки актуальны и заморожены
- `BUFFER_USAGE_LIMIT` размер буферного кольца вместо `vacuum_buffer_usage_limit`
  - в отличие от параметра конфигурации, `BUFFER_USAGE_LIMIT` можно установить в значение ноль и буферное кольцо не будет использоваться

Параметр `default_statistics_target`

- устанавливает
  - число наиболее часто встречающихся значений в столбцах таблиц (`pg_stats.most_common_vals`)
  - количество корзин в гистрограммах распределения значений в столбцах (`pg_stats.histogram_bounds`)
  - число строк (`default_statistics_target * 300`) случайной выборки по которым собирается статистика
- по умолчанию 100
- максимальное значение 10000
- можно установить для конкретного столбца таблицы или индекса по выражению:

```
alter table test alter column id set statistics 10000;
alter index test alter column  1 set statistics 10000;
```

Раздувание (bloat) таблиц и индексов

- автовакуум может не обработать таблицу из-за того, что:
  - горизонт базы данных долго не сдвигался
  - в момент обращения автовакуума к таблице на ней была установлена несовместимая с автовакуумом блокировка
- после того как автовакуум отработает, размеры файлов вряд ли уменьшатся
- блоки будут использоваться в будущем под новые версии строк
- слишком часто заниматься мониторингом не нужно, более актуальна проверка свободного места на дисках
- расширение для оценки числа версий строк:

```
create extension pgstattuple;
\dx+ pgstattuple
select relname, b.* from pg_class, pgstattuple_approx(oid) b WHERE relkind='r';
select relname, b.* from pg_class, pgstatindex(oid) b WHERE relkind='i' order by 10;
```

Внутристраничное обновление (HOT update)

- HOT update - при обновлении (`UPDATE`) строки не вносятся изменения в индексы
- условия HOT update:
  - изменяются только поля, не входящие ни в один из индексов (кроме индексов типа brin), созданных на таблице
  - новая версия строки размещается в том же блоке, что и прежняя версия
- если новая версия строки размещается в блоке, отличном от того в котором находится прежняя версия строки, то:
  - прежняя версия станет последней в цепочке HOT-версий
  - во всех индексах, созданных на таблицу будут созданы новые записи, указывающие на новую версию строки

Мониторинг HOT update

- статистика HOT доступна в представлениях `pg_stat_all_tables` и `pg_stat_user_tables`
- **счетчик HOT обновлений собирается по каждой таблице и отражается в столбце: `n_tup_hot_upd`**
- все обновления отражаются в столбце `n_tup_upd`
- **случаи, если при обновлении не нашлось места для новой версии строки и цепочка HOT была оборвана, а новая версия была вставлена в другой блок показывает `n_tup_newpage_upd`**
- статистика по базе обнуляется вызовом функции `pg_stat_reset();`
  - после вызова функции рекомендуется выполнить ANALYZE по всей базе

```
 select relname, n_tup_upd, n_tup_hot_upd, n_tup_newpage_upd,
round(n_tup_hot_upd*100/n_tup_upd,2) as hot_ratio from pg_stat_all_tables where n_tup_upd<>0
order by 5;
     relname     | n_tup_upd | n_tup_hot_upd | n_tup_newpage_upd | hot_ratio
 ----------------+-----------+---------------+-------------------+-----------
  pg_rewrite     |        14 |             9 |                 5 |     64.00
  pg_proc        |        33 |            23 |                10 |     69.00
  pg_class       |     71645 |         63148 |              8351 |     88.00
```

Внутристраничная очистка (HOT cleanup)

- чтобы HOT cleanup выполнился, предыдущий UPDATE строки должен превысить границу `min(90%, FILLFACTOR)`
- если размер строк в таблице такой, что в блок помещается меньше 9 строк, то восьмая строка не превысит границу 90%, а девятая строка будет больше 11% размера блока и она не поместится в блок
- наиболее эффективно при проектировании схем хранения данных делать размер строк небольшими. Разумный размер строки не больше 600 байт

Внутристраничная очистка в таблицах

- выполняя SELECT и UPDATE, серверный процесс может удалить dead tuples (версии строк, вышедшие за горизонт видимости базы данных, xmin horizon), выполнив реорганизацию версий строк внутри блока
- внутристраничная очистка совместима с HOT и может предварительно освобождать место, которое будет использоваться новыми версиями строк, появляющимися в результате UPDATE
- выполняется, если:
  - блок заполнен более чем на 90% или FILLFACTOR (по умолчанию 100%)
  - ранее выполнявшийся UPDATE не смог разместить новую версию строки в этом блоке
- приблизительная оценка по версиям строк, которые могут быть очищены: `pg_stat_all_tables.n_dead_tup`

Внутристраничная очистка в индексах

- выполняется при индексном сканировании
- если строка в таблице удалена, индексная запись на строку или цепочку версий может быть помечена флагом LP_DEAD
- пометка может ставиться командой SELECT
- журнальная запись не создается, но блок грязнится
- помеченная индексная запись игнорируется на мастере, но не на реплике
- помеченная индексная запись будет очищена при выполнении команд, которые меняют данные в таблице

```
create table t (id int primary key, c text) with (autovacuum_enabled = off);
insert into t SELECT i, 'simple delete ' || i from generate_series(1, 1000000) as i;
delete from t where id between 100 and 900000;
analyze t;
explain (analyze, buffers, costs off) select * from t where id between 1 and 900000;
 Index Scan using t_pkey on t (actual time=0.010..218.477 rows=99 loops=1)
   Buffers: shared hit=11489
 Execution Time: 218.600 ms
```

Эволюция индексов: создание, удаление, перестройка

- команды `create/drop/reindex index имя_индекса` устанавливают блокировку SHARE, не совместимую с внесением изменений в строки таблицы
- эти команды могут выполняться одновременно, они совместимы сами с собой, при этом с concurrently не совместимы
- автовакуум не совместим ни с `concurrently`, ни без
- для временных индексов на временные таблицы не надо использовать concurrently, так как блокировок на временные объекты нет
- `create/reindex concurrently` сканирует больше блоков таблицы, чем без `concurrently`
- `concurrently` позволяет выполняться командам `SELECT, WITH, INSERT, UPDATE, DELETE, MERGE` и позволяет использовать быстрый путь (fastpath) блокирования объектов (таблиц, индексов, секций)

Частичные (partial) индексы

- создаются по части строк таблицы
- предикат WHERE указывается при создании индекса и определяет индексируемые строки
- полезны тем, что позволяют избежать индексирования наиболее часто встречающихся значений
- частичный индекс может быть уникальным
- размер частичного индекса обычно меньше
- пример создания частичного индекса:

```
create unique index t1_idx1 ON t1 (c2 desc nulls first, upper(c1))
include (c3,c4) WHERE c2>0;
```

Команда REINDEX

- перестраивает индексы
- REINDEX блокирует практически любые запросы, кроме подготовленных, план которых был закэширован и которые не используют перестраиваемый индекс
- Для перестройки одного индекса:
- REINDEX INDEX имя_индекса;
- Если необходимо перестроить все индексы по таблице:
- REINDEX TABLE имя_таблицы;
- Также можно перестроить индексы в рамках конкретной схемы или даже всей базы данных:
- REINDEX SCHEMA имя_схемы;
- REINDEX DATABASE; перестроить можно инднексы на таблицы только текущей базы данных, кроме индексов таблиц системного каталога
- REINDEX SYSTEM; перестройка индексов на таблицы системного каталога

REINDEX CONCURRENTLY

- неблокирующее перестроение индексов
- для каждого перестраиваемого индекса выполняется первый проход, на котором строится индекс
- выполняется второй проход, на котором в индекс вносятся записи, добавленные в таблицу во время первого прохода
- ограничения целостности, использовавшие перестраиваемые индексы, переключаются на определение нового индекса и меняются имена индексов
- удаляются старые структуры индексов
- снимаются блокировки
- Перестройка может завершиться ошибкой, в этом случае REINDEX CONCURRENTLY прерывается, но оставляет после себя нерабочий новый индекс в дополнение к перестраиваемому. Этот индекс будет игнорироваться запросами, но будет обновляться при изменении данных, что повлечёт накладные расходы

Гипотетические индексы (расширение HypoPG)

- устанавливается командой `CREATE EXTENSION hypopg;`
- используется при настройке выполнения команд SQL
- позволяет создать определение индексов, существующих только в текущей сессии и не влияющие на работу других сессий
- позволяет выяснить будет ли планировщик использовать индекс при выполнении конкретных команд без создания индекса
- расширение позволяет скрыть любые существующие индексы в текущей сессии чтобы они не влияли на планировщик
- гипотетические индексы создаются функцией `hypopg_create_index('CREATE INDEX...')`, которой передаётся текст команды создания индекса

Счетчик транзакций

- Для 32-битного счетчика транзакций (xid) максимальное значение равно 4млрд.
- когда этот предел достигается, счетчик транзакций переходит "через ноль" и нумерация транзакций начинается со значения 3
- Чтобы не происходило переполнение счетчика версии строк "замораживаются" (freeze), что означает, что версия строки единственная, актуальная, видна во всех моментальных снимках

Практика

1. Обычная очистка таблицы
2. Анализ таблицы
3. Перестройка индекса
4. Полная очистка
5. Расширение HypoPG

2

2d Архитектура PostgreSQL

Выполнение запросов

SQL - декларативный язык

- SQL - декларативный язык программирования для работы с базами данных, где пользователь описывает желаемые результаты запроса, не указывая конкретные шаги выполнения
- Когда пользователь отправляет запрос SQL системе управления базами данных (СУБД) PostgreSQL, происходит следующее:
  - Парсинг (синтаксический и семантический разбор) запроса
  - Переписывание
  - Планирование
  - Выполнение

Синтаксический разбор

- Синтаксический разбор включает в себя следующие шаги:
  - лексический анализ
  - построение синтаксического дерева
  - проверка на соответствие грамматике

Семантический разбор

- определение смысла (семантики), проверка существования таблиц, столбцов, согласованность типов данных
- проверка прав доступа: имеет ли пользователь право на выполнение команды, права доступа к объектам, указанным в запросе: схемам, таблицам, функциям, представлениям и т.п.
  - идёт обращение к **таблицам системного каталога**, в которых хранятся определения объектов. Например, `pg_class`, `pg_attribute`, `pg_type`, `pg_depend`, `pg_constraint`, `pg_namespace`, `pg_inherits`, `pg_attrdef`, `pg_sequence`
  - Выбранные данные кэшируются в локальной памяти процесса, обслуживающего сессию пользователя

Трансформация (переписывание) запроса

- преобразование исходной структуры запроса в аналогичную с точки зрения получения результата в целях лучшей оптимизации на этапах планирования и выполнения в целях лучшей оптимизации на этапах планирования и выполнения
  - имена представлений, если такие были в запросе, заменяются запросами, на основе которых созданы представления

```
LOG:  rewritten parse tree:
DETAIL:  (
            {QUERY
            :commandType 1
            :querySource 0
            :canSetTag true
            :utilityStmt <>
            :resultRelation 0
            :hasAggs false
            :hasWindowFuncs false
...
```

Планирование выполнения (оптимизация)

- Цель этого шага - получить наилучший способ (план) выполнения запроса
- Генерируются возможные способы выполнения запроса, оценивается трудоемкость выполнения и выбирается способ (план) выполнения с наименьшей стоимостью
- Для оценки стоимости используется
- статистика, описывающая объекты
- весовые коэффициенты из параметров конфигурации
- параметры конфигурации, разрешающие использовать способы выборки и обработки строк
- В расчет стоимости включаются две части: вычислительная сложность (процессор) и ввод-вывод

Выполнение запроса

- Чтение данных: читаются строки из блоков таблиц, индексов, функций
- Обработка данных: фильтрации, сортировки, группировки, вычисления
- Соединение наборов строк: если запрос включает в себя соединение таблиц или других источников данных
- Группировка строк: например, если используются групповые функции типа COUNT, SUM, AVG
- Возврат результата: возврат строк клиенту или в код, пославший запрос на выполнение
- Освобождение ресурсов: снимаются блокировки с объектов и освобождается память, использованная при выполнении запроса

Команда EXPLAIN

- выдаёт план выполнения запроса
  - по умолчанию запрос не выполняется.
- Если указать опцию analyze, то запрос выполнится, после выполнения будет показан план с деталями выполнения.
  - без опции timing off "Execution Time" может выдать время выполнения, превышающее реальное в несколько раз
  - опция buffers показывает число буферов, которые считывались при планировании и выполнении

```
postgres=# explain (analyze, buffers) select * from t limit 1;
                                  QUERY PLAN
-------------------------------------------------------------------------------
Limit  (cost=0.03..0.04 rows=1 width=8) (actual time=0.048..0.067 rows=1 loops=1)
 Buffers: shared hit=2
 -> Seq Scan on t  (cost=0.00..14425.00 rows=1000000 width=8) (actual time=0.015..0.020 rows=1 loops=1)
    Buffers: shared hit=2
Planning Time: 0.040 ms
Execution Time: 0.198 ms
(6 rows)
```

Параметры команды EXPLAIN

- 12 параметров, указываются в круглых скобках

```
postgres=# EXPLAIN (analyze, verbose, buffers, serialize text, settings, memory, wal) SELECT *
FROM t WHERE i = 100 AND j = 10;
                      QUERY PLAN
-----------------------------------------------
Gather  (cost=1000.00..11676.00 rows=10 width=8) (actual time=0.421..65.852 rows=10 loops=1)
 Output: i, j
 Workers Planned: 2
 Workers Launched: 2
 Buffers: shared hit=4425
 ->Parallel Seq Scan on public.t(cost=0.00..10675.00 rows=4 width=8)(actual
time=37.521..57.814 rows=3 loops=3)
         Output: i, j
         Filter: ((t.i = 100) AND (t.j = 10))
         Rows Removed by Filter: 333330
         Buffers: shared hit=4425
         Worker 0:  actual time=55.948..55.955 rows=0 loops=1
           Buffers: shared hit=1396
         Worker 1:  actual time=56.416..56.422 rows=0 loops=1
           Buffers: shared hit=1399
Query Identifier: 687797574221341570
Planning:
  Memory: used=11kB  allocated=16kB
Planning Time: 0.054 ms
Serialization: time=0.042 ms  output=1kB  format=text
Execution Time: 66.150 ms
```

Параметры команды EXPLAIN

- 12 параметров, указываются в круглых скобках

```
postgres=# EXPLAIN (analyze, verbose, buffers, serialize text, settings, memory, wal) SELECT *
FROM t WHERE i = 100 AND j = 10;
                      QUERY PLAN
-------------------------------------------------
Gather  (cost=1000.00..11676.00 rows=10 width=8) (actual time=0.421..65.852 rows=10 loops=1)
 Output: i, j
 Workers Planned: 2
 Workers Launched: 2
 Buffers: shared hit=4425
 ->Parallel Seq Scan on public.t(cost=0.00..10675.00 rows=4 width=8)(actual
time=37.521..57.814 rows=3 loops=3)
         Output: i, j
         Filter: ((t.i = 100) AND (t.j = 10))
         Rows Removed by Filter: 333330
         Buffers: shared hit=4425
         Worker 0:  actual time=55.948..55.955 rows=0 loops=1
           Buffers: shared hit=1396
         Worker 1:  actual time=56.416..56.422 rows=0 loops=1
           Buffers: shared hit=1399
Query Identifier: 687797574221341570
Planning:
  Memory: used=11kB  allocated=16kB
Planning Time: 0.054 ms
Serialization: time=0.042 ms  output=1kB  format=text
Execution Time: 66.150 ms
```

Индексы для ограничений целостности

- для ограничений целостности PRIMARY KEY и UNIQUE необходим **уникальный** индекс **типа btree** со столбцами, входящими в ограничение целостности.
- остальные индексы могут использоваться для ускорения запросов ("аналитические индексы"), полнотекстового поиска
- индексы ускоряют поиск строк и замедляют добавление, изменение, удаление строк
- индексы используют место на диске, размер сопоставим с размером таблицы
- пример замены индекса другим индексом:

```
create table t3 (n int4 primary key, m int4);
Indexes:
    "t3_pkey" PRIMARY KEY, btree (n)
create unique index concurrently t3_pkey1 on t3 (m,n);
ALTER TABLE t3 DROP CONSTRAINT t3_pkey, ADD CONSTRAINT t3_pkey PRIMARY KEY USING INDEX t3_pkey1;
NOTICE:  ALTER TABLE / ADD CONSTRAINT USING INDEX will rename index "t3_pkey1" to "t3_pkey"
Indexes:
    "t3_pkey" PRIMARY KEY, btree (m, n)
```

Способы доступа к данным в плане запроса

- способы, которыми процесс получает данные (строки, записи) из источников данных
- источниками могут быть таблицы, внешние таблицы, функции и т.п.
- расширения могут создавать свои методы доступа
- для способа Bitmap строится битовая карта по индексу
  - построение карты отмечается строкой Bitmap Index Scan
  - по битовой карте сканируются строки или блоки таблицы, что отмечается строкой Bitmap Heap Scan в плане:

```
Bitmap Heap Scan on tab (cost=10..1000.51 rows=998 width=11)
  Recheck Cond: (col1 < '1000'::numeric)
  -> Bitmap Index Scan on t_col1_idx (cost=0.00..9.60 rows=998 width=0)
       Index Cond: (col1 < '1000'::numeric)
```

Методы доступа к строкам

```
\dA
List of access methods
   Name    | Type
-----------+-------
 bloom     | Index
 brin      | Index
 btree     | Index
 columnar  | Table
 gin       | Index
 gist      | Index
 hash      | Index
 heap      | Table
 spgist    | Index
(9 rows)
```

- два типа "методов" (способов) доступа к строкам таблиц: табличный и индексный
- Методы доступа можно добавлять расширениями:
  - `create extension pg_columnar;`
  - `create extension bloom;`
- Табличные методы доступа определяют способ хранения данных в таблицах и обычно считывают все строки
- Индексные методы обычно считывают часть блоков таблицы
- Для индексных методов (способов) нужно создать вспомогательный объект, называемый индексом
- Индексы создаются на один или несколько столбцов таблицы:

```
create table t (id int8, d date, s text);
create index t_idx on  t using btree (id int8_ops, d date_ops);
create index t_idx1 on t using btree (s text_ops);
create index if not exists t_idx2 on t (id, d);
```

Способы соединения наборов строк

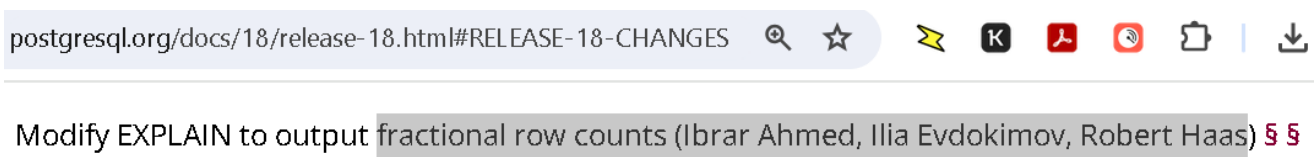
- Способы:
  - Соединение вложенным циклом (Nested Loop Join) с мемоизацией и без
  - Соединение хешированием (Hash Join) с использованием временных файлов и без использования
  - Соединение слиянием (Merge Join)
- Наборы строк всегда соединяются попарно

Кардинальность и селективность

- кардинальным числом отношения (сокращенно кардинальностью) или мощностью отношения называется число строк (они же tuples, кортежи)
  - в плане rows и actual rows
  - с 18 версии PostgreSQL rows - десятичное число

```
Gather (actual rows=2.00 loops=1)
-> Parallel Seq Scan on bookings (actual rows=0.67 loops=3)
```

- Селективность - доля (от нуля до 1) строк из выборки



Стоимость плана запроса

- Стоимость (cost) - числовая оценка трудоемкости выполнения узла плана или всего запроса
- Состоит из двух чисел, между которыми две точки
  - `cost=0.00..14425.00`
- Первое число (startup cost) - стоимость отдачи первой строки в выборке
- Второе число (total cost) - стоимость отдачи всех строк
- Значение стоимости имеет смысл только для сравнения планов одного и того же запроса
- Для всех запросов, кроме курсоров, выбирается план с наименьшим вторым числом
- Стоимость одного и того же запроса коррелирует с временем выполнения этого запроса, но нелинейно

Статистики

- используется планировщиком при выборе плана
- **базовая** и **расширенная** статистика хранится в таблицах системного каталога
- сбор расширенной статистики устанавливается вручную командой `CREATE STATISTICS` или автоматически расширением pg_stat_advisor
  - после установки расширенная статистика собирается вместе с базовой автовакуумом (автоанализом) или при выполнении команд VACUUM (ANALYZE)
- **накопительная** статистика хранится в разделяемой памяти, доступна через представления pg_stat_all_* и функциями pg_stat_get*
  - при остановке экземпляра сохраняется в файл в директории PGDATA/pg_stat
  - используется автовакуумом

Таблица pg_statistic

- Таблица pg_statistic хранит базовую статистику
- Собирается автоанализом и командой ANALYZE , используется для оптимизации запросов планировщиком
- По умолчанию пересобирается по default_statistics_target * 300 = 30000 строк
- содержатся данные по каждому столбцу таблиц
- Пример: доля строк с NULL в третьем столбце таблицы test:

```
select stanullfrac from pg_statistic
 where starelid = 'test'::regclass and staattnum = 3;
 stanullfrac
-------------
   0.9988884
```

Накопительная статистика

- по таблицам:

```
 select schemaname||'.'||relname name, seq_scan, idx_scan, idx_tup_fetch, autovacuum_count, autoanalyze_count from
pg_stat_all_tables where idx_scan is not null order by 3 desc limit 1;
           name           | seq_scan | idx_scan | idx_tup_fetch | autovacuum_count | autoanalyze_count
--------------------------+----------+----------+---------------+------------------+-------------------
 public.pgbench_accounts |        0 | 11183162 |      11183162 |             1512 |               266
  select relname name, n_tup_ins ins, n_tup_upd upd, n_tup_del del, n_tup_hot_upd hot_upd, n_tup_newpage_upd
newblock, n_live_tup live, n_dead_tup dead, n_ins_since_vacuum sv, n_mod_since_analyze sa from pg_stat_all_tables
where idx_scan is not null order by 3 desc limit 1;
        name        | ins |   upd   | del | hot_upd | newblock |  live  |  dead   | sv |  sa
--------------------+-----+---------+-----+---------+----------+--------+---------+----+------
 pgbench_tellers    |   0 | 5598056 |   0 | 5497197 |   100859 |     10 | 1456051 |  0 |  165
```

- В представлении `pg_statio_all_tables` статистика по всем индексам на таблицу. Статистику (чтение с подгрузкой с диска и из кэша буферов) по конкретному индексу можно посмотреть в представлении `pg_statio_all_indexes`.

Расширение pg_stat_statements

- детальная статистика работы экземпляра с точностью до команд SQL
- для установки нужно загрузить библиотеку и установить расширение:

```
alter system set shared_preload_libraries = pg_stat_statements;
create extension pg_stat_statements;
```

- в расширение входят 3 функции и 2 представления:

```
\dx+ pg_stat_statements
 function pg_stat_statements(boolean)
 function pg_stat_statements_info()
 function pg_stat_statements_reset(oid,oid,bigint,boolean)
 view pg_stat_statements
 view pg_stat_statements_info
```

- команды объединяются в одну строку в `pg_stat_statements`, когда они выполняются одним и тем же пользователем и имеют идентичные структуры запросов, то есть семантически равнозначны, за исключением литералов и переменных подстановки (literal constants)
- Например: `select * from t where id = 'a'` и `select * from t where id = 'b'` объединятся в `select * from t where id = $1`

Параметры расширения `pg_stat_statements`

- `pg_stat_statements.max` задаёт максимальное число команд, отслеживаемых расширением, то есть, максимальное число строк в представлении `pg_stat_statements`
- `pg_stat_statements.save` определяет, должна ли статистика сохраняться после перезагрузки сервера
- `pg_stat_statements.track` определяет, какие команды будут отслеживаться: top отслеживаются только команды верхнего уровня
- pg_stat_statements.track_planning устанавливает, будут ли отслеживаться операции планирования и длительность фазы планирования
- `pg_stat_statements.track_utility` будут ли отслеживаться команды, отличные от `SELECT, INSERT, UPDATE, DELETE, MERGE`

```
select name, setting, context, min_val, max_val from pg_settings where name like
'pg_stat_statements%';
               name                | setting |  context   | min_val |  max_val
-----------------------------------+---------+------------+---------+------------
 pg_stat_statements.max            | 5000    | postmaster | 100     | 1073741823
 pg_stat_statements.save           | on      | sighup     |         |
 pg_stat_statements.track          | top     | superuser  |         |
 pg_stat_statements.track_planning | on      | superuser  |         |
 pg_stat_statements.track_utility  | on      | superuser  |         |
```

Практика

1. Создание объектов для запросов
2. Извлечение данных последовательно
3. Возвращение данных по индексу
4. Низкая селективность
5. Использование статистики
6. Представление pg_stat_statements

2

2e Архитектура PostgreSQL

Расширения PostgreSQL

Расширяемость PostgreSQL

- добавление функционала без внесения изменений в файлы исходного кода и перекомпиляции
- можно создавать типы данных, операторы, групповые функции, приведения типов
- создание хранимых подпрограмм на большом числе языков программирования
- создание и использование расширений (extensions)
  - набор любых объектов базы данных, который можно установить или удалить как единое целое
- расширение функционала разделяемыми библиотеками
- обёртки внешних данных (FDW)

Директории файлов расширений и библиотек

- Файлы библиотек находятся в директории:
- `/opt/tantor/db/18/lib/postgresql`
- Файлы расширений (*.control и *.sql) находятся в директории:
- `/opt/tantor/db/18/share/postgresql/extension`
- Узнать местоположение можно командами:

```
postgres@tantor:/opt/tantor/db/18/lib$ pg_config --libdir
/opt/tantor/db/18/lib
postgres@tantor:/opt/tantor/db/18/lib$ pg_config --sharedir
/opt/tantor/db/18/share/postgresql
```

- Установка способом "PGXS":

```
root@tantor:~# export PATH=/opt/tantor/db/18/bin:$PATH
root@tantor:~# export USE_PGXS=1
root@tantor:~# cd директория_расширения
root@tantor:~# make
root@tantor:~# make install
```

- обычно расширения, библиотеки, утилиты, приложения устанавливаются пакетами deb и rpm

Установка расширений

- Расширения могут включать в себя разделяемую библиотеку и/или текстовые файлы: управляющий файл расширения и один или более файлов скриптов
- библиотеки загружаются параметрами или командой:

```
postgres=# \dconfig *librar*
 archive_library           |
 dynamic_library_path      | $libdir
 local_preload_libraries   |
 session_preload_libraries |
 shared_preload_libraries  | pg_stat_statements
postgres=# load 'библиотека';
LOAD
postgres=# \dx
```

- объекты расширения устанавливаются командой:

```
postgres=# create extension имя;
CREATE EXTENSION
```

- доступные к установке расширения перечислены в представлении `pg_available_extentions`

Файлы расширений

- пример управляющего файла расширения:

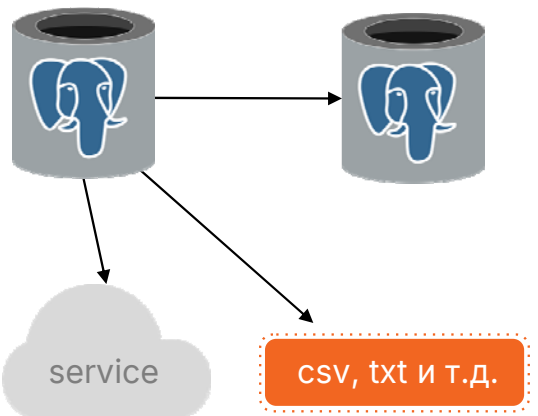
```
postgres@tantor:~$ cat
/opt/tantor/db/18/share/postgresql/extension/pg_stat_kcache.control
# pg_stat_kcache extension
comment = 'Kernel statistics gathering'
default_version = '2.3.1'
requires = 'pg_stat_statements'
module_pathname = '$libdir/pg_stat_kcache'
relocatable = true
```

- пример файла скрипта pg_stat_kcache--2.3.0.sql

```
\echo Use "CREATE EXTENSION pg_stat_kcache" to load this file. \quit
SET client_encoding = 'UTF8';
CREATE FUNCTION pg_stat_kcache_reset()
    RETURNS void
    LANGUAGE c COST 1000
    AS '$libdir/pg_stat_kcache', 'pg_stat_kcache_reset';
REVOKE ALL ON FUNCTION pg_stat_kcache_reset() FROM public;
...
```

Foreign Data Wrapper

- Обертка внешних данных - функционал для обращения к данным, находящимся вне базы данных из сессии с базой данных PostgreSQL стандартизованным и относительно простым способом
- В PostgreSQL включены две обертки (драйвера): **postgres_fdw** для работы с таблицами в базах данных PostgreSQL и **file_fdw** к содержимому текстовых файлов
- FDW устанавливается как расширение
- Существуют расширения: mysql_fdw, oracle_fdw, sqlite_fdw, mongo_fdw, redis_fdw
- команды psql для просмотра объектов FDW: **`\dew, \des, \deu, \det`**



Расширение postgres_fdw

```
create extension postgres_fdw;
CREATE EXTENSION
create database test_db;
CREATE DATABASE
\! pgbench -i -d test_db 2> /dev/null
create server test foreign data wrapper postgres_fdw options (host
'localhost', port '5432', dbname 'test_db', use_scram_passthrough
'true');
CREATE SERVER
create user mapping for postgres server test options (user 'postgres');
CREATE USER MAPPING
import foreign schema public limit to (pgbench_tellers) from server test
into public;
IMPORT FOREIGN SCHEMA
select * from pgbench_tellers limit 1;
 tid | bid | tbalance | filler
-----+-----+----------+--------
   1 |   1 |        0 |
```

Расширение file_fdw

- file_fdw позволяет создавать виртуальные таблицы на основе данных, хранящихся в текстовых файлах
- вносить изменения в файлы нельзя, только читать
- индексов нет, при каждом SELECT просматривается весь файл
- пример создания объектов FDW:

```
CREATE EXTENSION file_fdw;
CREATE SERVER csv_server FOREIGN DATA WRAPPER file_fdw;
CREATE FOREIGN TABLE t1 (
  column1 text,
  column2 numeric,
  ...
)
SERVER csv_server OPTIONS (filename '/path/to/file.csv', format 'csv');
```

Расширение dblink

- позволяет посылать команды на выполнение и получать результат

```
SELECT * FROM dblink('dbname=postgres user=postgres', $$ select 7; $$ ) as (col1 int);
 7
SELECT * FROM dblink_connect('connection1','host=/var/run/postgresql port=5432');
 OK
SELECT * FROM dblink_send_query('connection1', $$ select 8 from pg_sleep(1); $$ );
 1
SELECT dblink_is_busy('connection1');
 1
SELECT * FROM dblink_get_result('connection1') as t(col1 int);
 8
SELECT dblink_is_busy('connection1');
 0
SELECT * FROM dblink_exec('connection1', $$ CHECKPOINT; $$);
 CHECKPOINT
SELECT * FROM dblink_disconnect('connection1');
 OK
```

Практика

1. Определение директории с файлами расширения
2. Просмотр установленных расширений
3. Просмотр доступных расширений
4. Установка и удаление произвольного обновления
5. Просмотр доступных версий расширений
6. Обновление до актуальной версии
7. Обертки внешних данных

3

Конфигурирование PostgreSQL

Обзор

- имеется около 370 параметров
- влияют на работу экземпляра
- название нечувствительно к регистру
- расширения и приложения могут использовать свои параметры конфигурации, у таких параметров в названии присутствует точка
- настройка экземпляра - установка значений параметров конфигурации так, чтобы работа экземпляра была оптимальной при текущей нагрузке

Параметры конфигурации

- Первым просматривается основной файл конфигурации `postgresql.conf`
  Параметры могут дублироваться применяется последний
- Дальше применяются параметры `$PGDATA/postgresql.auto.conf`
- Параметры командной строки передаваемые процессу postgres параметром `-c` превалируют над значениями, установленными в файлах параметров:

```
postgres@tantor:~/tantor-se-18/data$
pg_ctl start -o "-c config_file=./postgresql.conf"
waiting for server to start.... done
server started
```

Просмотр параметров

- Команда `\dconfig`
  - можно искать используя символы * ?
  - можно использовать клавишу табуляции
  - без параметров выдаёт все параметры, которые отличаются от значений по умолчанию
- Команда `SHOW имя_параметра;`
  - символы * ? нельзя использовать
  - можно использовать клавишу табуляции
- функция `current_setting('имя параметра')`

```
postgres=# \dconfig d?ta_d*
               List of configuration parameters
      Parameter       |                    Value
----------------------+-----------------------------------
 data_directory       | /var/lib/postgresql/tantor-se-18/data
 data_directory_mode  | 0750
```

Просмотр параметров

- Команда `\dconfig`
- В представлениях `pg_settings pg_file_settings`
- Командой `SHOW ALL;`
  - нельзя отфильтровать параметры
  - аналог `select * from pg_settings;`
- В файлах `postgresql.conf postgresql.auto.conf`
- просмотр значения одного параметра на запущенном или остановленном экземпляре `postgres -C имя_параметра`

```
postgres@tantor:~$ postgres -C autovacuum_freeze_max_age
10000000000
postgres@tantor:~$ ls -CF $PGDATA
base/           pg_logical/    pg_stat/      pg_wal/
global/         pg_multixact/  pg_stat_tmp/  pg_xact/
pg_commit_ts/   pg_notify/     pg_subtrans/  postgresql.auto.conf
pg_dynshmem/    pg_replslot/   pg_tblspc/    postgresql.conf
pg_hba.conf     pg_serial/     pg_twophase/  postmaster.opts
pg_ident.conf   pg_snapshots/  PG_VERSION    postmaster.pid
```

Представления для просмотра параметров

- Представление `pg_settings` показывает текущие действующие значения параметров
- Представление `pg_file_settings`
  - параметры, которые явно указаны в файлах параметров
  - не показывает текущие значения, которые использует экземпляр
  - столбец `applied` имеет значение "f", если значение параметра: отличается от текущего и для применения значения из файла требуется перезапуск кластера
- Представление `pg_hba_file_rules` показывает содержимое файла `pg_hba.conf`

```
postgres=# select pg_read_file('./postgresql.auto.conf') \g
   (tuples_only=on format=unaligned)
# Do not edit this file manually!
# It will be overwritten by the ALTER SYSTEM command.
log_filename = 'postgresql-%F.log'
cluster_name = 'master'
logging_collector = 'on'
```

Основной файл параметров postgresql.conf

- основной файл, в котором хранятся параметры конфигурации кластера
- может содержать ~400 параметров плюс параметры расширений и разделяемых библиотек (*.so)
- Создаётся на основе файла postgresql.conf.sample утилитой initdb
- Можно редактировать
- Можно включать содержимое других файлов параметрами конфигурации include и include_dir заданными внутри postgresql.conf

Файл параметров postgresql.auto.conf

- Расположен в директории `PGDATA`
- Параметры добавляются командой
  - `ALTER SYSTEM SET параметр = значение;`
- Параметры убираются командой
  `ALTER SYSTEM RESET параметр;`
  `ALTER SYSTEM RESET ALL;`
- Чтобы новые значения вступили в силу после изменения командой `ALTER SYSTEM` нужно перечитать конфигурацию или перегрузить кластер

```
postgres=# select pg_reload_conf();
 pg_reload_conf
----------------
 t
postgres=# \! pg_ctl reload
server signaled
```

Применение изменений параметров

- Для применения изменений перечитываются все файлы конфигурации и применяются изменения которые можно провести без перезагрузки кластера
- Способы:
  - `SELECT pg_reload_conf();`
  - `pg_ctl reload`
  - `sudo killall -HUP postgres`
- Параметры в postgresql.auto.conf перекрывают значения параметров postgresql.conf
- Если в postgresql.conf параметр указан несколько раз, применяется указанный ближе к концу файла

Привилегии на изменение параметров

- Тип параметра указан в столбце `context` представления `pg_settings`
- Команду ALTER SYSTEM может выполнять роль с атрибутом SUPERUSER и те роли, которым была дана привилегия `GRANT ALTER SYSTEM on PARAMETER имя_параметра to роль;`
- Посмотреть список привилегий можно командой psql:

```
\dconfig+ имя_привилегии
      Parameter        | Value | Type |  Context   |  Access privileges
-----------------------+-------+------+------------+----------------------
 update_process_title | off   | bool | superuser | postgres=sA/postgres+
                      |       |      |           | user1=s/postgres
```

- или запросом select * from pg_parameter_acl;
- A - право на ALTER SYSTEM, s - право на SET
- параметром `allow_alter_system` можно отключить возможность использования команды ALTER SYSTEM

Классификация параметров: Контекст

- internal - параметры только для чтения
- postmaster - нужен перезапуск экземпляра кластера
- sighup - достаточно перечитать файлы конфигурации
- superuser - могут устанавливаться на уровне сессии, но у пользователя должен быть атрибут SUPERUSER или привилегия на изменение этого параметра
- superuser-backend - не могут быть изменены после создания сессии, но могут быть установлены для конкретной сессии в момент подсоединения, если есть привилегии
- backend - как и предыдущий, но привилегий не нужно
- user - можно менять в течение сессии или на уровне кластера

Параметры контекста internal

- 20 параметров в 18 версии
- только для чтения

```
postgres=# select name, setting from pg_settings where context='internal' order by 1;
              name                | setting
----------------------------------+----------
 block_size                       | 8192
 data_checksums                   | on
 data_directory_mode              | 0700
 debug_assertions                 | off
 huge_pages_status                | off
 in_hot_standby                   | off
 integer_datetimes                | on
 max_function_args                | 100
 max_identifier_length            | 63
 max_index_keys                   | 32
 num_os_semaphores                | 174
 segment_size                     | 131072
 server_encoding                  | UTF8
 server_version                   | 17.2
 server_version_num               | 170002
 shared_memory_size               | 189
 shared_memory_size_in_huge_pages | 95
 ssl_library                      | OpenSSL
 wal_block_size                   | 8192
 wal_segment_size                 | 16777216
(20 rows)
```

Классификация параметров: Уровни

- Кластера командой
  - `ALTER SYSTEM SET параметр = значение;`
- Базы данных
  - `ALTER DATABASE имя SET параметр = значение;`
- Роли
  - `ALTER ROLE роль SET параметр = значение;`
- Сессии создаваемой ролью, подсоединяющейся к конкретной базе данных
  - `ALTER ROLE роль IN DATABASE имя SET параметр = значение;`

Классификация параметров: Уровни

- Внутри сессии
  - `SET work_mem to '16MB';`
  - `SELECT set_config('work_mem', '16MB', false);`
- Транзакции
  - `SET LOCAL work_mem to '16MB';`
  - `SELECT set_config('work_mem', '16MB', true);`
- На время выполнения функции или процедуры
  - `CREATE {FUNCTION|PROCEDURE} .. SET параметр {TO значение | = значение | FROM CURRENT}`
  - `ALTER {PROCEDURE | FUNCTION}.. SET параметр {TO | = }{значение | DEFAULT};`

Параметры хранения на уровне таблиц

- Устанавливаются командой
  - `ALTER TABLE имя SET(параметр_хранения = значение);`
- Можно установить в команде CREATE TABLE
- Перекрывают аналогичные параметры конфигурации при работе автовакуума с конкретной таблицей и/или её TOAST таблицей
- Команда перекрывающая значение параметра конфигурации default_statistics_target для столбца таблицы:
  - `ALTER TABLE имя ALTER COLUMN имя SET STATISTICS число;`

Классификация параметров: Категории

- Категории описывают для чего предназначены параметры

```
select category, count(name) from pg_settings group by category
  order by 2 desc;
                      category                       | count
-----------------------------------------------------+-------
 Query Tuning / Planner Method Configuration         |    28
 Resource Usage / Memory                             |    28
 Client Connection Defaults / Statement Behavior     |    27
 Developer Options                                   |    27
 Reporting and Logging / What to Log                 |    22
 Preset Options                                      |    20
 Query Tuning / Planner Cost Constants               |    17
 Query Tuning / Other Planner Options                |    17
 Vacuuming / Automatic Vacuuming                     |    15
...
(47 rows)
```

Категория: "Для разработчиков"

- Категория параметров `Developer Options`
- Параметры этой категории не должны использоваться при промышленной эксплуатации
- Параметры из этой категории могут помочь получить данные в сложных случаях повреждения страниц
- `ignore_system_indexes` - игнорировать системные индексы при чтении системных таблиц (но все же обновлять индексы при изменении таблиц). Может пригодиться, если повреждения в системных индексах не позволяют создать сессию для устранения повреждений
- `zero_damaged_pages` - повреждения в служебной области блока не даёт прочесть данные из этого блока. Параметр позволяет считать блок пустым, как будто в нем нет строк

Категория: "Пользовательские настройки"

- Категория параметров `Customized Options`
- В категорию входят параметры расширений, библиотек или просто произвольные параметры, в названии которых присутствует точка
- разработчики библиотек и расширений дают возможность стандартным образом настраивать их функционал

```
postgres=# set myapp.par1 = '20';
SET
postgres=# show myapp.par1;
 myapp.par1
------------
 20
(1 row)
```

Названия и значения параметров конфигурации

- Названия параметров нечувствительны к регистру
- Значения параметров - один из пяти типов:
- boolean: значения можно указывать как `on`, `off`, `true`, `false`, `yes`, `no`, `1`, `0`
- строка: значения лучше задать в апострофах
- целое или десятичное число: в единицах измерения из столбца `unit` представления `pg_settings` или просто число
- `enum`: значение из списка в столбце `enumvals` представления pg_settings

Параметр конфигурации `transaction_timeout`

- позволяет отменить любую транзакцию, длительность которой превышает указанный период времени
- действие параметра распространяется как на явные транзакции (начатые с помощью команды BEGIN), так и на неявно начатые транзакции, соответствующие отдельной команде
- Значение ноль (по умолчанию) отключает таймаут
- Позволяет устанавливать ограничение на длительность транзакций
- `statement_timeout` позволяет установить максимальное время выполнения отдельной команды

Автономные транзакции

- В Tantor Postgres SE и SE 1C есть автономные транзакции
- используются в подпрограммах на plpgsql
- позволяют выполнить команды и сохранить результат их выполнения независимо от того зафиксируется, откатится или прервётся транзакция, в которой была вызвана автономная транзакция

```
postgres=# create table t (a int);
CREATE TABLE
postgres=# create or replace
procedure p() LANGUAGE plpgsql
AS $$
DECLARE
 PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
 insert into t values (1);
END;
$$;
CREATE PROCEDURE
postgres=# begin transaction;
BEGIN
postgres=*# call p();
CALL
postgres=*# rollback;
ROLLBACK
postgres=# select * from t;
 a
---
 1
(1 row)
```

Параметр конфигурации transaction_buffers

- задаёт размер общей памяти, используемый для кэширования содержимого `PGDATA/pg_xact` - подкаталога со статусом фиксации транзакций
- Значение по умолчанию - 0, что равно размеру разделяемого пула буферов, деленному на 512 (shared_buffers/512)

Параметры multixact_members_buffers и multixact_offsets_buffers

- задают размер общей памяти, используемой для кеширования содержимого содержимого двух поддиректорий `PGDATA/pg_multixact`
- Убедиться, что в кластере используются 64-битные идентификаторы транзакций можно по значениям параметров:

```
\dconfig autovacuum_*age
        List of configuration parameters
             Parameter                |    Value
--------------------------------------+-------------
 autovacuum_freeze_max_age            | 10000000000
 autovacuum_multixact_freeze_max_age | 20000000000
```

- если значения больше 4 миллиардов, то используются 64-битные счетчики транзакций и мультитранзакций

Параметр конфигурации subtransaction_buffers

- задаёт размер общей памяти, используемой для кеширования содержимого `PGDATA/pg_subtrans`
- подтранзакция создается:
  - командой SAVEPOINT
  - если в блоке на языке plpgsql присутствует секция EXCEPTION
  - в psql неявно перед каждой командой в транзакции, если включён (interactive или on) параметр командой:
    `\set ON_ERROR_ROLLBACK  interactive`
- наличие секции EXCEPTION неявно устанавливает точку сохранения перед началом блока (перед BEGIN)

Параметр конфигурации notify_buffers

- задаёт размер общей памяти, используемой для кеширования содержимого `PGDATA/pg_notify`
- Значение по умолчанию 8 блоков (64Кб)
- Используются в архитектуре NOTIFY/LISTEN обмена данными между процессами:

```
postgres=# listen abc;
LISTEN
postgres=# notify abc;
NOTIFY
Asynchronous notification "abc" received from
server process with PID 1234.
```

Задание параметров при создании кластера

- переменными окружения и параметрами утилиты initdb
- параметры **initdb --lc-collate, --lc-ctype, --encoding** нельзя изменить после создания кластера
- значения параметров локализации можно посмотреть командой psql `\l`
- включить и отключить подсчет контрольных сумм можно после создания кластера
- проверить статус в psql **`\dconfig data_checksums`**
- проверка контрольных сумм блоков данных выполняется утилитами pg_checksums и pg_verifybackup

Разрешения на директорию PGDATA

- Параметр `initdb -g` или `--allow-group-access` устанавливает разрешения `0750 (rwx r-x ---)`
- если не указать, то пересоздавать кластер не нужно, можно поменять разрешения на PGDATA (и поддиректории) на допустимые значения
- Допустимые значения 0750 и 0700:
- **`pg_ctl start`**

```
waiting for server to start....
FATAL:  data directory
  "/var/lib/postgresql/tantor/se16/data" has invalid
  permissions
DETAIL:  Permissions should be u=rwx (0700) or u=rwx,g=rx
  (0750).
stopped waiting
```

Размер блока данных PostgreSQL

- размер 8192 байт = 8Кб задан при компиляции
- синоним “страница”
- при нахождении в памяти занимает буфер в буферном кэше

Ограничения PostgreSQL

размер базы данных	не ограничен
количество баз данных в кластере	4,294,950,911
отношений в одной базе	1,431,650,303
размер отношения	32ТБ (блок 8КБ)
блоков в таблице	4,294,950,911
столбцов в таблице	1600
столбцов в выборке (SELECT)	1664
размер поля (в том числе text, bytea)	1ГБ
размер больших объектов (lo)	4ТБ
общий объем lo в базе	32ТБ (блок 8КБ)
длина идентификатора	63 байта
индексов на таблицу	не ограничено
размер строкового буфера	1Гб-1
столбцов в составном индексе	32

Ограничения на длину идентификаторов

- Максимальная длина идентификаторов (например, имен таблиц, столбцов, индексов и т. д.) составляет 63 символа
- Например, вы можете создать таблицу с именем, содержащим до 63 символов:
- `CREATE TABLE my_really_long_table_name_with_63_characters (...);`
- Или столбец с именем, также содержащим до 63 символов:
- `ALTER TABLE my_table_name ADD COLUMN my_really_long_column_name_with_63_characters INTEGER;`
- идентификаторы превышающие этот размер усекаются, о чем выдается предупреждение

Конфигурационные параметры

- установлены при сборке и не меняются
- можно посмотреть:
  - утилитой командной строки pg_config
  - в представлении pg_config
- Параметр `SHAREDIR` определяет директорию с файлами расширений
- `PKGLIBDIR` указывает на директорию разделяемых библиотек
- `BINDIR` определяет директорию с исполняемыми файлами

```
postgres=# select * from pg_config where name in
('BINDIR','LIBDIR','PKGLIBDIR','SHAREDIR','SYSCONFDIR');
    name     |                setting
-------------+-----------------------------------
 BINDIR      | /opt/tantor/db/18/bin
 LIBDIR      | /opt/tantor/db/18/lib
 PKGLIBDIR   | /opt/tantor/db/18/lib/postgresql
 SHAREDIR    | /opt/tantor/db/18/share/postgresql
 SYSCONFDIR | /opt/tantor/db/18/etc/postgresql
```

Демонстрация

- Просмотр параметров конфигурации

Практика

1. Обзор параметров конфигурации
2. Параметры конфигурации с единицей измерения
3. Параметры конфигурации логического типа
4. Конфигурационные параметры
5. Файл служб

4

4a

Логическая структура кластера

Кластер баз данных

- Кластер баз данных PostgreSQL - это набор баз данных
  - хранят свои файлы в директории PGDATA
  - обслуживаются экземпляром
- Приложение подсоединяется к одной базе данных и имеет доступ к её объектам
- Кластер PostgreSQL создаётся создаётся утилитой initdb
- Базы данных в кластере PostgreSQL можно создавать и удалять
- Кластер Patroni - это несколько кластеров PostgreSQL

Экземпляр

- Экземпляр это набор процессов и используемой ими памяти, обслуживающие один кластер баз данных
- Кластер баз данных обслуживается только одним экземпляром
- Первый и основной процесс экземпляра называется postgres или postmaster
- Необходимый параметр экземпляра - порт (port), по которому postmaster просушивает входящие соединения по протоколам TCP и локально
- По умолчанию параметр конфигурации port имеет значение 5432

База данных

- База данных - логическое место хранения объектов SQL
- База данных - часть кластера
- Содержимое базы данных может быть выгружено на логическом уровне (в виде команд SQL) и загружено в другую базу данных этого или другого кластера
- Приложение создаёт сессию (session) с одной базой данных
- В сессии есть доступ к объектам одной базы данных
- Доступа к объектам других баз данных кластера в одной сессии нет

Список баз данных

- Получение списка это команда psql:
  `\l` или `\l+`

```
postgres=# \l
                                 List of databases
   Name    |  Owner   | Encoding | Locale Provider |   Collate   |    Ctype    |
-----------+----------+----------+-----------------+-------------+-------------+
 postgres  | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
 template1 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
 testdb    | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
(4 rows)

postgres=# SELECT datname FROM pg_database;
  datname
-----------
 postgres
 template1
 template0
 testdb
```

Создание базы данных

- команда CREATE DATABASE или утилита createdb
- два режима создания базы: WAL_LOG и FILE_COPY
- создание базы это клонирование другой базы со всем содержимым
- у базы есть роль-владелец (OWNER)
- владельца и имя базы можно сменить после её создания
- можно создать базу со своими параметрами локализации (кодировкой)
- кодировку после создания базы изменить нельзя
- свойство базы "шаблонная" (IS_TEMPLATE) влияет на возможность ее клонирования непривилегированной ролью и удаление
- Функция pg_import_system_collations('pg_catalog') добавляет параметры локализации, появившиеся в linux,после создания кластера

Изменение свойств базы данных

- Командой ALTER DATABASE имя можно поменять некоторые свойства существующей базы данных
- Часть свойств можно поменять только, если нет ни одной сессии к базе данных
- Командой ALTER DATBASE имя SET можно поменять конфигурационные параметры сессий, создаваемых с этой базой
- на уровне базы данных можно установить около 190 параметров конфигурации
- база данных - общий объект кластера и менять её свойства можно, подсоединившись к любой базе данных в кластере

Команда ALTER DATABASE

- Синтаксис команды по которому видно какие свойства можно поменять:
-
```
ALTER DATABASE имя_базы
  ALLOW_CONNECTIONS true|false
  CONNECTION LIMIT -1/число
  IS_TEMPLATE true|false
  RENAME TO новое
  OWNER TO новый
  SET TABLESPACE новое
  REFRESH COLLATION VERSION
  SET параметр=значение
```

Удаление базы данных

- Выполняется командой DROP DATABASE или утилитой-оберткой dropdb
- Для выполнения команды нужно подсоединиться к любой базе кластера, отличной от удаляемой
- База данных удаляется со всем содержимым
- Команда нетранзакционна - откатить её нельзя
- Физически удаляются файлы в которых хранятся данные объектов базы

Схемы в базе данных

- синоним схемы - пространство имён (namespace)
- используются для упорядочивания хранения объектов базы данных
- Схема - локальный объект базы данных
- Схемы позволяют иметь несколько таблиц и других типов объектов с одинаковыми именами в одной и той же базе данных
- Объект не может находиться в нескольких схемах
- Схемы могут использоваться для логики объединения (пакетирования) подпрограмм
- Объектов "пакеты" в PostgreSQL нет

Создание и изменение схем

- Схемы не связаны с пользователями, но имеют владельца
- При создании схемы можно назначить пользователя-владельца:
  `CREATE SCHEMA имя_схемы AUTHORIZATION владелец;`
- Можно сменить владельца:
  `ALTER SCHEMA имя_схемы OWNER TO пользователь;`
- Можно переименовать схему:
  `ALTER SCHEMA имя_схемы RENAME TO имя;`
- На схемы можно выдавать привилегии CREATE, USAGE ролям

Путь поиска объектов в схемах

- задаёт список схем, в которых ищется объект
- устанавливается параметром `search_path`, который может быть изменён на любом уровне
- значение по умолчанию `"$user", public`

```
postgres=# show search_path;
   search_path
-----------------
 "$user", public
postgres=# select current_schema , current_schema(),
current_schemas(true), current_schemas(false);
 current_schema | current_schema |   current_schemas    | current_schemas
----------------+----------------+----------------------+----------------
 public         | public         | {pg_catalog,public} | {public}
(1 row)
```

Специальные схемы

- **pg_catalog** - в этой схеме расположены объекты "системного каталога"
- **information_schema** - схема, описанная в стандарте SQL
- **pg_toast** - схема для особых таблиц TOAST
- **pg_temp** (ссылка на pg_tempN, где N число) - схема для временных таблиц
- **pg_toast_temp** (ссылка на pg_toast_tempN, где N число) - схема для временных TOAST таблиц на временные таблицы

```
postgres=# \dnS
            List of schemas
        Name         |        Owner
---------------------+---------------------
 information_schema | postgres
 pg_catalog         | postgres
 pg_toast           | postgres
 public             | pg_database_owner
```

Определение текущего пути поиска

- командой psql `SHOW search_path;`
  - выдает установленный для этого места путь поиска
- функцией `current_schemas(false)`
  - выдает действующий в этом месте путь поиска в виде массива
- функция `current_schemas(true)`
  - добавляет служебные схемы, а именно pg_catalog и pg_temp_N
- функция `current_schema` или `current_schema()`
  - выдает одно имя первой по порядку схемы в пути поиска или NULL, если путь поиска пустой

В какой схеме будет создан объект

- имя этой схемы для обычных объектов выдает функция `current_schema()`
- временные объекты создаются в служебных схемах
- служебные схемы для временных объектов создаются автоматически

Путь поиска в подпрограммах SECURITY DEFINER

- при использовании $user в пути поиска в теле подпрограммы будет подставлено имя владельца подпрограммы
- перед вызовом такой подпрограммы вызывающий ее может поменять значение в пути поиска и убрать $user, в этом случае в теле подпрограммы будет использоваться значение установленное вызывающим и подпрограмма может начать работать с другими объектами
- На уровне любой (INVOKER и DEFINER) подпрограммы можно установить значение для параметров конфигурации, которые могут меняться на уровне сессии

Маскировка объектов схем

- Если в командах перед именем объектов не используется имя схемы, добавление в путь поиска имея схемы, где создан объект с тем же именем маскирует исходный объект
- По умолчанию, временная схема и схема системного каталога неявно присутствуют в начале пути поиска и создание временного объекта маскирует любую таблицу, представление, последовательность

```
postgres=# \dt pg_authid
              List of tables
   Schema   |   Name    | Type  |  Owner
------------+-----------+-------+---------
 pg_catalog | pg_authid | table | postgres
(1 row)

postgres=# create temporary table
pg_authid(t text);
CREATE TABLE
postgres=# select * from pg_authid;
 t
---
(0 rows)

postgres=# select current_schemas(true);
        current_schemas
-------------------------------
 {pg_temp_2,pg_catalog,public}
(1 row)
```

Системный каталог

- Системный каталог это таблицы, представления, функции, индексы и другие объекты которые используются для хранения метаданных
- Объекты системного каталога располагаются в схеме `pg_catalog`
- Объекты системного каталога (кроме глобальных) всегда располагаются в табличном пространстве по умолчанию для базы данных
- названия объектов приводятся к нижнему регистру
- Таблицы системного каталога неявно используют процессы кластера на этапе выполнения команд SQL

Общие объекты кластера

- это объекты (таблицы и вспомогательные объекты) системного каталога, хранящие данные об общих объектах кластера
- располагаются в табличном пространстве `pg_global`
- Общие объекты кластера:
- базы данных
- роли
- табличные пространства
- подписки логической репликации
- источники логической репликации

Использование системного каталога

- из таблиц и представлений системного каталога можно получать информацию командами SELECT, WITH, командами psql (начинаются на символ \), графическими приложениями и Платформой Tantor
- `\?` справка по командам psql
- Вносить изменения в таблицы системного каталога если, это не описано в документации нежелательно
- Зная архитектуру PostgreSQL, понятия, термины вы сможете легко получать информацию командами psql

Обращение к системному каталогу

- Выполняется командами SELECT и WITH
- можно соединять таблицы и представления
- столбец с именем oid - первичный ключ
- для связей между таблицами используется столбец oid
- максимальное число строк в таблицах системного каталога 4 миллиарда
- первые три символа в названии столбцов - сокращенное название таблицы

reg-типы

- созданы для 11 таблиц системного каталога
- используются для приведения текстового имени объекта к значению типа oid и обратно
- позволяют проще писать запросы к таблицам системного каталога уменьшая количество соединений таблиц
- Пример: найти название TOAST таблицы созданной для таблицы pg_tablespace:

```
select reltoastrelid, reltoastrelid::regclass from pg_class
where relname='pg_tablespace';
 reltoastrelid |     reltoastrelid
---------------+-----------------------
          4185 | pg_toast.pg_toast_1213
```

Часто используемые команды psql

- \l - список баз данных
- \du \dg - список ролей кластера
- \dn - список схем базы
- \db - список табличных пространств
- \dconfig *имя* - список параметров конфигурации
- \dfS pg* - список системных функций и процедур, полезных для администрирования. Часть информации о работе экземпляра и кластера можно получить только с помощью функций. Некоторые служебные представления используют функции
- \dvS pg* - полезные служебные представления
- символ + в конце команды показывает больше информации пример: \db+ покажет размер и привилегии

Демонстрация

- Просмотр списка баз данных кластера
- Создание базы данных
- Переименование базы данных
- Ограничение на соединение с базой
- форматирование вывода psql

Практика

1. Установка параметров конфигурации на различных уровнях
2. Установка пути поиска в функциях и процедурах

4

4b

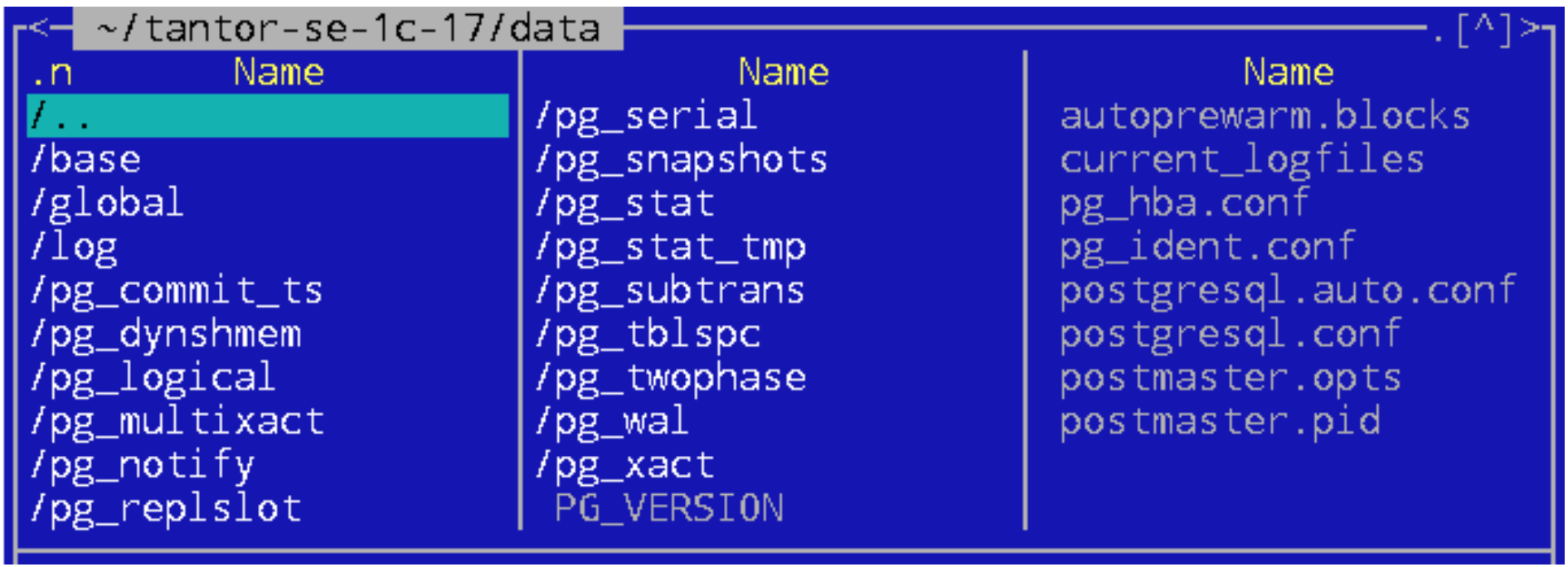
Физическая структура кластера

Директория файлов кластера PGDATA

- Кластер хранит свои файлы в файловой системе
- Директория кластера называется PGDATA
- Директория или её поддиректории могут быть точками монтирования, жесткими, символическими ссылками
- Блочные устройства и неформатированные разделы жесткого диска не используются
- По умолчанию при работе с файлами кластера используется кэш операционной системы

Директории и файлы в PGDATA

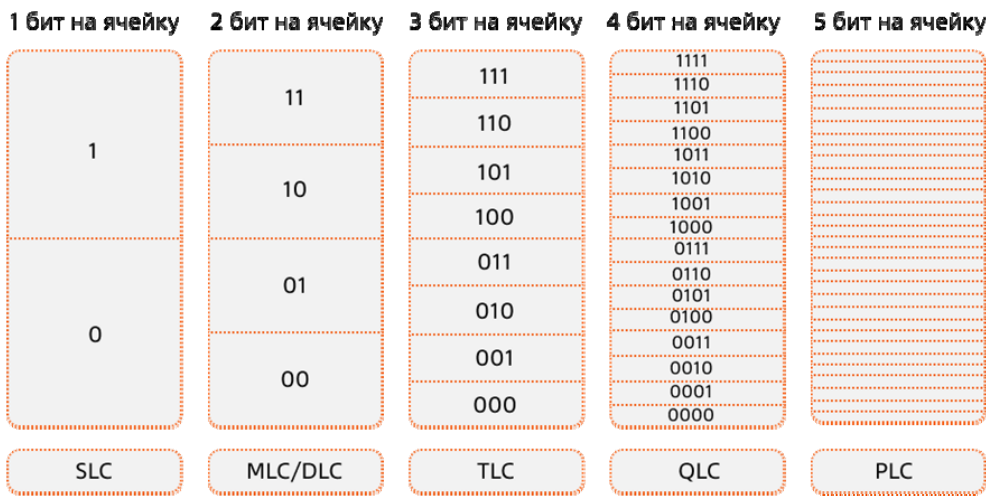
- содержит файлы параметров
- поддиректории



Файлы журнала предварительной записи (WAL)

- Параметры `wal_recycle` и `wal_init_zero` определяют работу с файлами WAL журнала
- По умолчанию, файлы повторно используются, вновь создаваемые заполняются нулями
- **PGDATA/pg_wal** хранит файлы журнала
- При использовании SSD большая часть раздела файловой системы должна быть свободна

```
wal_block_size              | 8192
wal_buffers                 | 4MB
wal_compression             | off
wal_consistency_checking    |
wal_init_zero               | on
wal_level                   | replica
wal_recycle                 | on
wal_segment_size            | 16MB
wal_sync_method             | fdatasync
```



Директория с файлами журнала

- `PGDATA/pg_wal` может быть символической ссылкой, указывающей на смонтированный раздел диска
- утилите `initdb` можно указать в параметре `--waldir=путь` к директории, где будут храниться файлы журнала
- после создания кластера можно создать символическую ссылку вручную
- Можно включить сжатие образов страниц параметром `wal_compression`, алгоритмы: `pglz, lz4, zstd`

```
postgres@tantor:~/data$ initdb -D . --waldir=/var/lib/postgresql/wal
...
Success. You can now start the database server using:
    pg_ctl -D . -l logfile start
postgres@tantor:~/data$ ls -l pg_wal
lrwxrwxrwx 1 postgres postgres 44 pg_wal -> /var/lib/postgresql/wal
```

Табличные пространства

- предназначены для того, чтобы кластер мог располагаться на нескольких устройствах хранения
- физически это директория в файловой системе
- являются общими объектами кластера
- могут хранить файлы нескольких баз данных кластера

Табличные пространства: характеристики

- Поддиректории создаются автоматически
- При создании кластера создаются табличные пространства **pg_global** и **pg_default**
- Список табличных пространств:
  команда **\db** и таблица **pg_tablespace**
- содержимое списка одинаково во всех базах данных кластера
- **pg_tablespace** - глобальная таблица системного каталога

Табличные пространства: характеристики

- Без кластера не существуют
- Не могут перемещаться между кластерами
- Не могут резервироваться отдельно от кластера
- Не являются объектами схем
- Имеют роль-владельца
- Директория табличного пространства:
- создаётся вручную в операционной системе
- должна располагаться в файловых системах, устойчивых к потере питания
- рекомендуется создавать вне PGDATA
- в директории не стоит удалять вручную файлы, они удаляются автоматически командами SQL

Команды управления табличными пространствами

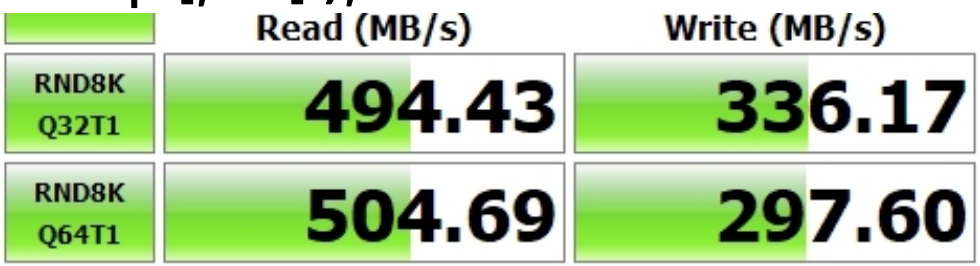
- Команда создания:
```
CREATE TABLESPACE имя [ OWNER роль ] LOCATION
'директория'
 [ WITH ( параметр = значение [, ...] ) ]
```
- Смена владельца:
```
ALTER TABLESPACE имя OWNER TO роль;
```
- Переименование:
```
ALTER TABLESPACE имя RENAME TO имя;
```
- База данных имеет табличное пространство "по умолчанию", в нем находятся файлы объектов системного каталога этой базы данных
- Команда изменения табличного пространства по умолчанию для базы данных:
```
ALTER DATABASE база SET TABLESPACE имя;
```

Изменение директории табличного пространства

- Кластер определяет директорию табличного пространства по символической ссылке в директории **PGDATA/pg_tblspc**
- Процедура замены:
  - Найдите название символической ссылки в директории **PGDATA/pg_tblspc**, указывающую на директорию табличного пространства
  - Остановите экземпляр
  - Переместите директорию табличного пространства
  - Обновите символическую ссылку
  - Запустите экземпляр

Параметры табличных пространств

- Четыре параметра: **seq_page_cost, random_page_cost, effective_io_concurrency, maintenance_io_concurrency**
- Перекрывают значения параметров, установленных на уровне кластера
- Влияют на стоимость плана выполнения команд SQL
- Можно задать при создании табличного пространства:
  CREATE TABLESPACE имя WITH (параметр = значение [, ...]);
- Можно изменить позже:
  ALTER TABLESPACE имя SET (параметр = значение [, ...]);
- Можно сбросить установленное значение:
  ALTER TABLESPACE имя RESET (параметр [, ...]);



Временные файлы

- создаются в директориях табличных пространств
- табличные пространства для временных файлов задаются параметром `temp_tablespaces`
- рекомендуется создать отдельное табличное пространство для временных файлов
- временные файлы создаются если обрабатываемые данные не помещаются в память процесса
- временные файлы создаются для временных таблиц и их индексов, при выполнении команд SQL обрабатывающих большие объемы данных (например, сортировка, создание индекса)

Основной слой хранения данных

- состоит из файлов размером до 1Гб
- новые файлы создаются при достижении предыдущего файла слоя 1Гб
- Максимальный размер файлов слоя 32Тб
- Работа с файлами всех слоёв объектов постоянного хранения происходит через буферный кэш
- Работа с файлами всех слоёв временных объектов происходит через буфер в локальной памяти серверного процесса
- Файлы всех слоёв располагаются в одной директории одного табличного пространства

Дополнительные слои

Карта свободного пространства (fsm)
- не создается для хэш-индексов
- создается вакуумированием

Карта видимости (vm)
- не создается для индексов
- создается вакуумированием
- два бита на блок основного слоя
- установленный первый бит означает что все строки блока актуальны и нет старых версий
- установленный второй бит означает что все строки блока заморожены

Слой инициализации (init)
- создается для нежурналируемых таблиц и индексов
- файл размером один блок без данных

Расположение файлов объектов

- Если объект расположен в `pg_default` то файлы располагаются в директории:
  `PGDATA/base/{oid базы данных}`
- Если в других табличных пространствах (значение столбца reltablespace в pg_class не равно нулю), то:
  `PGDATA/pg_tblspc/{reltablespace}/{oid базы данных}`
- Имена файлов объектов начинаются на relfilenode
- Для получения местоположения (относительно PGDATA) первого файла данных объекта используется функция `pg_relation_filepath(oid)`

Размеры табличных пространств и баз данных

- размер табличных пространств кластера `\db+` или функция `pg_tablespace_size()`

```
postgres=# \db+
                                  Список табличных пространств
    Имя     | Владелец | Расположение | Права доступа | Параметры | Размер
------------+----------+--------------+---------------+-----------+--------
 pg_default | postgres |              |               |           | 30 MB
 pg_global  | postgres |              |               |           | 565 kB
postgres=# SELECT spcname, pg_size_pretty(pg_tablespace_size(oid)) FROM pg_tablespace;
  spcname   | pg_size_pretty
------------+----------------
 pg_default | 30 MB
 pg_global  | 565 kB
```

- размер баз данных: `\l+` или функция `pg_database_size()`

```
postgres=# SELECT datname, pg_size_pretty(pg_database_size(datname)) FROM pg_database;
    datname    | pg_size_pretty
---------------+----------------
 postgres      | 7737 kB
 template1     | 7609 kB
 template0     | 7377 kB
 lab01iso88595 | 7537 kB
```

Функции определения размера

- Для получения размера файлов объектов есть набор функций
- **`pg_relation_size(regclass, 'main' | 'vm' | 'fsm' | 'init')`** выдаёт размеры слоёв
- **`pg_indexes_size()`** размер всех индексов созданных на таблицу
- **`pg_table_size()`** размер таблицы (TOAST и всех слоёв) без индексов
- **`pg_total_relation_size()`** размер таблицы включая TOAST, все индексы и слои

Перемещение объектов

- можно переместить файлы таблиц, индексов, материализованных представлений
- в новые файлы копируется содержимое файлов всех слоёв
- устанавливает монопольную блокировку, несовместимую даже с командой `SELECT`
- Команда `ALTER тип ALL IN TABLESPACE` перемещает все объекты одного типа в текущей базе данных

Смена схемы и владельца

- можно менять владельца объекта и схему у тех объектов, у которых они должны быть
- Для смены владельца используется команда:
  `ALTER тип_объекта имя OWNER TO роль;`
- Для смены схемы используется команда:
  `ALTER тип_объекта имя SET TO схема;`
- Для массового переназначения всех объектов ролей в одной базе на другую роль используется команда:
  `REASSIGN OWNED BY роль TO роль;`
- удаление объектов принадлежащих роли в базе:
  `DROP OWNED BY имя [CASCADE];`

Реорганизация и перемещение таблиц утилитой pg_repack

- утилита командной строки pg_repack
- имеется в Tantor Postgres
- нужно установить расширение в базах данных, где нужно реорганизовать таблицы
- реорганизация файлов таблиц с перемещением в другое табличное пространство или без перемещения
- в процессе реорганизации устанавливается блокировка ACCESS SHARE, позволяющая выполнять команды VACUUM, ANALYZE, SELECT, DML
- в конце реорганизации на короткое время устанавливается эксклюзивная блокировка

Уменьшение размера файлов таблиц утилитой pgcompacttable

- Утилита написанная на языке Perl
- использует стандартное расширение pgstattuple
- Отличия от pg_repack:
  - свободное место равно размеру самого большого индекса, а не двойной размер таблицы и индексов
  - таблицы обрабатываются с адаптивной задержкой, а не с полной нагрузкой
  - не может перемещать файлы в другое табличное пространство

TOAST (The Oversized-Attribute Storage Technique)

- Обычные таблицы (heap tables) хранят данные "построчно"
- Если строка не помещается в блоке данных, то используется технология TOAST
- Четыре режима хранения на уровне каждого столбца: `PLAIN`, `EXTERNAL`, `EXTENDED`, `MAIN`
- Для типов данных небольшого размера не предусмотрено хранение в TOAST и режим единственный: `PLAIN`
- для большинства остальных типов данных по умолчанию установлен режим `EXTENDED`
- используется сжатие на уровне полей, для полей небольшого размера сжатие неэффективно

```
create table t(n numeric storage main, t text storage plain);
alter table t alter column n set storage plain;
```

TOAST (The Oversized-Attribute Storage Technique)

- TOAST это техника хранения "атрибутов" (полей) большого размера
- поля большого размера - это поля не помещающиеся в блок
- позволяет хранить поля размером до 1Гб
- если в таблице есть заведомо большое поле или вставляется строка с большим полем, то создается toast-таблица и индекс на toast-таблицу
- в TOAST выносятся отдельные поля строк
- вынесенные поля делятся на части (chunk) по 1996 байт:

```
postgres@tantor:~$ pg_controldata | grep TOAST
Maximum size of a TOAST chunk:          1996
postgres@vanilla-x32:~$ pg_controldata | grep TOAST
Maximum size of a TOAST chunk:          2000
```

Поля переменной длины

- строка должна поместиться в один блок
- поля varlena, которые не помещаются в блок выносятся в таблицу TOAST
- типы данных фиксированной длины не сжимаются и не выносятся в TOAST
- стратегию хранения можно установить командой `ALTER TABLE имя ALTER COLUMN имя SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN | DEFAULT }`.
- сжатие поддерживается для MAIN и EXTENDED
- 1 байт в начале поля переменной длины хранит длину поля длиной до 127 байт
- 4 байта в начале поля переменной длины хранит длину поля для полей длиннее 126 байт

Вытеснение полей в TOAST

- выносятся поля строк длиной больше 2032 байт
  - при этом поля режутся на части по 1996 байт
- алгоритм (очередность) вытеснения зависит от порядка следования столбцов
- при доступе к каждому вытесненному полю дополнительно читается 2-3 блока TOAST-индекса
- функция проверки, вытеснено ли поле в TOAST: **pg_column_toast_chunk_id()**

```
select reltoastrelid, reltoastrelid::regclass
from pg_class where relname='t';
 reltoastrelid |      reltoastrelid
---------------+-------------------------
         74295 | pg_toast.pg_toast_74292
\d+ pg_toast.pg_toast_74292
TOAST table "pg_toast.pg_toast_74292"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_id   | oid     | plain
 chunk_seq  | integer | plain
 chunk_data | bytea   | plain
Owning table: "public.t"
Indexes:
    "pg_toast_74292_index" PRIMARY KEY,
            btree (chunk_id, chunk_seq)
Access method: heap
select chunk_id, chunk_seq, length(chunk_data)
from pg_toast.pg_toast_74292;
 chunk_id | chunk_seq | length
----------+-----------+--------
    74297 |         0 |   1996
    74297 |         1 |      9
```

Алгоритм вытеснения полей в TOAST

- при вставке строки в таблицу она полностью размещается в памяти серверного процесса в строковом буфере размера 1Гб (или 2Гб
- при обновлении строки обработка выполняется по затрагиваемым командой полям в пределах строкового буфера. Поля, не затрагиваемые командой, представлены в буфере заголовком длиной 18 байт.
- после обработки (сжатия или выноса) каждого поля проверяется размер строки. Если размер не превышает `toast_tuple_target` (по умолчанию 2032 байт) строка сохраняется в буфер и обработка строки заканчивается
- обработка начинается с поля EXTENDED и EXTERNAL от наибольшего размера к меньшему
- сжатие и вынос полей MAIN выполняется только после выноса всех полей EXTENDED и EXTERNAL
- По умолчанию, используется сжатие pglz

TOAST chunk

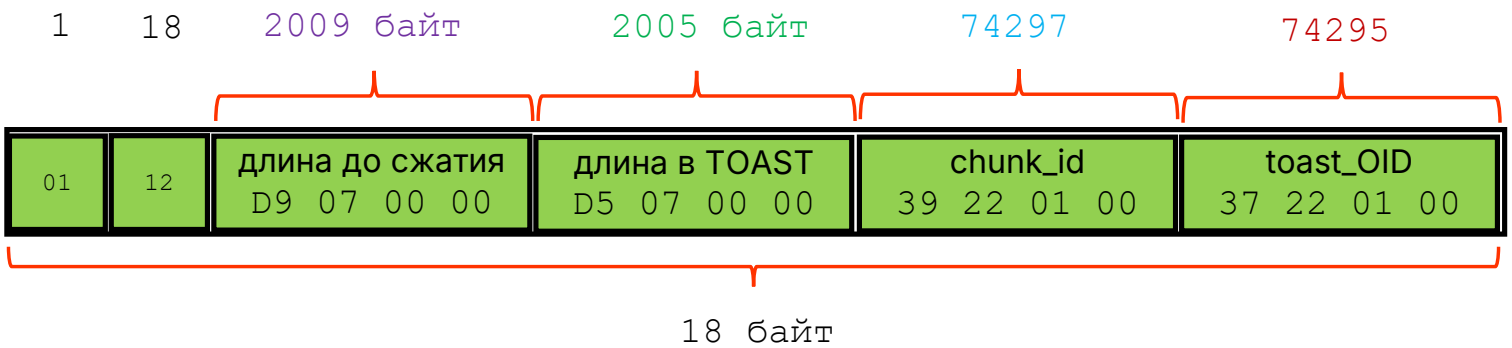
- при использовании EXTERNAL поля размером от 1997 байт создают второй чанк небольшого размера из-за которого в блок TOAST помещается только 3 чанка большого размера
- для полей EXTERNAL с размером от 1997 до ~2300 байт есть вероятность менее плотного хранения

```
select reltoastrelid, reltoastrelid::regclass
from pg_class where relname='t';
 reltoastrelid |      reltoastrelid
---------------+-------------------------
         74295 | pg_toast.pg_toast_74292
\d+ pg_toast.pg_toast_74292
TOAST table "pg_toast.pg_toast_74292"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_id   | oid     | plain
 chunk_seq  | integer | plain
 chunk_data | bytea   | plain
Owning table: "public.t"
Indexes:
    "pg_toast_74292_index" PRIMARY KEY,
            btree (chunk_id, chunk_seq)
Access method: heap
select chunk_id, chunk_seq, length(chunk_data)
from pg_toast.pg_toast_74292;
 chunk_id | chunk_seq | length
----------+-----------+--------
    74297 |         0 |   1996
    74297 |         1 |      9
```

Ограничения TOAST

- в таблицу TOAST может быть вынесено не больше 2^32 полей (~4млрд.)
- для вынесенного в TOAST поля в блоке таблицы хранится 18 байт, они не выравниваются, но вся строка выравнивается
- значение, остающееся в блоке таблицы, после выноса поля в TOAST:

```
select lp_len, t_data from heap_page_items(get_raw_page('t', 0));
 lp_len |                    t_data
--------+-------------------------------------------
     42 | \x0112d9070000d5070000392201003722 0100
```



Колоночное хранение: общая информация

- снизить трудоёмкость доступа к данным в столбцах путём совместного хранения значений столбцов
- возможно эффективно сжимать данные по сравнению со сжатием полей в heap tables
- Для использования колоночного способа хранения достаточно указать при создании таблицы:
  `CREATE TABLE имя (...) USING columnar;`
- расширение создаёт табличный метод доступа **`columnar`**
- список методов доступа есть в таблице `pg_am`

Колоночное хранение: особенности использования

- поддерживаются UPDATE и DELETE
- поддерживаются команды: TRUNCATE, INSERT (в том числе одной строки), COPY
- параллельное сканирование реализовано
- поддерживаются типы индексов, используемые в ограничениях целостности: btree, hash
- остальные типы индексов не поддерживаются: gist, gin, spgist, brin
- совместимость с секционированием таблиц: секции могут иметь разные форматы хранения
- нет служебных столбцов xmin, xmax
- есть служебный столбец ctid

Колоночное хранение: параметры

- у расширения есть параметры с префиксом "**`columnar.`**"
- последовательная запись в таблицу упорядоченных данных может улучшить сжатие, значительно сократить размер индексов и объем сканируемых блоков данных
- данные часто упорядочиваются по времени и называются "Time Series"
- наиболее эффективный алгоритм сжатия zstd, он установлен по умолчанию
- при чтении небольших объемов данных использование индексов может оказаться более эффективным

Демонстрация

- Директория для временных файлов
- Перемещение директории табличного пространства

Практика

1. Создание соединения с базой данных
2. Содержимое табличного пространства
3. Файл последовательности
4. Перемещение таблицы в другое табличное пространство
5. Перемещение таблицы в другое табличное пространство утилитой pg_repack
6. Использование pgcompacttable
7. Колоночное хранение pg_columnar

5

Диагностический журнал

Диагностический журнал

- накапливает диагностические сообщения от процессов экземпляра
- используется для:
  - диагностики проблем
    - мониторинга и настройки производительности
  - аудита безопасности
  - исторического анализа того, что происходило при работе экземпляра
  - анализа выполнения запросов

```
postgres@tantor:~$ cat $PGDATA/log/postgresql-*.log
23:17:09.415 [784] LOG:  database system is ready to accept connections
23:17:09.732 [791] LOG:  autoprewarm successfully prewarmed 13763 of 13763 previously-loaded blocks
23:27:09.762 [786] LOG:  checkpoint starting: time
23:27:19.982 [800] STATEMENT:  select * from tickets1 where ticket_no='0005432020304';
23:27:21.200 [800] ERROR:  index "tickets1_ticket_no_idx" contains unexpected zero page at block 3
23:27:21.200 [800] HINT:  Please REINDEX it.
```

Уровни важности сообщений

- В коде ядра PostgreSQL, коде библиотек расширений, коде plpgsql сообщения помечаются уровнями важности (severity level)
- **log_min_messages** устанавливает сообщения каких уровней важности будут передаваться в диагностический журнал
  - › допустимые значения и порядок важности для этого параметра: DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, INFO, NOTICE, WARNING, ERROR, LOG, FATAL, PANIC
- **client_min_messages** устанавливает сообщения каких уровней важности будут передаваться клиенту, создавшему сессию
  - › допустимые значения и порядок важности для этого параметра: DEBUG5, DEBUG4, DEBUG3, DEBUG2, DEBUG1, LOG, NOTICE, WARNING, ERROR

```
postgres=# \dconfig *_min_messages
List of configuration parameters
      Parameter       |  Value
----------------------+---------
 client_min_messages | notice
 log_min_messages    | warning
(2 rows)
```

Расположение журнала

- `log_destination` определяет местоположение, куда будут выводиться диагностические сообщения
  - допустимые значения: `stderr, csvlog, jsonlog, syslog`
- если местоположений несколько, то они будут выводиться одновременно во все местоположения
- `logging_collector=on` запускает фоновый процесс **`logger`**, который перехватывает stderr и направляет сообщения в директорию `log_directory`
- в корне `PGDATA` создаётся текстовый файл с названием `current_logfiles`, в который записывается местоположение и текущие (куда сейчас идет запись) названия файлов диагностического журнала
- `log_filename` задаёт название файла (в `log_destination` одно значение `stderr`) или файлов лога
  - удобно значение `log_filename = postgresql-%F.log`

Передача сообщений syslog

- при `log_destination = syslog` сообщения передаются службе операционной системы syslog
- в вывод syslog попадают не все сообщения, в **`logger`** все
  - **`logger`** предпочтительнее - он не теряет сообщения
- параметры конфигурации для взаимодействия с syslog:

```
postgres=# \dconfig syslog*
  List of configuration parameters
        Parameter         |  Value
--------------------------+----------
 syslog_facility          | local0
 syslog_ident             | postgres
 syslog_sequence_numbers  | on
 syslog_split_messages    | on
(4 rows)
```

Ротация файлов диагностического журнала

- при использовании **`logger`** файлы журнала могут ротироваться
  - старые файлы не удаляются
- Параметр `log_truncate_on_rotation` позволяет при ротации по времени (но не при ротации по размеру или при запуске экземпляра) перезаписывать существующие файлы лога, а не дописывать в их конец
- Параметр `log_file_mode` задаёт разрешения на файлы диагностического журнала (по умолчанию `0600`)
  - значение `0640` разрешит читать файлы членам группы postgres

```
postgres=# \dconfig *rotation*
 List of configuration parameters
        Parameter          | Value
---------------------------+-------
 log_rotation_age          | 1d
 log_rotation_size         | 10MB
 log_truncate_on_rotation | off
(3 rows)
```

Диагностический журнал

- `logging_collector` (по умолчанию `off`) рекомендуется установить значение в `on`
- запустится фоновый процесс **`logger`**, который собирает (collect) сообщения, отправленные в stderr и записывает их в файлы лога
- `log_min_messages=WARNING` по умолчанию, что означает логирование сообщений с уровнями `WARNING, ERROR, LOG, FATAL, PANIC`
- `log_min_error_statement=ERROR` по умолчанию. Минимальный уровень важности для команд SQL, которые завершились с ошибкой
- `log_directory=log` (PGDATA/log) по умолчанию. Задаёт путь к директории файлов лога, можно поменять на точку монтирования, скорость и объем которых достаточны для приема логов. Есть параметры, настраивающие ротацию файлов лога

```
psql -c "alter system set logging_collector = on;"
sudo systemctl restart tantor-se-server-18
ps -ef | grep logger
postgres   21861   21860  0 09:37  00:00:00 postgres: logger
```

Параметры диагностики

- параметров конфигурации логирования больше 35
- также есть больше 8 параметров для отладки команд SQL
- расширения могут иметь параметры для логирования
- основные параметры:

```
alter system set logging_collector=on;
alter system set log_min_duration_statement='8s';
alter system set log_statement=ddl;
alter system set log_min_error_statement=ERROR;
alter system set log_temp_files='1MB';
alter system set cluster_name='main';
alter system set log_autovacuum_min_duration='10s';
alter system set log_disconnections=on;
alter system set log_connections=on;
alter system set log_lock_waits=true;
alter system set deadlock_timeout='60s';
alter system set log_recovery_conflict_waits=on;
select pg_reload_conf();
```

```
\dconfig *debug*
debug_assertions                          | off
debug_discard_caches                      | 0
debug_io_direct                           |
debug_logical_replication_streaming       | buffered
debug_parallel_query                      | off
debug_pretty_print                        | on
debug_print_parse                         | off
debug_print_plan                          | off
debug_print_rewritten                     | off
jit_debugging_support                     | off
(10 rows)
\dconfig log*
log_autovacuum_min_duration               | 10min
log_checkpoints                           | on
log_connections                           | off
log_disconnections                        | off
log_duration                              | off
log_error_verbosity                       | default
log_executor_stats                        | off
logging_collector                         | off
log_lock_waits                            | off
log_file_mode                             | 0600
log_filename                              | postgresql-
%Y-%m-%d_%H%M%S.log
logging_collector                         | on
log_hostname                              | off
logical_decoding_work_mem                 | 64MB
log_line_prefix                           | %m [%p]
log_lock_waits                            | off
log_min_duration_sample                   | -1
log_min_duration_statement                | -1
```

Отслеживание использования временных файлов

- `cluster_name = 'main'` По умолчанию пусто. Рекомендуется установить. Значение добавляется к названию процессов экземпляра, что упрощает их идентификацию. На реплике по умолчанию используется для идентификации `wal_receiver`
- `log_temp_files='1MB'` (по умолчанию отключено) логирует имена и размеры создаваемых временных файлов **в момент их удаления**
- при нулевом значении логируются файлы любого размера
- временные файлы создаются в директории табличных пространств, указанных в параметре `temp_tablespaces`
  - можно ограничить параметром `temp_file_limit`
  - рекомендуется установить `log_temp_files` и `temp_file_limit`
- пример сообщения о том, что временный файл дорос до 97Мб:

```
STATEMENT:  CREATE INDEX ON test(id);
LOG:  temporary file: path "base/pgsql_tmp/pgsql_tmp137894.0.fileset/0.0",
size 101810176
```

Отслеживание работы автовакуума и автоанализа

- `log_autovacuum_min_duration`, по умолчанию установлен в 10 минут. Если автовакуум превысит это время при обработке таблицы, то в лог кластера запишется сообщение. При возникновении таких сообщений стоит выяснять причину долгого вакуумирования таблицы

```
LOG:  automatic vacuum of table "postgres.public.test": index scans: 37
pages: 0 removed, 88496 remain, 88496 scanned (100.00% of total)
tuples: 10000000 removed, 10000000 remain, 0 are dead but not yet removable
removable cutoff: 799, which was 0 XIDs old when operation ended
new relfrozenxid: 798, which is 2 XIDs ahead of previous value
frozen: 1 pages from table (0.00% of total) had 82 tuples frozen
index scan needed: 44249 pages from table (50.00% of total) had 10000000 dead item identifiers removed
index "test_id_idx": pages: 54840 in total, 0 newly deleted, 0 currently deleted, 0 reusable
avg read rate: 518.021 MB/s, avg write rate: 47.473 MB/s
buffer usage: 385747 hits, 1864788 misses, 170895 dirtied
WAL usage: 231678 records, 83224 full page images, 248448578 bytes
system usage: CPU: user: 25.72 s, system: 0.38 s, elapsed: 28.12 s

LOG:  automatic analyze of table "postgres.public.test"
avg read rate: 498.808 MB/s, avg write rate: 0.018 MB/s
buffer usage: 2906 hits, 27199 misses, 1 dirtied
system usage: CPU: user: 0.42 s, system: 0.00 s, elapsed: 0.42 s
```

Наблюдение за контрольными точками

- первая запись появляется в момент начала контрольной точки
- total = 09:31:35.070 - 09:27:05.095
  - 270 секунд = checkpoint_completion_target * checkpoint_timeout = 0.9*300
- total=write+sync время записи в WAL-файлы
- sync= время, затраченное на вызовы fdatasync по WAL-файлам
- sync files=15 (синхронизировано файлов) - число файлов данных, в которые велась запись, по ним в конце контрольной точке посылается вызов fsync
- longest=0.003 s (самая_долгая синхр.) - наибольшая длительность обработки одного файла

```
09:27:05.095 LOG:  checkpoint starting: time
09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%), wrote 1 SLRU
buffers; 0 WAL file(s) added, 0 removed, 6 recycled; write=269.938 s, sync=0.009 s,
total=269.976 s; sync files=15, longest=0.003 s, average=0.001 s; distance=109699 kB,
estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/BC98978
```

Описание записей log_checkpoints

- `wrote 4315 buffers` число грязных буферов, которые записаны по контрольной точке. Одновременно с checkpointer грязные блоки могут записывать другие процессы экземпляра
- `(26.3%)` процент от общего количества буферов буферного кэша, задаваемых параметром `shared_buffers`
- `file(s) added, 0 removed, 6 recycled` число созданных, удалённых, повторно использованных WAL файлов
- `distance=109699 kB` (расстояние) - объем записей WAL между **началом предыдущей** контрольной точки и началом текущей

```
 09:22:05.087 LOG: checkpoint starting: time
 09:26:35.066 LOG: checkpoint complete: wrote 3019 buffers (18.4%), wrote 1 SLRU buffers; 0
WAL file(s) added, 0 removed, 6 recycled; write=269.951 s, sync=0.009 s, total=269.980 s;
sync files=14, longest=0.004 s, average=0.001 s; distance=99467 kB, estimate=108859 kB;
lsn=8/AA004C8, redo lsn=8/5177990
 09:27:05.095 LOG: checkpoint starting: time
 09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%); 0 WAL file(s) added, 0
removed, 6 recycled;
 write=269.938 s, sync=0.009 s, total=269.976 s; sync files=15, longest=0.003 s,
average=0.001 s; distance=109699 kB, estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/BC98978
```

Описание записей log_checkpoints

- estimate=109699 kB (расстояние которое ожидалось) рассчитывается, чтобы оценить сколько WAL сегментов будет использовано в следующей контрольной точке
- если нули в "0 WAL file(s) added, 0 removed", то оценка estimate верная. Сколько файлов удалить, определяется параметрами min_wal_size, max_wal_size, wal_keep_size, max_slot_wal_keep_size, wal_init_zero=on, wal_recycle=on

```
 09:22:05.087 LOG: checkpoint starting: time
 09:26:35.066 LOG: checkpoint complete: wrote 3019 buffers (18.4%), wrote 1 SLRU
buffers; 0 WAL file(s) added, 0 removed, 6 recycled; write=269.951 s, sync=0.009 s,
total=269.980 s; sync files=14, longest=0.004 s, average=0.001 s; distance=99467 kB,
estimate=108859 kB; lsn=8/AA004C8, redo lsn=8/5177990
 09:27:05.095 LOG: checkpoint starting: time
 09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%); 0 WAL file(s)
added, 0 removed, 6 recycled;
 write=269.938 s, sync=0.009 s, total=269.976 s; sync files=15, longest=0.003 s,
average=0.001 s;
 distance=109699 kB, estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/BC98978
```

Утилита pg_waldump и записи log_checkpoints

- для просмотра записей в WAL-файлах используется утилита `pg_waldump`. По умолчанию утилита ищет WAL-файлы в "." (текущей директории откуда она запущена), дальше в `./pg_wal`, `$PGDATA/pg_wal`
- в логе и выводе `pg_controldata` в LSN ведущие нули после "/" не печатаются
- в выводе `pg_waldump` в lsn и prev ноль печатается, а в redo не печатается

```
pg_controldata | grep check | head -n 3
Latest checkpoint location:           8/1164B2E8
Latest checkpoint's REDO location:    8/0BC98978
Latest checkpoint's REDO WAL file:    000000010000000080000000B
```

```
pg_waldump -s 8/0B000000 | grep CHECKPOINT
rmgr: XLOG len (rec/tot): 148/148, tx: 0,
lsn: 8/1164B2E8, prev 8/1164B298, desc: CHECKPOINT_ONLINE redo 8/0BC98978;
tli 1; prev tli 1; fpw true; xid 8064948; oid 33402; multi 1; offset 0; oldest xid 723 in DB 1;
oldest multi 1 in DB 5; oldest/newest commit timestamp xid: 0/0; oldest running xid 8064947; online
pg_waldump: error: error in WAL record at 8/1361C488: invalid record length at 8/1361C4B0:
expected at least 26, got 0
```

```
09:27:05.095 LOG:  checkpoint starting: time
09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%), wrote 1 SLRU buffers; 0 WAL file(s) added,
0 removed, 6 recycled; write=269.938 s, sync=0.009 s, total=269.976 s; sync files=15, longest=0.003 s,
average=0.001 s; distance=109699 kB, estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/0BC98978
```

Утилита pg_waldump и записи log_checkpoints

- lsn 8/1164B2E8 запись о конце контрольной точки
- redo 8/0BC98978 запись о начале контрольной точки, с которой начнется восстановление в случае сбоя экземпляра
- prev 8/1164B298 адрес начала предыдущей записи в журнале
- distance объем журнала от начала предыдущей до начала завершенной контрольной точки '8/0BC98978'::pg_lsn-'8/05177990'::pg_lsn

```
pg_controldata | grep check | head -n 3
Latest checkpoint location:           8/1164B2E8
Latest checkpoint's REDO location:    8/BC98978
Latest checkpoint's REDO WAL file:    000000010000000080000000B
```

```
pg_waldump -s 8/0B000000 | grep CHECKPOINT
rmgr: XLOG len (rec/tot): 148/148, tx: 0,
lsn: 8/1164B2E8, prev 8/1164B298, desc: CHECKPOINT_ONLINE redo 8/BC98978;...
```

```
 09:26:35.066 LOG: checkpoint complete: wrote 3019 buffers (18.4%), wrote 1 SLRU buffers; 0 WAL file(s) added,
0 removed, 6 recycled; write=269.951 s, sync=0.009 s, total=269.980 s; sync files=14, longest=0.004 s,
average=0.001 s; distance=99467 kB, estimate=108859 kB; lsn=8/AA004C8, redo lsn=8/5177990
 09:27:05.095 LOG:  checkpoint starting: time
 09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%); 0 WAL file(s) added, 0 removed,
 6 recycled; write=269.938 s, sync=0.009 s, total=269.976 s; sync files=15, longest=0.003 s, average=0.001 s;
 distance=109699 kB, estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/BC98978
```

Логирование соединений

- выявляет нет ли слишком частых соединений и коротких сессий
- порождение сессии для выполнения одного запроса не оптимально
- Для логирования соединений используются параметры:

```
log_connections={all, authentification, receipt,
  authorization, setup_duration, on, off}
log_disconnections=on
pgaudit.log_connections=on
pgaudit.log_disconnections=on
```

```
LOG: connection received: host=[local]
LOG: connection authorized: user=postgres database=db2 application_name=psql
FATAL: database "db2" does not exist
LOG: connection received: host=[local]
LOG: connection authorized: user=alice database=alice application_name=psql
FATAL: role "alice" does not exist
```

Параметр log_connections

- `log_connections` записывает в диагностический журнал кластера попытки подключения к экземпляру, попытки аутентификации и успешную аутентификацию
- изменить значение можно только на уровне кластера
- ни на уровне роли, ни на уровне базы данных параметра не может устанавливаться:

```
postgres=# alter user alice set log_connections = 'all';
ERROR:  parameter "log_connections" cannot be set after connection start
```

- пример сообщений:

```
LOG:  connection received: host="10.0.2.15"
LOG:  connection authorized: user=fff database=fff application_name=psql
FATAL:  role "fff" does not exist
```

Параметр log_disconnections

- `log_disconnections=on` записывает в диагностический журнал одно сообщение при остановке серверного процесса, обслуживавшего сессию
- изменить значение можно только на уровне кластера, а также на клиенте перед установкой соединения:

```
export PGOPTIONS="-c log_disconnections=on"
```

- ни на уровне роли, ни на уровне базы данных параметра не может устанавливаться:

```
postgres=# alter database postgres set log_disconnections = 'all';
ERROR:  parameter "log_disconnections" cannot be set after connection start
```

- пример сообщений:

```
 LOG:  disconnection: session time: 0:00:00.007 user=postgres database=postgres
host=127.0.0.1 port=34298
```

Расширения pgaudit и pgaudittofile

- расширениями `pgaudit` и `pgauditlogtofile` можно направить сообщения о создании сессий и их длительности в отдельный файл или файлы аудита
- для работы расширений нужно загрузить их библиотеки:

```
alter system set shared_preload_libraries = pgaudit, pgauditlogtofile;
```

- Расширения работают независимо и параллельно с журналом кластера и управляются собственными параметрами, которые имеют префикс "`pgaudit.`"
- параметры `pgaudit.log_connections` и `pgaudit.log_disconnections` аналогичны одноимённым параметрам PostgreSQL и могут создавать аналогичные записи в отдельном файле аудита

Конфигурирование расширений pgaudit и pgaudittofile

- восемь параметров относятся к библиотеке `pgauditlogtofile`
- чтобы создавался журнал аудита нужно установить параметр `pgaudit.log` как минимум в значение `'misc'`
  - `'none'` - файл журнала аудита не создаётся
  - `'role'` и `'ddl'` `pgaudit.log_connections` и `pgaudit.log_disconnections` не действуют

```
postgres=# \dconfig pgaudi*
            Parameter              |    Value
-----------------------------------+----------
 pgaudit.ddl_notemp                | off
 pgaudit.dml_notemp                | off
 pgaudit.log                       | none
 pgaudit.log_autoclose_minutes     | 0
 pgaudit.log_catalog               | on
 pgaudit.log_client                | off
 pgaudit.log_connections           | off
 pgaudit.log_directory             | log
 pgaudit.log_disconnections        | off
 pgaudit.log_filename              | audit-%F.log
 pgaudit.log_format                | csv
 pgaudit.log_level                 | log
 pgaudit.log_parameter             | off
 pgaudit.log_parameter_max_size    | 0
 pgaudit.log_relation              | off
 pgaudit.log_rotation_age          | 1d
 pgaudit.log_rotation_size         | 0
 pgaudit.log_rows                  | off
 pgaudit.log_statement             | on
 pgaudit.log_statement_once        | off
 pgaudit.marking_log_directory     |
 pgaudit.marking_log_filename      | pgaudit_mark-
%Y-%m-%d.csv
 pgaudit.marking_rules_enabled     | off
 pgaudit.marking_rules_max         | 1000
 pgaudit.role                      |
 pgaudit.whitelist_role            |
(26 rows)
```

Диагностика частоты соединений с базой данных

- `log_disconnections=on` записывает в лог событие завершения сессии. Записывается та же информация, что `log_connections` плюс длительность сессии.
  - преимущество в том, что выводится одна строка, что не замусоривает лог
  - позволяет идентифицировать короткие по времени сессии
  - пример сообщения о длительности сессии сессии 4 секунды:

```
LOG: disconnection: session time: 0:00:04.056 user=oleg database=db1 host=[vm1]
```

- `log_connections=on` записывает в лог попытки установить сессию
  - недостаток в том, что неудачные попытки отличаются только дополнительной строкой:

```
LOG: connection received: host=[local]
LOG: connection authorized: user=postgres database=db2 application_name=psql
FATAL: database "db2" does not exist
LOG: connection received: host=[local]
LOG: connection authorized: user=alice database=alice application_name=psql
FATAL: role "alice" does not exist
```

Диагностика блокирующих ситуаций

- `log_lock_waits=true`. По умолчанию отключен. Рекомендуется включить, чтобы получать сообщения в диагностический журнал что какой-либо процесс ждет дольше, чем: `deadlock_timeout`
- `deadlock_timeout='60s'`. По умолчанию 1 секунда, что слишком мало и на нагруженных экземплярах создает значительные издержки
- `log_startup_progress_interval='10s'` не стоит отключать
- `log_recovery_conflict_waits=on`. По умолчанию `off`. Процесс `startup` запишет сообщение в лог реплики, если не сможет применить WAL к реплике дольше, чем `deadlock_timeout`

```
LOG: recovery still waiting after 60.555 ms: recovery conflict on lock
DETAIL: Conflicting process: 5555.
```

- Наличие конфликтов можно увидеть в представлении (мало деталей):

```
select * from pg_stat_database_conflicts where datname='postgres';
datid|datname |tblspc|confl_lock|confl_snapshot|confl_bufferpin|deadlock
-----+--------+------+----------+--------------+---------------+--------
13842|postgres|     0 |         0 |             1 |              1 |       0
```

Практика

1. Какая информация попадает в журнал
2. Расположение журналов сервера
3. Как информация попадает в журнал
4. Добавление формата csv
5. Включение коллектора сообщений

6

Безопасность

Пользователи (роли) в кластере баз данных

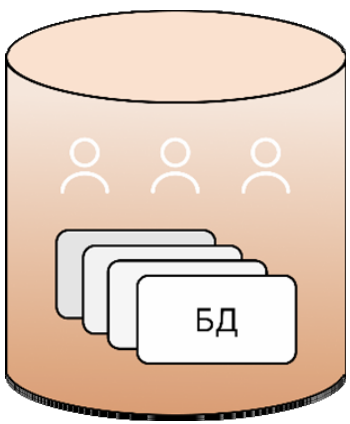
- роль синоним пользователя
- общий объект кластера
- у объектов может быть только одна роль-владелец
- у ролей нет роли-владельца
- ролям можно предоставлять привилегии на объекты
- у ролей есть атрибуты
- разница между командами CREATE USER и CREATE ROLE:

```
postgres=# create user alice;
CREATE ROLE
postgres=# create role bob;
CREATE ROLE
postgres=# \du
                                 List of roles
 Role name |   Attributes
-----------+-------------------------------------
 alice     |
 bob       | Cannot login
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS
```

Пользователи (роли)

- Список ролей можно посмотреть командами **\duS \dgS**
- Список ролей хранится в глобальной таблице **pg_authid**
- Есть псевдороль public, в которую входят все роли
  - ей можно выдавать привилегии
- Атрибуты роли postgres:

```
postgres=# select * from pg_authid where rolname='postgres'\gx
-[ RECORD 1 ]--+------------------------------
oid            | 10
rolname        | postgres
rolsuper       | t
rolinherit     | t
rolcreaterole  | t
rolcreatedb    | t
rolcanlogin    | t
rolreplication | t
rolbypassrls   | t
rolconnlimit   | -1
rolpassword    | SCRAM-SHA-256$4096:oejDqb5w...
rolvaliduntil  |
```



Атрибуты (параметры, свойства) пользователей

- есть 7 атрибутов типа включён/выключен
- LOGIN - право создания начального соединения с базами данных
- SUPERUSER - обходит проверки прав доступа, кроме начального соединения
- REPLICATION LOGIN - (без атрибута LOGIN атрибут REPLICATION бесполезен) имеет право резервировать весь кластер
- CREATEROLE - может создавать новые роли
  - у ролей нет владельца
  - созданная роль грантуется создающей роли с опцией ADMIN OPTION, но без SET и без INHERIT, поэтому ни грант WITH ADMIN, ни атрибут CREATEROLE не позволяют поднять своей роли привилегии путем создания и переключения в созданную роль
- Грант с ADMIN OPTION не даёт право менять атрибуты CREATEROLE, BYPASSRLS, REPLICATION, CREATEDB, SUPERUSER
  - эти атрибуты роль сможет менять у ролей, на которые есть ADMIN OPTION, только, если у неё имеется такой же атрибут

Параметр конфигурации `createrole_self_grant`

- По умолчанию, создатель не получает SET и INHERIT, только ADMIN

```
postgres@postgres=# create user alice createrole;
CREATE ROLE
postgres@postgres=# grant all on schema public to alice with grant option;
GRANT
postgres@postgres=# \c postgres alice
You are now connected to database "postgres" as user "alice".
alice@postgres=> create user bob createrole;
CREATE ROLE
alice@postgres=> set role bob;
ERROR:  permission denied to set role "bob"
alice@postgres=> grant all on schema public to bob;
GRANT
alice@postgres=> drop user bob;
ERROR:  role "bob" cannot be dropped because some objects depend on it
DETAIL:  privileges for schema public
alice@postgres=> revoke all on schema public from bob;
REVOKE
alice@postgres=> drop user bob;
DROP ROLE
```

Выданные пользователям привилегии

- Зависимости объектов хранятся в `pg_shdepend` и `pg_depend`
- Функция `pg_get_acl()` выдаёт список прав доступа

```
\pset xheader_width 20
Expanded header width is 20.
select pg_describe_object(classid, objid, objsubid) obj,
pg_describe_object(refclassid, refobjid, 0) ref, pg_get_acl(classid,
objid, objsubid) acl from pg_shdepend\gx
-[ RECORD 1 ]-------
obj | schema public
ref | role alice
acl | {pg_database_owner=UC/pg_database_owner,=U/pg_database_owner,
alice=U*C*/pg_database_owner, bob=UC/pg_database_owner}
-[ RECORD 2 ]-------
obj | schema public
ref | role bob
acl | {pg_database_owner=UC/pg_database_owner,=U/pg_database_owner,
alice=U*C*/pg_database_owner, bob=UC/pg_database_owner}
```

Атрибут INHERIT и GRANT WITH INHERIT

- Атрибут INHERIT устанавливается по умолчанию
- Позволяет наследовать права, выданные на объекты баз данных
- Если роли установить атрибут NOINHERIT, то она не будет наследовать привилегии ролей, которые ей даны
- При выдаче роли можно явно задать
  - будут ли работать права грантованной роли у сессий грантуемого
  - может ли грантуемый переключиться в грантованную роль

```
postgres=# grant postgres to alice with inherit false, set true;
GRANT ROLE
postgres=# \c postgres alice
You are now connected to database "postgres" as user "alice".
postgres=> set role postgres;
SET
postgres=#
postgres=# grant postgres to bob with inherit true, set false;
GRANT ROLE
postgres=# \c postgres bob
You are now connected to database "postgres" as user "bob".
postgres=> set role postgres;
ERROR:  permission denied to set role "postgres"
```

Переключение сессии в другую роль и смена ролей

- `SET [SESSION | LOCAL] SESSION AUTHORIZATION роль;` переключает сессию в другую роль
- Команду может выполнить только суперпользователь
  - используется для того, чтобы суперпользователь мог переключить сессию в другого пользователя и затем вернуться в исходную
- Текущего пользователя можно изменить командой `SET ROLE`
  - проверка прав на объекты выполняется для текущего пользователя

```
postgres=# set session authorization alice;
postgres=> select current_user, session_user, user;
 current_user | session_user | user
--------------+--------------+-------
 alice        | alice        | alice
postgres=> set role bob;
postgres=> select current_user, session_user, user;
 bob          | alice        | bob
postgres=> reset role;
postgres=> select current_user, session_user, user;
 alice        | alice        | alice
postgres=> reset session authorization;
postgres=# select current_user, session_user, user;
 postgres     | postgres     | postgres
```

Предопределённые (служебные) роли

- появились в PostgreSQL начиная с версии 14
- 15 ролей плюс псевдороль public
- может грантовать роль с атрибутами SUPERUSER или роль, имеющая право WITH ADMIN на грантуемую роль
- удалить служебные роли нельзя
- полезны чтобы не давать привилигею суперпользователя для выполнения востребованных действий

```
postgres=# \duS
                              List of roles
           Role name            |   Attributes
--------------------------------+---------------
 pg_checkpoint                  | Cannot login
 pg_create_subscription         | Cannot login
 pg_database_owner              | Cannot login
 pg_execute_server_program      | Cannot login
 pg_maintain                    | Cannot login
 pg_monitor                     | Cannot login
 pg_read_all_data               | Cannot login
 pg_read_all_settings           | Cannot login
 pg_read_all_stats              | Cannot login
 pg_read_server_files           | Cannot login
 pg_signal_backend              | Cannot login
 pg_stat_scan_tables            | Cannot login
 pg_use_reserved_connections    | Cannot login
 pg_write_all_data              | Cannot login
 pg_write_server_files          | Cannot login
```

Права на объекты

- При создании объекта назначается владелец
- По умолчанию владелец и суперпользователь имеют права на созданный объект
- Другим ролям нужны права
- Для каждого типа объекта есть свой набор прав
  - названия прав приведены в таблице
- Права выдаются и отзываются командами GRANT и REVOKE

право		типы объектов
SELECT	r	LARGE OBJECT, SEQUENCE, TABLE (и подобные им), COLUMN
INSERT	a	TABLE, столбец
UPDATE	w	LARGE OBJECT, SEQUENCE, TABLE, COLUMN
DELETE	d	TABLE
TRUNCATE	D	TABLE
REFERENCES	x	TABLE, COLUMN
TRIGGER	t	TABLE
CREATE	C	DATABASE, SCHEMA, TABLESPACE
CONNECT	c	DATABASE
TEMPORARY	T	DATABASE
EXECUTE	X	FUNCTION, PROCEDURE
USAGE	U	DOMAIN, FDW, FOREIGN SERVER, LANGUAGE, SCHEMA, SEQUENCE, TYPE
SET	s	PARAMETER
ALTER SYSTEM	A	PARAMETER
MAINTAIN	m	TABLE

Просмотр прав на объекты в psql

- Права выводятся в виде списка:
- кому_дана=привилегии/кто_дал
- Если перед значком "=" пусто, это означает public (предоставлено всем)
- Значок "*" после буквы означает, что право предоставлено с правом передачи WITH GRANT OPTION.
- Значок "+" в конце обозначает, что это не последний элемент и список продолжается на следующей строке

```
postgres=# \l
                              List of databases
   Name |  Owner   |   Access privileges
--------+----------+-----------------------
 demo   | postgres | postgres=CTc/postgres+
        |          | alice=C*c/postgres
postgres=# GRANT ALL PRIVILEGES ON ALL TABLES IN
SCHEMA public TO alice, bob WITH GRANT OPTION;
GRANT
```

Тип объекта	Все права	Права PUBLIC	Команда psql
DATABASE	CTc	Tc	\l
DOMAIN	U	U	\dD+
FUNCTION, PROCEDURE	X	X	\df+
FDW	U	нет	\dew+
FOREIGN SERVER	U	нет	\des+
LANGUAGE	U	U	\dL+
LARGE OBJECT	rw	нет	\dl+
PARAMETER	sA	нет	\dconfig+
SCHEMA	UC	нет	\dn+
SEQUENCE	rwU	нет	\dp
TABLE (и подобные им)	arwdDxtm	нет	\dp
COLUMN	arwx	нет	\dp
TABLESPACE	C	нет	\db+
TYPE	U	U	\dT+

Привилегии по умолчанию (DEFAULT PRIVILEGES)

- `ALTER DEFAULT PRIVILEGES` позволяет задавать права, применяемые к объектам, которые будут создаваться в будущем для схем, таблиц, представлений, внешних таблиц, последовательностей, подпрограмм, типов (включают в себя домены)
- роль `public` получает права:
  - CONNECT и TEMPORARY (создание временных таблиц) для баз данных
  - EXECUTE для функций и процедур
  - USAGE для языков, типов данных, доменов

```
alter default privileges REVOKE ALL on routines from public;
alter default privileges REVOKE ALL on types from public;
revoke all on database demo from public;
revoke connect on database p2 from public;
\l
                              List of databases
   Name    |  Owner   | .. |   Access privileges
-----------+----------+----+-----------------------
 demo      | postgres | .. | postgres=CTc/postgres+
           |          | .. | alice=c/postgres
 p2        | postgres | .. | =T/postgres           +
           |          | .. | postgres=CTc/postgres
revoke all on language plpgsql from public;
\dL+
                              List of languages
  Name   |  Owner   | Trusted | .. |  Access privileges
---------+----------+---------+----+--------------------
 plpgsql | postgres | t       | .. | postgres=U/postgres
(1 row)
```

Защита на уровне строк (Row-level security, RLS)

- по умолчанию отключена
- RLS не является мандатным контролем доступа (MAC)
- если RLS включается, а разрешающих политик нет, то доступ запрещён
- политики создаются командой CREATE POLICY, например:
  ```
  CREATE POLICY имя ON таблица AS PERMISSIVE FOR ALL TO роль
  USING (предикат);
  ```
  - политик на таблице может быть несколько, они могут быть PERMISSIVE и/или RESTRICTIVE и могут комбинироваться AND и OR
- RLS включают на уровне таблиц командой:
  ```
  ALTER TABLE имя [ENABLE | DISABLE | FORCE |NO FORCE ] ROW
  LEVEL SECURITY;
  ```
- включение с опцией ENABLE:
  - RLS действуют на всех, кроме владельца и ролей с атрибутом SUPERUSER или BYPASSRLS
- RLS не применяется к проверкам ограничений целостности
  - ограничения целостности не могут нарушаться

Подсоединение к экземпляру

- Параметры начальной аутентификации устанавливаются в двух файлах `pg_hba.conf` и `pg_ident.conf`
- Файлы редактируются вручную
- содержимое файла `pg_hba.conf` можно в представлении `pg_hba_file_rules`
- Расположение файлов можно посмотреть параметрами конфигурации hba_file и ident_file:

```
postgres=# \dconfig *_file
                          List of configuration parameters
        Parameter         |                              Value
--------------------------+-----------------------------------------------------
 config_file              | /var/lib/postgresql/tantor-se-1c-17/data/postgresql.conf
 enable_delayed_temp_file | off
 external_pid_file        |
 hba_file                 | /var/lib/postgresql/tantor-se-1c-17/data/pg_hba.conf
 ident_file               | /var/lib/postgresql/tantor-se-1c-17/data/pg_ident.conf
```

Файл pg_hba.conf

- Формат файла - одна запись на строку
- Запись состоит из нескольких полей, разделённых пробелами и/или символами табуляции
- Содержимое полей может быть заключено в двойные кавычки
- записи (строки) ближе к началу файла превалируют, в отличие от файлов параметров (postgresql.conf)
- Текущее содержимое файла можно посмотреть через представление:

```
postgres=# select rule_number r, right(file_name, 11) file_name, line_number l, type, database, user_name
user, address, left(netmask, 15) netmask, auth_method auth, options opt, error from pg_hba_file_rules;
 r |  file_name  |  l  | type  |   database    | user  |  address  |     netmask     | auth  | opt | error
---+-------------+-----+-------+---------------+-------+-----------+-----------------+-------+-----+-------
 1 | pg_hba.conf | 117 | local | {all}         | {all} |           |                 | trust |     |
 2 | pg_hba.conf | 119 | host  | {all}         | {all} | 127.0.0.1 | 255.255.255.255 | trust |     |
 3 | pg_hba.conf | 121 | host  | {all}         | {all} | ::1       | ffff:ffff:ffff: | trust |     |
 4 | pg_hba.conf | 124 | local | {replication} | {all} |           |                 | trust |     |
 5 | pg_hba.conf | 125 | host  | {replication} | {all} | 127.0.0.1 | 255.255.255.255 | trust |     |
 6 | pg_hba.conf | 126 | host  | {replication} | {all} | ::1       | ffff:ffff:ffff: | trust |     |
(6 rows)
```

Содержимое pg_hba.conf

- тип соединения:
  - local - соединение через UNIX-сокет
  - host - любые (с шифрованием и без) соединения по TCP/IP
- название базы данных:
  - all - все базы
- replication - подключение по протоколу физической репликации
- IP-адрес с маской подсети или без маски:
  - all - все адреса IPv4 и IPv6
  - samehost - с IP-адресов хоста, на котором работает экземпляр

```
postgres@tantor:~$ tail $PGDATA/pg_hba.conf
# TYPE  DATABASE        USER        IP-ADDRESS    IP-MASK     METHOD
local   sameuser        all                                   peer
local   samerole        all                                   md5
host    all             all         127.0.0.1/32              trust
host    db1,db2,@file1  +bob,@file2 10.0.0.0      255.0.0.0   scram-sha-256
host    "/^db\d{2,4}$"  /^.*desk$   samehost                  ident map=omicron
host    all             all         all                       reject
```

Содержимое pg_hba.conf (продолжение)

- метод аутентификации:
- `trust` - установить соединение без проверок, в том числе пароля
- `reject` - безусловный отказ в соединении
- `peer` - имя пользователя операционной системы клиента должно совпадать с именем роли, есть опциональный параметр `map`
- `scram-sha-256` или `md5` - проверить пароль
- `gss` - аутентификация по протоколу kerberos
- `ldap` - аутентификация ldap-сервером
- `cert` - запрашивать у клиента сертификат SSL

```
postgres@tantor:~$ tail $PGDATA/pg_hba.conf
# TYPE  DATABASE        USER        IP-ADDRESS   IP-MASK     METHOD
local   sameuser        all                                  peer map=map1
local   samerole        all                                  md5
host    all             all         127.0.0.1/32             trust
host    db1,db2,@file1  +bob,@file2 10.0.0.0     255.0.0.0   scram-sha-256
hostssl "/^db\d{2,4}$"  /^.*desk$   localhost                cert
host    all             all         1.1.0.0/16               reject
```

Файл сопоставления имён pg_ident.conf

- Для методов **peer, gss, ident** можно сопоставить имя, возвращаемое службой аутентификации и роли кластера, под которой хочет установить сессию клиент
- на названия `MAPNAME` ссылается параметр `map=map1` в записях файла `pg_hba.conf`
- Представление `pg_ident_file_mappings` позволяет посмотреть текущее содержимое файла:

```
postgres=# select map_number r, right(file_name, 13) file_name, line_number l,
map_name, sys_name, pg_username, error from pg_ident_file_mappings;
 r |   file_name   | l  | map_name | sys_name | pg_username | error
---+---------------+----+----------+----------+-------------+-------
 1 | pg_ident.conf | 73 | map1     | astra    | postgres    |
 2 | pg_ident.conf | 75 | map1     | astra    | alice       |
(2 rows)
postgres=# \! tail -n 4 $PGDATA/pg_ident.conf
# MAPNAME SYSTEM-USERNAME PG-USERNAME
map1      astra           postgres
# astra также может подключаться с ролью alice
map1      astra           alice
```

Практика

1. Создание новой роли
2. Установка атрибутов
3. Создание групповой роли
4. Создание схемы и таблицы
5. Выдача роли доступа к таблице
6. Удаление созданных объектов
7. Расположение файлов конфигурации
8. Просмотр правил аутентификации
9. Локальные изменения для аутентификации
10. Проверка корректности настройки
11. Очистка ненужных объектов

7

7a

Физическое резервирование

Виды резервных копий

- Горячие - без простоя, без остановки экземпляра, без приостановки обслуживания клиентских сессий
- Холодные - на корректно (с финальной контрольной точкой) остановленном экземпляре
- Автономные или самодостаточные
- Инкрементальные

Холодные бэкапы

- Экземпляр должен быть корректно остановлен
- Утилиты `pg_basebackup` и `wal-g` не бэкапят, если экземпляр остановлен
- Используются утилиты операционной системы cp, rsync, tar
- Недостаток - прерывание обслуживания приложений, нужно учитывать символические ссылки
- Пример создания холодного бэкапа, если нет символических ссылок:

```
pg_ctl stop
cp -pr $PGDATA/* $HOME/backup
sudo systemctl start tantor-se-server-18
```

- Пример создания горячего бэкапа утилитой pg_basebackup:

```
pg_basebackup -D $HOME/backup
```

- Пример создания горячего бэкапа утилитой wal-g:

```
wal-g backup-push $PGDATA
```

Что нужно резервировать

- Директорию PGDATA
  - временные файлы не резервируются
- Директории табличных пространств
- Файлы WAL от начала контрольной точки перед копированием файлов кластера до момента, на который нужно восстановиться
- Для полного восстановления нужно настроить "архив журналов"
  - утилитой `pg_receivewal`
  - и/или параметрами `archive_command` и `archive_mode=on`

Ограничения при создании бэкапа

- Пока создаётся бэкап не стоит менять базы данных, на основе которых создаются новые базы данных
- После создания или удаления табличного пространства рекомендуется сделать бэкап
- Если нужно создатьавтономный бэкап, то нужно включить в бэкап журналы, которые включают изменения с момента начала контрольной точки до момента окончания копирования файлов кластера

Архив журналов

- Способы организации архива журналов:
- прием (pull) потока журнальных записей `pg_receivewal`

```
pg_receivewal --create-slot --slot=arch
pg_receivewal -D $HOME/archivelog --slot=arch --synchronous
```

- передача (push) командой в параметре `archive_command`

```
alter system set archive_command = 'wal-g wal-push %p >>
$PGDATA/log/archive_command.log 2>&1';
alter system set archive_mode=on;
```

- Команда копирования журналов из архива при восстановлении:

```
alter system set restore_command = 'wal-g wal-fetch %f %p >>
$PGDATA/log/restore_command.log 2>&1 || cp $HOME/archivelog/%f %p || cp
$HOME/archivelog/%f.partial %p';
```

Процедура восстановления

- Остановить экземпляр
- Восстановить из полной резервной копии PGDATA и директории табличных пространств
- Применить инкрементальные бэкапы, если они создавались
- Создать файл `standby.signal` и установить значение параметра конфигурации `restore_command`, чтобы файлы журнала копировались и применялись из архива

```
pg_ctl stop -m immediate
rm -rf $PGDATA/*
wal-g backup-fetch $PGDATA LATEST
touch $PGDATA/standby.signal
echo "restore_command = 'wal-g wal-fetch %f %p >>
$PGDATA/log/restore_command.log 2>&1 || cp $HOME/archivelog/%f %p || cp
$HOME/archivelog/%f.partial %p'" >> $PGDATA/postgresql.auto.conf
```

Процедура восстановления (продолжение)

- Запустить экземпляр
- Проверить, что все журналы, которые есть, были применены
- Послать сигнал promote экземпляру, он откроется на чтение-запись

```
sudo systemctl start tantor-se-server-18
psql -c "select pg_last_wal_replay_lsn()"
psql -c "checkpoint"
psql -c "select pg_promote(true, 60)"
```

Пример восстановления

- удобнее использовать файл `standby.signal`, а не `recovery.signal`
- в примере делается бэкап

```
cd $HOME/backup
pg_ctl stop -m immediate -D .
rm -rf $HOME/backup/*
pg_basebackup -c fast -D .
touch standby.signal
echo "restore_command = 'cp /var/lib/postgresql/tantor-se-
18/data/pg_wal/%f %p || cp /var/lib/postgresql/tantor-se-
18/data/%f.partial %p'" >> ./postgresql.auto.conf
echo "port = 5433" >> ./postgresql.auto.conf
chmod 750 $HOME/backup
pg_ctl start -D .
psql -p 5433 -c "select pg_last_wal_replay_lsn();"
psql -c "checkpoint"
pg_ctl promote  -t 60 -D .
```

Применение журнальных записей (WAL)

- журнальные записи применяет только процесс startup
- управляется файлами `standby.signal`, `recovery.signal` и параметрами `restore_command` и `recovery_*`
- журнальная запись, с которой начинается применение (накат) берётся из текстового файла `PGDATA/backup_label` или бинарного управляющего файла `PGDATA/global/pg_control`
- процессу startup неизвестно какая журнальная запись формировалась последней перед сбоем кластера, процесс пытается применять все записи и файлы, которые имеются в `PGDATA/pg_wal`

Линии времени

- При изменении линии времени в директории `PGDATA/pg_wal` создаются текстовые файлы `00000000N.history` с информацией о линиях времени, их нельзя стирать и они будут резервироваться
- Новая линия времени появляется, если выполнялось восстановление или реплика стала мастером
- Цель линий времени:
  - чтобы при восстановлении на момент в прошлом новые файлы журналов не затирали прежние
  - чтобы была возможность вернуться на прежние линии времени
- Файлы с линями времени может удалять утилита `pg_archivecleanup` запущенная с параметром `-b`

Файлы журнала предварительной записи (WAL)

- Находятся в `PGDATA/pg_wal`
- Не дублируются
- В них пишут серверные и фоновые процессы экземпляра
- могут располагаться вне PGDATA
- Размер текущего файла журнала такой же, как остальных
- `pg_switch_wal()` - переключение на следующий WAL-файл

```
postgres=# select * from pg_ls_waldir() order by 1;
            name              |   size   |      modification
------------------------------+----------+------------------------
 000000010000000000000005D | 16777216 | 2026-05-13 09:36:52+03
 000000010000000000000005E | 16777216 | 2026-05-13 09:41:59+03
 000000010000000000000005F | 16777216 | 2026-05-13 09:34:28+03
postgres=# select pg_current_wal_lsn()\gx
-[ RECORD 1 ]-----+-----------
pg_current_wal_lsn | 0/5E0001C0
```

LSN (Log Sequence Number)

- Журнал можно представить себе как конкатенированный набор файлов журнала (большей части которые уже удалены) начиная с самого первого файла, сформированного при создании кластера
- LSN - 64битное число, монотонно возрастающее, указывающее адрес (смещение от первого байта первого файла журнала) с точностью до байта в журнале кластера
- Процессы экземпляра пишут в журнал записи переменной длинны, LSN используется для указания адреса начала журнальной записи
- физически запись в файлы осуществляется блоками по 8Кб (`wal_block_size`)

Названия журнальных файлов и LSN

- LSN представляют в виде двух 32-битных чисел, записанных в шестнадцатеричной форме (HEX), разделенных слэшем: `XXXXXXXX/YYZZZZZZ`
- `XXXXXXXX` - "старшие" 32 бита LSN
- `YY` - "старшие" 8 бит "младшего" 32-битного числа
- `ZZZZZZ` - смещение в файле 16-мегабайтного журнала относительно его начала
- Размер файла журнала определяет максимальный размер журнального буфера (wal_buffers)
- `00000001XXXXXXXX000000YY` - названия 16-мегабайтных файлов журнала

Функции для работы с журналами

- **`pg_switch_wal()`** переключает запись на новый WAL-файл
- **`pg_create_restore_point('текст')`** создаёт в журнале запись LSN с текстовой меткой, которую можно использовать в параметре `recovery_target_name`
- **`pg_walfile_name('LSN')`** выдает название WAL-файла в котором есть запись LSN
- **`pg_current_wal_flush_lsn()`** LSN конца последней журнальной записи, которая считается надёжно сохранённой
- **`pg_wal_lsn_diff(LSN,LSN)`** байт между двумя LSN
- **`pg_last_wal_replay_lsn()`** какой LSN применён
- **`pg_last_wal_receive_lsn()`** какой LSN принят

Отсутствие потерь (Durability)

- Архитектура резервирования должна гарантировать полное восстановление зафиксированных транзакций
- Архитектура резервирования должна защищать от удаления бэкапов в случае взлома хоста с экземпляром кластера: инициатором бэкапов должен быть хост отличный от того на котором работает экземпляр кластера
- Текущий журнальный файл не должен быть точкой сбоя (single point of failure)
- Использование `pg_receivewal` или физической реплики - возможность гарантировать отсутствие потерь при повреждении текущего журнала

Утилита pg_receivewal

- подсоединяется по протоколу репликации
- принимает журнальные записи без задержек
- рекомендуется использовать слот репликации
- может получать журналы как с мастера, так и с физической реплики
- скрипты для автозапуска (**systemd**) отсутствуют
- может сжимать журнальные записи
- преимущество в том, что является инициатором соединения, что позволяет обеспечить безопасность
- текущему журнальному файлу даёт суффикс **.partial**

Zero data loss (RPO=0)

- `pg_receivewal --synchronous` может подтверждать транзакции
- Для гарантии отсутствия потерь транзакций при полной потере кластера используются параметры:
  - `synchronous_commit = {on, remote_write}`
  - `synchronous_standby_names`

```
pg_receivewal --create-slot --slot=arch
pg_receivewal -D $HOME/archivelog --slot=arch --synchronous
psql -c "alter system set synchronous_standby_names = pg_receivewal"
psql -c "alter system set synchronous_commit = remote_write"
psql -c "select pg_reload_conf()"
```

Запуск pg_receivewal как службу

```
[Unit]
Description=postgres pg_receivewal
After=network.target
[Service]
Type=simple
Environment=PGDATA=/var/lib/postgresql/tantor-se-18/data
WorkingDirectory=/var/lib/postgresql/archivelog
ExecStartPre=/opt/tantor/db/18/bin/pg_receivewal --create-slot --if-not-exists
--slot=arch
ExecStart=/opt/tantor/db/18/bin/pg_receivewal -D /var/lib/postgresql/archivelog
--slot=arch --synchronous -w -v
Restart=on-failure
RestartSec=10s
User=postgres
Group=postgres
UMask=047
StandardOutput=append:/var/lib/postgresql/archivelog/pg_receivewal.log
StandardError=inherit
[Install]
WantedBy=multi-user.target
```

Слот репликации

- три вида: физический, временный физический, логический
- используются для удержания журнальных файлов
- Если клиент не принимает журнальные данные (остановился), то файлы журналов будут удерживаться и заполнят всё свободное место в директории `PGDATA/pg_wal`. Чтобы этого не произошло стоит установить ограничение параметром **`max_slot_wal_keep_size`**
- представление **`pg_replication_slots`** содержит список всех слотов репликации

Утилита резервирования pg_basebackup

- создаёт резервную копию кластера на работающем экземпляре
- создать копию отдельных баз данных или объектов базы нельзя, только всего кластера
- использует безопасный режим резервирования "pull" - приём файлов, а не "push" - передачу файлов
- подсоединяется к экземпляру через TCP или Unix-сокет
- использует протокол репликации
- стоит использовать слоты репликации
- может резервировать реплику, не нагружая мастер
- прогресс в представлении `pg_stat_progress_basebackup`

Создание резервной копии

- **`pg_basebackup`** лучше запустить на хосте, где будет храниться резервная копия (резервный хост)
- параметр **`-D`** задает путь к директории куда бэкапить
  - Директория создаётся на том хосте, где запущена утилита
  - Если директория существует, то она должна быть пустой
- по умолчанию, создаёт два соединения к двум процессам `wal-sender`. По первому соединению передаются файлы с данными, по второму журналы
- Можно ограничить скорость резервирования файлов данных
- Полезный параметр `-P` или `--progress` покажет в какой фазе резервирования находится утилита

Создание резервной копии (продолжение)

- Ждёт завершения контрольной точки перед началом резервирования
  - можно выполнить контрольную точку максимально быстро, указав параметр `-c fast` или `--checkpoint=fast`
  - На реплике утилита не может инициировать контрольную точку и ждёт, пока она выполнится на мастере
- Создаёт файлы **`backup_manifest`** и **`backup_label`**
- Принимает файлы ("pull"), а не передаёт ("push")
  - Приём (pull) - **безопаснее**
  - `pg_receivewal` тоже работает в режиме "pull"

Параметр конфигурации `full_page_writes`

- по умолчанию включён
- не рекомендуется отключать
- защищает от разрыва (torn) блоков данных
- при резервировании реплики должен быть включён на мастере

Инкрементальные бэкапы

- могут быть полезны, если размер кластера большой
- увеличивает сложность процедур резервирования, вероятность ошибки
- Процедура:
  - `alter system summarize_wal=on;` перечитать конфигурацию
  - создать полный бэкап утилитой `pg_basebackup`
  - инкрементальные бэкапы создаются утилитой `pg_basebackup` с парметром `-i манифест` или `--incremental=манифест`
  - для восстановления инкрементальные бэкапы накладываются на полный бэкап утилитой `pg_combinebackup`

Пример инкрементального резервирования

```
pg_basebackup -D $HOME/backup/full1 -P -R -c fast -C --slot=r
39909/39909 kB (100%), 1/1 tablespace
pg_basebackup -D $HOME/backup/incr -P -R -c fast --slot=r -i
$HOME/backup/full1/backup_manifest
 6487/39891 kB (16%), 1/1 tablespace
pg_combinebackup --link $HOME/backup/full1 $HOME/backup/incr -o $HOME/backup/full2
pg_combinebackup: warning: manifest file
"/var/lib/postgresql/backup/incr/backup_manifest" contains no entry for file
"standby.signal"
pg_combinebackup: warning: --link mode was used; any modifications to the output
directory might destructively modify input directories
rm -rf $HOME/backup/full1
rm -rf $HOME/backup/incr
pg_basebackup -D $HOME/backup/incr -P -R -c fast --slot=r -i
$HOME/backup/full2/backup_manifest
 6487/39891 kB (16%), 1/1 tablespace
pg_combinebackup --link $HOME/backup/full2 $HOME/backup/incr -o $HOME/backup/full1
pg_combinebackup: warning: --link mode was used; any modifications to the output
directory might destructively modify input directories
rm -rf $HOME/backup/full2
rm -rf $HOME/backup/incr
```

Утилита pg_verifybackup

- проверяет файлы в бэкапах созданных утилитой `pg_basebackup`
- рассчитывает контрольные суммы файлов (CRC32C) и сравнивает со значениями в файле manifest_file
- сравнивает файлы со списком файлов в файле манифеста
- без файла манифеста не работает
- Проверяет наличие журнальных записей нужных для синхронизации файлов бэкапа - автономность бэкапа
- не проверяет файлы `postgresql.auto.conf`, `standby.signal`, `recovery.signal`

Утилита резервирования WAL-G

- Свободно распространяемая утилита командной строки для резервирования кластеров PostgreSQL
- Поставляется с Tantor Postgres
- Резервирует и восстанавливает с высокой скоростью в несколько потоков
- Использует всокоэффективный алгоритм сжатия бэкапов и архивных журналов
- Поддерживает протокол S3 и файловые системы
- Создаёт инкрементальные бэкапы
- Бэкапит данные и журналы в режимах push и pull

Демонстрация

- Изменение размера WAL файлов

Практика

1. Создание базовой резервной копии кластера
2. Запуск экземпляра на копии кластера
3. Файлы журнала
4. Проверка целостности резервной копии
5. Согласованная резервная копия
6. Удаление файлов журнала
7. Создание архива журнала утилитой pg_receivewal
8. Синхронная фиксация транзакций и pg_receivewal
9. Минимизация потерь данных транзакций

7

7b

Логическое резервирование

Логическое резервирование

- формирование текстового файла или файлов позволяющие воссоздать объекты
- Сохраняет объекты в том состоянии, в котором они были на начало выгрузки
- Данные выгружаются согласованно - на один момент
- Используются утилиты:
  - **`pg_dump`**
  - **`pg_restore`**
  - **`pg_dumpall`**
  - **`psql`**
  - команда **`COPY`**
  - команда утилиты `psql` **`\copy`**

Примеры использования

- Переход на новую основную версию
- Проверка целостности данных
- Получение скрипта воссоздания объектов
- Резервирование кластера и общих объектов кластера
- Выгрузка части кластера: отдельной базы данных, содержимого схемы и других объектов

Сравнение логического и физического резервирования

возможность	Л	Ф
выгружает содержимое отдельной базы данных	+	-
восстановление части объектов	+	-
восстановление на произвольный момент времени	-	+
не зависит от версии, сборки, производителя софта	+	-
обеспечение отказоустойчивости (Durability)	-	+
использует репликационный протокол	-	+
резервирование по сети	+	+
простота использования	+	+

Команда COPY .. TO

- используется для высокоэффективной выгрузки и загрузки данных в одну таблицу
- Можно выгрузить содержимое таблицы или результат любой команды SQL, возвращающей данные
- Данные выгружаются в:
  - файл в файловой системе сервера
  - передаются на стандартный вход (`stdin`) утилите командной строки сервера
  - стандартный вывод `stdout`
- Если команда `COPY` вызывается через сетевое соединение, то стандартный вывод передается через сетевое соединение клиентской программе

Команда COPY .. FROM

- Загрузка выполняется в одну таблицу
- Данные загружаются из:
  - файла на хосте где запущен экземпляр
  - стандартного вывода произвольной исполняемой программы
  - из стандартного ввода `stdin`
- В процессе загрузки можно сразу установить признак заморозки строк
- Можно задать опциональное выражение `WHERE`, в нём нельзя использовать подзапросы
- представление `pg_stat_progress_copy` - отслеживание работы команды и при загрузке и при выгрузке

Команда psql `\copy`

- `\copy` это команда psql
- синтаксис похож на команду `COPY`
- команда использует нижний регистр и чувствительна к нему
- набирается одной строкой
- ";" в конце команды ставить не нужно
- работает с файлами там, где запущена утилита psql, команда COPY работает с файлами через серверный процесс на хосте сервера
- для больших объемов данных COPY эффективнее

Утилита pg_dump

- создаёт логическую резервную копию (дамп) базы данных или её части
- выгружает данные одной базы данных согласованно - на один момент времени
- для выгрузки данных из таблиц по умолчанию использует команду COPY
- выгружает в одном из четырёх форматов: **plain**, **custom**, **directory**, **tar**
- в формате **directory** может выгружать в несколько потоков
- для **custom** и **directory** параметром `-Z` можно выбрать алгоритм и уровень сжатия `zstd, lz4, gzip, none`
- можно не создавать файл, а использовать пайп (pipe):

```
pg_dump параметры | psql параметры
pg_dump -F c параметры | pg_restore параметры
```

Параллельная выгрузка

- возможна только в режиме directory
- количество рабочих процессов задаётся параметром **`-j`** или **`--jobs`**
- прервётся, если после начала выгрузки будут даны команды, устанавливающие эксклюзивную блокировку на выгружаемые объекты
- На время выгрузки устанавливаются блокировки **`ACCESS SHARE`** на все выгружаемые объекты
- При большом количестве объектов можно увеличить значения параметров **`max_locks_per_transaction`**, **`max_connections`**

Утилита pg_restore

- обрабатывает дампы в формате **custom, directory, tar**
- может создавать из архивов текстовый файл формата **plain**
- можно восстановить только определения объектов, не загружая данные указав параметр **--schema-only**
- есть вариант загрузки с помощью файла "оглавления" (TOC, title of contents):
  - сформировать файл оглавления архива параметром **--list**
  - закомментировать (удалить, переместить) строки объектов
  - загрузить архив указав оглавление **--use-list**
  - объекты, не указанные в оглавлении, не будут загружены

Возможности pg_restore

- можно восстановить только определения объектов не загружая данные
- загрузить объекты только части схем
- не восстанавливать права владения
- загружать в табличное пространство по умолчанию для базы данных
- восстановить часть секций секционированных таблиц
- восстановить только указанные таблицы, индексы, представления, последовательности, функции, процедуры, триггеры

Утилита pg_dumpall

- Создаёт единственный скрипт, позволяющий восстановить образ всего кластера
- Выгружает общие объекты кластера
- Последовательно запускает pg_dump в режиме plain для всех баз данных, которые нужно выгрузить
- Скрипт текстовый, содержит команды SQL
- Скрипт можно выполнить в psql
- В скрипте нет команды создания кластера, кластер нужно создать, запустить экземпляр и указать любую базу данных для начального подсоединения psql
- большая часть параметров утилиты относится к pg_dump, который будет запускать утилита
- выгружает в один поток
- можно использовать пайп (pipe):
  pg_dumpall параметры | psql параметры

Возможности pg_dumpall

- имеет много параметров, большая часть из них является параметрами **pg_dump**
- можно выгрузить только общие объекты кластера **--globals-only**
- можно выгрузить только определения ролей **--roles-only**
- можно выгрузить только определения табличных пространств **--tablespaces-only**
- можно не выгружать часть баз данных **--exclude-database=шаблон**
- не добавлять в команды имена табличных пространств **--no-tablespaces**

Строки большого размера

- Типы данных text и bytea могут хранить данные размером до 1Гб
- Строки размером больше 1Гб могут требовать особого обращения
- Поля при выгрузке в текстовом виде могут увеличиваться в размерах

Параметр `enable_large_allocations`

- параметр СУБД Tantor Postgres, который увеличивает размер StringBuffer с 1 гигабайта до 2 гигабайт

```
postgres=# select * from pg_settings where name like '%large%'\gx
name        | enable_large_allocations
setting     | off
category    | Resource Usage / Memory
short_desc | whether to use large memory buffer greater than 1Gb, up to 2Gb
context     | superuser
vartype     | bool
boot_val    | off
```

- может устанавливаться на уровне сессии и утилитами pg_dump, pg_dumpall

```
postgres@tantor:~$ pg_dump --help | grep alloc
  --enable-large-allocations   enable memory allocations with size up to 2Gb
```

- проблема возникает со строкой таблицы config приложений 1С:ERP, Комплексная автоматизация, Управление производственным предприятием

Демонстрация

- Обработка строк большого размера

Практика

1. Использование утилиты pg_dump
2. Формат custom и утилита pg_restore
3. Формат directory
4. Сжатие и скорость резервирования
5. Команда COPY

8a

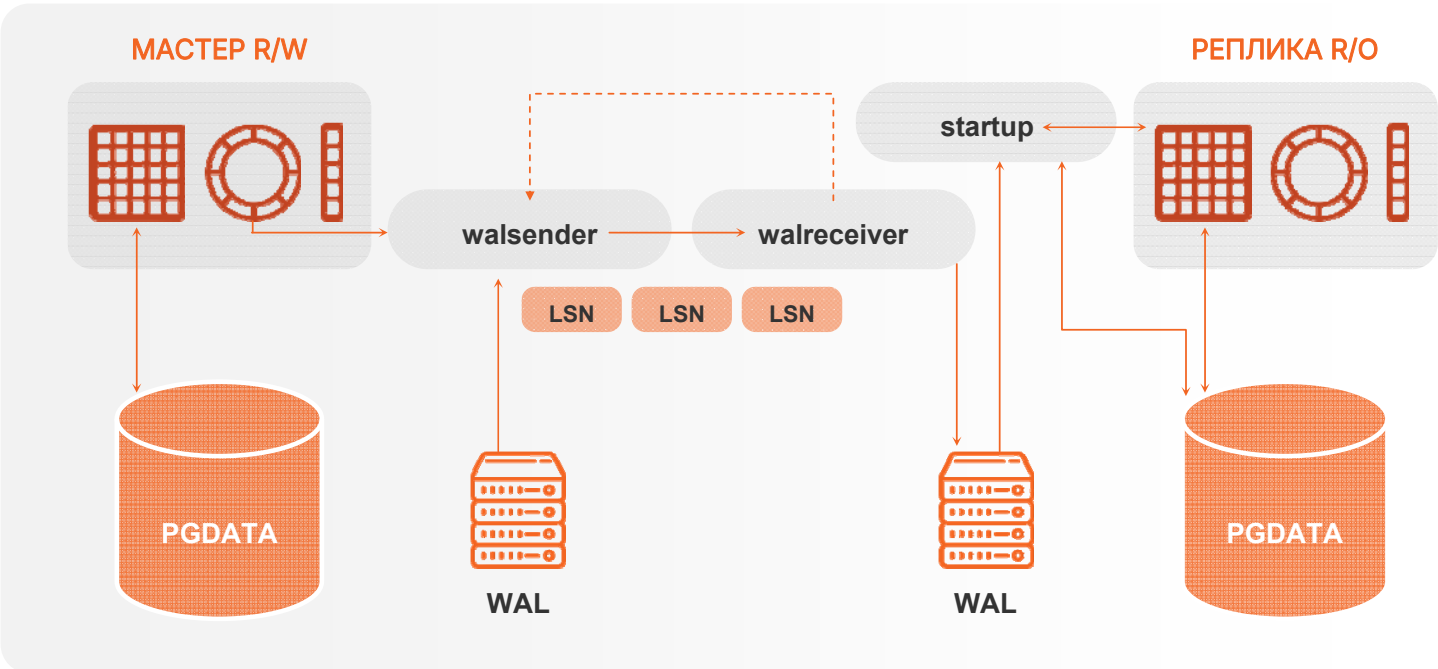
Физическая репликация

Физическая репликация

- Один основной (primary, master, ведущий) кластер баз данных - допускает внесение изменений в данные
- Один или несколько резервных (standby, ведомых) кластеров
- Резервные кластера (физические реплики или просто реплики) получают журнальные данные и применяют к своим файлам
- Реплики могут обслуживать запросы на чтение - режим горячего резерва (hot standby)
- Реплики являются физической резервной копией основного кластера, которая обновляется

Мастер и реплика

- мастер и реплики должны использовать ту же основную версию PostgreSQL
- кластер реплицируется целиком
- директории табличных пространств могут отличаться



Реплики и архив журнала

- Обычно используется поточная репликация с использованием слота репликации: процесс walreceiver подсоединяется к процессу walsender
- Может использоваться архив журналов, заполняемый командой в параметре `archive_command` или формируемый утилитой `pg_receivewal`
- Если реплика не сможет получить журнал через walreceiver, то использует команду `restore_command`

Настройка мастера

- Настроить возможность подсоединения реплик по протоколу репликации
- Проверить значения параметров конфигурации:
  - **`wal_level`** по умолчанию replica
  - **`max_walsenders`** по умолчанию 10
  - **`max_replication_slots`** по умолчанию 10
  - **`max_slot_wal_keep_size`** по умолчанию -1 без ограничений, стоит установить ограничение

Создание реплики

- создание резервной копии утилитой pg_basebackup
- параметр -R устанавливает конфигурационные параметры для реплики
- параметр **--create-slot** (**-C**) и **--slot=имя** (**-S**) создаёт слот для удержания WAL-журналов до запуска экземпляра реплики
- установить параметр конфигурации cluster_name
- проверить значения параметров конфигурации приведя их в соответствие с возможностями хоста реплики
- Настроить службу для автоматического запуска экземпляра реплики и запустить экземпляр реплики

Слоты репликации

- Процесс walreceiver реплики использует физический слот
- список слотов - в представлении `pg_replication_slots`
- Функция создания `pg_create_physical_replication_slot('имя')`
- Удаление слота любого типа `pg_drop_replication_slot('имя')`
- Создание копии слота `pg_copy_physical_replication_slot('имя', 'имя_создаваемого')`

Параметры конфигурации на репликах

- на мастере не играют роли, но могут быть заранее установлены
- многие не меняются без рестарта экземпляра реплики
- реплика обслуживает запросы (**`hot_standby=on`**)
- реплика применяет журнальные записи без задержки (**`recovery_min_apply_delay=0`**)
- `walreceiver_status_interval` по умолчанию 10 секунд. Обратная связь будут отправляться не чаще и горизонт событий баз данных на мастере будет сдвигаться не чаще, чем это значение
- `wal_retrieve_retry_interval` по умолчанию - 5 секунд. Время ожидания репликой поступления журнальных данных из любых источников

Конфликты на реплике

- обратная связь отключена (**`hot_standby_feedback=off`**)
- `walreceiver_timeout` по умолчанию 60 секунд
  - `walreceiver` реплики может обнаружить отсутствие ответа от `walsender`, и заново переподсоединиться
- `max_standby_streaming_delay` и `max_standby_archive_delay` по умолчанию равны 30 секунд. Максимально допустимое время задержки применения WAL

Горячая реплика

- по умолчанию включена (параметр **`hot_standby=on`**)
- реплика принимает соединения и выполняет запросы
- утилита **pg_basebackup** может выполнять резервирование реплики вместо мастера (backup offloading)
- временные таблицы на реплике нельзя использовать, расширение pg_variables можно

Обратная связь с мастером

- обратная связь по умолчанию отключена
  `hot_standby_feedback=off`
- обратная связь удерживает горизонт очистки всего мастера в длительность самого долгого запроса на базах реплики или транзакции (уровень транзакций REPEATABLE READ)
- Вместо обратной связи можно использовать параметры
  - `max_standby_streaming_delay` и
  - `max_standby_archive_delay`

Мониторинг горизонта

- Оценка горизонта текущей базы данных:
  - **backend_xmin** из **pg_stat_activity**
  - Длительность самого долгого запроса или транзакции: **max(now()-xact_start)** из **pg_stat_activity**
  - Столбец **xmin** представления **pg_replication_slots** при использовании слотов репликации
  - Столбец **backend_xmin** представления **pg_stat_replication** на мастере - что получили walsender по обратной связи

```
select age(backend_xmin), extract(epoch from (clock_timestamp()-xact_start)) secs, pid, datname database,
state from pg_stat_activity where backend_xmin IS NOT NULL OR backend_xid IS NOT NULL order by
greatest(age(backend_xmin), age(backend_xid)) desc;
  age   |    secs     |  pid   | database |        state
--------+-------------+--------+----------+---------------------
 175455 | 1425.651346 | 255554 | postgres | idle in transaction
      1 |    0.001878 | 255547 | postgres | active
      1 |    0.001213 | 255626 | postgres | active
```

Мониторинг горизонта

- горизонт баз данных кластера в количестве номеров транзакций, отстоящих от текущей:

```
select datname, greatest(max(age(backend_xmin)), max(age(backend_xid))) from pg_stat_activity
where backend_xmin is not null or backend_xid is not null group by datname order by datname;
```

- длительность самого долгого запроса или транзакции, который и удерживает горизонт:

```
select datname, extract(epoch from max(clock_timestamp()-xact_start)) from pg_stat_activity
where backend_xmin is not null or backend_xid is not null group by datname order by datname;
```

- удержание горизонта (удерживают на всех базах) физическими слотами репликации, если включена обратная связь (hot_standby_feedback=on

```
select max(age(xmin)) from pg_replication_slots;
select backend_xmin, application_name from pg_stat_replication order by  age(backend_xmin) desc;
```

- в самих репликах искать процессы, выполняющие команды, удерживающие горизонт можно так же, как и на мастере - запросом к `pg_stat_activity`

```
select age(backend_xmin), extract(epoch from (clock_timestamp()-xact_start)) secs, pid, datname database,
state from pg_stat_activity where backend_xmin IS NOT NULL OR backend_xid IS NOT NULL order by
greatest(age(backend_xmin), age(backend_xid)) desc;
```

Параметры `max_slot_wal_keep_size` и `transaction_timeout`

- `max_slot_wal_keep_size` по умолчанию -1 (без ограничений) максимальный размер журнальных файлов, который может оставаться в каталоге pg_wal после выполнения контрольной точки для слотов репликации
- `transaction_timeout` по умолчанию ноль (таймаут отключен). Позволяет отменить любую транзакцию или одиночную команду, длительность которой превышает указанный период времени, а не только простаивающую. Защищает от удержания горизонта базы данных и раздувания файлов (bloat)

Параметры мастера, которые должны быть синхронизированы с репликами

- Если менять значения этих параметров на мастере, то на репликах значения пяти параметров должны быть не меньше
- Изменения в этих параметрах записываются в журнал
- если реплика обнаружит, что значение на мастере стало больше, то наложение журнальных записей приостановится или экземпляр реплики остановится
- Пять параметров:
- `max_connections`, `max_prepared_transactions`, `max_locks_per_transaction` ограничивают максимальное количество блокировок на уровне объектов
- `max_walsenders`
- `max_worker_processes`

Смена ролей мастер-реплика

- запланированная когда мастер работоспособен называют switchover
- если мастер неработоспособен называют failover
- убедиться что мастер остановлен перед тем как дать сигнал одной из реплик стать мастером
- исключить или минимизировать потерю транзакций
- **`pg_last_wal_replay_lsn()`** - журнальная запись, которая была восстановлена последней
- **`pg_last_wal_receive_lsn()`** - последний LSN, полученный репликой

Повышение реплики до мастера

- запустить утилиту **`pg_ctl promote`**
- вызвать функцию **`pg_promote()`**
- повышение до мастера увеличивает линию времени на единицу
- при переходе на новую линию времени в директории кластеров `PGDATA/pg_wal` создаётся текстовый файл `0000000N.history`

Файлы истории линий времени

- Находятся в директории `PGDATA/pg_wal`
- Имеют название `0000000N.history`
- Удалять эти файлы не стоит
- при повышении реплики до мастера создаётся новая линия времени и новый файл истории линий времени
- используются утилитами и процессами в целях резервирования и восстановления
- запрашиваются и передаются по протоколу репликации наравне с файлами журналов
- резервируются наравне с файлами журналов

Утилита pg_rewind

- синхронизирует директорию `PGDATA` и табличных пространств с исходным кластером, в том числе заменяет файлы параметров
- Утилита ищет файл `0000000N.history` обоих кластеров с целью найти точку, в которой линии времени двух кластеров разошлись
- Читает журнальные файлы в `PGDATA/pg_wal`, начиная с последней контрольной точки перед моментом, когда линии времени разошлась и до текущего файла журнала того кластера, чью директорию утилита будет синхронизировать
- По журнальным записям определяет все блоки, в которые были внесены изменения
- Копирует эти блоки с другого кластера

Утилита pg_rewind

- используется для того, чтобы вернуть в работу бывший мастер после незапланированной смены ролей
- смена ролей должна быть выполнена с переходом на новую линию времени
- требует включенных `full_page_writes`, подсчета контрольных сумм (или `wal_log_hints`)
- создаёт файл `backup_label` в котором указываются параметры для восстановления согласованности
- копирует все файлы параметров, которые находятся в `PGDATA` исходного кластера. Если содержимое файлов параметров синхронизируемого кластера важно, то стоит сохранить эти файлы до запуска утилиты

Процессы экземпляра реплики

- **postgres** основной процесс, запускает другие процессы, принимает соединения
- **checkpointer** выполняет точки рестарта
- **background writer** записывает грязные страницы на диск
- startup накатывает журнальные записи из файлов в `PGDATA/pg_wal`
- **walreceiver** принимает поток журнальных записей и записывает их в файлы в `PGDATA/pg_wal`

Отложенная репликация

- Можно установить задержку для реплики в применении журнальных записей
- по умолчанию задержки нет
- устанавливается параметром `recovery_min_apply_delay`
- Приём журнальных записей репликой (процесс `walreceiver`) выполняется без задержки, задерживается их накат
- параметр устанавливается на уровне экземпляра реплики, при изменении достаточно перечитать файлы параметров

Восстановление повреждённых блоков данных с реплики

- Расширение **page_repair** содержит функцию **pg_repair_page** для восстановления повреждённых блоков
- за один вызов процедуры восстанавливается один блок
- можно восстановить блоки слоёв **main, vm, fsm**
- в процессе восстановления на отношение устанавливается монопольная блокировка
- блок запрашивается с реплики через соединение
- параметры соединения передаются функции

Демонстрация

- Создание реплики и запуск её экземпляра

Практика

1. Создание реплики
2. Слоты репликации
3. Изменение имени кластера
4. Создание второй реплики
5. Выбор реплики на роль мастера
6. Подготовка к переключению на реплику
7. Переключение на реплику
8. Включение обратной связи
9. Утилита pg_rewind

8b

Логическая репликация

Логическая репликация

- поддерживает репликацию изменений в строках обычных и секционированных таблиц
- Логическая и физическая репликация могут работать одновременно
- Создаётся два типа объектов: публикации и подписки
- Публикация включает в себя набор таблиц одной базы данных
- Набор таблиц можно менять без остановки репликации
- Реплицируются изменения, а не команды
- Одна и та же таблица быть источником и приемником изменений

Применение логической репликации

Используется для:

- переноса данных в архивные или аналитические базы данных
- консолидации данных: переноса данных из региональных баз данных в центральную
- организации резервной базы данных на случай потери основной
- снижения нагрузки на базу данных за счёт переноса части запросов на резервные базы данных

Физическая и логическая репликация

Возможности логической репликации:

- можно реплицировать наборы таблиц, а не весь кластер целиком
- нечувствительность к основным версиям, платформам, сборкам
- передаются журнальные записи не всего кластера, а только по реплицируемым таблицам
- есть двунаправленная репликация
- журнальные данные **могут** передаваться с физической реплики

Идентификация строк

- для репликации только вставок строк (INSERT) и TRUNCATE идентификаторы строк не нужны
- для обновления и удаления строк в таблице должен быть:
- первичный ключ
- либо уникальный индекс и ограничения целостности NOT NULL на каждом из столбцов уникального индекса
- либо использовать все столбцы для идентификации строки

Способы идентификации строк

- Если в таблице нет первичного ключа, нужно установить способ идентификации строк командой
  `ALTER TABLE имя REPLICA IDENTITY`
  - `DEFAULT;` использовать первичный ключ
  - `USING INDEX имя;` уникального, не частичного, не отложенного индекса, на всех индексируемых столбцах установить NOT NULL
  - `FULL;` передавать значения всех столбцов
  - `NOTHING;` установлено для таблиц системного каталога, для пользовательских таблиц не имеет смысла

Действия для создания логической репликации

- Установить значение параметра `wal_level=logical` на мастере и физической реплике к которой будет подсоединяться подписка (изменение параметра требует рестарт экземпляра)
- выбрать таблицы для репликации
- проверить наличие первичных ключей или выбрать способ идентификации строк
- создать таблицы в базах данных-приёмниках
- создать публикации на базе данных-источнике
- создать подписки на базах данных-приёмниках и выбрать параметры подписки

Создание публикации

- выбрать таблицы, которые будут включены в одну публикацию
- Можно указать какие команды будут реплицироваться: вставки, изменение, удаление строк, усечение таблиц
- В публикацию можно включить:
  - все таблицы базы данных
  - все таблицы в схемах
  - список таблиц
- При задании списка таблиц можно указать набор столбцов и фильтр для строк

Создание подписки

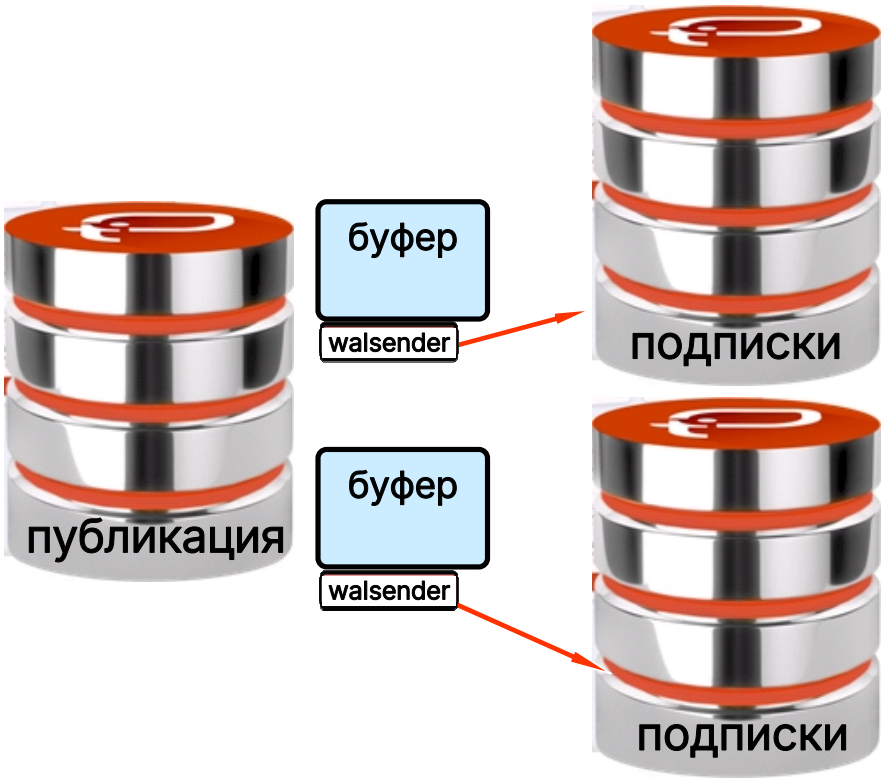
- Подписка создаётся в базе данных где находятся таблицы-приёмники изменений
- Каждая подписка должна использовать свой слот логической репликации в базе данных-источнике
- Таблицы источника и приёмника и их столбцы сопоставляются по именам и должны быть одинаковыми
- В таблице-приёмнике могут быть дополнительные столбцы. Они могут заполняться значениями по умолчанию или триггерами

Свойства подписки

- При создании подписки указывается строка соединения с базой публикации
- В одной подписке можно указать несколько публикаций если они в одной базе
- При создании подписки строки таблиц публикации можно скопировать в таблицы-приёмники
- Логическая репликация работает только через слот
- Слот может быть создан заранее и указан при создании подписки
- Слот может быть создан при создании подписки
- По умолчанию имя слота такое же как имя подписки

Нагрузка на экземпляр

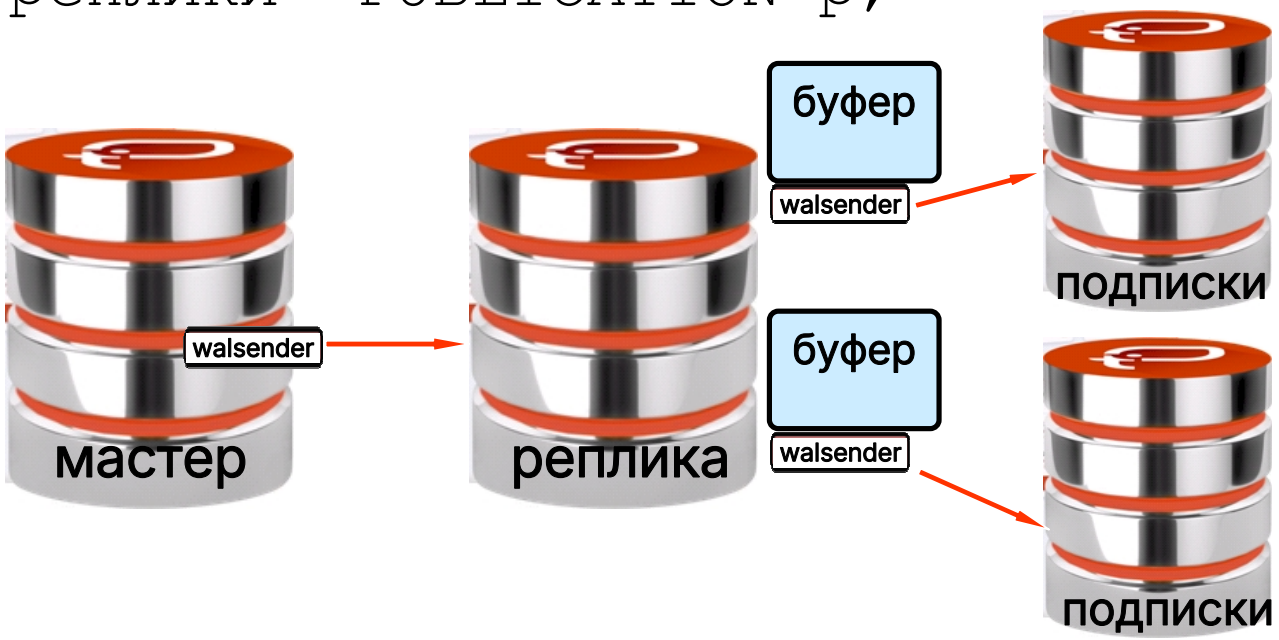
- Для каждой подписки запускается процесс **walsender**, выполняющий буферизацию и логическое декодирование



Получение журнальных данных с реплики

- При создании подписки достаточно указать адрес физической реплики:

```
CREATE SUBSCRIPTION s CONNECTION 'dbname=имя
host=реплики port=реплики' PUBLICATION p;
```



Конфликты

- Конфликт - если процесс **logical replication worker** не может внести изменения из-за нарушения ограничений целостности или по другой причине
- репликация во всей подписке приостанавливается
- Автоматического разрешения конфликтов нет
- Устранить конфликт можно вручную изменив данные или пропустить (не применять) транзакцию, в которой выполняется команда, приведшая к ошибке
- Информация об ошибке попадает в логе кластера

Двунаправленная репликация

- Если менять значения этих параметров на мастере, то на репликах эти значения должны соответствовать значениям на мастере
- Изменения в этих параметрах записываются в журнал
- если реплика обнаружит, что значение на мастере стало больше, то наложение журнальных записей приостановится или экземпляр реплики остановится
- Список параметров:
- max_connections, max_prepared_transactions, max_locks_per_transaction ограничивают максимальное количество блокировок на уровне объектов
- max_walsenders
- max_worker_processes

Демонстрация

- Однонаправленная репликация
- Двунаправленная репликация

Практика

1. Репликация таблицы
2. Репликация без первичного ключа
3. Добавление таблицы в публикацию
4. Двунаправленная репликация

9

Обзор Платформы Tantor

Инструменты мониторинга

Универсальные системы мониторинга, неадаптированные для PostgreSQL

Grafana

okmeter

ZABBIX

new relic

MUNIN

cacti

DATADOG

Платформа "Tantor"

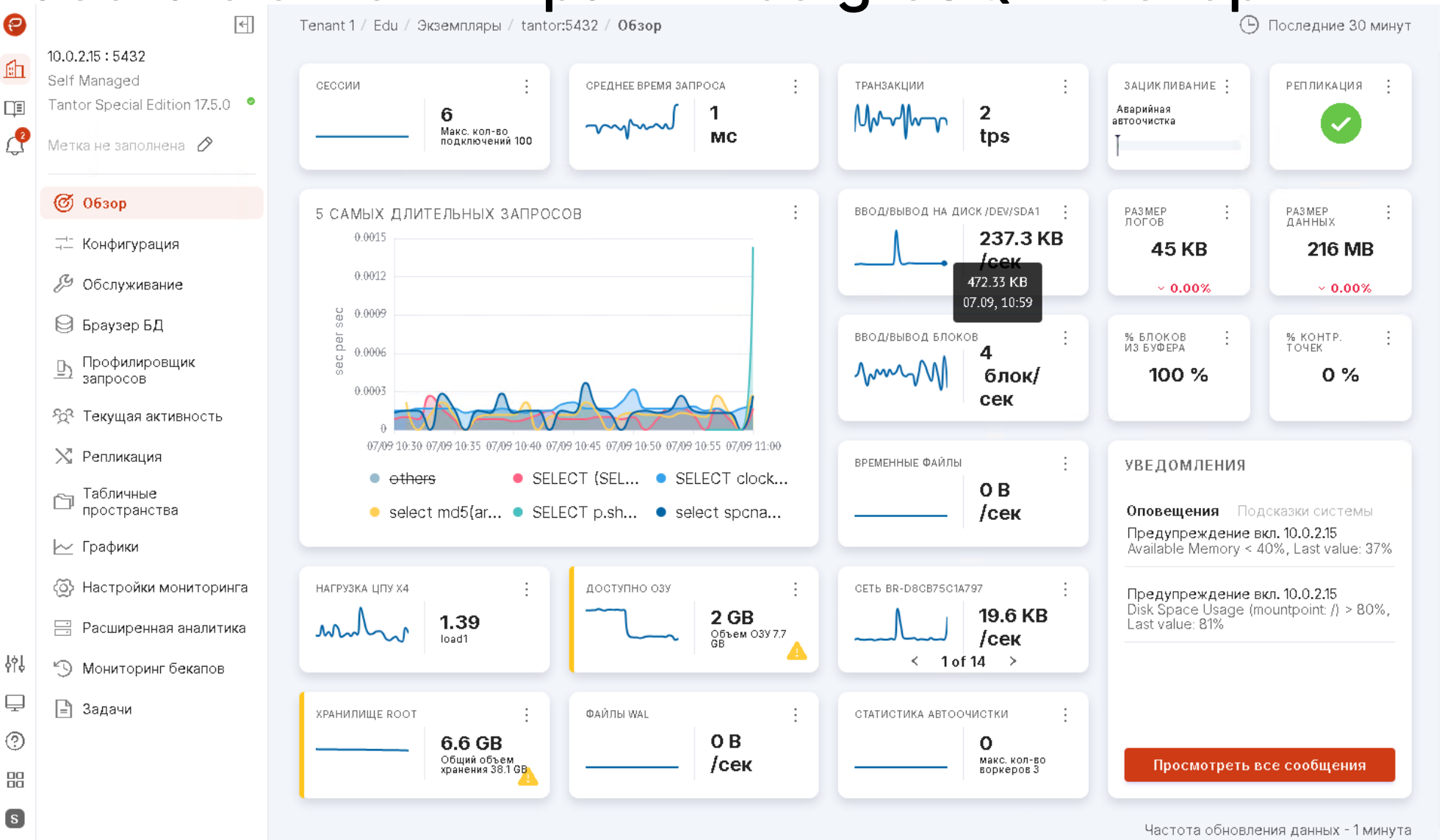
Платформа Tantor

- Функциональное ПО с графическим пользовательским интерфейсом, созданное для удобного администрирования экземпляров PostgreSQL и кластеров Patroni
- Необходима организациям, которые используют множество БД, каждая из которых обслуживает определенную информационную систему или сервис
- Российские компании переходят на российские аналоги

Возможности Платформы Tantor

- Dashboard (панель): графики для наиболее важных метрик
- Конфигуратор: рекомендации по установке значений параметров PostgreSQL
- Анализатор планов запросов Tensor: встроенный анализатор планов запросов с рекомендациями исправления запросов и визуализацией планов
- Аналитика: анализ диагностических журналов PostgreSQL
- Управление кластерами Patroni: просмотр и изменение параметров, мониторинг репликации и событий failover
- Интеграция с системами
  - уведомлений
  - мониторинга: по REST API (Swagger/OpenAPI)
  - службами каталогов по протоколу LDAP
  - системами резервирования: RuBackup и Backman

Работа с экземплярами PostgreSQL: Обзор



Список кластеров Patroni и их экземпляров

Tenant 1 / Edu / **Кластеры**

КЛАСТЕРЫ ЭКЗЕМПЛЯРЫ АГЕНТЫ

Поиск по имени

Добавить кластер Patroni

ИМЯ КЛАСТЕРА	СОСТОЯНИЕ	PATRONI	КОЛ-ВО ЦПУ	ОБЪЕМ ОЗУ	ДИСК ОСНОВНОГО ЭКЗЕМПЛЯРА	ЭКЗЕМПЛЯРЫ
patroni	partial	4.0.4	12	23.3 GB	10.2 GB / 38.1 GB	3 / 3

Tenant 1 / Edu / Кластеры / patroni / **Обзор и управление**

ОБЗОР И УПРАВЛЕНИЕ МОНИТОРИНГ

Отправлена команда возобновить на кластер

Отправлена команда пауза на кластер

ВЕРСИЯ PATRONI	ВЕРСИЯ POSTGRES	ОБЪЕМ ОЗУ	КОЛ-ВО ЦПУ	ДИСК ОСНОВНОГО ЭКЗЕМПЛЯРА	ЭКЗЕМПЛЯРЫ	РЕПЛИКИ
4.0.4	17.3	23.3 GB	12	38.1 GB	3 / 3	2 / 2

Возобновить

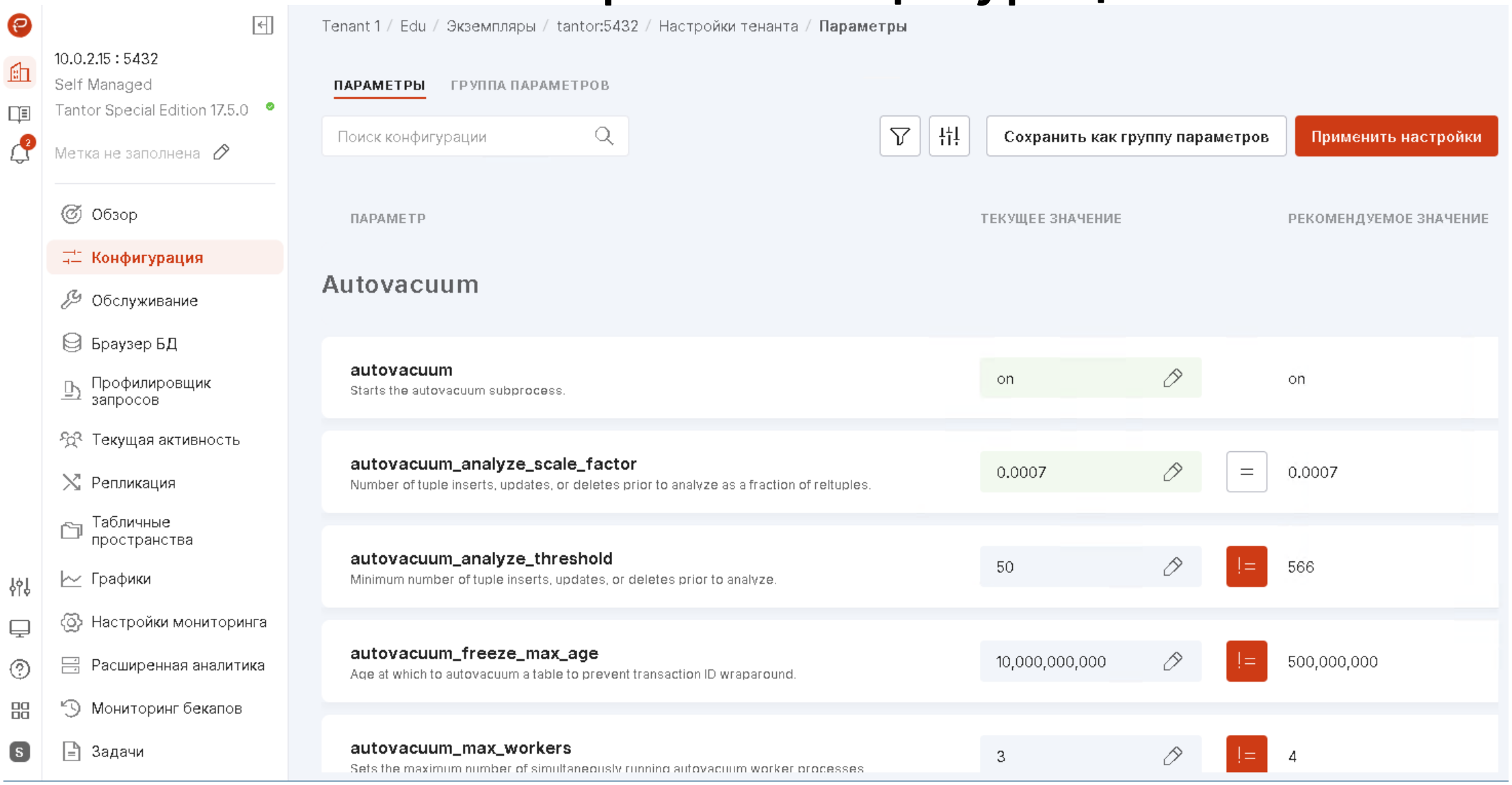
Конфигурация кластера Пауза/Обслуживание

Экземпляры кластера

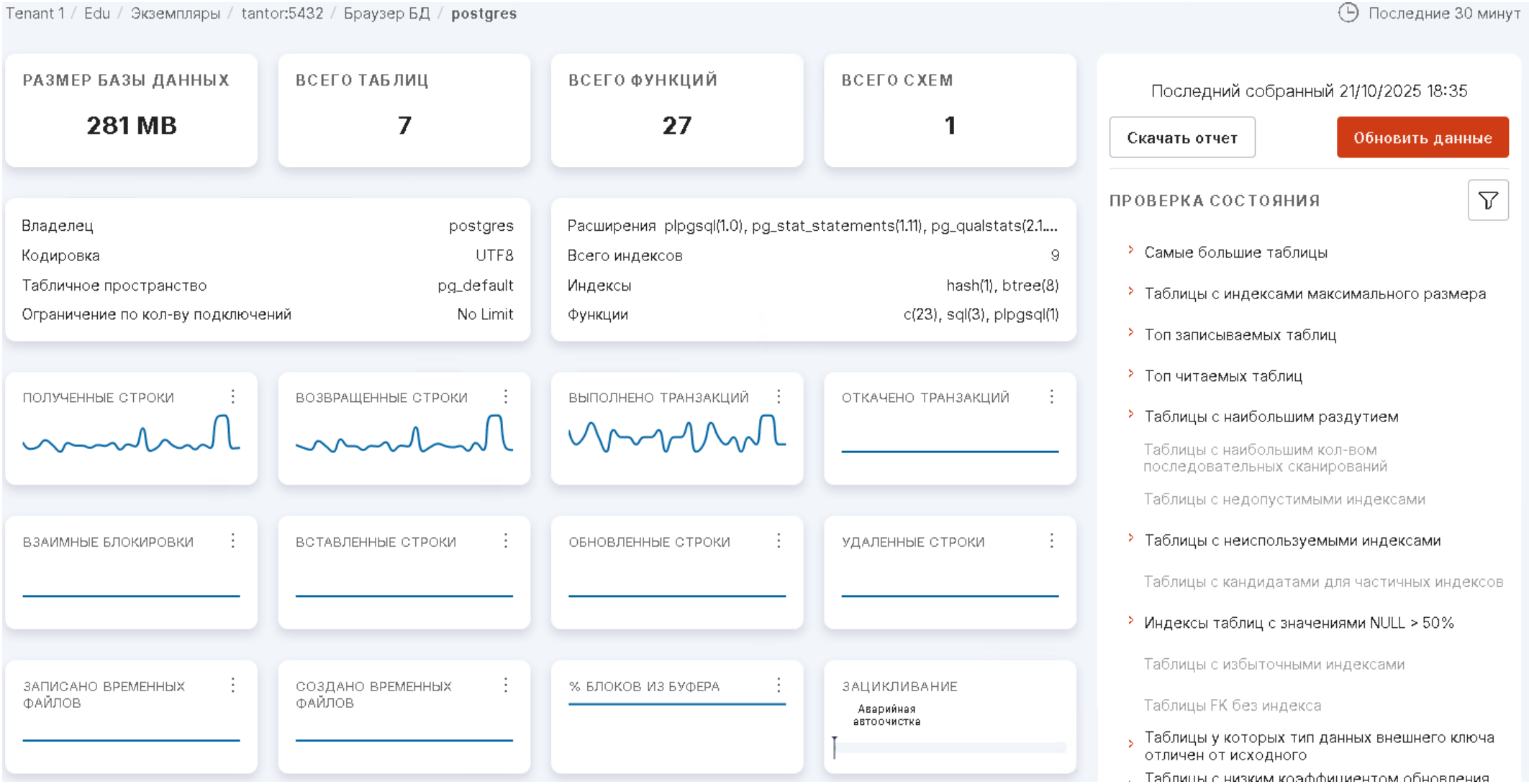
Частота обновления - 1 минута

ИД	СОСТОЯНИЕ	РОЛЬ	ОС	ТИП	ВЕРСИЯ	ЦПУ	МЕТКА	КОНЕЧНАЯ ТОЧКА	ПОРТ	TTL	ЗАДЕРЖКА
43	streaming	replica	Debian GNU/Linux	Postgres	17.3	4		172.21.1.3	5432	30	0 B
44	running	primary	Debian GNU/Linux	Postgres	17.3	4		172.21.1.1	5432	30	0 B

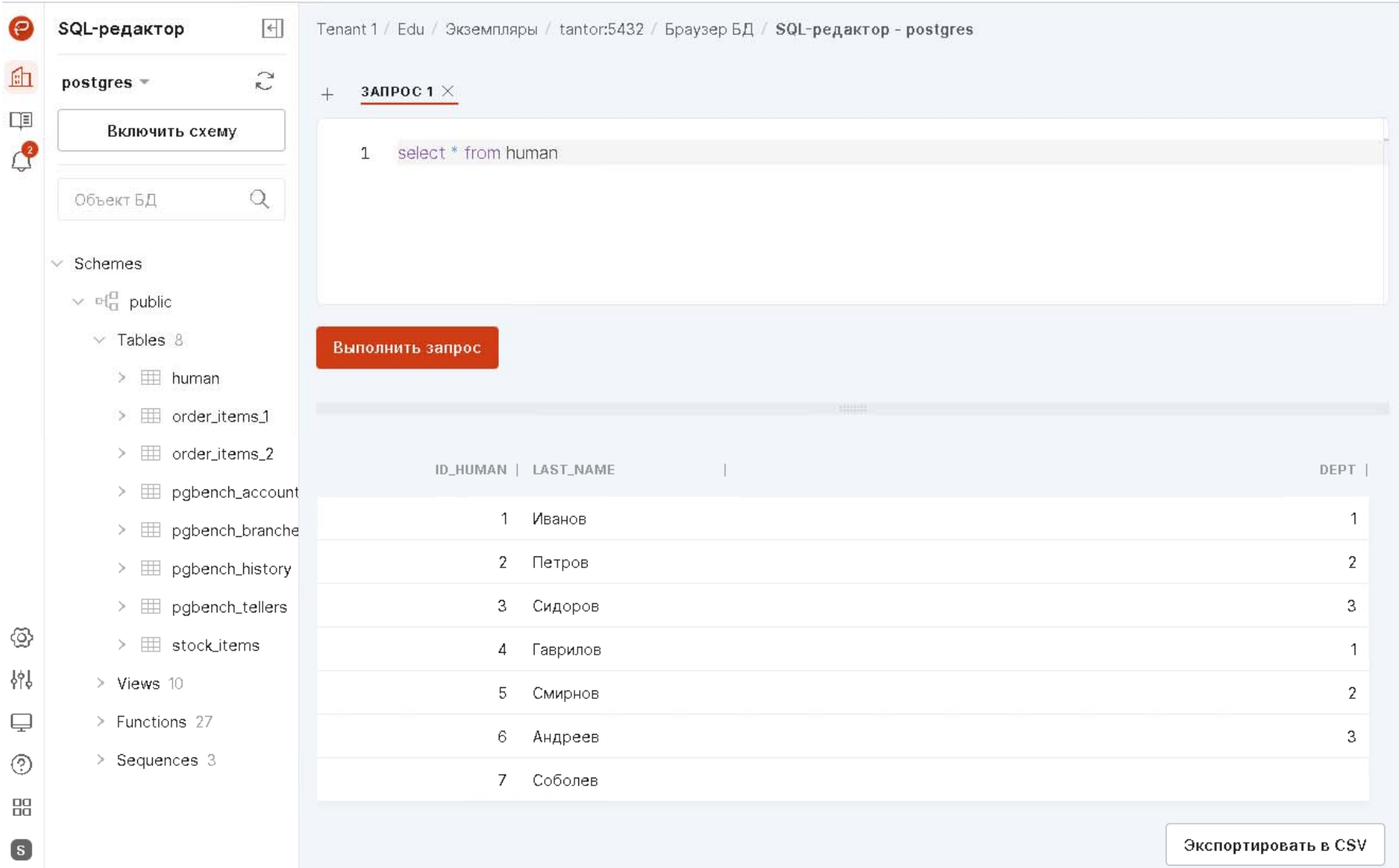
Работа с экземплярами: Конфигурация



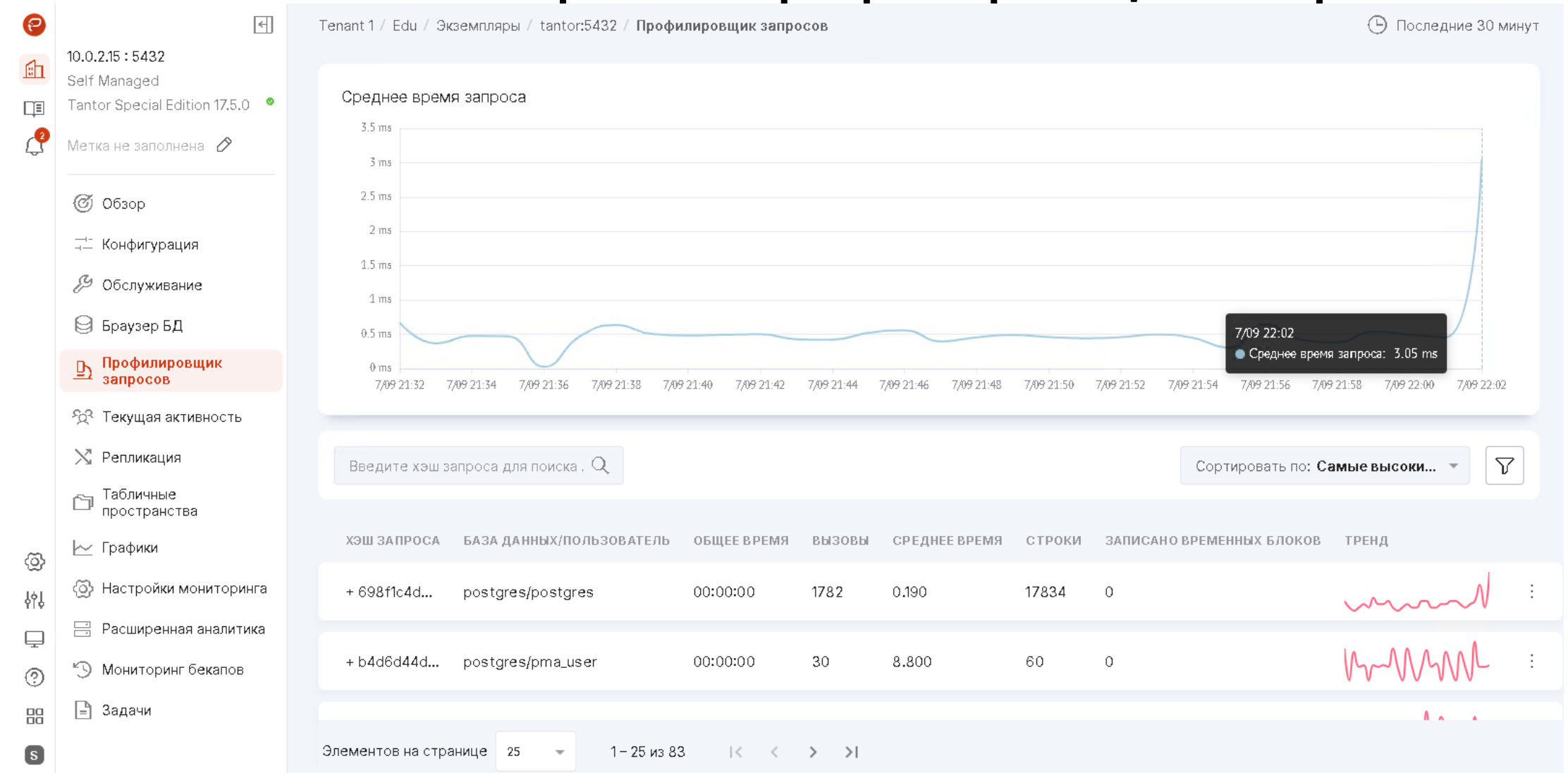
Браузер БД -> Аудит



Браузер БД -> SQL-редактор и Схема

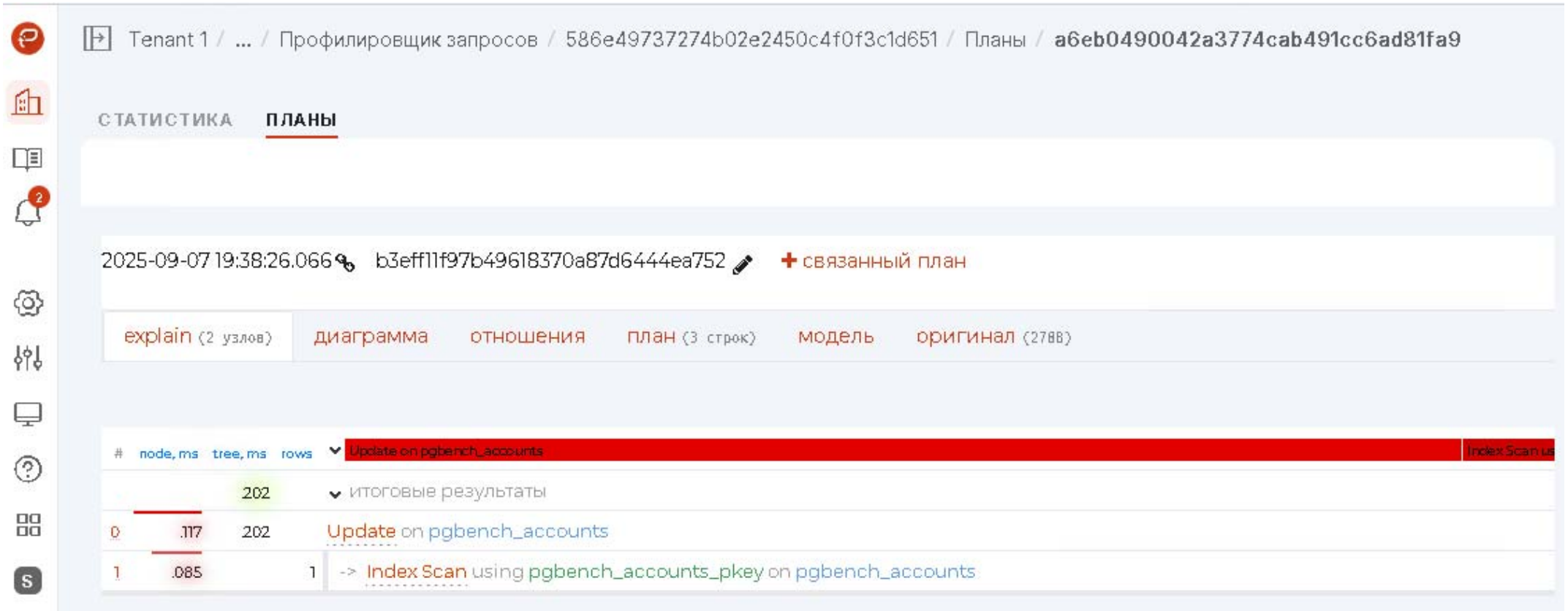


Работа с экземплярами: Профилировщик запросов

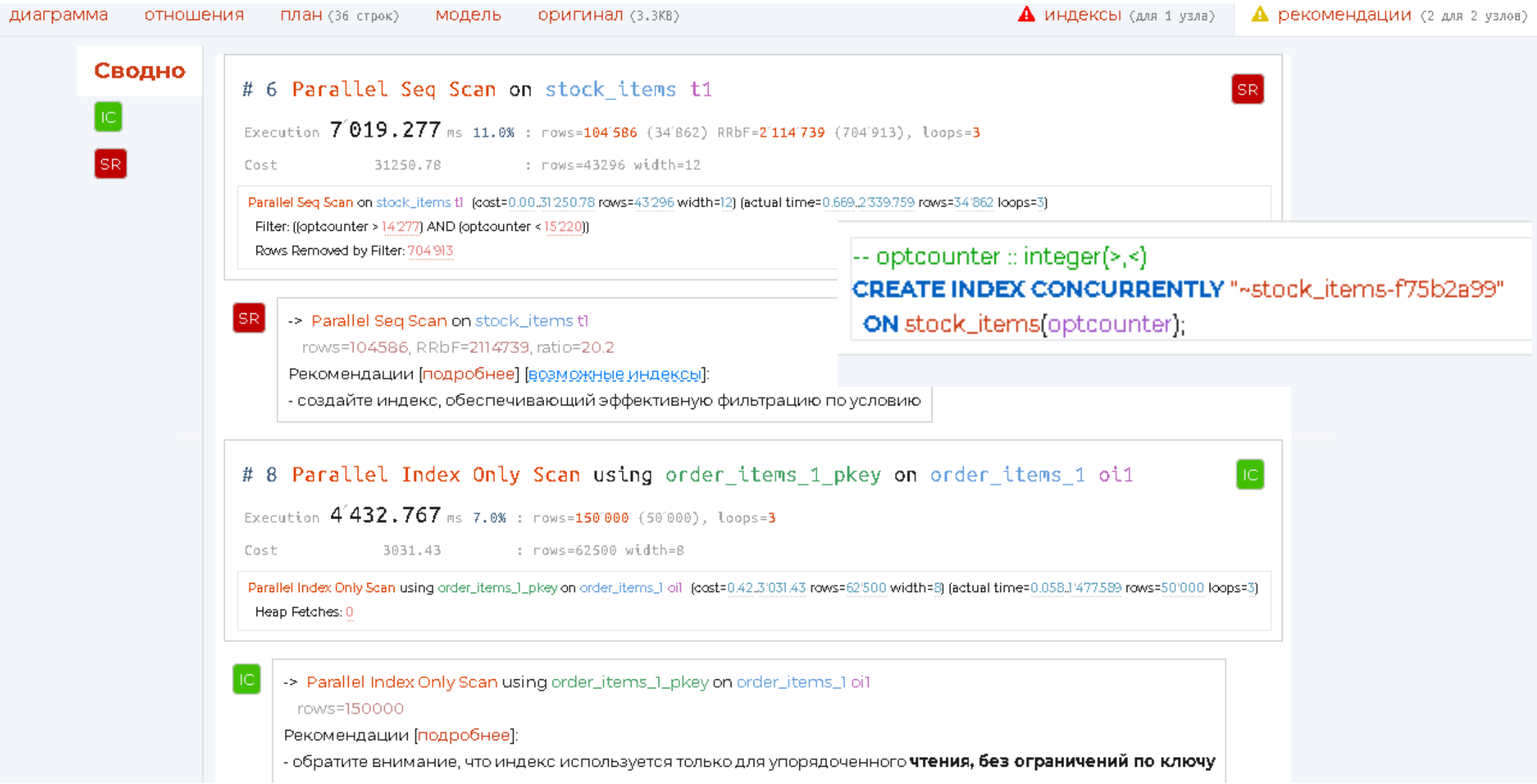


Профилировщик запросов -> Планы

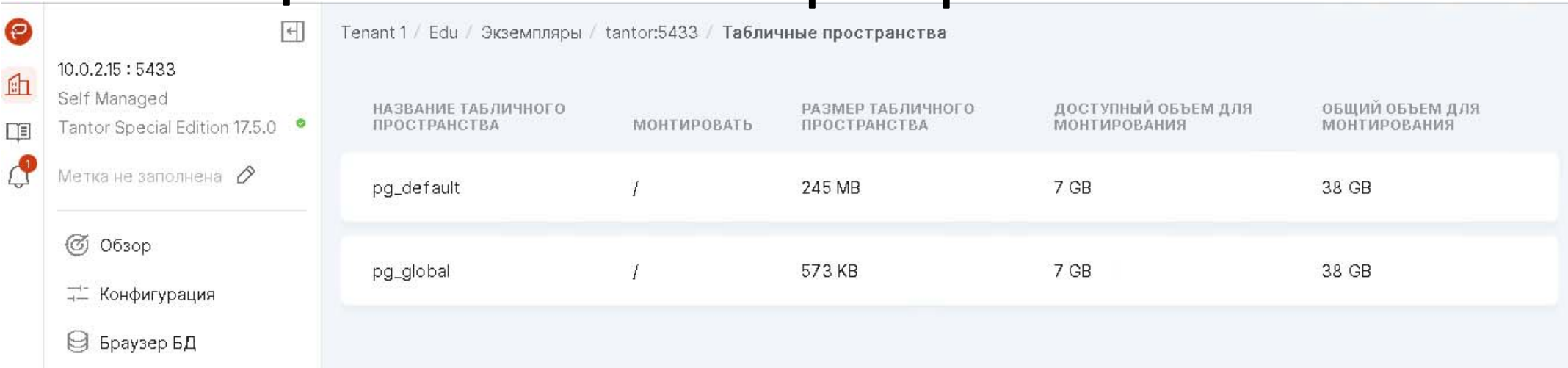
- В Платформу Tantor встроен полнофункциональный анализатор запросов Tensor
- Запросы не передаются на внешние сервера

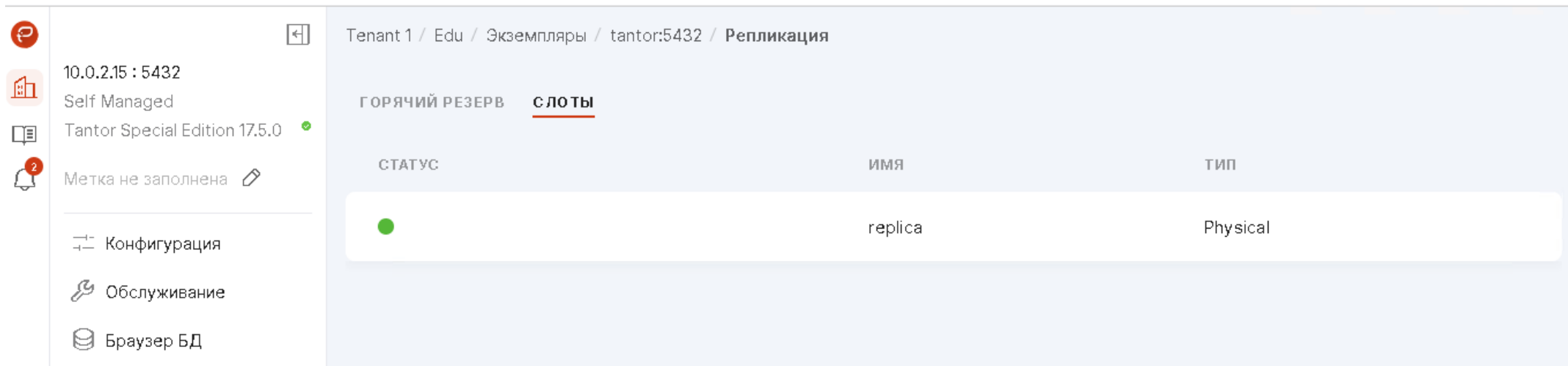


Экземпляр: Профилировщик запросов: рекомендации

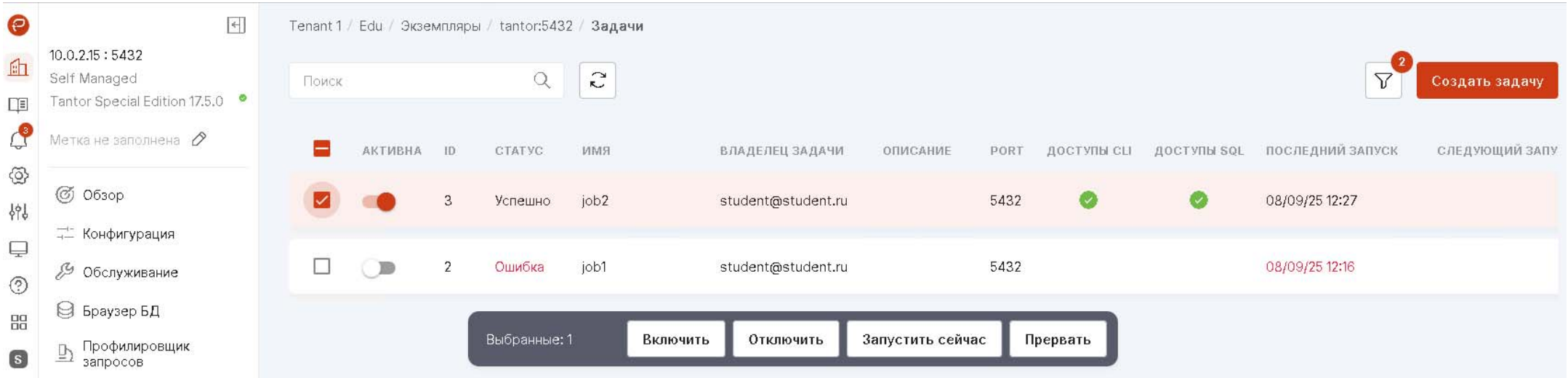


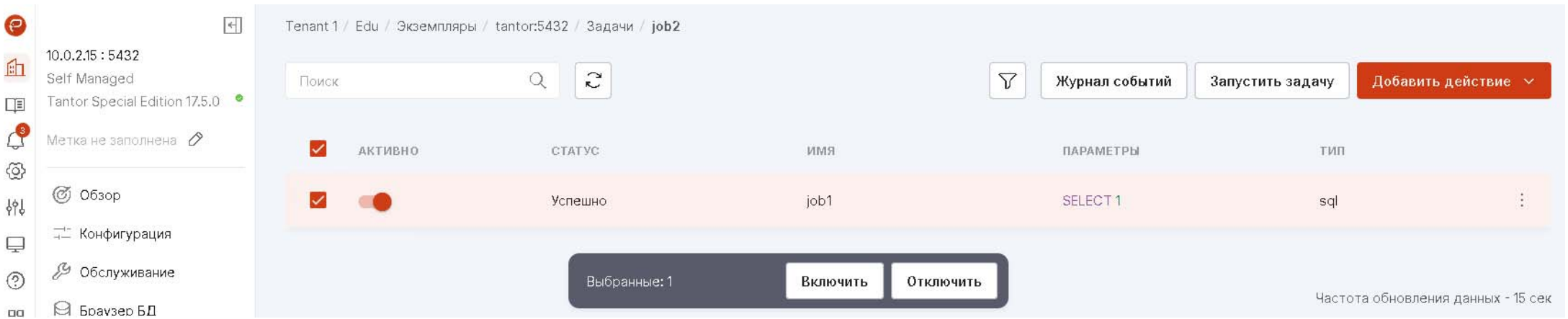
Репликация и Табличные пространства



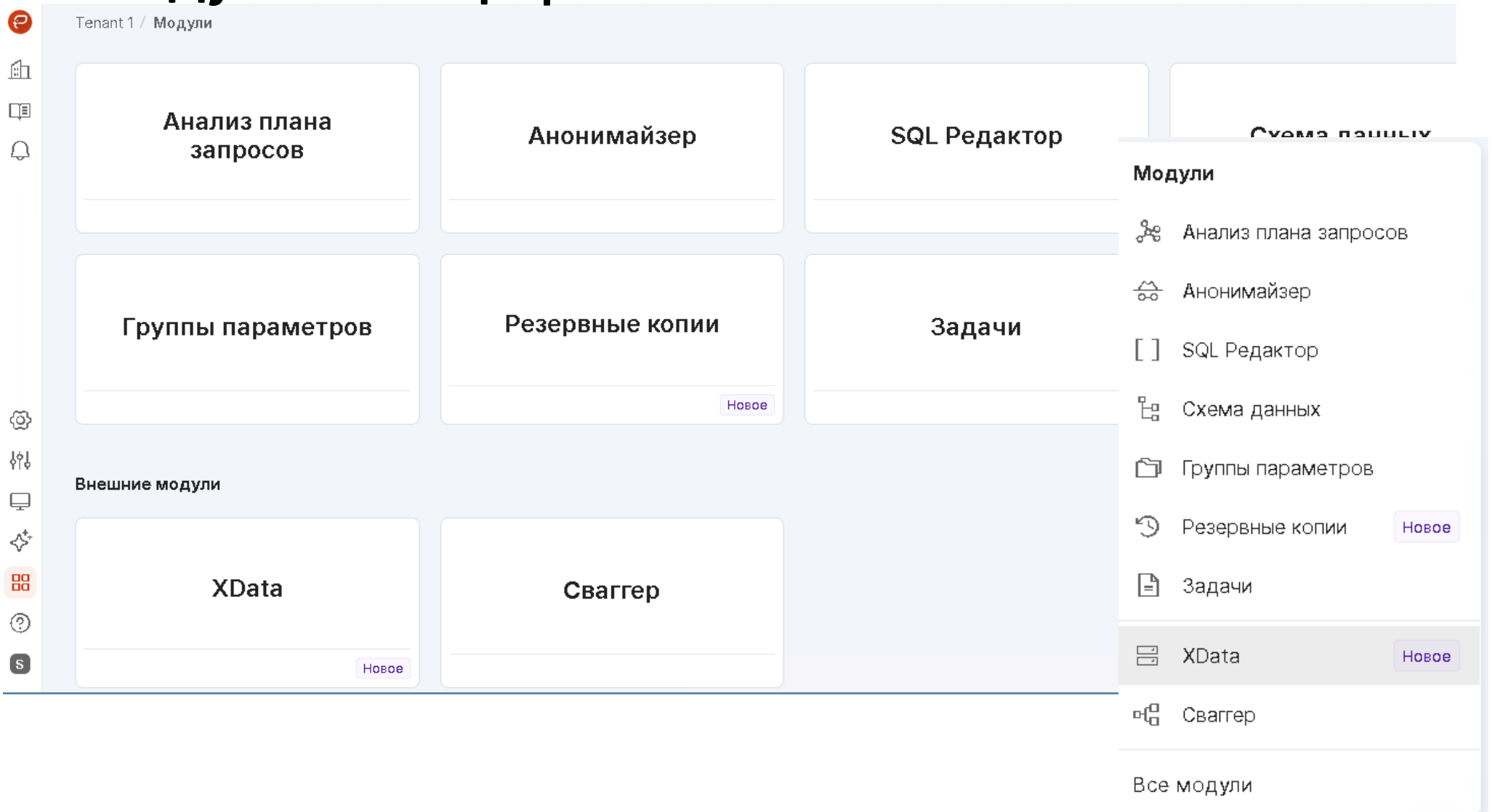


Работа с экземплярами: Задачи

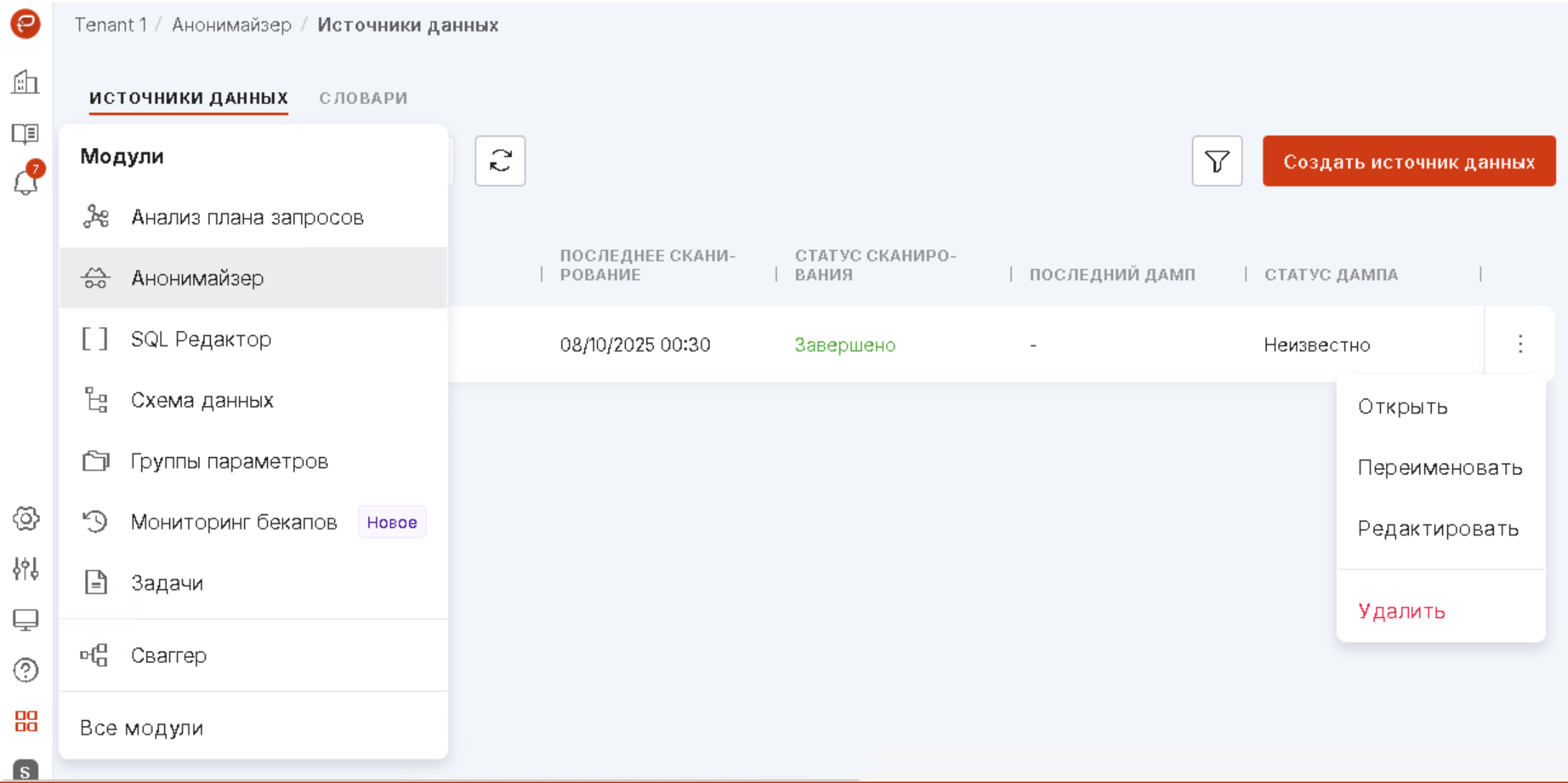




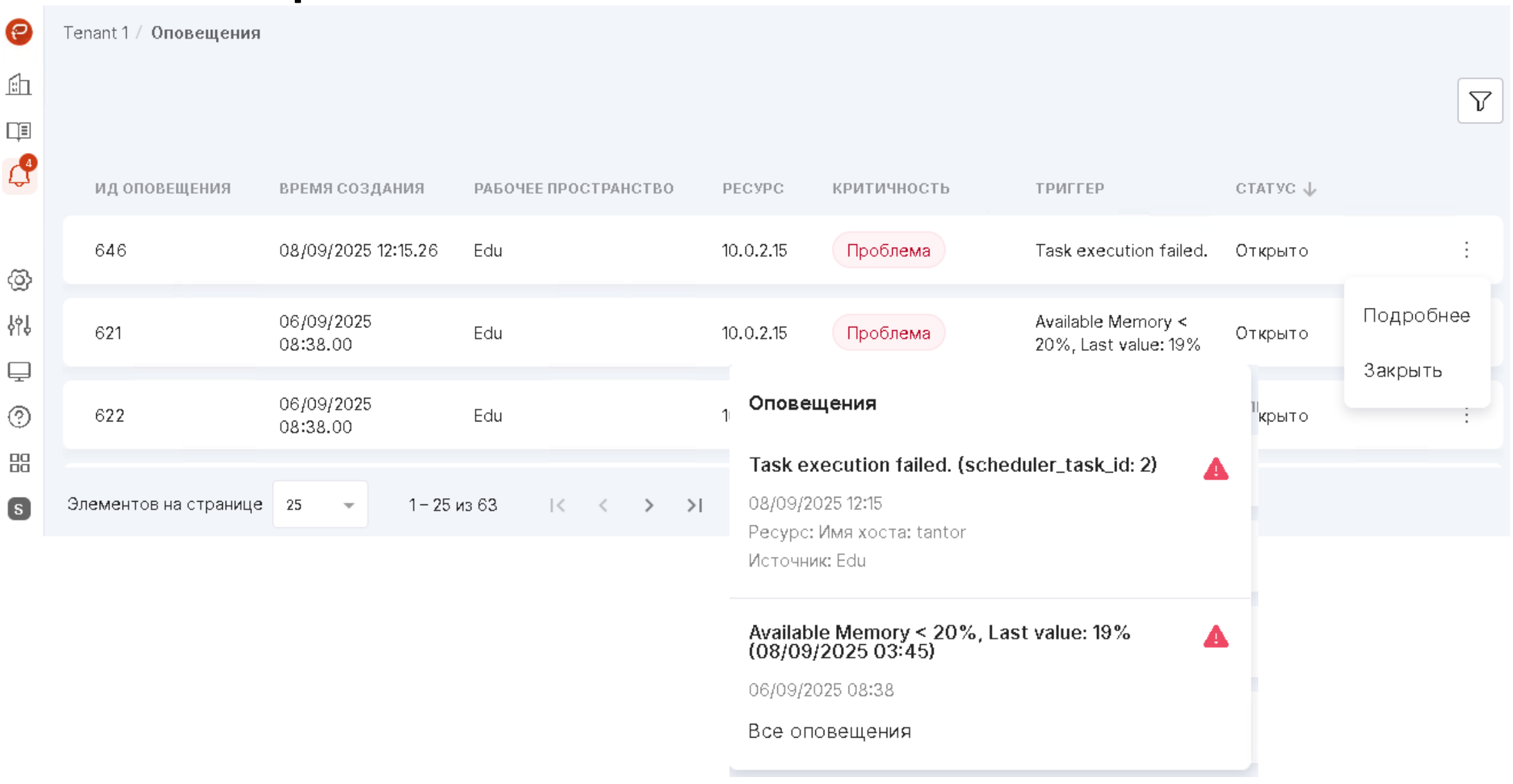
Модули Платформы Tantor



Анонимайзер

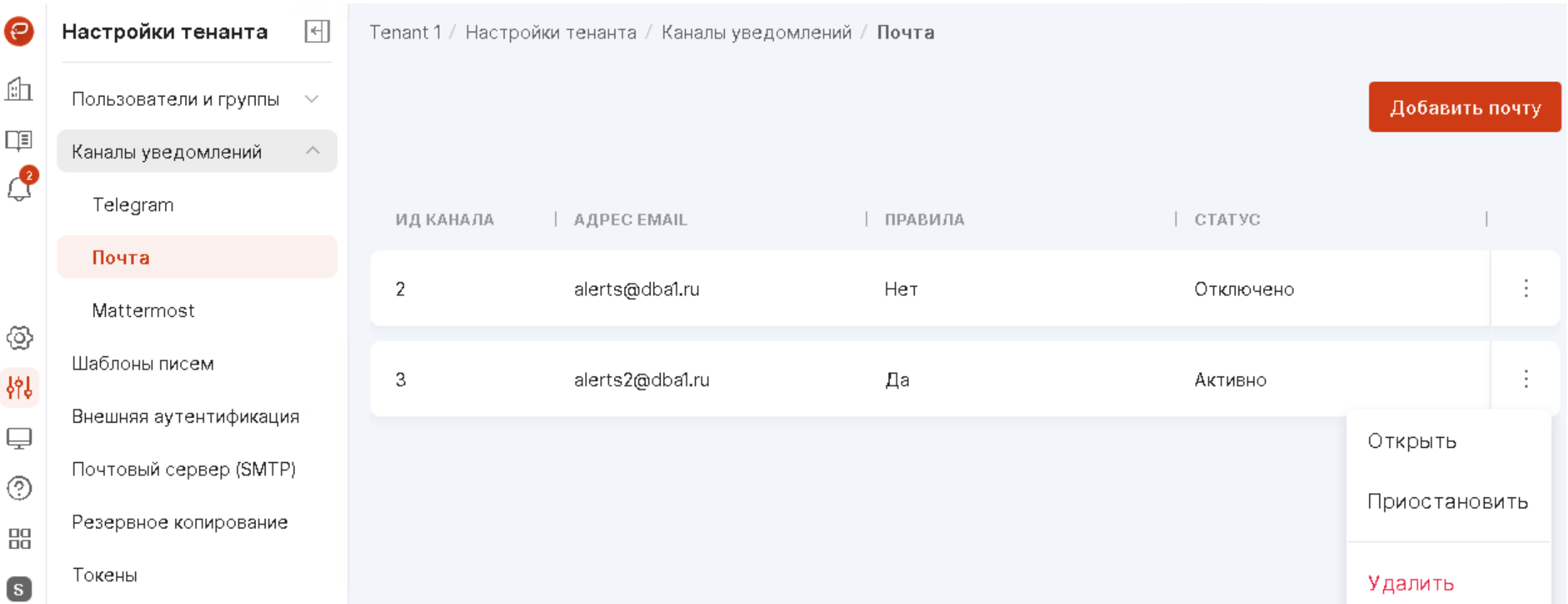


Оповещения


Tenant 1 / Оповещения
ИД ОПОВЕЩЕНИЯ
ВРЕМЯ СОЗДАНИЯ
РАБОЧЕЕ ПРОСТРАНСТВО
РЕСУРС
КРИТИЧНОСТЬ
ТРИГГЕР
СТАТУС
646
08/09/2025 12:15.26
Edu
10.0.2.15
Проблема
Task execution failed.
Открыто
621
06/09/2025 08:38.00
Edu
10.0.2.15
Проблема
Available Memory < 20%, Last value: 19%
Открыто
622
06/09/2025 08:38.00
Edu
Подробнее
Закрыть
Элементов на странице
25
1 – 25 из 63
Оповещения
Task execution failed. (scheduler_task_id: 2)
08/09/2025 12:15
Ресурс: Имя хоста: tantor
Источник: Edu
Available Memory < 20%, Last value: 19% (08/09/2025 03:45)
06/09/2025 08:38
Все оповещения

Интеграция со службами сообщений

- уведомления об оповещениях могут передаваться по:
  - e-mail, Telegram, Mattermost



Курс по Платформе Tantor

- Возможности Платформы Tantor изучаются в учебном курсе PL6: Платформа Tantor 6
  - длительность курса 2 дня
- Темы курса:
  - Возможности Платформы Tantor
  - Архитектура Платформы Тантор (4 части)
  - Мониторинг (8 частей)
  - Конфигурирование и обслуживание PostgreSQL (7 частей)
  - Установка Платформы, Prometheus, Grafana (5 частей)

10

Возможности Tantor Postgres

Tantor Postgres - ветвь PostgreSQL

СУБД Tantor Postgres включает в себя:

- все возможности "ванильного" PostgreSQL
- возможности, которые появятся в будущих версиях PostgreSQL
- дополнительные расширения и утилиты
- изменения в ядре PostgreSQL, которые нужны для высоконагруженных СУБД и приложений, формирующих сложные запросы (1C:ERP)
- собственные доработки

Доработки в Tantor Postgres

- Доработки позволяют улучшить производительность и отказоустойчивость при промышленной эксплуатации
- Доработки вносятся так, чтобы Tantor Postgres минимально отличался от основной ветви PostgreSQL
  - выбирается реализация, которая имеет наибольшую вероятность появления в основной ветви
  - наименее меняющая код PostgreSQL и его настройки по умолчанию
- Администрируя Tantor Postgres можно применять опыт администрирования PostgreSQL
- Тантор Лабс избегает доработок, которые могли бы привязать приложения ("vendor lock") и затруднить работу приложений в ванильном PostgreSQL

Дополнительные параметры конфигурации

- В Tantor Postgres 17.5 есть 15 дополнительных параметров, влияющих на создание и выбор планов запросов и функционал:

```
postgres=# \dconfig enable_*
                Parameter                 | Value
------------------------------------------+------
enable_convert_exists_as_lateral_join | on
enable_convert_in_values_to_any       | on
enable_group_by_reordering            | on
enable_index_path_selectivity         | on
enable_join_pushdown                  | on
enable_self_join_removal              | on
```

```
backtrace_on_internal_error           | off
enable_delayed_temp_file              | off
enable_large_allocations              | off
enable_temp_memory_catalog            | off
libpq_compression                     | off
wal_sender_stop_when_crc_failed       | off
pg_stat_statements.sample_rate        | 1
pg_stat_statements.mask_const_arrays | off
pg_stat_statements.mask_temp_tables   | off
```

- Параметры, появившиеся в PostgreSQL 17 версии:

```
allow_alter_system, commit_timestamp_buffers, huge_pages_status, io_combine_limit,
max_notify_queue_pages, ultixact_member_buffers, multixact_offset_buffers, notify_buffers,
restrict_nonsystem_relation_kind, serializable_buffers, subtransaction_buffers,
summarize_wal, sync_replication_slots, synchronized_standby_slots,
trace_connection_negotiation, transaction_buffers, transaction_timeout, wal_summary_keep_time
```

Расширения Tantor Postgres SE и SE 1C

- Ядра Tantor Postgres SE и SE 1C унифицированы
  - все возможности и расширения SE 1C есть в SE
- Tantor Postgres SE и SE 1C включают:
  - расширения credcheck, cube, fasttrun, fulleq, hypopg, mchar, page_repair, pg_cron, pg_hint_plan, pg_repack, pg_stat_kcache, pg_store_plans, pg_trace, pg_uuidv7, pg_wait_sampling, pgaudit, pgauditтofile, transp_anon
  - библиотеки dbcopies_decoding, oauth_base_validator, online_analyze, pg_query_id, pg_stat_advisor, plantuner, wal2json
  - утилиты pgcompacttable, pgcopydb, pg_diag, pg_repack
- в отдельных пакетах поставляются программы: pg_anon, wal-g, pg_configurator, pg_cluster, pg_diag_setup, pg_sec_check
- Tantor Postgres SE дополнительно включает расширения:
  - http, orafce, pgl_ddl_deploy, pgq, vector, pg_archive, pg_columnar, pg_ivm, pg_partman, pg_qualstats, pg_tde, pg_throttle, pg_variables, pg_background

Параметры оптимизатора запросов

- В 17 версии Tantor Postgres появилось 6 параметров, которыми можно включать оптимизации планирования запросов
- Оптимизации позволяют существенно уменьшать время выполнения встречающихся на практике запросов
- Параметры добавлены, чтобы была гибкость настройки оптимизатора запросов
- По умолчанию оптимизации включены

```
postgres=# \dconfig enable_*
                Parameter                 | Value
------------------------------------------+------
enable_convert_exists_as_lateral_join | on
enable_convert_in_values_to_any       | on
enable_group_by_reordering            | on
enable_index_path_selectivity         | on
enable_join_pushdown                  | on
enable_self_join_removal              | on
```

Библиотека pg_stat_advisor

- автоматически обнаруживает запросы, где планировщик недооценивает или переоценивает число возвращаемых строк (actual rows отличается от планируемых rows)
- если коэффициент расхождения превышает заданный порог создает расширенную статистику на таблицу

```
set pg_stat_advisor.suggest_statistics_threshold = 0.33;
set pg_stat_advisor.min_duration = 0;
create table t (i int, j int);
insert into t select i/10, i/100 from generate_series(1, 1000000) i;
analyze t;
explain (analyze, buffers, timing off) select * from t where i = 100 and j = 10;
select pg_sleep(1);
\dX
                    List of extended statistics
 Schema |    Name    | Definition  | Ndistinct | Dependencies |   MCV
--------+------------+-------------+-----------+--------------+--------
 public | t_i_j_stat | i, j FROM t | defined   | defined      | defined
\! cat $PGDATA/log/postgresql-*.log | grep pg_stat_advisor
LOG:  pg_stat_advisor: successfully created extended statistics  from public.t
```

Параметры enable_temp_memory_catalog и enable_delayed_temp_file

- `enable_temp_memory_catalog` позволяет сохранять метаданные временных объектов в локальной памяти того процесса, который работает с ними и не вносить изменения в таблицы системного каталога
  - скорость доступа к метаданным уже созданной временной таблицы выше
  - способ хранения данных во временных объектах и способ доступа к данным не меняются
- `enable_delayed_temp_file` ускоряет работу с временными таблицами, позволяя не создавать файлы временных таблиц, пока хватает памяти локального буфера серверного процесса

Алгоритм сжатия pglz

- В Tantor Postgres оптимизирован алгоритм сжатия данных pglz
- Оптимизация удаляет потенциально избыточные операции, повышая скорость сжатия в 1.4 раза.
- Сжатие используется только для типов данных переменной ширины и используется по умолчанию для большинства типов данных, которые могут использовать сжатие
- Алгоритм сжатия pglz применяется по умолчанию для сжатия TOAST

```
postgres=# \dconfig *compress*
    Список параметров конфигурации
         Параметр           | Значение
----------------------------+----------
 default_toast_compression | pglz
 libpq_compression         | off
 wal_compression           | off
```

Параметр libpq_compression

- значение по умолчанию `off`
- определяет список поддерживаемых алгоритмом сжатия сетевого трафика
- допустимые значения: `off, on, lz4, zlib`
- можно задавать уровень сжатия, например:

```
alter system set libpq_compression =
'lz4:1,zlib:2';
```

Параметр wal_sender_stop_when_crc_failed

- снижает вероятность передачи поврежденных журнальных записей на реплики
- по умолчанию отключён
- при включении процессы walsender будут проверять контрольные суммы журнальных записей перед передачей клиентам. Если контрольная сумма не совпадёт, то процессы попытаются прочесть запись из журнального буфера (WAL buffer). Если в журнальном буфере уже не будет записи или контрольная сумма не совпадёт, то `walsender` остановится

Параметр backtrace_on_internal_error

- Параметр относится к группе Developer Options, то есть не используется при промышленной эксплуатации
- Если этот параметр включен и возникает ошибка с кодом XX000 (internal_error), то трассировка стека записывается в диагностический журнал вместе с сообщением об ошибке
  - Это полезно при отладке внутренних ошибок, которые обычно не возникают в рабочей среде
- По умолчанию выключен.

Расширение uuid_v7

- расширение добавляет функцию **высокоскоростной** генерации uuid_v7
- uuid_v7 генерирует возрастающие значения, что **позволяет** использовать оптимизацию вставки в индекс btree
  - при очень частых вставках в сессии высокая скорость, не медленнее, чем с bigserial
  - структура индекса остаётся оптимальной и компактной
- uuid занимает 16 байт, bigint 8 байт
- пример использования:

```
create extension if not exists "uuid-ossp";
create extension if not exists pg_uuidv7;
create table tt1 (id uuid default uuidv7() primary key, data bigint);
create table tt2 (id bigint generated by default as identity primary key, data bigint);
select count(*), pg_indexes_size('tt1') from tt1;
```

Расширение pg_tde (Transparent Data Encryption)

- Реализует "прозрачное" шифрование данных (Transparent Data Encryption)
- Прозрачность означает, что клиент получает и передает незашифрованные данные
- Не шифрует данные в памяти (в буферном кэше) и при передаче по сети
- Шифрование выполняется протоколами: AES, Magma, Kuznyechik, ChaCha20
- Физическая и логическая репликация поддерживаются
- Таблицы системного каталога не шифруются
- pg_rewind пока не работает с зашифрованным WAL
- WAL шифруются полностью

Валидатор oauth_base_validator

- В Tantor Postgres 17 есть способ аутентификации oauth (OAuth 2.0), который появится в PostgreSQL 18 версии
- Этот способ использует внешнюю службу для аутентификации
- Метод oauth вставляется в четвертое поле строки в файле `pg_hba.conf`

```
#TYPE DATABASE USER ADDRESS METHOD
local all      all          oauth issuer="http://1.1.1.1:80/realms/a" scope="openid" map="o1"
```

- Для использования способа аутентификации oauth нужно на языке C написать "валидатор"
- В Tantor Postgres библиотека с валидатором поставляется
- Пример аутентификации в psql:

```
psql "user=alice dbname=postgres oauth_issuer=http://1.1.1.1:80/realms/a oauth_client_id=user1
oauth_client_secret=AbCdEf123GhIjKl"
Visit http://1.1.1.1:80/realms/a/device and enter the code: XYZX-XYZO
postgres=>
```

Библиотека credcheck

- Использует библиотеку, которая может загружаться на уровне кластера (параметр `shared_preload_libraries`) и для одной сессии (командой `LOAD credcheck`)
- При загрузке регистрирует 30 параметров конфигурации, которыми можно задать проверки сложности пароля, защиту от подбора пароля, параметры повторного использования паролей, список ролей на которые проверки не действуют и т.п.
- В расширении 8 функций и 2 представления
- Расширение срабатывает при создании роли, переименовании, изменения пароля, аутентификации

```
postgres=# \dconfig credcheck.*
               Parameter                | Value
----------------------------------------+-------
 credcheck.auth_delay_ms                | 0
 credcheck.encrypted_password_allowed   | off
 credcheck.max_auth_failure             | 0
...
 credcheck.whitelist                    |
(30 rows)
```

Расширения fasttrun и online_analyze

- Усечение временных таблиц обновляет таблицу pg_class
- Расширение fasttrun содержит функцию fasttruncate
  - у файлов не меняется название
  - таблицы системного каталога не меняются
  - может использоваться вместо команды TRUNCATE
  - работает только с временными таблицами
- После усечения или других команд можно собрать статистику
- Для автоматического сбора статистики можно использовать расширение online_analyze
  - расширение настраивается параметрами конфигурации

```
select fasttruncate('tt');
INFO:  analyzing "pg_temp_5.tt"
INFO:  "tt": scanned 0 of 0 pages, containing 0 live rows and 0 dead rows; 0 rows in sample, 0 estimated
total rows
INFO:  analyze "tt" took 0.00 seconds
 fasttruncate
--------------

(1 row)
```

Расширение mchar

- Добавляет поддержку типов mchar, mvarchar для совместимости с Microsoft SQL Server
- Для типов mchar и mvarchar определяет функции и операторы:
  - length()
  - substr(str, pos[, length])
  - || конкатенация с разными типами (mchar || mvarchar)
  - < <= = >= > сравнение без чувствительности к регистру (ICU)
  - &< &<= &= &>= &> сравнение с чувствительностью к регистру (ICU)
  - LIKE
  - SIMILAR TO
  - ~ (регулярные выражения)
- Добавлят неявное приведение типов mchar к mvarchar и обратно
- поддержку типов индексами b-tree и hash
- Использование индексов для выполнения оператора LIKE

Расширение fulleq

- При использовании оператора "=" для сравнения значений, если хотя бы один из операндов имеет значение NULL, результатом будет NULL
- В приложениях 1С часто используется оператор "==", который возвращает true, когда операнды равны или оба имеют значение NULL, что удобно при работе с базами данных, особенно с 1С, где операторы и семантика работы с NULL отличаются от стандарта SQL
- Оператор “==” из расширения fulleq позволяет высокоэффективно выполнять сравнение значений с использованием нужной логики

Расширение orafce

- Расширение, предоставляющее функции, упрощающие миграцию кода приложений с Oracle Database
- orafce эмулирует часть функционала и пакетов Oracle Database
- Создаёт большое количество функций и других объектов и типов данных, которые работают так же, как их аналоги в Oracle Database
- Расширение упрощает миграцию кода приложений в Tantor Postgres с Oracle Database
- Расширение создаёт 15 схем, в которых находятся объекты расширения. Имена восьми схем соответствуют именам пакетов в Oracle Database

Расширение http

- расширение http предоставляет возможность выполнять HTTP и HTTPS запросы прямо из SQL
- устанавливается в базу данных командой `create extension http;`
- можно создать триггер, обращающийся к веб-службе, передать данные и получить результат, который можно использовать в логике триггера
- pgsql-http может быть полезен в:
  - при интеграции с внешними API
  - в интерактивных приложениях
  - для обработки данных в реальном времени

Расширение pg_store_plans

- Расширение для отслеживания статистики выполнения SQL-запросов
- сохраняет полные планы запросов, а не только их статистику или текст
- использование pg_store_plans может увеличить нагрузку на систему из-за сбора и хранения дополнительных данных

Расширение pg_variables

- Расширение, позволяющее создавать и использовать переменные в SQL-запросах
- по умолчанию не транзакционны, но могут быть транзакционными
- хранятся в памяти серверного процесса
- существуют только в текущем сеансе пользователя
- могут использоваться на физических репликах, временные таблицы не могут
- скорость работы сравнима с временными таблицами
- отсутствие накладных расходов (не требуют реальный номер транзакции, не создают файлы)
- альтернатива set_config()/current_setting()

Производительность при использовании pg_variables

- При сравнении скорости доступа к таблицам в памяти, временным, обычным результаты не очевидны
- Расширение pg_variables не является стандартным и не пользуется популярностью из-за неудобного синтаксиса использования

```
 select * from pgv_select('bookings', 'tickets', '0005432020304'::char(13)) as (ticket_no character(13),
book_ref character(6), passenger_id character varying(20), passenger_name text, contact_data jsonb);
 Time: 0.281 ms
 select * from tickets where ticket_no='0005432020304';
 Time: 0.266 ms
 select book_ref from tickets where passenger_name like '%G IVANOV' limit 1;
 Time: 0.463 ms
 select book_ref from tickets1 where passenger_name like '%G IVANOV' limit 1;
 Time: 1.169 ms
 select book_ref from pgv_select('bookings', 'tickets', '0005432020304'::char(13)) as (ticket_no
character(13), book_ref character(6), passenger_id character varying(20), passenger_name text, contact_data
jsonb) where passenger_name like '%G IVANOV' limit 1;
 Time: 0.185 ms
```

Преимущества расширения pg_variables

- Временные таблицы нельзя использовать на репликах
- функции расширения pg_variables можно использовать на репликах точно так же, как на мастере
- pg_variables хранит данные только в локальной памяти серверного процесса и не использует временные файлы
- Переменные могут быть транзакционными и нет
- На сохраняемые объекты действует ограничение строкового буфера - 1Гб

```
select pgv_insert('bookings','t2', pgbench_branches) from pgbench_branches;
select * from pgv_select('bookings','t2',1) as (bid int, bbalance int, filler character(88));
 bid | bbalance | filler
-----+----------+--------
   1 |        0 |
select pgv_select('bookings','t2',1);
 pgv_select
------------
 (1,0,)
```

Расширение pg_stat_kcache

- дополняет `pg_stat_statements` и зависит от него
- собирает статистику linux, выполняя системный вызов `getrusage` после выполнения каждой команды
- в отличие от утилит операционной системы, расширение собирает статистики с детальностью до команды
- позволяет различать читался блок с диска или из страничного кэша
- использует два буфера в разделяемой памяти

```
 select * from (select *,lead(off) over(order by off)-off as diff from pg_shmem_allocations) as a where name
like 'pg_%';
           name            |    off    | size  | allocated_size |  diff
---------------------------+-----------+-------+----------------+---------
 pg_stat_statements        | 148162816 |    64 |            128 |     128
 pg_stat_statements hash   | 148162944 |  2896 |           2944 | 2188544
 pg_stat_kcache            | 150351488 |   992 |           1024 |    1024
 pg_stat_kcache hash       | 150352512 |  2896 |           2944 | 1373056
select name, setting, context, min_val, max_val from pg_settings where name like '%kcache%';
             name              | setting |  context   | min_val |  max_val
-------------------------------+---------+------------+---------+------------
 pg_stat_kcache.linux_hz       | 333333  | user       | -1      | 2147483647
 pg_stat_kcache.track          | top     | superuser  |         |
 pg_stat_kcache.track_planning | off     | superuser  |         |
```

Статистики, собираемые `pg_stat_kcache`

Статистики в представлениях `pg_stat_kcache` и `pg_stat_kcache_detail`:

- `reads_blks` reads, in 8K-blocks
- `writes_blks` writes, in 8K-blocks
- `user_time` user CPU time used
- `system_time` system CPU time used
- `minflts` page reclaims (soft page faults)
- `majflts` page faults (hard page faults)
- `nswaps` swaps
- `msgsnds` IPC messages sent
- `msgrcvs` IPC messages received
- `nsignals` signals received
- `nvcsws` voluntary context switches
- `nivcsws` involuntary context switches

```
alter system set shared_preload_libraries = pg_stat_statements, pg_wait_sampling, pg_stat_kcache;
create extension pg_stat_kcache;
\dx+ pg_stat_kcache
 function pg_stat_kcache()
 function pg_stat_kcache_reset()
 view pg_stat_kcache
 view pg_stat_kcache_detail
```

Расширение pg_wait_sampling

- входит во все сборки Tantor Postgres
- выдает статистику по событиям ожиданий всех процессов экземпляра
- для установки нужно загрузить библиотеку и установить расширение:

```
alter system set shared_preload_libraries = pg_stat_statements, pg_stat_kcache, pg_wait_sampling;
create extension if not exists pg_wait_sampling;
```

- библиотека `pg_wait_sampling` должна быть указана позже `pg_stat_statements`, чтобы расширение не перезаписало идентификаторы запросов (queryid)
- расширение использует фоновый процесс `pg_wait_sampling collector`
- процесс опрашивает состояние всех процессов экземпляра
- в расширение входят 4 функции и 3 представления:

```
\dx+ pg_wait_sampling
 function pg_wait_sampling_get_current(integer)
 function pg_wait_sampling_get_history()
 function pg_wait_sampling_get_profile()
 function pg_wait_sampling_reset_profile()
 view pg_wait_sampling_current
 view pg_wait_sampling_history
 view pg_wait_sampling_profile
```

История событий ожидания

- расширение использует "сэмплрование" с частотой от 1 миллисекунды (по умолчанию 10 миллисекунд)
- история доступна через представление:

```
select count(*) from pg_wait_sampling_history;
count
-----
5000
```

- по умолчанию сохраняется история 5000 последних событий ожидания
- расширение также использует разделяемую память под хранение своих трёх структур:
  - очередь (MessageQueue) фиксированного размера 16Кб
  - память под список PID
  - память под идентификаторы команд (`queryid`), выполняющихся процессами

```
 select * from (select *, lead(off) over(order by off)-off as diff from pg_shmem_allocations) as a
where name like '%wait%';
        name        |    off    | size  | allocated_size | diff
--------------------+-----------+-------+----------------+-------
 pg_wait_sampling | 148145920 | 17536 |          17536 | 17536
```

Расширение pg_background

- Расширение позволяет асинхронно (в фоновом режиме) выполнять произвольную команду и реализовать произвольные задачи, которые нужно выполнить приложению или администратору. Задачи будут выполняться фоновыми процессами экземпляра.
- Расширение содержит функции :
  - `pg_background_launch()`
  - `pg_background_result()`
  - `pg_background_detach()`

Расширения pgaudit и pgaudittofile

- расширениями `pgaudit` и `pgauditlogtofile` можно направить сообщения о создании сессий и их длительности в отдельный файл или файлы аудита
- для работы расширений нужно загрузить их библиотеки:

```
alter system set shared_preload_libraries = pgaudit, pgauditlogtofile;
```

- Расширения работают независимо и параллельно с журналом кластера и управляются собственными параметрами, которые имеют префикс "`pgaudit.`"
- параметры `pgaudit.log_connections` и `pgaudit.log_disconnections` аналогичны одноимённым параметрам PostgreSQL и могут создавать аналогичные записи в отдельном файле аудита

Конфигурирование расширений pgaudit и pgaudittofile

- семь параметров относятся к библиотеке `pgauditlogtofile`
- чтобы создавался журнал аудита нужно установить параметр `pgaudit.log` как минимум в значение `'misc'`
  - `'none'` - файл журнала аудита не создаётся
  - `'role'` и `'ddl'` `pgaudit.log_connections` и `pgaudit.log_disconnections` не действуют

```
postgres=# \dconfig pgaudi*
       List of configuration parameters
            Parameter              |    Value
-----------------------------------+----------
 pgaudit.log                       | none
 pgaudit.log_autoclose_minutes     | 0
 pgaudit.log_catalog               | on
 pgaudit.log_client                | off
 pgaudit.log_connections           | off
 pgaudit.log_directory             | log
 pgaudit.log_disconnections        | off
 pgaudit.log_filename              | audit-%F.log
 pgaudit.log_level                 | log
 pgaudit.log_parameter             | off
 pgaudit.log_parameter_max_size    | 0
 pgaudit.log_relation              | off
 pgaudit.log_rotation_age          | 1d
 pgaudit.log_rotation_size         | 0
 pgaudit.log_rows                  | off
 pgaudit.log_statement             | on
 pgaudit.log_statement_once        | off
 pgaudit.role                      |
(18 rows)
```

Утилита pgcopydb

- Утилита автоматизирует копирование базы данных на другой кластер
- Типичный случай использования pgcopydb - миграция на новую основную версию PostgreSQL с минимизацией простоя
- Утилита реализует задачу распараллеливания с поточной передачей данных по логике "pg_dump -jN | pg_restore -jN" между двумя работающими кластерами, дирижируя этими утилитами
- Поддерживает параллельное создание индексов, отслеживание изменений и их применение, возобновление прерванной перегрузки, фильтрацию объектов
- Проект с открытым исходным кодом

Утилита pg_anon

Программа, написанная на языке python выполняет:

- поиск конфиденциальных данных в базе данных
- создание словаря на основе результатов поиска
- сохранение и восстановление с использованием словаря
- синхронизация содержимого или структуры указанных таблиц между исходной и целевой базами данных
- выгрузка данных с маскировкой (обезличиванием, анонимизацией) по заданным шаблонам

Утилита pg_configurator

- Представляет собой скрипт pg_configurator на языке python, устанавливаемый по пути /usr/bin
- предлагает рекомендуемые параметры конфигурации на основе характеристик аппаратных ресурсов, таких как доступная память, количество процессоров и дисковое пространство и т.д. Это позволяет оптимально использовать имеющиеся ресурсы и увеличить производительность экземпляра

Утилита pg_diag_setup.py

- Задача, решаемая утилитой: вызвать утилиту на хосте кластера баз данных, которая установит и настроит параметры расширений в соответствии с шаблоном и зарезервирует значения параметров, чтобы можно было восстановить изменения
- Проверяет доступность расширений через `pg_available_extensions`
- Обновляет значение `shared_preload_libraries` без перезаписи существующих библиотек
- Добавляет новые параметры в конец `postgresql.conf`
- Создает текстовый файл бэкапа со значениями конфигурируемых параметров с временной меткой
- Позволяет откатить изменения к любому бэкапу

Утилита pg_sec_check

- позволяет автоматизировать процесс проверки параметров безопасности, от настроек операционной системы до параметров конфигурации PostgreSQL
- создаёт отчеты о выявленных проблемах и рекомендации по их устранению
- проверки выполняются скриптами `.sql` и `.sh`
- результаты проверок валидируются скриптами на языке **Lua**, ими же формируются отчеты и рекомендации
- проверяет целостность своих файлов контрольными суммами
- В утилите имеется 68 проверок
- можно создавать собственные проверки путем написания скриптов `.sql .sh .lua`

Утилита WAL-G (Write-Ahead Log Guard)

- утилита для создания шифрованных резервных копий кластера баз данных (полных и инкрементальных) и архивирования WAL-сегментов, их высокоэффективной отправки/получения по протоколу S3 "из" и "в" хранилища (облачные в сети предприятия или внешние) напрямую без создания промежуточных файлов в файловой системе.
- Используя утилиту WAI-G можно:
  - создавать резервные копии кластера и WAL-сегментов
  - восстанавливать кластер на выбранный момент времени в прошлом
  - управлять резервными копиями по протоколу S3: удалять ненужные бэкапы и связанные с ними файлы журналов

Другие расширения

- dbcopies_decoding - библиотека 1С, предоставляет слоты логической репликации при копировании баз данных баз данных 1С
- vector - полная поддержка типа данных векторов большой размерности
- pg_partman - автоматизация поддержки секционированных таблиц
- pg_qualstats - ведёт статистику по предикатам, найденным в операторах WHERE и в предложениях JOIN
- pg_hint_plan - подсказки оптимизатору в запросах
- plantuner скрывает индексы от планировщика
- pg_cron - планировщик внутри экземпляра
- pg_throttle - ограничение объема читаемых строк в запросах, позволяющее снизить конкуренцию за ввод-вывод
- pg_trace - трассировки работающих запросов, может быть полезна для анализа запросов в 1С
- pg_ddl_deploy - расширение для логической репликации
- pqg - очередь в базе данных

Практика

1. Расширение orafce
2. Расширение pg_variables
3. Расширение page_repair
   1. Подготовка реплики
   2. Подготовка таблицы
   3. Восстановление страницы с помощью page_repair
   4. Обнуление страницы
4. Отладка подпрограмм
   1. Установка расширения из исходных кодов на примере pldebugger
   2. Отладка функции в pgAdmin
   3. Отладка подпрограмм в DBeaver
5. Обработка строк большого размера StringBuffer
6. Поиск осиротевших файлов
7. Резервирование и восстановление WAL-G
   1. Установка WAL-G
   2. Конфигурирование кластера для архивирования журналов
   3. Восстановление из бэкапа, созданного WAL-G
   4. Остановка архивирования журналов