DBA1-18: Администрирование PostgreSQL 18

Демонстрации

tantor

Автор: Олег Иванов

Оглавление

Авторские права

Автор: Олег Иванов

Создан: **28 апреля 2026 г.**

По вопросам обучения обращайтесь: **edu@tantorlabs.ru**

Глава 1. Установка Tantor Postgres

Часть 1. Инсталляция СУБД Tantor

В виртуальной машине курса предустановлена версия Tantor Postgres SE для целей обучения.

Продемонстрируем установку Tantor Postgres BE.

1) Откроем терминал с правами root:

```
astra@tantor:~$ su -
Password: root
```

2) Выполним предварительные проверки.

Число ядер процессора (результат может отличаться от приведенных как пример значений):

```
root@tantor:~# cat /proc/cpuinfo | grep cores
cpu cores   : 4
cpu cores   : 4
```

Оперативной памяти:

```
root@tantor:~# cat /proc/meminfo | grep Mem
MemTotal:        2981796 kB
MemFree:         1439696 kB
MemAvailable:    2286608 kB
```

Свободное место в точке монтирования "/":

```
root@tantor:~# df -HT | grep /$
/dev/sda1      ext4   50G   17G   31G  35% /
```

Свободно 31 ГБ.

При промышленной эксплуатации рекомендуется иметь 4 ядра.

Оперативной памяти: по крайней мере 4 ГБ.

Свободного места на системе хранения («диске»): 40 ГБ.

3) Скачаем инсталлятор:

```
root@tantor:~# wget https://public.tantorlabs.ru/db_installer.sh
https://public.tantorlabs.ru/db_installer.sh
Resolving public.tantorlabs.ru (public.tantorlabs.ru)... 84.201.157.208
Connecting to public.tantorlabs.ru (public.tantorlabs.ru)|84.201.157.208|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 18312 (18K) [application/octet-stream]
Saving to: 'db_installer.sh.1'

db_installer.sh              100%[============================>]  17,88K  --.-
KB/s  in 0s
'db_installer.sh' saved [22273/22273]]
```

4) Посмотрим разрешение на исполнение инсталлятора:

```
root@tantor:~# ls -al db_installer.sh
-rwxr--r-- 1 root root 18353 db_installer.sh
```

Файл инсталлятора был в виртуальной машине на случай отсутствия интернет. Инсталлятор со временем обновляется, можно использовать любой.

5) Установка разрешения на исполнение инсталлятора:

```
root@tantor:~# chmod +x db_installer.*
```

6) Удалим директорию кластера, если она осталась с предыдущих попыток установки:

```
root@tantor:~# rm -rf /var/lib/postgresql/tantor-be-18
```

7) Проверим версию инсталлятора и обратим внимание на выделенные параметры:

```
root@tantor:~# ./db_installer.sh --help
=========================================================
Usage: db_installer.sh [OPTIONS]
Installer version: 25.07.14
This script will perform installation of the Tantor DB on current host.
If the Tantor DB is already installed, no actions will be taken.
 Available options:
  --help                  Show this help message.
---------------------------------------------------------
  --edition=                Set edition (be, se, se-1c, certified, certified-2).
                          "se" is default.
  --major-version=        Set major version (14, 15, 16, 17)
  --maintenance-version=  Set maintenance version (15.2.4).
                          By default latest version will be installed.

  --do-initdb             After installation run initdb with checksums.
   --package=             Set specific package (all, client, libpq5).
                          "all" is default.
-------------------------------------------------------------------
  --from-file=            Install package from local file (rpm, deb)
                          May be used with --do-initdb option
=========================================================
Example for commercial use
=========================================================
export NEXUS_USER="user_name"
export NEXUS_USER_PASSWORD="user_password"
export NEXUS_URL="nexus.tantorlabs.ru"
./db_installer.sh \
      --do-initdb \
    --major-version=15 \
      --edition=se
=========================================================
Example for evaluation use (without login and password)
Only for Basic Edition
=========================================================
export NEXUS_URL="nexus-public.tantorlabs.ru"
./db_installer.sh \
      --do-initdb \
    --major-version=15 \
      --edition=be
 =========================================================
Examples how to install from file
=========================================================
./db_installer.sh \
    --from-file=./packages/tantor-be-server-15_15.4.1.jammy_amd64.deb
./db_installer.sh \
      --do-initdb \
    --from-file=/tmp/tantor-be-server-15_15.4.1.jammy_amd64.deb
```

8) Так как уже установлен Tantor Postgres SE 18, остановим основной кластер, сохраним директорию, файл профиля:

```
root@tantor:~# systemctl stop tantor-se-server-18
mv /opt/tantor/db /opt/tantor/db.SAV
cp /var/lib/postgresql/.bash_profile /var/lib/postgresql/bash_profile
```

Инсталлятор скачан, порт по умолчанию 5432 свободен, адрес репозитория с дистрибутивами установлен. Можно приступить к установке.

9) Установка адреса расположения дистрибутивов:

```
root@tantor:~# export NEXUS_URL="nexus-public.tantorlabs.ru"
```

10) Установка без скачивания дистрибутива, с созданием базы данных:

```
root@tantor:~# ./db_installer.sh --edition=be  --major-version=18 --do-initdb --from-file=/root/tantor-be-server-18_18.3.0_amd64.deb
```

Если есть доступ в интернет и хочется продемонстрировать установку со скачиванием дистрибутива:

```
root@tantor:~# ./db_installer.sh --edition=be  --major-version=18 --do-initdb
Hit:1 https://download.astralinux.ru/astra/stable/1.8_x86-64/repository-extended 1.8_x86-64 InRelease
Hit:2 https://download.astralinux.ru/astra/stable/1.8_x86-64/repository-main 1.8_x86-64 InRelease
Reading package lists... Done
dpkg is installed (found in PATH).
Warning: apt-key is deprecated. Manage keyring files in trusted.gpg.d instead (see apt-key(8)).
OK
deb [arch=amd64] https://nexus-public.tantorlabs.ru/repository/astra-1.8 1.8_x86-64 main
Hit:1 https://download.astralinux.ru/astra/stable/1.8_x86-64/repository-extended 1.8_x86-64 InRelease
Hit:2 https://download.astralinux.ru/astra/stable/1.8_x86-64/repository-main 1.8_x86-64 InRelease
Get:3 https://nexus-public.tantorlabs.ru/repository/astra-1.8 1.8_x86-64 InRelease [2,062 B]
Get:4 https://nexus-public.tantorlabs.ru/repository/astra-1.8 1.8_x86-64/main amd64 Packages [18.8 kB]
Get:5 https://nexus-public.tantorlabs.ru/repository/astra-1.8 1.8_x86-64/main all Packages [1,692 B]
Fetched 22.5 kB in 1s (33.4 kB/s)
Reading package lists... Done
W: https://nexus-public.tantorlabs.ru/repository/astra-1.8/dists/1.8_x86-64/InRelease: Key is stored in legacy
trusted.gpg keyring (/etc/apt/trusted.gpg), see the DEPRECATION section in apt-key(8) for details.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
 tantor-be-server-18
0 upgraded, 1 newly installed, 0 to remove and 2101 not upgraded.
Need to get 22.2 MB of archives.
After this operation, 0 B of additional disk space will be used.
Get:1 https://nexus-public.tantorlabs.ru/repository/astra-1.8 1.8_x86-64/main amd64 tantor-be-server-18 amd64
18.1.0 [22.2 MB]
Fetched 22.2 MB in 1s (22.4 MB/s)
Selecting previously unselected package tantor-be-server-18.
(Reading database ... 316583 files and directories currently installed.)
Preparing to unpack .../tantor-be-server-18_18.1.0_amd64.deb ...
+ echo ----------------------------------------
+ echo 'tantor-be-server-18 is getting installed'
tantor-be-server-18 is getting installed
+ echo ----------------------------------------
+ getent group postgres
+ getent passwd postgres
++ getent passwd postgres
```

```
++ awk -F: '{print $6}'
+ current_home=/var/lib/postgresql
+ '[' /var/lib/postgresql '!=' /var/lib/postgresql ']'
+ mkdir -p /var/lib/postgresql
+ chown postgres:postgres /var/lib/postgresql
+ chmod 700 /var/lib/postgresql
+ mkdir -p /var/run/postgresql
+ chown postgres:postgres /var/run/postgresql
+ '[' '!' -d /usr/lib/tmpfiles.d ']'
+ echo 'D /run/postgresql 0755 postgres postgres - -'
+ tee /usr/lib/tmpfiles.d/tantor-db.conf
+ echo ----------------------------------------
+ set +vx
Unpacking tantor-be-server-18 (18.1.0) ...
Setting up tantor-be-server-18 (18.1.0) ...
+ echo ----------------------------------------
+ echo 'tantor-be-server-18 is getting installed'
tantor-be-server-18 is getting installed
+ echo ----------------------------------------
+ /usr/sbin/ldconfig
+ /usr/sbin/ldconfig -n /opt/tantor/db/18/lib
+ /bin/systemctl daemon-reload
+ '[' '!' -f /var/lib/postgresql/.bash_profile ']'
+ '[' -f /var/lib/postgresql/.bash_profile ']'
++ grep /opt/tantor/db/18/bin /var/lib/postgresql/.bash_profile
+ '[' -z 'export PATH=/opt/tantor/db/18/bin:$PATH' ']'
+ '[' '!' -d /var/lib/postgresql/tantor-be-18/data ']'
+ '[' '!' -d /var/lib/postgresql/data ']'
+ mkdir -p /var/lib/postgresql/tantor-be-18/data
+ chown postgres:postgres /var/lib/postgresql/tantor-be-18/data
+ chmod 700 /var/lib/postgresql/tantor-be-18/data
+ echo ----------------------------------------
+ set +vx
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.

The database cluster will be initialized with locale "en_US.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".

Data page checksums are enabled.

fixing permissions on existing directory /var/lib/postgresql/tantor-be-17/data ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default "max_connections" ... 100
selecting default "shared_buffers" ... 128MB
selecting default time zone ... Europe/Moscow
creating configuration files ... ok
running bootstrap script ... ok
performing post-bootstrap initialization ... ok
syncing data to disk ... ok

initdb: warning: enabling "trust" authentication for local connections
initdb: hint: You can change this by editing pg_hba.conf or using the option -A, or --auth-local
and --auth-host, the next time you run initdb.

Success. You can now start the database server using:

    /opt/tantor/db/17/bin/pg_ctl -D /var/lib/postgresql/tantor-be-18/data -l logfile start

Created symlink /etc/systemd/system/multi-user.target.wants/tantor-be-server-18.service
/lib/systemd/system/tantor-be-server-18.service.
 tantor-be-server-18.service - Tantor Basic database server 18
     Loaded: loaded (/lib/systemd/system/tantor-be-server-18.service; enabled; preset: enabled)
     Active: active (running) since Thu 2026-04-23 18:06:08 MSK; 25ms ago
       Docs: https://docs.tantorlabs.ru/tdb/ru/
    Process: 8224 ExecStartPre=/opt/tantor/db/18/bin/postgresql-check-db-dir ${PGDATA}
(code=exited, status=0/SUCCESS)
    Process: 8226 ExecStart=/opt/tantor/db/18/bin/pg_ctl start -D ${PGDATA} -s -w -t
${PGSTARTTIMEOUT} (code=exited, status=0/SUCCESS)
   Main PID: 8228 (postgres)
      Tasks: 6 (limit: 3390)
     Memory: 18.5M
```

```
        CPU: 68ms
     CGroup: /system.slice/tantor-be-server-18.service
             ├─8228 /opt/tantor/db/18/bin/postgres -D /var/lib/postgresql/tantor-be-18/data
             ├─8230 "postgres: checkpointer "
             ├─8231 "postgres: background writer "
             ├─8233 "postgres: walwriter "
             ├─8234 "postgres: autovacuum launcher "
             └─8235 "postgres: logical replication launcher "

tantor systemd[1]: Starting tantor-be-server-18.service - Tantor Basic database server 18...
tantor pg_ctl[8228]: MSK [8228] LOG:  starting Tantor Basic Edition 18.1.0 e4cea675 on x86_64-pc-
linux-gnu, compiled by gcc (Astra 12.2.0-14.astra3) 12.2.0, 64-bit
tantor pg_ctl[8228]: [8228] LOG:  listening on IPv4 address "127.0.0.1", port 5432
tantor pg_ctl[8228]: LOG:  listening on IPv6 address "::1", port 5432
tantor pg_ctl[8228]: [8228] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
tantor pg_ctl[8232]: [8232] LOG:  database system was shut down at 2026-04-23 18:06:06 MSK
tantor pg_ctl[8228]: [8228] LOG:  database system is ready to accept connections
tantor systemd[1]: Started tantor-be-server-18.service - Tantor Basic database server 18.

Pager usage is off.
       tantor_version
-----------------------------
 Tantor Basic Edition 18.3.0
(1 row)

Installation successfully completed.
```

При создании кластера инсталлятором включается подсчет контрольных сумм для блоков данных.

Файлы, относящиеся к этой СУБД, будут принадлежать пользователю "postgres".

От его имени также будет запускаться процесс сервера.

Кластер баз данных инициализирован со следующими параметрами локали:

```
The database cluster will be initialized with locale "en_US.UTF-8".
The default database encoding has accordingly been set to "UTF8".
The default text search configuration will be set to "english".
```

Кодировка БД по умолчанию, выбранная в соответствии с настройками: "UTF8".

Выбрана конфигурация текстового поиска по умолчанию "english".

Контроль целостности страниц данных включён.

initdb: предупреждение: включение метода аутентификации "trust" для локальных подключений.

initdb: подсказка: Другой метод можно выбрать, отредактировав `pg_hba.conf` или ещё раз запустив `initdb` с ключом -A, `--auth-local` или `--auth-host`.

Если в переменной окружения `PATH` пользователя `postgres` (файлы профиля `/var/lib/postgresql/.bash_profile`) присутствовала директория другой сборки (`export PATH=/opt/tantor/db/18/bin:$PATH`), то кластер будет создан и запущен из-под этой сборки (`Tantor Special Edition 18.1.0`)

11) Проверяем, что путь к исполняемым файлам в файле профиля пользователя `postgres` в конце файла установлен в PATH. Если этого нет, то добавляем PATH и PGDATA, чтобы было так, как приведено ниже:

```
root@tantor:~# mcedit /var/lib/postgresql/.bash_profile
```

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

12) Проверка того, что кластер работает:

```
root@tantor:~# su - postgres -c 'psql -c "select tantor_version()"'
Pager usage is off.
       tantor_version
-----------------------------
 Tantor Basic Edition 18.1.0
(1 row)


root@tantor:~#
```

Версия кластера Basic Edition.

Демонстрация установки выполнена.

Часть 2. Деинсталляция

1) Остановим экземпляр кластера Tantor BE:

```
root@tantor:~# systemctl stop tantor-be-server-18
```

2) Запретим автоматический запуск службы:

```
root@tantor:~# systemctl disable tantor-be-server-18
Removed /etc/systemd/system/multi-user.target.wants/tantor-be-server-18.service.
```

3) Посмотрим список установленного программного обеспечения Tantor:

```
root@tantor:~# apt list | grep tantor | grep installed
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

tantor-be-server-18/1.8_x86-64,now 18.1.0 amd64 [installed]
```

3a) Сохраним директорию Tantor BE:

```
root@tantor:~# cp -a /opt/tantor/db /opt/tantor/db.be
```

4) Деинсталлируем то, что установили:

```
root@tantor:~# apt remove tantor-be-server-18 -y

      Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages will be REMOVED:
 tantor-be-server-18
0 upgraded, 0 newly installed, 1 to remove and 2101 not upgraded.
After this operation, 0 B of additional disk space will be used.
(Reading database ... 320063 files and directories currently installed.)
Removing tantor-be-server-18 (18.1.0) ...
+ echo ----------------------------------------
+ echo 'tantor-be-server-18 is getting removed'
tantor-be-server-18 is getting removed
+ echo ----------------------------------------
+ /bin/systemctl --no-reload disable tantor-be-server-18
+ /bin/systemctl stop tantor-be-server-18
+ echo ----------------------------------------
+ set +vx
```

```
+ echo ----------------------------------------
+ echo 'tantor-be-server-18 is getting removed'
tantor-be-server-18 is getting removed
+ echo ----------------------------------------
+ /usr/sbin/ldconfig
+ /bin/systemctl daemon-reload
+ '[' -f /var/lib/postgresql/.bash_profile ']'
++ grep /opt/tantor/db/18/bin /var/lib/postgresql/.bash_profile
+ '[' '!' -z 'export PATH=/opt/tantor/db/18/bin:$PATH' ']'
+ sed -i 's|/opt/tantor/db/18/bin:*||g' /var/lib/postgresql/.bash_profile
+ sed -i '/^PATH=:*\($PATH\)*:*$/d' /var/lib/postgresql/.bash_profile
+ /usr/sbin/ldconfig
+ echo ----------------------------------------
+ set +vx
```

5) Проверим, как изменился список установленного программного обеспечения Tantor:

```
root@tantor:~# apt list | grep tantor | egrep "(inst|config)"
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


pg-configurator-tantor-all/1.8_x86-64 26.1.21-1astra1.8-1 all
tantor-be-server-18/1.8_x86-64,now 18.1.0 amd64 [residual-config]
tantor-free-server-16/stable 16.6.2-1 amd64
```

Пакет деинсталлирован, но конфигурационные файлы были оставлены.

6) Посмотрим, есть ли ещё пакеты `residual config`:

```
root@tantor:~# aptitude search ~c
c   tantor-be-server-18        - Tantor Basic database server installation
package
```

7) Удалим эти пакеты:

```
root@tantor:~# aptitude -y purge ~c
The following packages will be REMOVED:
 tantor-be-server-18{p}
0 packages upgraded, 0 newly installed, 1 to remove and 1893 not upgraded.
Need to get 0 B of archives. After unpacking 0 B will be used.
(Reading database ... 321507 files and directories currently installed.)
Purging configuration files for tantor-be-server-18 (18.1.0) ...
+ echo ----------------------------------------
+ echo 'tantor-be-server-18 is getting removed'
tantor-be-server-18 is getting removed
+ echo ----------------------------------------
+ /usr/sbin/ldconfig
+ /bin/systemctl daemon-reload
+ '[' -f /var/lib/postgresql/.bash_profile ']'
++ grep /opt/tantor/db/18/bin /var/lib/postgresql/.bash_profile
+ '[' '!' -z '' ']'
+ /usr/sbin/ldconfig
+ echo ----------------------------------------
+ set +vx
dpkg: warning: while removing tantor-be-server-18, directory '/opt/tantor' not
empty so not removed
```

8) Вернем директорию софта Tantor Postgres SE 18 и файл профиля:

```
root@tantor:~# mv /opt/tantor/db.SAV /opt/tantor/db
mv /var/lib/postgresql/bash_profile /var/lib/postgresql/.bash_profile
```

9) Запустим кластер Tantor Postgres SE, который останавливали:

```
root@tantor:~# systemctl start tantor-se-server-18
```

10) Проверим, что кластер работоспособен:

```
root@tantor:~# su - postgres -c "psql --command='select tantor_version();'"
       tantor_version
------------------------------
Tantor Special Edition 18.3.0
(1 row)
```

11) Директория PGDATA кластера Tantor BE не была стёрта:

```
root@tantor:~# root@tantor:~# ls /var/lib/postgresql/tantor-be-18/data
base          pg_hba.conf    pg_notify     pg_stat      pg_twophase  postgresql.auto.conf
global        pg_ident.conf  pg_replslot   pg_stat_tmp  PG_VERSION   postgresql.conf
pg_commit_ts  pg_logical     pg_serial     pg_subtrans  pg_wal       postmaster.opts
pg_dynshmem   pg_multixact   pg_snapshots  pg_tblspc    pg_xact
```

12) Выйдем из root:

```
root@tantor:~# exit
exit
astra@tantor:~$
```

Глава 3. Конфигурирование

Просмотр параметров конфигурации

1) Переключимся в пользователя postgres:

```
astra@tantor:~$ su - postgres
Password: postgres
postgres@tantor:~$
```

2) Проверим стартовый файл psql, команды из которого выполняются при каждом запуске утилиты psql:

```
root@tantor:~# cat /var/lib/postgresql/.psqlrc
\setenv PAGER 'less -XS'
\setenv PSQL_EDITOR '/usr/bin/mcedit'
\pset pager off
```

Если содержимое файла другое, то отредактируйте его.

3) Посмотрим названия столбцов в представлении со списком конфигурационных параметров:

```
postgres@tantor:~$ psql
Pager usage is off.
psql (18.3)
Type "help" for help.

postgres=# \d pg_settings
                    Представление "pg_catalog.pg_settings"
     Столбец      |   Тип    | Правило со
------------------+----------+-----------
 name             | text     |
 setting          | text     |
 unit             | text     |
 category         | text     |
 short_desc       | text     |
 extra_desc       | text     |
 context          | text     |
 vartype          | text     |
 source           | text     |
 min_val          | text     |
 max_val          | text     |
 enumvals         | text[]   |
 boot_val         | text     |
 reset_val        | text     |
 sourcefile       | text     |
 sourceline       | integer  |
 pending_restart  | boolean  |
```

4) Число параметров в текущей версии:

```
postgres=# select count(*) from pg_settings;
 count
-------
   462
```

```
(1 строка)
```

429 или больше параметров, **включая** параметры загруженных библиотек (параметр конфигурации `shared_preload_libraries`) и параметры специфичные для Tantor Postgres SE и SE 1C.

5) Посмотрим, какие типы значений параметров есть:

```
postgres=# select distinct unit, vartype from pg_settings order by unit;
 unit | vartype
------+---------
8kB   | integer
B     | integer
kB    | integer
MB    | integer
min   | integer
ms    | real
ms    | integer
s     | integer
      | integer
      | enum
      | bool
      | real
      | int64
      | string
(14 rows)
```

6) Есть параметры с единицами измерения и без.

Посмотрим сколько есть параметров каждого типа:

```
postgres=# select unit, vartype, count(*) from pg_settings group by unit, vartype
order by 3;
unit | vartype | count
-----+---------+-------
ms   | real    |     2
min  | integer |     3
B    | integer |     5
MB   | integer |     6
     | int64   |     8
s    | integer |    12
kB   | integer |    12
8kB  | integer |    20
ms   | integer |    21
     | real    |    28
     | enum    |    48
     | string  |    74
     | integer |    75
     | bool    |   148
(14 rows)
```

7) Посмотрим, какие два параметра измеряются по умолчанию в долях миллисекунд:

```
postgres=# select name, setting from pg_settings where unit='ms' and
vartype='real';
             name              | setting
-------------------------------+---------
 autovacuum_vacuum_cost_delay | 2
```

```
 vacuum_cost_delay             | 0
(2 строки)
```

Это параметры, настраивающие задержку в работе процессов вакуумирования.

8) Есть параметры типа enum. Посмотрим какие значения бывают для параметров этого типа:

```
postgres=# select distinct enumvals from pg_settings;
                            enumvals
-----------------------------------------------------------
{local,remote_write,remote_apply,on,off}
{md5,scram-sha-256}
{none,pl,all}
{pause,promote,shutdown}
{none,top,all}
{postgres,postgres_verbose,sql_standard,iso_8601}
{sync,worker}
{debug5,debug4,debug3,debug2,debug1, log,notice,warning,error}
{sysv,mmap}
{origin,replica,local}
{always,on,off}
{TLSv1,TLSv1.1,TLSv1.2,TLSv1.3}
{serializable,"repeatable read","read committed","read uncommitted"}
{disabled,debug5,debug4,debug3,debug2, debug1,log,notice,warning,error}
{auto,force_generic_plan,force_custom_plan}
{content,document}
{posix_fallocate,write_zeros}
{pglz,lz4}
{local0,local1,local2,local3,local4, local5,local6,local7}
{none,ddl,mod,all}
{none,top,all,verbose}
{none,cache,snapshot}
{off,on,try}
{"",TLSv1,TLSv1.1,TLSv1.2,TLSv1.3}
{minimal,replica,logical}
{fsync,syncfs}
{auto,regress,on,off}
{copy,clone}
{shmem,file}
{safe_encoding,on,off}
{exhaust,consistency,startup}
{buffered,immediate}
{debug5,debug4,debug3,debug2,debug1 ,info,notice,warning,error, log,fatal,panic}
{terse,default,verbose}
{CE17,CE18}
{off,on,unknown}
{base64,hex}
{posix,sysv,mmap}
{raw,text,json,yaml,xml}
{partition,on,off}
{fsync,fdatasync,open_sync,open_datasync}
{off,on,regress}
{pglz,lz4,zstd,on,off}
{escape,hex}
(45 rows)
```

В основном, используются английские слова, обозначающие названия технологий и алгоритмов. Например, алгоритмов сжатия `pglz,lz4,zstd`.

9) Какие **контексты** параметров есть:

```
postgres=# select distinct context from pg_settings;
```

```
      context
------------------
 postmaster
 superuser-backend
 user
 internal
 backend
 sighup
 superuser
(7 строк)
```

Контекст указывает, можно ли изменить значение параметра, и если можно, то каким образом.

10) Параметры расширений и библиотек имеют в названии точку.

Посмотрим параметр `plpgsql.variable_conflict`:

```
postgres=# show plpgsql.variable_conflict;
ERROR:  unrecognized configuration parameter "plpgsql.variable_conflict"
```

Параметр неизвестен. Неизвестные параметры можно устанавливать в `postgresql.conf`, но не командой `ALTER SYSTEM`.

11) Загрузим библиотеку расширения («модуль»). Апострофы в строковых параметрах обязательны:

```
postgres=# load 'plpgsql';
LOAD
postgres=# show plpgsql.variable_conflict;
 plpgsql.variable_conflict
---------------------------
 error
(1 строка)
```

12) Посмотрим, какие параметры конфигурации были зарегистрированы при загрузке модуля:

```
postgres=# show plpgsql.<TAB><TAB>
plpgsql.check_asserts       plpgsql.extra_errors      plpgsql.extra_warnings
plpgsql.print_strict_params plpgsql.variable_conflict
```

Также можно посмотреть командой:

```
postgres=# \dconfig plpgsql.*
     Список параметров конфигурации
          Параметр            | Значение
------------------------------+----------
 plpgsql.check_asserts        | on
 plpgsql.extra_errors         | none
 plpgsql.extra_warnings       | none
 plpgsql.print_strict_params  | off
 plpgsql.variable_conflict    | error
(5 строк)
```

Глава 4a. Логическая структура кластера

Часть1. Просмотр списка баз данных кластера

1) Запустим стандартно поставляемую утилиту `oid2name`:

```
postgres@tantor:~$ oid2name
All databases:
    Oid  Database Name  Tablespace
----------------------------------
      5       postgres  pg_default
      4      template0  pg_default
      1      template1  pg_default
```

Утилита, запущенная без параметров, выдаёт список баз данных, название табличного пространства по умолчанию для каждой из баз данных, **oid** базы данных, который соответствует поддиректории в директории табличного пространства.

2) Подключимся к экземпляру:

```
postgres@tantor:~$ psql
psql (18.3)
Введите "help", чтобы получить справку.
```

3) Посмотрим список баз командой **\l**:

```
postgres=# \l
                                                    List of databases
   Name    |  Owner   | Encoding | Locale Provider |   Collate   |    Ctype    | ICU Locale | ICU Rules |   Access privileges
-----------+----------+----------+-----------------+-------------+-------------+------------+-----------+-----------------------
 postgres  | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           |
 template0 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +
           |          |          |                 |             |             |            |           | postgres=CTc/postgres
 template1 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +
           |          |          |                 |             |             |            |           | postgres=CTc/postgres
(3 строки)
```

4) Посмотрим, что нам выдаст команда **\l**, если добавить символ "+", означающий дополнительные данные:

```
postgres=# \l+
                                                                    List of databases
   Name    |  Owner   | Encoding | Locale Provider |   Collate   |    Ctype    | ICU Locale | ICU Rules |   Access privileges   |  Size   | Tablespace |                Description
-----------+----------+----------+-----------------+-------------+-------------+------------+-----------+-----------------------+---------+------------+--------------------------------------------
 postgres  | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           |                       | 7511 MB | pg_default | default administrative connection database
 template0 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +| 7353 kB | pg_default | unmodifiable empty database
           |          |          |                 |             |             |            |           | postgres=CTc/postgres |         |            |
 template1 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |            |           | =c/postgres          +| 7583 kB | pg_default | default template for new databases
           |          |          |                 |             |             |            |           | postgres=CTc/postgres |         |            |

(3 rows)
```

Что добавилось?

Добавились столбцы с размером, табличным пространством по умолчанию, описанием.

5) Посмотрим список баз данных командой `SELECT`:

```
postgres=# SELECT datname FROM pg_database;
  datname
-----------
 postgres
 template1
 template0
(4 rows)
```

Часть 2. Создание базы данных

6) Создадим базу данных командой SQL:

```
postgres=# CREATE DATABASE db01;
CREATE DATABASE
```

Показать, что можно стрелкой вверх на клавиатуре повторить предыдущие команды, и убедиться, что новая база данных выводится.

На основе какой шаблонной базы была создана база данных `db01`?

На основе `template1`.

Часть 3. Переименование базы данных

7) Переименуем базу:

```
postgres=# ALTER DATABASE db01 RENAME TO db02;
ALTER DATABASE
```

8) Убедимся, что можем подсоединиться к базе `db02`:

```
postgres=# \c db02
You are now connected to database "db02" as user "postgres".
db02=# \c postgres
You are now connected to database "postgres" as user "postgres".
```

Помните о том, что нажимая клавишу табуляции *<TAB>* можно завершать команды.

Часть 4. Ограничение на соединение с базой

9) Установим максимальное число подсоединений в ноль:

```
postgres=# ALTER DATABASE db02 CONNECTION LIMIT 0;
ALTER DATABASE
```

10) Как пользователь с атрибутом `SUPERUSER` мы можем подсоединиться:

```
postgres=# \c db02
You are now connected to database "db02" as user "postgres".
db02=# \c postgres
You are now connected to database "postgres" as user "postgres".
```

11) Воспользуемся свойством базы данных `ALLOW_CONNECTIONS`:

```
postgres=# ALTER DATABASE db02 ALLOW_CONNECTIONS false;
ALTER DATABASE
postgres=# \c db02
connection to server on socket "/var/run/postgresql/.s.PGSQL.5432" failed: FATAL:
database "db02" is not currently accepting connections
Previous connection kept
```

Теперь нельзя подсоединиться.

Часть 5. Форматирование вывода psql

12) Проверим стартовый файл psql, команды из которого выполняются при каждом запуске утилиты psql:

```
root@tantor:~# cat /var/lib/postgresql/.psqlrc
\setenv PAGER 'less -XS'
\setenv PSQL_EDITOR '/usr/bin/mcedit'
\pset pager off
```

Если содержимое файла другое, то отредактируйте его.

13) В виртуальной машине для курса установили эти параметры. По умолчанию файл отсутствует. Установим значение переменной в значения по умолчанию и посмотрим, как будет выдаваться результат:

```
postgres=# \setenv PAGER 'more'
\pset pager on
Pager is used for long output.
```

14) Посмотрим список встроенных функций, которые полезны для администрирования:

```
postgres=# \dfS pg*
```

Результат нечитаемый. Для выхода из утилиты постраничного вывода more нажмем клавишу q

15) Настроим вывод и повторим:

```
postgres=# \pset format wrapped
postgres=# \dfS pg*
                                List of functions
   Schema   |           Name           | Result data type | Argument data types | Type
------------+--------------------------+------------------+---------------------+------
 pg_catalog | pg_advisory_lock         | void             | bigint              | func
 pg_catalog | pg_advisory_lock         | void             | integer, integer    | func
 pg_catalog | pg_advisory_lock_shared  | void             | bigint              | func
 pg_catalog | pg_advisory_lock_shared  | void             | integer, integer    | func
```

Отображение поменялось. При использовании утилиты more клавишами <PgUp> и <PgDown> нельзя пользоваться.

16) Вернем формат в значение по умолчанию :

```
postgres=# \pset format aligned
Формат вывода: aligned.
```

Вернем переменную, задающую программу постраничного вывода вместо использовавшейся утилиты "more":

```
postgres=# \setenv PAGER 'less -XS'
```

17) Повторим и убедимся, что вывод стал читаемым. Можно использовать клавиши <PgUp> и <PgDown> или z и b:

```
postgres=# \dfS pg*
```

Нажмите клавиши <PgDn>, клавишу <h>: обратите внимание на то, что высветилась подсказка по команде less, прочтите, что для выхода из режима помощи можно нажать клавишу "q" и нажмите ее два раза <q><q>.

```
                    SUMMARY OF LESS COMMANDS

      Commands marked with * may be preceded by a number, N.
      Notes in parentheses indicate the behavior if N is given.
      A key preceded by a caret indicates the Ctrl key; thus ^K is ctrl-K.
```

```
 h  H                 Display this help.
 q  :q  Q  :Q  ZZ     Exit.
---------------------------------------------

                           MOVING

 e  ^E  j  ^N  CR  *  Forward  one line   (or N lines).
 y  ^Y  k  ^K  ^P  *  Backward one line   (or N lines).
 f  ^F  ^V  SPACE  *  Forward  one window (or N lines).
 b  ^B  ESC-v      *  Backward one window (or N lines).
 z                 *  Forward  one window (and set window to N).
 w                 *  Backward one window (and set window to N).
 ESC-SPACE         *  Forward  one window, but don't stop at end-of-file.
 d  ^D             *  Forward  one half-window (and set half-window to N).
 u  ^U             *  Backward one half-window (and set half-window to N).
 ESC-)  RightArrow *  Right one half screen width (or N positions).
 ESC-(  LeftArrow  *  Left  one half screen width (or N positions).
 ESC-}  ^RightArrow   Right to last column displayed.
 ESC-{  ^LeftArrow    Left  to first column.
HELP -- Press RETURN for more, or q when done
```

18) Удалите созданную базу данных:

```
postgres=# drop database db02;
DROP DATABASE
```

Глава 4b. Физическая структура кластера

Часть 1. Директория для временных файлов

1) Запустим стандартно поставляемую утилиту `oid2name`:

```
postgres@tantor:~$ oid2name
All databases:
    Oid  Database Name  Tablespace
----------------------------------
      5       postgres  pg_default
      4      template0  pg_default
      1      template1  pg_default
```

Утилита, запущенная без параметров, выдаёт список баз данных, название табличного пространства по умолчанию, `oid` базы данных, который соответствует поддиректории в директории табличного пространства.

2) Создадим большую временную таблицу:

```
postgres@tantor:~$ time psql -c "create temp table x as select * from generate_series(1, 1000000);"
SELECT 1000000

real    0m0.737s
user    0m0.007s
sys     0m0.003s
```

3) Посмотрим какие директории есть в табличном пространстве `pg_default`:

```
postgres@tantor:~$ ls --color -w 1 $PGDATA/base
1
4
5
pgsql_tmp
```

Зачем нужна директория `pgsql_tmp`?

Это поддиректория для временных файлов, которая создаётся в директории табличного пространства.

Для временных файлов лучше использовать отдельное табличное пространство, которое стоит создать отдельно.

Как устанавливается табличное пространство для временных файлов?

Параметром конфигурации **temp_tablespaces**

Дожидаться ответа слушателей не нужно, достаточно чтобы они задались вопросом в целях запоминания.

4) Перейдите в директорию `$PGDATA` и удобными средствами (`mc`) покажите директории и поддиректории, и дайте короткий обзор того, что хранится в директориях и файлах.

Часть 2. Перемещение директории табличного пространства

1) Создадим табличное пространство. Для этого создадим директорию:

```
postgres@tantor:~$ mkdir $PGDATA/u01
```

2) Проверим, что пользователь `postgres` может читать-писать в неё:

```
postgres@tantor:~$ ls -al $PGDATA/u01
```

```
total 8
drwxr-xr-x  2 postgres postgres 4096  .
drwxr-x--- 20 postgres postgres 4096  ..
```

3) Запустим `psql`:

```
postgres@tantor:~$ psql
psql (18.3)
Введите "help", чтобы получить справку.
```

4) Попытаемся создать табличное пространство:

```
postgres=# CREATE TABLESPACE u01tbs LOCATION 'u01';
ERROR:  tablespace location must be an absolute path
```

Относительный путь не подходит, **нужно указать абсолютный.**

5) Укажем:

```
postgres=# CREATE TABLESPACE u01tbs LOCATION '/var/lib/postgresql/tantor-se-18/data/u01';
WARNING:  tablespace location should not be inside the data directory
CREATE TABLESPACE
```

Табличное пространство создано, но выдано предупреждение, что **не стоит** директорию `u01` располагать в `PGDATA`. Также не стоит располагать и другие директории (**например, логирования**), чтобы они с большим количеством ненужных файлов не попали в бэкап.

6) Создадим в табличном пространстве таблицу:

```
postgres=# CREATE TABLE t (id bigserial, t text) TABLESPACE u01tbs;
CREATE TABLE
```

7) Во втором окне терминала покажите, что появилось три файла, один из них размером 8192 байт, а другие нулевого размера:

Перейдите в директорию табличного пространства и поддиректорию с `oid` базы данных:

```
postgres@tantor:~$ cd $PGDATA/u01/PG_18_642601132/5/
postgres@tantor:~/tantor-se-18/data/u01/PG_18_642601132/5$ ls -l -w 1
итого 8
-rw-r----- 1 postgres postgres    0  16401
-rw-r----- 1 postgres postgres    0  16406
-rw-r----- 1 postgres postgres 8192  16407
```

Что это за файлы?

Это файл основного слоя таблицы `t`, основной слой её `TOAST`-таблицы и `TOAST`-индекса. `TOAST`-таблица и `TOAST`-индекс были созданы автоматически, так как есть столбец типа `text`.

8) Проверим, к чему относится какой файл:

```
postgres@tantor:~/tantor-se-18/data/u01/PG_18_642601132/5$ oid2name -f 16401
From database "postgres":
  Filenode  Table Name
----------------------
    16401           t

postgres@tantor:~/tantor-se-18/data/u01/PG_18_642601132/5$ oid2name -f 16406
From database "postgres":
```

```
  Filenode       Table Name
---------------------------
    16406  pg_toast_16401

postgres@tantor:~/tantor-se-18/data/u01/PG_18_642601132/5$ oid2name -f 16407
From database "postgres":
  Filenode             Table Name
---------------------------------
    16407  pg_toast_16401_index
```

8-килобайтный файл относится к индексу.

9) Перенесем директорию с остановкой экземпляра:

```
postgres@tantor:~/tantor-se-18/data/u01/PG_18_642601132/5$ cd $PGDATA
postgres@tantor:~/tantor-se-18/data$ pg_ctl stop
waiting for server to shut down.... done
server stopped
postgres@tantor:~/tantor-se-18/data$ mv u01 ..
```

10) Посмотрим список символических ссылок на табличные пространства:

```
postgres@tantor:~/tantor-se-18$ ls -al $PGDATA/pg_tblspc
total 8
drwx------  2 postgres postgres 4096 .
drwxr-x--- 19 postgres postgres 4096 ..
lrwxrwxrwx  1 postgres postgres   41 16399 -> /var/lib/postgresql/tantor-se-18/data/u01
```

На директорию `u01` указывает ссылка с названием 16399. В вашем случае название файла ссылки будет другое.

11) Пересоздадим ссылку, чтобы она указывала на уже перемещенную директорию:

```
postgres@tantor:~/tantor-se-18$ ln -fs $PGDATA/../u01 $PGDATA/pg_tblspc/16399
```

12) Убедимся, что символическая ссылка указывает на содержимое директории табличного пространства:

```
postgres@tantor:~/tantor-se-18/data$ ls  $PGDATA/pg_tblspc/16399

PG_18_642601132
```

13) Запустим экземпляр:

```
postgres@tantor:~/tantor-se-18/data$ sudo systemctl start tantor-se-server-18
```

14) Переподсоединимся в окне `psql` и проверим, что содержимое таблицы доступно:

```
postgres@tantor:~/tantor-se-18/data$ psql -c "select * from t;"
Pager usage is off.
id | t
----+---
(0 rows)
```

Директория табличного пространства успешно перенесена.

Глава 7a. Физическое копирование

Изменение размера WAL файлов

1) Корректно остановим экземпляр кластера:

```
postgres@tantor:~$ pg_ctl stop
waiting for server to shut down.... done
server stopped
```

2) Проверим, что остановка выполнена корректно:

```
postgres@tantor:~$ pg_controldata | grep state
postgres@tantor:~$ pg_controldata | grep Состояние
Database cluster state:               shut down
```

Переключение раскладки: справа внизу окна виртуальной машины кликнуть мышкой на Eng.

3) Сохраним значения из управляющего файла для последующего сравнения со значениями, которые изменятся:

```
postgres@tantor:~$ pg_controldata > 16MB.txt
```

4) Меняем размер WAL-сегментов с 16 Мб на 256 Мб:

```
postgres@tantor:~$
 pg_resetwal --wal-segsize=256 /var/lib/postgresql/tantor-se-18/data
Журнал предзаписи сброшен (Write-ahead log reset)
```

5) Сохраним значения из управляющего файла для сравнения:

```
postgres@tantor:~$ pg_controldata > 256MB.txt
```

Сравним:

```
postgres@tantor:~$ diff 16MB.txt 256MB.txt
5,8c5,8
< Последнее обновление pg_control:      02:29:38 PM MSK
< Положение последней конт. точки:      0/1C7A468
< Положение REDO последней конт. точки: 0/1C7A468
---
> Последнее обновление pg_control:      02:34:53 PM MSK
> Положение последней конт. точки:      0/10000028
> Положение REDO последней конт. точки: 0/10000028
23c23
< Время последней контрольной точки:    02:29:38 PM MSK
---
> Время последней контрольной точки:    02:34:53 PM MSK
30c30
< Значение wal_level:                   replica
---
> Значение wal_level:                   minimal
42c42
< Байт в сегменте WAL:                  16777216
---
> Байт в сегменте WAL:                  268435456
```

Пример на английском языке:

```
< pg_control last modified:             12:43:57 AM MSK
< Latest checkpoint location:           0/1C7A468
```

```
< Latest checkpoint's REDO location:    0/1C7A468
---
> pg_control last modified:             12:48:17 AM MSK
> Latest checkpoint location:           0/10000028
> Latest checkpoint's REDO location:    0/10000028
23c23
< Time of latest checkpoint:            12:43:57 AM MSK
---
> Time of latest checkpoint:            12:48:17 AM MSK
30c30
< wal_level setting:                    replica
---
> wal_level setting:                    minimal
42c42
< Bytes per WAL segment:                16777216
---
> Bytes per WAL segment:                268435456
```

Значение **minimal** поменяет своё значение после запуска экземпляра.

6) Попытаемся запустить экземпляр:

```
postgres@tantor:~$ pg_ctl start -c "-c logging_collector=off"
ожидание запуска сервера....
[10094] ВАЖНО:  "min_wal_size" должен быть минимум вдвое больше "wal_segment_size"
[10094] СООБЩЕНИЕ:  система БД выключена
 прекращение ожидания
pg_ctl: не удалось запустить сервер
Изучите протокол выполнения.

waiting for server to start....
[10094] FATAL:  "min_wal_size" must be at least twice "wal_segment_size"
[10094] LOG:  database system is shut down
 stopped waiting
pg_ctl: could not start server
Examine the log output
```

Мы не учли, что от размера WAL-сегментов может что-то зависеть.

7) Установим значение параметра:

```
postgres@tantor:~$ echo "min_wal_size=512MB" >> $PGDATA/postgresql.auto.conf
```

8) Запустим экземпляр:

```
postgres@tantor:~$ pg_ctl start -o "-c logging_collector=off"

waiting for server to start....
[10741] LOG:  database system is ready to accept connections
done
server started
```

Экземпляр запустился.

9) В `psql` переключим файл журнала:

```
postgres@tantor:~$ psql
postgres=# select pg_switch_wal();
 pg_switch_wal
---------------
 115/D000015A
(1 row)

postgres=# select pg_switch_wal();
```

```
 pg_switch_wal
---------------
 115/E000008A
(1 row)

[8505] LOG:  checkpoint starting: wal
```

Теперь после слэша меняется не два символа, а один. Остальные символы укажут на смещение в 256-мегабайтном файле:

```
LOG:  checkpoint complete: wrote 0 buffers (0.0%), wrote 3 SLRU
buffers; 0 WAL file(s) added, 0 removed, 2 recycled; write=0.002 s, sync=0.001 s,
total=0.323 s; sync files=2
, longest=0.001 s, average=0.001 s; distance=524287 kB, estimate=524287 kB;
lsn=0/60000098, redo lsn=0/60000028
```

10) Выйдем из `psql`, остановим кластер и вернем обратно размер журнала:

```
postgres=# \q
postgres@tantor:~$ pg_ctl stop
waiting for server to shut down...
[8504] LOG:  received fast shutdown request
[8504] LOG:  aborting any active transactions
[8504] LOG:  background worker "logical replication launcher" (PID 8510) exited with exit
code 1
[8505] LOG:  shutting down
[8505] LOG:  checkpoint starting: shutdown immediate
[8505] LOG:  checkpoint complete: wrote 0 buffers (0.0%), wrote 0 SLRU buffers; 0
WAL file(s) added, 0 removed, 0 recycled; write=0.001 s, sync=0.001 s, total=0.005 s;
sync files=0, longest=0.000 s, average=0.000 s; distance=0 kB, estimate=471858 kB;
lsn=0/60000178, redo lsn=0/60000178
[8504] LOG:  database system is shut down
done
server stopped
```

11) Проверяем корректность остановки:

```
postgres@tantor:~$ pg_controldata | grep state
postgres@tantor:~$ pg_controldata | grep Состояние
Состояние кластера БД:               выключен
```

12) Меняем размер обратно на 16 Мб:

```
postgres@tantor:~$
pg_resetwal --wal-segsize=16 /var/lib/postgresql/tantor-se-18/data
Журнал предзаписи сброшен
```

13) Запустим экземпляр через службы:

```
postgres@tantor:~$ sudo systemctl start tantor-se-server-18
```

14) Проверяем, как изменилось содержимое выдаваемых LSN (значения LSN даны для примера):

```
postgres@tantor:~$ psql
psql (18.3)
Type "help" for help.

postgres=# select pg_switch_wal();
 pg_switch_wal
---------------
 116/15A
(1 row)
```

```
postgres=# select pg_switch_wal();
 pg_switch_wal
---------------
 116/100008A
(1 row)
```

15) LSN может выводиться коротким, как в данном примере. Почему LSN был с виду «коротким» **116/15A**? И в **116/100008A** после слэша 7 символов, а не 8.

Потому, что название WAL-сегмента приняло значение ноль в конце.

Реальное значение: **116/0000015A** и **116/0100008A**.

```
postgres@tantor:~$ ls $PGDATA/pg_wal
000000010000011600000000  000000010000011600000001  000000010000011600000002
000000010000011600000003  archive_status
```

16) Посмотрим, какие записи есть в файлах журнала (выберите несколько):

```
postgres@tantor:~$ pg_waldump 000000010000011600000000

rmgr: XLOG        len (rec/tot):    148/   148, tx:                  0, lsn: 116/00000028, prev 0/00000000,
desc: CHECKPOINT_SHUTDOWN redo 116/28; tli 1; prev tli 1; fpw true; xid 35741; oid 390998; multi 502936; offset
2034077; oldest xid 723 in DB 1; oldest multi 1 in DB 1; oldest/newest commit timestamp xid: 0/0; oldest running
xid 0; shutdown
rmgr: XLOG        len (rec/tot):     56/    56, tx:                  0, lsn: 116/000000C0, prev 116/00000028,
desc: PARAMETER_CHANGE max_connections=100 max_worker_processes=8 max_wal_senders=10 max_prepared_xacts=0
max_locks_per_xact=64 wal_level=replica wal_log_hints=off track_commit_timestamp=off
rmgr: Standby     len (rec/tot):     68/    68, tx:                  0, lsn: 116/000000F8, prev 116/000000C0,
desc: RUNNING_XACTS nextXid 35741 latestCompletedXid 35740 oldestRunningXid 35741
rmgr: XLOG        len (rec/tot):     26/    26, tx:                  0, lsn: 116/00000140, prev 116/000000F8,
desc: SWITCH

postgres@tantor:~$ pg_waldump 000000010000011600000001

rmgr: Standby     len (rec/tot):     68/    68, tx:                  0, lsn: 116/01000028, prev 116/00000140,
desc: RUNNING_XACTS nextXid 35741 latestCompletedXid 35740 oldestRunningXid 35741
rmgr: XLOG        len (rec/tot):     26/    26, tx:                  0, lsn: 116/01000070, prev 116/01000028,
desc: SWITCH

postgres@tantor:~$ pg_waldump 000000010000011600000002

rmgr: Standby     len (rec/tot):     68/    68, tx:                  0, lsn: 116/02000028, prev 116/01000070,
desc: RUNNING_XACTS nextXid 35741 latestCompletedXid 35740 oldestRunningXid 35741
rmgr: XLOG        len (rec/tot):     26/    26, tx:                  0, lsn: 116/02000070, prev 116/02000028,
desc: SWITCH
```

Текущий файл журнала (**03**):

```
postgres@tantor:~$ pg_waldump  000000010000011600000003

rmgr: Standby     len (rec/tot):     68/    68, tx:                  0, lsn: 116/03000028, prev 116/02000070,
desc: RUNNING_XACTS nextXid 35741 latestCompletedXid 35740 oldestRunningXid 35741
rmgr: Standby     len (rec/tot):     68/    68, tx:                  0, lsn: 116/03000070, prev 116/03000028,
desc: RUNNING_XACTS nextXid 35741 latestCompletedXid 35740 oldestRunningXid 35741
rmgr: XLOG        len (rec/tot):    148/   148, tx:                  0, lsn: 116/030000B8, prev 116/03000070,
desc: CHECKPOINT_ONLINE redo 116/3000070; tli 1; prev tli 1; fpw true; xid 35741; oid 390998; multi 502936;
offset 2034077; oldest xid 723 in DB 1; oldest multi 1 in DB 1; oldest/newest commit timestamp xid: 0/0; oldest
running xid 35741; online
rmgr: Standby     len (rec/tot):     68/    68, tx:                  0, lsn: 116/03000150, prev 116/030000B8,
desc: RUNNING_XACTS nextXid 35741 latestCompletedXid 35740 oldestRunningXid 35741
pg_waldump: error: error in WAL record at 116/3000150: invalid record length at 116/3000198:
expected at least 26, got 0
```

Или на русском языке:

```
pg_waldump: ошибка: ошибка в записи WAL в позиции 116/3000150: неверная длина записи в
позиции 9/A3000198: ожидалось минимум 26, получено 0
```

Глава 7b. Логическое резервирование

Обработка строк большого размера

1) Выполните команды:

```
drop table if exists t2;
create table t2 (c1 text, c2 text);
insert into t2 (c1)
VALUES (repeat('a', 1024*1024*512));
update t2 set c2 = c1;
select * from t2;
```

При выполнении команды `select` появится ошибка:

```
ERROR:  out of memory
DETAIL:  Cannot enlarge string buffer containing 536870922 bytes by 536870912 more bytes.

[31089] ERROR:  out of memory
[31089] DETAIL:  Cannot enlarge string buffer containing 536870922 bytes by 536870912
more bytes.
[31089] STATEMENT:  select * from t2;
```

При выборке в строковый буфер выбиралось значение поля `c1` плюс 10 байт. Для выборки значения второго поля `c2` буфер пытался увеличиться на размер поля `c2`.

2) Попробуем с меньшими полями:

```
drop table if exists t1;
create table t1 (c1 text, c2 text, c3 text, c4 text);
insert into t1 (c1) VALUES (repeat('a', 1024*1024*256));
update t1 SET c2=c1;
update t1 SET c3=c1;
update t1 SET c4=c1;
select * from t1;
```

Появится ошибка:

```
ERROR:  out of memory
DETAIL:  Cannot enlarge string buffer containing 805306386 bytes by 268435456 more bytes.
```

При выборке в строковый буфер выбирались значения полей `c1, c2, c3`. Буфер достиг размера трёх полей плюс 18 байт. При увеличении размера буфера на размер поля `c4`, возникла ошибка превышения границы 1 ГБ.

3) Выполните команду:

```
postgres=# COPY t2 TO '/tmp/test';
ERROR:  string buffer exceeds maximum allowed length (1073741823 bytes)
DETAIL:  Cannot enlarge string buffer containing 536870913 bytes by 536870912 more bytes.
```

В логе кластера будут сообщения:

```
20:17:50.015 MSK [31089] ERROR:  string buffer exceeds maximum allowed length
(1073741823 bytes)
20:17:50.015 MSK [31089] DETAIL:  Cannot enlarge string buffer containing 536870913 bytes
by 536870912 more bytes.
20:17:50.015 MSK [31089] STATEMENT:  COPY t2 TO '/tmp/test';
```

Возникла та же самая ошибка.

4) В Tantor Postgres начиная с версии 16.4 строки большого размера можно выгрузить используя параметр **enable_large_allocations**:

```
postgres=# set enable_large_allocations = on;
SET
postgres=# COPY t2 TO '/tmp/test';
COPY 1
postgres=# \! rm /tmp/test
```

5) Выполните:

```
postgres=# set enable_large_allocations = off;
drop table if exists t2;
create table t2 (c1 text);
insert into t2 (c1) VALUES (repeat(E'a\n', 357913941));
COPY t2 TO '/tmp/test';
```

Появится ошибка:

```
ERROR:  string buffer exceeds maximum allowed length (1073741823 bytes)
DETAIL:  Cannot enlarge string buffer containing 1073741822 bytes by 1 more bytes.
```

Было превышено на 1 байт ограничение на память строкового буфера.

В логе кластера будут сообщения:

```
20:23:51.783 MSK [31089] ERROR:  string buffer exceeds maximum allowed length (1073741823
bytes)
20:23:51.783 MSK [31089] DETAIL:  Cannot enlarge string buffer containing 1073741822
bytes by 1 more bytes.
20:23:51.783 MSK [31089] STATEMENT:  COPY t2 TO '/tmp/test';
```

Размер поля - треть гигабайта с округлением в меньшую сторону.

При выгрузке в текстовом виде содержимое поля будет выглядеть так:

`a\na\na\n` и размер поля увеличится в три раза до 1073741823 байт, что на 1 байт превышает максимальную границу.

6) При использовании формата `binary` поле можно выгрузить:

```
postgres=# COPY t2 TO '/tmp/test' WITH BINARY;
\! rm /tmp/test
COPY 1
```

7) удалите таблицы:

```
postgres=# drop table t1;
postgres=# drop table t2;
```

Примечание:

Если на виртуальной машине не хватает физической памяти для выделения буфера обработки строк, то экземпляр может аварийно остановиться.

Следующий пример опционален и его можно не выполнять:

```
postgres=# drop table if exists t2;
create table t2 (c1 text, c2 text);
insert into t2 (c1) values (repeat('a', 1024*1024*1024-69));
```

В процессе выполнения команды `insert`, если успеть, то можно во втором окне показать, как менялся объем памяти:

```
postgres@tantor:~$ free -b -w
```

```
postgres@tantor:~/tantor-se-18/data/base/5$ free -b -w
               total        used        free      shared     buffers       cache   available
Mem:     8325275648  3656470528  2537848832  1463402496    77914112  2053042176  2886438912
Swap:             0           0           0
               total        used        free      shared     buffers       cache   available
Mem:     8325275648  5789761536   412213248  1463402496    80195584  2043105280   761610240
Swap:             0           0           0
```

Использование памяти увеличилось примерно на 2 ГБ (2125635584 байт). Свободной памяти в примере осталось 400 Мб.

```
postgres=# update t2 set c2 = c1;
select * from t2;
```

psql может быть убит процессом oom-kill, тогда окно терминала закроется. Если не закрылось, то будут сообщения:

```
сервер неожиданно закрыл соединение
        Скорее всего сервер прекратил работу из-за сбоя
        до или в процессе выполнения запроса.
Подключение к серверу потеряно. Попытка восстановления неудачна.
Подключение к серверу потеряно. Попытка восстановления неудачна.
!?> \q
```

```
postgres@tantor:~$ psql
psql (18.3)
Введите "help", чтобы получить справку.
```

Если psql не подключается, то oom-kill убил процесс postgres. Запустите экземпляр (если терминал закрылся, то откройте новый терминал и переключитесь в пользователя postgres):

```
postgres@tantor:~$ sudo systemctl start tantor-se-server-18
```

Служба может запускаться некоторое время, выполняя восстановление кластера.
Удалите таблицу:

```
postgres@tantor:~$ psql
```

```
postgres=# drop table t2;
DROP TABLE
```

Такая ошибка возникнет при нехватке физической памяти. Серверный процесс пытается выделить чуть меньше 4 ГБ памяти, а свободной памяти в данном примере 2.5 ГБ. **oom-kill** (out of memory killer) убил серверный процесс. Процесс `postgres` остановил все процессы и запустил фоновые процессы.

Сообщение в журнале операционной системы:

```
postgres@tantor:~$ sudo dmesg
[ 9952.535780] oom-kill: constraint=CONSTRAINT_NONE,nodemask=(null),
cpuset=user.slice, mems_allowed=0,global_oom
,task_memcg=/system.slice/tantor-se-server-
18.service,task=postgres,pid=10891,uid=113
[ 9952.535807] Out of memory: Killed process 10891 (postgres) total-vm:3470568kB,
anon-rss:2535552kB, file-rss:1280kB, shmem-rss:37248kB, UID:113 pgtables:5208kB
oom_score_adj:0
[ 9952.576141] systemd-journald[239]: /dev/kmsg buffer overrun, some messages
lost.
```

В примере `oom-kill` посылает сигнал 9 (`SIGKILL`) серверному процессу, но он может послать этот сигнал и другим процессам, которые выделили много памяти. Процесс `postgres` останавливает все процессы и снова запускает процессы, как при запуске экземпляра.

Сообщения в логе кластера:

```
[31030] LOG:  server process (PID 31038) was terminated by signal 9: Killed
[31030] DETAIL:  Failed process was running: COPY t1 TO '/tmp/test' WITH BINARY;
[31030] LOG:  terminating any other active server processes
[31030] LOG:  all server processes terminated; reinitializing
[31039] LOG:  database system was interrupted; last known up at 19:58:59 MSK
[31042] FATAL:  the database system is in recovery mode
Failed.
[31039] LOG:  database system was not properly shut down; automatic recovery in progress
[31039] LOG:  redo starts at 116/CE344C0
[31039] LOG:  invalid record length at 116/DF34798: expected at least 26, got 0
[31039] LOG:  redo done at 116/DF34770 system usage: CPU: user: 0.02 s, system: 0.12 s, elapsed:
0.15 s
[31040] LOG:  checkpoint starting: end-of-recovery immediate wait
[31040] LOG:  checkpoint complete: wrote 2105 buffers (12.8%); 0 WAL file(s) added, 0 removed, 0
recycled; write=0.025 s, sync=0.003 s, total=0.031 s; sync files=25, longest=0.001 s, average=0.001
s; distance=17408 kB, estimate=17408 kB; lsn=116/DF34798, redo lsn=116/DF34798

        [31030] LOG:  database system is ready to accept connections
```

Глава 8a. Физическая репликация

До выполнения демонстрации проверьте, есть ли табличные пространства:

```
postgres=# \db
                          List of tablespaces
    Name    |  Owner   |                    Location
------------+----------+-----------------------------------------------
 pg_default | postgres |
 pg_global  | postgres |
 u01tbs     | postgres | /var/lib/postgresql/tantor-se-18/data/../u01
(3 rows)
```

Если есть созданные ранее табличные пространства, то удалите их. Если табличное пространство не содержит объектов, то оно удалится командой:

```
postgres=# drop tablespace u01tbs;
DROP TABLESPACE
```

Если не удалится, так как есть объекты, то список отношений в текущей базе данных можно получить командой:

```
ppostgres=# SELECT n.nspname, relname
FROM pg_class c
 LEFT JOIN pg_namespace n ON n.oid = c.relnamespace,
 pg_tablespace t
WHERE relkind IN ('r','m','i','S','t') AND
 n.nspname <> 'pg_toast' AND t.oid = reltablespace AND
 t.spcname = 'u01tbs';
nspname | relname
---------+---------
public  | t
(1 row)
```

Удалить объекты и потом удалить табличное пространство:

```
postgres=# drop table t;
DROP TABLE
postgres=# drop tablespace u01tbs;
DROP TABLESPACE
```

Создание физической реплики

1) Сделаем бэкап с параметрами

-P или **--progress** - показывает прогресс резервирования;

-S или **--slot** - создает слот;

-c fast или **--checkpoint=fast** - вызывает контрольную точку;

-R или **--write-recovery-conf** - создает файлы конфигурации для реплики:

```
postgres@tantor:~$ pg_basebackup -D /var/lib/postgresql/tantor-se-18-
replica/data1 -P -R -C -c fast -S replica1
48902/48902 kB (100%), 1/1 tablespace
```

Если резервирование прервать, то нужно будет удалить директорию:

```
postgres@tantor:~$ rm -rf /var/lib/postgresql/tantor-se-18-replica/data1
```

И слот на мастере:

```
postgres@tantor:~$ psql -c "select pg_drop_replication_slot('replica1');"

pg_drop_replication_slot
--------------------------


(1 row)
```

2) После успешного создания бэкапа нужно установить порт для экземпляра реплики.

Обязательно использовать две угловые скобки, если будет одна, то файл затрётся:

```
postgres@tantor:~$ echo "port=5433" >> /var/lib/postgresql/tantor-se-18-replica/data1/postgresql.auto.conf
```

3) Можно запустить реплику:

```
postgres@tantor:~$ pg_ctl start -D /var/lib/postgresql/tantor-se-18-replica/data1

waiting for server to start...
[56348] LOG:  Auto detecting pg_stat_kcache.linux_hz parameter...
[56348] LOG:  pg_stat_kcache.linux_hz is set to 250000
[56348] LOG:  redirecting log output to logging collector process
[56348] HINT:  Future log output will appear in directory "log".
```

4) **На мастере** посмотрим, что физический слот репликации создан и активен:

```
postgres@tantor:~$ psql

postgres=# select * from pg_replication_slots \gx
-[ RECORD 1 ]-------+-----------
slot_name           | replica1
plugin              |
slot_type           | physical
datoid              |
database            |
temporary           | f
active              | t
active_pid          | 56358
xmin                |
catalog_xmin        |
restart_lsn         | 0/BA002080
confirmed_flush_lsn |
wal_status          | reserved
safe_wal_size       |
two_phase           | f
two_phase_at        |
inactive_since      |
conflicting         |
invalidation_reason |
failover            | f
synced              | f
```

5) Посмотрим ещё одно представление для мониторинга репликации:

```
postgres=# select * from pg_stat_replication \gx
-[ RECORD 1 ]----+------------------------------
pid              | 56358
usesysid         | 10
usename          | postgres
application_name | walreceiver
client_addr      |
```

```
client_hostname  |
client_port      | -1
backend_start    | 2026-04-25 08:38:54.737172+03
backend_xmin     |
state            | streaming
sent_lsn         | 0/BA0021D0
write_lsn        | 0/BA0021D0
flush_lsn        | 0/BA0021D0
replay_lsn       | 0/BA0021D0
write_lag        |
flush_lag        |
replay_lag       |
sync_priority    | 0
sync_state       | async
reply_time       | 2026-04-25 08:45:11.020033+03
```

Имя приложения по умолчанию **walreceiver**.

6) Подключимся к реплике:

```
postgres=# \q
postgres@tantor:~$ psql -p 5433
```

7) Проверим название слота:

```
postgres=# \dconfig primary_slot_name
List of configuration parameters
     Parameter      |  Value
--------------------+----------
 primary_slot_name | replica1
(1 строка)
```

8) Посмотрим значение параметра `cluster_name`:

```
postgres=# \dconfig cluster_name
List of configuration parameters
  Parameter    | Value
---------------+-------
 cluster_name |
```

Значение параметра пусто, поэтому **application_name=walreceiver**

9) Посмотрим значение параметра **primary_conninfo**:

```
postgres=# show primary_conninfo;
       primary_conninfo
------------------------------
user=postgres passfile='/var/lib/postgresql/.pgpass' channel_binding=prefer
port=5432 sslmode=prefer sslcompression=0 sslcertmode=allow sslsni=1
ssl_min_protocol_version=TLSv1.2 gssencmode=prefer krbsrvname=postgres
gssdelegation=0 compression=off target_session_attrs=any
load_balance_hosts=disable
```

Значение было сгенерировано автоматически утилитой `pg_basebackup` при использовании параметра **-R** на основе параметров, с которыми утилита подсоединялась к экземпляру, с которого создавала бэкап.

10) Удалим реплику и слот репликации:

```
postgres=# \q
postgres@tantor:~$ pg_ctl stop -D /var/lib/postgresql/tantor-se-18-replica/data1
waiting for server to shut down....done
server stopped
postgres@tantor:~$ rm -rf /var/lib/postgresql/tantor-se-18-replica/data1
postgres@tantor:~$ psql -c "select pg_drop_replication_slot('replica1')"
 pg_drop_replication_slot
--------------------------

(1 строка)
```

Глава 8b. Логическая репликация

Часть 1. Однонаправленная репликация

1) Подсоединимся к базе данных мастера и создадим таблицу, которую будем реплицировать:

```
postgres@tantor:~$ psql
postgres=# create table t (t text);
CREATE TABLE
```

2) Посмотрим список таблиц, для которых не задан способ идентификации строк:

```
postgres=#  SELECT relnamespace::regnamespace||'.'||relname "table"
 FROM pg_class
 WHERE relreplident IN ('d','n') -- d первичный ключ, n никакие
  AND relkind IN ('r','p') -- r таблица, p секционированная
  AND oid NOT IN (SELECT indrelid FROM pg_index WHERE indisprimary)
  AND relnamespace <> 'pg_catalog'::regnamespace
  AND relnamespace <> 'information_schema'::regnamespace
ORDER BY 1;

  table
----------
public.t
public.t2
(2 rows)
```

По этим таблицам могут реплицироваться только вставки строк (INSERT) и TRUNCATE.

3) Установим параметр конфигурации wal_level=logical. Изменение параметра требует перезапуск экземпляра:

```
postgres=# alter system set wal_level=logical;
ALTER SYSTEM
postgres=# \q
postgres@tantor:~$ pg_ctl stop -D /var/lib/postgresql/tantor-se-18/data
postgres@tantor:~$ sudo systemctl start tantor-se-server-18
```

4) Создадим публикацию:

```
postgres@tantor:~$ psql
postgres=# create publication t for table t with (publish= 'insert,truncate');
```

Репликация UPDATE и DELETE рассматривается в практике.

5) Создадим определение таблицы t в какой-нибудь базе данных этого же кластера:

```
postgres=# create database test_db;
CREATE DATABASE
postgres=# \! pg_dump -t t --schema-only | psql -d test_db
```

Список баз можно посмотреть командой \l

6) Создадим слот логической репликации в базе источника:

```
postgres=# select pg_create_logical_replication_slot('s','pgoutput');
 pg_create_logical_replication_slot
------------------------------------
 (s,9/BC0739E8)
(1 строка)
```

7) В базе-приёмнике создадим подписку и укажем имя слота:

```
postgres=# \c test_db
You are now connected to database "test_db" as user "postgres".
test_db=# create subscription t connection 'dbname=postgres user=postgres'
publication t with (origin=none, create_slot=false, slot_name=s);
CREATE SUBSCRIPTION
```

Слот создали отдельно, потому что, если публикация и подписка в одном и том же кластере, то создание подписки подвиснет на создании слота. Можно создать репликацию даже между таблицами той же самой базы данных, но в разных схемах, так как имена таблиц должны быть одинаковыми.

8) Проверим, что вставка строк из одной базы в другую реплицируется:

```
test_db=# \c postgres
You are now connected to database "postgres" as user "postgres".
postgres=# insert into t values ('a');
```

9) Проверим, что строка реплицировалась:

```
postgres=# \! psql -d test_db -c "select * from t;"
 t
---
 a
(1 row)
```

10) Аналогично проверим, что реплицируется команда `TRUNCATE`:

```
postgres=# truncate t;
TRUNCATE TABLE

postgres=# \c test_db
You are now connected to database "test_db" as user "postgres".
test_db=# select * from t;
 t
---
(0 rows)
```

Часть 2. Двунаправленная репликация

1) Создадим репликацию в обратном направлении с зеркальными настройками.

Обратите внимание, что имя слота должно быть уникальным в конфигурации:

```
test_db=# select pg_create_logical_replication_slot('reverses','pgoutput');
 pg_create_logical_replication_slot
------------------------------------
 (reverses,9/BC0817D8)
(1 строка)

test_db=# create publication t for table t with (publish= 'insert,truncate');
CREATE PUBLICATION
```

2) Создадим подписку:

```
test_db=# \c postgres
You are now connected to database "postgres" as user "postgres".
postgres=# create subscription t connection 'dbname=test_db user=postgres'
publication t with (origin=none, create_slot=false, slot_name=reverses);
WARNING:  subscription "t" requested copy_data with origin = NONE but might copy
data that had a different ori
gin
DETAIL:  The subscription being created subscribes to a publication ("t") that
contains tables that are writte
n to by other subscriptions.
HINT:  Verify that initial data copied from the publisher tables did not come
from other origins.
CREATE SUBSCRIPTION
```

Предупреждение говорит о том, что при создании подписки данные будут скопированы из таблиц публикующей базы данных. Если в таблицах подписчика уже есть эти строки и строки синхронизированы, то стоило бы создавать подписку с параметром `copy_data=off`.

В обеих таблицах нет ни одной строки, поэтому нет разницы.

Использование параметра `copy_data=off` рассматривается в практике.

3) Проверим, что репликация работает в обе стороны:

```
postgres=# insert into t values ('a');
INSERT 0 1
postgres=# select * from t;
 t
---
 a
(1 row)
postgres=# \c test_db
You are now connected to database "test_db" as user "postgres".
test_db=# select * from t;
 t
---
 a
(1 row)
test_db=# insert into t values ('b');
INSERT 0 1
test_db=# \c postgres
You are now connected to database "postgres" as user "postgres".
postgres=# select * from t;
 t
---
 a
 b
(2 rows)
```

4) Удалим все строки:

```
postgres=# delete from t;
DELETE 2
```

5) Удаление не реплицируется потому, что в публикации указали **publish='insert,truncate'**

```
postgres=# \! psql -d test_db -c "select * from t;"
 t
---
 a
 b
(2 rows)
```

6) Вставим строку:

```
postgres=# insert into t values ('a');
INSERT 0 1
```

7) Проверим, что строка вставилась:

```
postgres=# select * from t;
 t
---
 a
(1 строка)
```

8) Проверим, какие строки есть в таблице второй базе:

```
postgres=# \! psql -d test_db -c "select * from t;"
 t
---
 a
 b
 a
(3 rows)
```

Возникла рассинхронизация. Строки на второй таблице не удаляются, но при этом новые строки вставляются. На первой таблице удалили 2 строки, потом вставили одну и получилась одна строка. На второй таблице две строки осталось, и добавилась еще одна строка, получилось три строки.

9) Удалим объекты:

```
postgres=# drop subscription t;
NOTICE:  dropped replication slot "reverses" on publisher
DROP SUBSCRIPTION
postgres=# drop publication t;
DROP PUBLICATION
postgres=# drop table t;
DROP TABLE
postgres=# \c test_db
You are now connected to database "test_db" as user "postgres".
test_db=# drop publication t;
DROP PUBLICATION
test_db=# drop subscription t;
NOTICE:  dropped replication slot "s" on publisher
DROP SUBSCRIPTION
test_db=# drop table t;
DROP TABLE
```