2.2

Создаем стенд

Текущий стенд состоит из 1-ого узла: `vs01.sXXXXXX`, (XXXXXX - ваш номер студента).
Работа стенда 6 часов. 2 попытки запуска стенда.

1. Над текстом нажмите кнопку "Создать стенд". Каждый стенд создан под определенную тему курса. Сейчас вы находитесь в теме - **2. Логическая резервная копия** , этот стенд не подойдет под практические занятия из других пунктов курса.

Запуск обычно идёт до 10 минут, в редких случаях - до 30 минут.

2. После создания стенда вам нужно зайти по SSH на **sbox.slurm.io** с логином и паролем из личного кабинета (https://edu.slurm.io/):

```
ssh s000001@sbox.slurm.io
```

s000001 нужно заменить на ваш номер студента

Далее вам нужно перейти на сам стенд (sbox является просто jump-хостом и не понадобится кроме как для входа):

```
ssh vs01.s000001
```

s000001 снова нужно заменить на ваш личный номер студента

На хосте vs01 у вас есть полные права (включая беспарольный sudo):

```
sudo -i
```

Будьте осторожны и не сломайте стенд до прохождения всех заданий :)

Практика: Логическая резервная копия

Можете запустить скрипт на стенде `~/practice/lecture1/1.logical_backups.sh`, который запускал спикер в видео.

Или пройтись по шагам текущей практике
- https://gitlab.slurm.io/postgres/slurm_course/-

/blob/main/practice/lecture1/1.logical_backups.md, также представлена она далее ниже по тексту.

Инициализация кластера PostgreSQL

1. Мы зашли на стенд. Теперь все действия с СУБД PostgreSQL мы будем выполнять от локального пользователя postgres. Для этого выполним команду

```
student$ sudo -u postgres -i
```

2. Перейдем в домашний каталог. Если этого не сделать, то будут постоянно появляться ошибки, связанные с отсутствием прав на домашнюю папку student.

```
postgres$ cd ~
```

3. Проверяем установленные пакеты.

```
postgres$ yum list installed | grep postgres
```

Должен быть примерно такой результат

```
postgresql12.x86_64              12.6-1PGDG.rhel7              @pgdg12
postgresql12-contrib.x86_64      12.6-1PGDG.rhel7              @pgdg12
postgresql12-docs.x86_64         12.6-1PGDG.rhel7              @pgdg12
postgresql12-libs.x86_64         12.6-1PGDG.rhel7              @pgdg12
postgresql12-pglogical.x86_64    2.3.3-1.el7                   @barman
postgresql12-server.x86_64       12.6-1PGDG.rhel7              @pgdg12
```

4. Инициализируем кластер. Мы это делаем специально для того, чтобы потом было удобно разворачивать другие экземпляры кластера на других портах для прохождения практики. К тому же нам нужно включить контрольные суммы. Это важный параметр, о нем мы будем говорить чуть позже.

```
postgres$ pg_ctl initdb "-D" "/var/lib/pgsql/12/main5432" -o "--data-checksums"
```

Вывод будет примерно таким:

```
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.

creating directory /var/lib/pgsql/12/main5432 ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... UTC
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
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:

    /usr/pgsql-12/bin/pg_ctl -D /var/lib/pgsql/12/main5432 -l logfile start
```

5. Последняя команда показывает, как надо запустить кластер. Меняем немного команду, для нашего удобства. Путь до файлов кластера баз данных должен быть заканчиваться на `main5432`, номер порта на котором будет запущен кластер. И флаг `-l logfile` нам не нужен, будем писать в стандартный файл логов postgres-а, путь до которого можно посмотреть в конфигурационном файле `postgresql.conf`.

Запускаем кластер

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 start
```

И в результате

```
waiting for server to start....2021-03-30 10:41:21.319 UTC [18476] LOG:  starting
PostgreSQL 12.6 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat
4.8.5-44), 64-bit
2021-03-30 10:41:21.319 UTC [18476] LOG:  listening on IPv4 address "127.0.0.1", port
5432
2021-03-30 10:41:21.322 UTC [18476] LOG:  listening on Unix socket
"/var/run/postgresql/.s.PGSQL.5432"
2021-03-30 10:41:21.330 UTC [18476] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.5432"
2021-03-30 10:41:21.343 UTC [18476] LOG:  redirecting log output to logging collector
process
2021-03-30 10:41:21.343 UTC [18476] HINT:  Future log output will appear in directory
"log".
 done
server started
```

6. Проверяем состояние базы данных

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 status
pg_ctl: server is running (PID: 18476)
/usr/pgsql-12/bin/postgres "-D" "/var/lib/pgsql/12/main5432"
```

7. Инициализируем еще один кластер, но уже на порту 5433

```
postgres$ pg_ctl initdb "-D" "/var/lib/pgsql/12/main5433" -o "--data-checksums"
```

8. Так как на порту 5432 у нас уже есть кластер PostgreSQL, то меняем порт с 5432 на порт 5433

```
postgres$ sed -i 's/#port = 5432/port = 5433/'
/var/lib/pgsql/12/main5433/postgresql.conf
```

9. Запускаем кластер

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5433 start
pg_ctl: server is running (PID: 18527)
/usr/pgsql-12/bin/postgres "-D" "/var/lib/pgsql/12/main5433"
```

10. Проверяем состояние базы данных

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5433/ status
```

11. Создаем базу данных

```
postgres$ psql
5432=> CREATE DATABASE dbcopy1;
```

```
5432=> \c dbcopy1
You are now connected to database "dbcopy1" as user "postgres".
```

12. Создадим таблицу test и внесем несколько записей

```
5432=> CREATE TABLE test(id bigserial PRIMARY KEY, name TEXT);
5432=> INSERT INTO test(name) VALUES (null), ('');
5432=> INSERT INTO test(name) SELECT 'Hello, world!';
```

Каждая таблица создает файлы на диске. Сколько создалось файлов данной командой?

13. Посмотрим наши записи

```
5432=> SELECT * FROM test;
id |     name
----+--------------
  1 | Hello, world!
  2 |
  3 |
(3 rows)
```

14. Их можно также выдать на экран с помощью команды copy

```
5432=> COPY test TO STDOUT;
1    Hello, world!
2
3    \N
```

Можно заметить отличие в выводе пустых и неопределенных значений.

15. Также в команде COPY можно настраивать формат вывода. Например, изменить разделитель, представление неопределенных значений и настраивать другие параметры вывода/ввода:

```
5432=> COPY test TO STDOUT WITH (NULL '<nil>', DELIMITER ';');
1;Hello, world!
2;
3;<nil>
```

16. Но, наверное, самое главное свойство команды COPY – это указывать вместо наименования таблицы произвольный запрос. Как на примере ниже:

```
5432=> COPY (SELECT * FROM test WHERE name='Hello, world!') TO STDOUT;
1    Hello, world!
```

Часто это свойство используют, когда нужно перекинуть данные между разными базами и кластерами БД.

17. Команда COPY позволяет выводить данные в формате CSV (FORMAT CSV) или в бинарном виде (FORMAT binary)

```
5432=> COPY test TO STDOUT WITH (FORMAT CSV);
1,"Hello, world!"
2,""
3,
```

Утилита pg_dump

18. Если запускать утилиту pg_dump без дополнительных параметров, то выдаются команды SQL, которые создают базу данных и заполняют ее данными:

```
postgres$ pg_dump -d dbcopy1
```

Примерный вывод

```
--
-- PostgreSQL database dump
--

-- Dumped from database version 12.6 (Ubuntu 12.6-1.pgdg18.04+1)
-- Dumped by pg_dump version 12.6 (Ubuntu 12.6-1.pgdg18.04+1)

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

SET default_tablespace = '';

SET default_table_access_method = heap;

--
-- Name: test; Type: TABLE; Schema: public; Owner: student
--

CREATE TABLE public.test (
    id bigint NOT NULL,
    name text
);


ALTER TABLE public.test OWNER TO student;

--
-- Name: test_id_seq; Type: SEQUENCE; Schema: public; Owner: student
--

CREATE SEQUENCE public.test_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER TABLE public.test_id_seq OWNER TO student;

--
-- Name: test_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: student
--

ALTER SEQUENCE public.test_id_seq OWNED BY public.test.id;


--
-- Name: test id; Type: DEFAULT; Schema: public; Owner: student
--
```

```
ALTER TABLE ONLY public.test ALTER COLUMN id SET DEFAULT
nextval('public.test_id_seq'::regclass);


--
-- Data for Name: test; Type: TABLE DATA; Schema: public; Owner: student
--

COPY public.test (id, name) FROM stdin;
1    Hello, world!
2
3    \N
\.


--
-- Name: test_id_seq; Type: SEQUENCE SET; Schema: public; Owner: student
--

SELECT pg_catalog.setval('public.test_id_seq', 3, true);


--
-- Name: test test_pkey; Type: CONSTRAINT; Schema: public; Owner: student
--

ALTER TABLE ONLY public.test
    ADD CONSTRAINT test_pkey PRIMARY KEY (id);


--
-- PostgreSQL database dump complete
--
```

Можно увидеть, что pg_dump содержит скрипт создания таблицы test и скрипт команды copy для заполнения таблицы данными. Утилита pg_dump позволяет заменить команду copy командой insert. Для этого нужно воспользоваться флагом `--column-inserts`

19. Важный момент: в выгрузку попадают и изменения, сделанные в шаблонной БД template1. Поэтому восстанавливать резервную копию лучше на базе данных, созданной из template0. При использовании ключа `--create` это учитывается автоматически:

```
postgres$ pg_dump --create -d dbcopy1 | grep 'CREATE DATABASE'
CREATE DATABASE dbcopy1 WITH TEMPLATE = template0 ENCODING = 'UTF8' LOCALE =
'en_US.utf8';
```

20. У pg_dump есть много различных параметров. Мы не будем перечислять их тут. Для этого можно обратиться к документации postgresql. Укажем только два, так как она могут быть нам полезны.

Ключ для выбора объектов, которые должны попасть в резервную копию:
`-n, --schema` - шаблон для имен схем;
`-t, --table` - шаблон для имен таблиц.
Восстановим таблицу `test` в другой базе данных на другом сервере.

```
postgres$ psql -p 5433
5433$ CREATE DATABASE dbcopy2;
CREATE DATABASE
```

21. Завершаем сеанс соединения с БД

```
5433$ \q
```

22. Делаем копию таблицы и переносим ее на другой кластер 5433

```
postgres$ pg_dump --table=test -d dbcopy1 | psql -p 5433 -d dbcopy2
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
ALTER TABLE
COPY 3
 setval
--------
      3
(1 row)

ALTER TABLE
```

23. Подключаемся к кластеру на порту 5433

```
postgres$ psql -p 5433 -d dbcopy2
5433=> SELECT * FROM test;
id |     name
----+---------------
  1 | Hello, world!
  2 |
  3 |
(3 rows)
```

Утилита pg_dump - формат custom

24. Одним из серьезных ограничений формата plain заключается в том, что выбирать объекты нужно в момент создания резервной копии. Нет возможности восстановить избранные объекты из резервной копии. Однако в формате custom отсутствует этот недостаток, что позволяет сначала сделать полную копию, а потом выбирать объекты при загрузке.

```
postgres$ pg_dump --format=custom -d dbcopy1 -f ./dbcopy1.custom
```

25. Для того, чтобы восстановить объекты из резервной копии, у нас есть утилита pg_restore. Воспользуемся ей для восстановления таблицы test:

```
postgres$ psql -p 5433 -d dbcopy2
5433=> DROP TABLE test;
DROP TABLE

postgres$ pg_restore --table=test -p 5433 -d dbcopy2 ./dbcopy1.custom
```

Кроме того, утилита pg_restore позволяет восстановить не только конкретную таблицу, но и индекс, функцию или триггер.

26. Проверяем, что данные восстановились

```
postgres$ psql -p5433 -d dbcopy2
5433=> SELECT * FROM test;
 id |      s
----+-------------
  1 | Hello, world!
  2 |
  3 |
(3 rows)
```

27. Восстановим целиком исходную базу данных dbcopy1 на сервере, который находится на порту 5433. Заметим, что новую резервную копию мы не делаем, используем старый файл копии.

```
postgres$ pg_restore --create -p 5433 -d postgres ./dbcopy1.custom
```

28. Проверим:

```
postgres$ psql -p5433 -l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 dbcopy1   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 dbcopy2   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | postgres=CTc/postgres+
           |          |          |             |             | =c/postgres
(5 rows)
```

Утилита pg_dump - формат directory

29. Прежде чем перейдем к данному формату давайте вначале создадим еще одну таблицу на сервере 5432 в новой схеме.

```
postgres$ psql -d dbcopy1
5432=> CREATE SCHEMA dump;
5432=> CREATE TABLE dump.direct(id int, number int);
```

А сколько тут создалось файлов на диске?

```
5432=> INSERT INTO dump.direct(id, number) select id, random()*1000 from
generate_series(1,10) id;
5432=> \q
```

30. Формат directory интересен тем, что позволяет выгружать данные в несколько параллельных потоков.

```
postgres$ pg_dump --format=directory --jobs=2 -d dbcopy1 -f ./dbcopy1.directory
```

31. Заглянем внутрь каталога:

```
postgres$ ls -l ./dbcopy1.directory
total 20
drwx------ 2 postgres postgres 4096 Mar 30 11:49 ./
drwx------ 8 postgres postgres 4096 Mar 30 11:49 ../
-rw-r--r-- 1 postgres postgres   51 Mar 30 11:49 3692.dat.gz
-rw-r--r-- 1 postgres postgres 2138 Mar 30 11:49 3693.dat.gz
-rw-r--r-- 1 postgres postgres 3031 Mar 30 11:49 toc.dat
```

32. В нем находится файл оглавления и по одному файлу на каждый выгружаемый объект (у нас он всего один):

```
postgres$ zcat ./dbcopy1.directory/3693.dat.gz
1    871
2    824
3    909
4    446
5    475
6    400
7    7
8    45
9    41
10    360
\.
```

33. Восстановление из резервной копии:

```
postgres$ pg_restore --clean --create --jobs=2 -p 5433 -d postgres
./dbcopy1.directory
```

В данной команде мы добавили ключ `--clean`, который генерирует команду удаления БД, и команду `--create`, которая создает нашу базу данных dbcopy1

Утилита pg_dump - формат tar

34. В данном случае формат tar мы не рассматриваем в связи с тем, что он очень похож на формат plain.

Утилита pg_dumpall

35. Как уже было сказано выше, утилита pg_dump годится для выгрузки одной базы данных, но никогда не выгружает общие объекты кластера БД, такие, как роли и табличные пространства. Чтобы сделать полную копию кластера, нужна утилита pg_dumpall.

Давайте в таблицу test сервера 5432 еще внесем несколько записей, чтобы проверить работу утилиты pg_dumpall.

```
postgres$ psql -d dbcopy1
5432=> INSERT INTO test(name) VALUES('Новая запись'), ('Самая новая запись');
```

36. Проверяем

```
5432=> select * from test;
id  |        name
----+--------------------
  1 |
```

```
  2 |
  3 | Hello, world!!!
  4 | Новая запись
  5 | Самая новая запись
(5 rows)
```

37. Восстановление выполняется с помощью psql - никакой другой формат не поддерживается.

```
postgres$ pg_dumpall --clean -U postgres -f ./db5432.sql
```

38. В копию кластера дополнительно попадают такие команды, как:

```
postgres$ cat ./db5432.sql | grep ROLE
DROP ROLE postgres;
CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION
BYPASSRLS;
student$ psql -p 5433 -U postgres -f ./db5432.sql
SET
SET
SET
DROP DATABASE
psql:./db5432.sql:23: ERROR:  current user cannot be dropped
psql:./db5432.sql:30: ERROR:  role "postgres" already exists
ALTER ROLE
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
UPDATE 1
DROP DATABASE
CREATE DATABASE
ALTER DATABASE
You are now connected to database "template1" as user "postgres".
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
COMMENT
ALTER DATABASE
You are now connected to database "template1" as user "postgres".
```

```
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
REVOKE
GRANT
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
CREATE DATABASE
ALTER DATABASE
You are now connected to database "dbcopy1" as user "postgres".
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
CREATE SCHEMA
ALTER SCHEMA
SET
SET
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
ALTER TABLE
COPY 10
COPY 3
 setval
```

```
--------
       3
(1 row)

ALTER TABLE
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
SET
DROP DATABASE
CREATE DATABASE
ALTER DATABASE
You are now connected to database "postgres" as user "postgres".
SET
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
SET
COMMENT
```

Как уже понятно, pg_dumpall может работать только в одном режиме `plain` и значит создавать резервную копию в один поток, что сильно осложняет его использование. Также восстановление возможно в один поток. Это сильно затрудняет использование данной утилиты. Однако есть выход из данной ситуации. Если мы хотим создать логическую резервную копию всего кластера PostgreSQL, то можно в начале создать утилитой pg_dumpall копию глобальных объектов, для этого используем ключ `--globals-only`. И далее создаем копию каждой БД с помощью утилиты pg_dump нужного нам формата (custom или directory). Соответственно, восстановление должно проходить таким образом. В начале создаем глобальные объекты (роли и табличные пространства). Затем восстанавливаем базы данных pg_restore.

39. Проверяем, что у нас появились новые записи

```
postgres$ psql -p 5433 -d dbcopy1
5433=> \list
  datname
-----------
 postgres
```

```
 template0
 dbcopy2
 dbcopy1
 template1
(5 rows)

5433=> \c dbcopy1
You are now connected to database "dbcopy1" as user "student".
5433=> SELECT * FROM test;
id |        name
----+--------------------
  1 |
  2 |
  3 | Hello, world!!!
  4 | Новая запись
  5 | Самая новая запись
(5 rows)
```

2.3

Домашнее задание.

Логическая копия данных.

1. Создаем базу данных `db_dumpall` на порту 5432, нового пользователя, три таблицы и выдаем права к ним для нового пользователя;
2. Создаем резервную копию
3. Накатываем последовательно сначала глобальные объекты, потом данные объекта базы данных на сервера на порту 5433

Чек-лист

1. Создана новая база данных на порту 5432 и три таблицы в этой базе данных;
2. Создан новый пользователь и выданы ему три разных разрешения на таблицы (на первую – чтение, на вторую – чтение и запись, на третью – не выдаем права);
3. Создана с `pg_dumpall` резервная копия только глобальных объектов;
4. Создана с помощью `pg_dump` резервная копия созданной базы данных
5. Последовательно выполнены скрипт глобальных объектов, а потом скрипт разворачивания данной базы данных на сервере на порту 5433
6. Проверено, что права корректно перенеслись

Решение по ссылке в конце файла

Логическая резервная копия

Инициализация кластера PostgreSQL

1. Мы зашли на стенд. Теперь все действия с СУБД PostgreSQL мы будем выполнять от локального пользователя postgres. Для этого выполним команду

```
student$ sudo -u postgres -i
```

2. Перейдем в домашний каталог. Если этого не сделать, то будут постоянно появляться ошибки, связанные с отсутствием прав на домашнюю папку student.

```
postgres$ cd ~
```

3. Проверяем установленные пакеты.

```
postgres$ yum list installed | grep postgres
```

Должен быть примерно такой результат

```
postgresql12.x86_64             12.6-1PGDG.rhel7              @pgdg12
postgresql12-contrib.x86_64     12.6-1PGDG.rhel7              @pgdg12
postgresql12-docs.x86_64        12.6-1PGDG.rhel7              @pgdg12
postgresql12-libs.x86_64        12.6-1PGDG.rhel7              @pgdg12
postgresql12-pglogical.x86_64   2.3.3-1.el7                   @barman
postgresql12-server.x86_64      12.6-1PGDG.rhel7              @pgdg12
```

4. Инициализируем кластер. Мы это делаем специально для того, чтобы потом было удобно разворачивать другие экземпляры кластера на других портах для прохождения практики. К тому же нам нужно включить контрольные суммы. Это важный параметр, о нем мы будем говорить чуть позже.

```
postgres$ pg_ctl initdb "-D" "/var/lib/pgsql/12/main5432" -o "--data-
checksums"
```

Вывод будет примерно таким:

```
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.

creating directory /var/lib/pgsql/12/main5432 ... ok
creating subdirectories ... ok
selecting dynamic shared memory implementation ... posix
selecting default max_connections ... 100
selecting default shared_buffers ... 128MB
selecting default time zone ... UTC
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
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:

    /usr/pgsql-12/bin/pg_ctl -D /var/lib/pgsql/12/main5432 -l logfile start
```

5. Последняя команда показывает, как надо запустить кластер. Меняем немного команду, для нашего удобства. Путь до файлов кластера баз данных должен быть заканчиваться на main5432, номер порта на котором будет запущен кластер. И флаг `-l logfile` нам не нужен, будем писать в стандартный файл логов postgres-а, путь до которого можно посмотреть в конфигурационном файле `postgresql.conf`.

Запускаем кластер

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 start
```

И в результате

```
waiting for server to start....2021-03-30 10:41:21.319 UTC [18476] LOG:
starting PostgreSQL 12.6 on x86_64-pc-linux-gnu, compiled by gcc (GCC) 4.8.5
20150623 (Red Hat 4.8.5-44), 64-bit
2021-03-30 10:41:21.319 UTC [18476] LOG:  listening on IPv4 address
"127.0.0.1", port 5432
```

```
2021-03-30 10:41:21.322 UTC [18476] LOG:  listening on Unix socket
"/var/run/postgresql/.s.PGSQL.5432"
2021-03-30 10:41:21.330 UTC [18476] LOG:  listening on Unix socket
"/tmp/.s.PGSQL.5432"
2021-03-30 10:41:21.343 UTC [18476] LOG:  redirecting log output to logging
collector process
2021-03-30 10:41:21.343 UTC [18476] HINT:  Future log output will appear in
directory "log".
 done
server started
```

6. Проверяем состояние базы данных

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5432 status

pg_ctl: server is running (PID: 18476)

/usr/pgsql-12/bin/postgres "-D" "/var/lib/pgsql/12/main5432"
```

7. Инициализируем еще один кластер, но уже на порту 5433

```
postgres$ pg_ctl initdb "-D" "/var/lib/pgsql/12/main5433" -o "--data-
checksums"
```

8. Так как на порту 5432 у нас уже есть кластер PostgreSQL, то меняем порт с 5432 на порт 5433

```
postgres$ sed -i 's/#port = 5432/port = 5433/'
/var/lib/pgsql/12/main5433/postgresql.conf
```

9. Запускаем кластер

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5433 start
```

10. Проверяем состояние базы данных

```
postgres$ pg_ctl -D /var/lib/pgsql/12/main5433/ status
pg_ctl: server is running (PID: 18527)
/usr/pgsql-12/bin/postgres "-D" "/var/lib/pgsql/12/main5433"
```

Команда copy

11. Создаем базу данных

```
postgres$ psql

5432=> CREATE DATABASE dbcopy1;
5432=> \c dbcopy1
You are now connected to database "dbcopy1" as user "postgres".
```

12. Создадим таблицу test и внесем несколько записей

```
5432=> CREATE TABLE test(id bigserial PRIMARY KEY, name TEXT);
5432=> INSERT INTO test(name) VALUES (null), ('');
5432=> INSERT INTO test(name) SELECT 'Hello, world!';
```

Каждая таблица создает файлы на диске. Сколько создалось файлов данной командой?

13. Посмотрим наши записи

```
5432=> SELECT * FROM test;
id |     name
---+--------------
  1 | Hello, world!
  2 |
  3 |
(3 rows)
```

14. Их можно также выдать на экран с помощью команды copy

```
5432=> COPY test TO STDOUT;
1    Hello, world!
2
3    \N
```

Можно заметить отличие в выводе пустых и неопределенных значений.

15. Также в команде COPY можно настраивать формат вывода. Например, изменить разделитель, представление неопределенных значений и настраивать другие параметры вывода/ввода:

```
5432=> COPY test TO STDOUT WITH (NULL '<nil>', DELIMITER ';');
1;Hello, world!
2;
3;<nil>
```

16. Но, наверное, самое главное свойство команды COPY – это указывать вместо наименования таблицы произвольный запрос. Как на примере ниже:

```
5432=> COPY (SELECT * FROM test WHERE name='Hello, world!') TO STDOUT;
1    Hello, world!
```

Часто это свойство используют, когда нужно перекинуть данные между разными базами и кластерами БД.

17. Команда COPY позволяет выводить данные в формате CSV (FORMAT CSV) или в бинарном виде (FORMAT binary)

```
5432=> COPY test TO STDOUT WITH (FORMAT CSV);
1,"Hello, world!"
2,""
3,
```

Утилита pg_dump

18. Если запускать утилиту pg_dump без дополнительных параметров, то выдаются команды SQL, которые создают базу данных и заполняют ее данными:

```
postgres$ pg_dump -d dbcopy1
```

Примерный вывод -- -- PostgreSQL database dump --

-- Dumped from database version 12.6 (Ubuntu 12.6-1.pgdg18.04+1) -- Dumped by pg_dump version 12.6 (Ubuntu 12.6-1.pgdg18.04+1)

SET statement_timeout = 0; SET lock_timeout = 0; SET idle_in_transaction_session_timeout = 0; SET client_encoding = 'UTF8'; SET standard_conforming_strings = on; SELECT pg_catalog.set_config('search_path', '', false); SET check_function_bodies = false; SET xmloption = content; SET client_min_messages = warning; SET row_security = off;

SET default_tablespace = '';

SET default_table_access_method = heap;

-- -- Name: test; Type: TABLE; Schema: public; Owner: student

CREATE TABLE public.test (id bigint NOT NULL, name text);

ALTER TABLE public.test OWNER TO student;

-- -- Name: test_id_seq; Type: SEQUENCE; Schema: public; Owner: student

CREATE SEQUENCE public.test_id_seq START WITH 1 INCREMENT BY 1 NO MINVALUE NO MAXVALUE CACHE 1;

ALTER TABLE public.test_id_seq OWNER TO student;

-- -- Name: test_id_seq; Type: SEQUENCE OWNED BY; Schema: public; Owner: student

ALTER SEQUENCE public.test_id_seq OWNED BY public.test.id;

-- -- Name: test id; Type: DEFAULT; Schema: public; Owner: student

ALTER TABLE ONLY public.test ALTER COLUMN id SET DEFAULT nextval('public.test_id_seq'::regclass);

-- -- Data for Name: test; Type: TABLE DATA; Schema: public; Owner: student

COPY public.test (id, name) FROM stdin; 1 Hello, world! 2 3 \N .

-- -- Name: test_id_seq; Type: SEQUENCE SET; Schema: public; Owner: student

SELECT pg_catalog.setval('public.test_id_seq', 3, true);

-- -- Name: test test_pkey; Type: CONSTRAINT; Schema: public; Owner: student

ALTER TABLE ONLY public.test ADD CONSTRAINT test_pkey PRIMARY KEY (id);

-- -- PostgreSQL database dump complete

Можно увидеть, что pg_dump содержит скрипт создания таблицы test и скрипт команды copy для заполнения таблицы данными. Утилита pg_dump позволяет заменить команду copy командой insert. Для этого нужно воспользоваться флагом `--column-inserts`

19. Важный момент: в выгрузку попадают и изменения, сделанные в шаблонной БД template1. Поэтому восстанавливать резервную копию лучше на базе данных, созданной из template0. При использовании ключа `--create` это учитывается автоматически:

```
postgres$ pg_dump --create -d dbcopy1 | grep 'CREATE DATABASE'
CREATE DATABASE dbcopy1 WITH TEMPLATE = template0 ENCODING = 'UTF8' LOCALE =
'en_US.utf8';
```

20. У pg_dump есть много различных параметров. Мы не будем перечислять их тут. Для этого можно обратиться к документации postgresql. Укажем только два, так как она могут быть нам полезны. Ключ для выбора объектов, которые должны попасть в резервную копию: `-n, --schema` - шаблон для имен схем; `-t, --table` - шаблон для имен таблиц. Восстановим таблицу test в другой базе данных на другом сервере.

```
postgres$ psql -p 5433
5433$ CREATE DATABASE dbcopy2;
CREATE DATABASE
```

21. Завершаем сеанс соединения с БД

```
5433$ \q
```

22. Делаем копию таблицы и переносим ее на другой кластер 5433

```
postgres$ pg_dump --table=test -d dbcopy1 | psql -p 5433 -d dbcopy2
SET
SET
SET
SET
 set_config
------------
```

```
(1 row)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
ALTER TABLE
COPY 3
 setval
--------
      3
(1 row)

ALTER TABLE
```

23. Подключаемся к кластеру на порту 5433

```
postgres$ psql -p 5433 -d dbcopy2
5433=> SELECT * FROM test;
id |     name
---+---------------
  1 | Hello, world!
  2 |
  3 |
(3 rows)
```

Утилита pg_dump - формат custom

24. Одним из серьезных ограничений формата plain заключается в том, что выбирать объекты нужно в момент создания резервной копии. Нет возможности восстановить избранные объекты из резервной копии. Однако в формате custom отсутствует этот недостаток, что позволяет сначала сделать полную копию, а потом выбирать объекты при загрузке.

```
postgres$ pg_dump --format=custom -d dbcopy1 -f ./dbcopy1.custom
```

25. Для того, чтобы восстановить объекты из резервной копии, у нас есть утилита pg_restore. Воспользуемся ей для восстановления таблицы test:

```
postgres$ psql -p 5433 -d dbcopy2
5433=> DROP TABLE test;
DROP TABLE


postgres$ pg_restore --table=test -p 5433 -d dbcopy2 ./dbcopy1.custom
```

Кроме того, утилита pg_restore позволяет восстановить не только конкретную таблицу, но и индекс, функцию или триггер.

26. Проверяем, что данные восстановились

```
postgres$ psql -p5433 -d dbcopy2
5433=> SELECT * FROM test;
 id |      s
----+--------------
  1 | Hello, world!
  2 |
  3 |
(3 rows)
```

27. Восстановим целиком исходную базу данных dbcopy1 на сервере, который находится на порту 5433. Заметим, что новую резервную копию мы не делаем, используем старый файл копии.

```
postgres$ pg_restore --create -p 5433 -d postgres ./dbcopy1.custom
```

28. Проверим:

```
postgres$ psql -p5433 -l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access
privileges
-----------+----------+----------+-------------+-------------+--------------
--------
 dbcopy1   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 dbcopy2   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres
+
           |          |          |             |             |
postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
postgres=CTc/postgres+
           |          |          |             |             | =c/postgres
(5 rows)
```

Утилита pg_dump - формат directory

29. Прежде чем перейдем к данному формату давайте вначале создадим еще одну таблицу на сервере 5432 в новой схеме.

```
postgres$ psql -d dbcopy1
5432=> CREATE SCHEMA dump;
5432=> CREATE TABLE dump.direct(id int, number int);
```

А сколько тут создалось файлов на диске?

```
5432=> INSERT INTO dump.direct(id, number) select id, random()*1000 from
generate_series(1,10) id;
5432=> \q
```

30. Формат directory интересен тем, что позволяет выгружать данные в несколько параллельных потоков.

```
postgres$ pg_dump --format=directory --jobs=2 -d dbcopy1 -f
./dbcopy1.directory
```

31. Заглянем внутрь каталога:

```
postgres$ ls -l ./dbcopy1.directory
total 20
drwx------ 2 postgres postgres 4096 Mar 30 11:49 ./
drwx------ 8 postgres postgres 4096 Mar 30 11:49 ../
-rw-r--r-- 1 postgres postgres   51 Mar 30 11:49 3692.dat.gz
-rw-r--r-- 1 postgres postgres 2138 Mar 30 11:49 3693.dat.gz
-rw-r--r-- 1 postgres postgres 3031 Mar 30 11:49 toc.dat
```

32. В нем находится файл оглавления и по одному файлу на каждый выгружаемый объект (у нас он всего один):

```
postgres$ zcat ./dbcopy1.directory/3693.dat.gz
1    871
2    824
3    909
4    446
5    475
6    400
7    7
8    45
9    41
10    360
\.
```

33. Восстановление из резервной копии:

```
postgres$ pg_restore --clean --create --jobs=2 -p 5433 -d postgres
./dbcopy1.directory
```

В данной команде мы добавили ключ `--clean`, который генерирует команду удаления БД, и команду `--create`, которая создает нашу базу данных dbcopy1

Утилита pg_dump - формат tar

34. В данном случае формат tar мы не рассматриваем в связи с тем, что он очень похож на формат plain.

Утилита pg_dumpall

35. Как уже было сказано выше, утилита pg_dump годится для выгрузки одной базы данных, но никогда не выгружает общие объекты кластера БД, такие, как роли и табличные пространства. Чтобы сделать полную копию кластера, нужна утилита pg_dumpall. Давайте в таблицу test сервера 5432 еще внесем несколько записей, чтобы проверить работу утилиты pg_dumpall.

```
postgres$ psql -d dbcopy1
5432=> INSERT INTO test(name) VALUES('Новая запись'), ('Самая новая запись');
```

36. Проверяем

```
5432=> select * from test;
id  |        name
----+--------------------
  1 |
  2 |
  3 | Hello, world!!!
  4 | Новая запись
  5 | Самая новая запись
(5 rows)
```

37. Восстановление выполняется с помощью psql - никакой другой формат не поддерживается.

```
postgres$ pg_dumpall --clean -U postgres -f ./db5432.sql
```

38. В копию кластера дополнительно попадают такие команды, как:

```
postgres$ cat ./db5432.sql | grep ROLE
DROP ROLE postgres;
CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN
REPLICATION BYPASSRLS;
student$ psql -p 5433 -U postgres -f ./db5432.sql
SET
SET
SET
DROP DATABASE
psql:./db5432.sql:23: ERROR:  current user cannot be dropped
psql:./db5432.sql:30: ERROR:  role "postgres" already exists
ALTER ROLE
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
UPDATE 1
DROP DATABASE
CREATE DATABASE
ALTER DATABASE
You are now connected to database "template1" as user "postgres".
SET
SET
SET
SET
SET
 set_config
------------

(1 row)
```

```
SET
SET
SET
SET
COMMENT
ALTER DATABASE
You are now connected to database "template1" as user "postgres".
SET
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
REVOKE
GRANT
SET
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
CREATE DATABASE
ALTER DATABASE
You are now connected to database "dbcopy1" as user "postgres".
SET
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
CREATE SCHEMA
ALTER SCHEMA
SET
SET
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
```

```
ALTER SEQUENCE
ALTER TABLE
COPY 10
COPY 3
 setval
--------
       3
(1 row)

ALTER TABLE
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
DROP DATABASE
CREATE DATABASE
ALTER DATABASE
You are now connected to database "postgres" as user "postgres".
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
COMMENT
```

Как уже понятно, pg_dumpall может работать только в одном режиме `plain` и значит создавать резервную копию в один поток, что сильно осложняет его использование. Также восстановление возможно в один поток. Это сильно затрудняет использование данной утилиты. Однако есть выход из данной ситуации. Если мы хотим создать логическую резервную копию всего кластера PostgreSQL, то можно в начале создать утилитой pg_dumpall копию глобальных объектов, для этого используем ключ `--globals-only`. И далее создаем копию каждой БД с помощью утилиты pg_dump нужного нам формата (custom или directory). Соответственно, восстановление должно проходить таким образом. В начале создаем глобальные объекты (роли и табличные пространства). Затем восстанавливаем базы данных pg_restore.

39. Проверяем, что у нас появились новые записи

```
postgres$ psql -p 5433 -d dbcopy1
5433=> \list
  datname
```

```
-----------
 postgres
 template0
 dbcopy2
 dbcopy1
 template1
(5 rows)
```



```
5433=> \c dbcopy1
You are now connected to database "dbcopy1" as user "student".
5433=> SELECT * FROM test;
id |        name
---+--------------------
  1 |
  2 |
  3 | Hello, world!!!
  4 | Новая запись
  5 | Самая новая запись
(5 rows)
```

Домашнее задание.

Логическая копия данных.

1. Создаем базу данных db_dumpall на порту 5432, нового пользователя, три таблицы и выдаем права к ним для нового пользователя;
2. Создаем резервную копию
3. Накатываем последовательно сначала глобальные объекты, потом данные объекта базы данных на сервера на порту 5433

Чек-лист

1. Создана новая база данных на порту `5432` и три таблицы в этой базе данных;
2. Создан новый пользователь и выданы ему три разных разрешения на таблицы (на первую – чтение, на вторую – чтение и запись, на третью – не выдаем права);
3. Создана с `pg_dumpall` резервная копия только глобальных объектов;
4. Создана с помощью `pg_dump` резервная копия созданной базы данных
5. Последовательно выполнены скрипт глобальных объектов, а потом скрипт разворачивания данной базы данных на сервере на порту `5433`
6. Проверено, что права корректно перенеслись

Решение:

1. Создаем базу данных `db_dumpall` на порту `5432`, нового пользователя, три таблицы и выдаем права к ним для нового пользователя;

```
student$ sudo -iu postgres psql -p5432 -c "create database db_dumpall;"
CREATE DATABASE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "create user
dumpall_user;"
```

```
CREATE ROLE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "create table t1(id
int primary key)"
CREATE TABLE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "create table t2(id
int primary key)"
CREATE TABLE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "create table t3(id
int primary key)"
CREATE TABLE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "grant select on
table t1 to dumpall_user"
GRANT

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "grant select, insert
on table t2 to dumpall_user"
GRANT

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "\dt+"
                    List of relations
 Schema | Name | Type  |  Owner   |  Size   | Description
--------+------+-------+----------+---------+-------------
 public | t1   | table | postgres | 0 bytes |
 public | t2   | table | postgres | 0 bytes |
 public | t3   | table | postgres | 0 bytes |
(3 rows)

student$ sudo -iu postgres psql -p5432 -d db_dumpall -U dumpall_user -c
"select * from t1"
 id
----
(0 rows)

student$ sudo -iu postgres psql -p5432 -d db_dumpall -U dumpall_user -c
"select * from t2"
id
----
(0 rows)

student$ sudo -iu postgres psql -p5432 -d db_dumpall -U dumpall_user -c
"select * from t3"
ERROR:  permission denied for table t3
```

2. Создаем резервную копию

```
student$ sudo -iu postgres pg_dumpall --globals-only -p5432 >
/tmp/global.dump
student$ sudo -iu postgres pg_dump -Cc --if-exists -d db_dumpall -p5432 >
/tmp/db_dumpall.dump
```

3. Накатываем последовательно сначала глобальные объекты, потом данные объекта базы данных на сервера на порту `5433`

```
student$ sudo -iu postgres psql -p5433 -f /tmp/global.dump
SET
SET
SET
CREATE ROLE
```

```
ALTER ROLE
psql:/tmp/global.dump:16: ERROR:  role "postgres" already exists
ALTER ROLE

student$ sudo -iu postgres psql -p5433 -f /tmp/db_dumpall.dump
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
DROP DATABASE
CREATE DATABASE
ALTER DATABASE
You are now connected to database "db_dumpall" as user "postgres".
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
COPY 0
COPY 0
COPY 0
ALTER TABLE
ALTER TABLE
ALTER TABLE
GRANT
GRANT

student$ sudo -iu postgres psql -p5433 -d db_dumpall -U dumpall_user -c
"select * from t1"
id
----
(0 rows)

student$ sudo -iu postgres psql -p5433 -d db_dumpall -U dumpall_user -c
"select * from t2"
id
----
```

```
(0 rows)

student$ sudo -iu postgres psql -p5433 -d db_dumpall -U dumpall_user -c
"select * from t3"
ERROR:  permission denied for table t3
```

Решение:

1. Создаем базу данных `db_dumpall` на порту `5432`, нового пользователя, три таблицы и выдаем права к ним для нового пользователя;

```
student$ sudo -iu postgres psql -p5432 -c "create database db_dumpall;"
CREATE DATABASE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "create user
dumpall_user;"
CREATE ROLE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "create table t1(id
int primary key)"
CREATE TABLE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "create table t2(id
int primary key)"
CREATE TABLE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "create table t3(id
int primary key)"
CREATE TABLE

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "grant select on
table t1 to dumpall_user"
GRANT

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "grant select, insert
on table t2 to dumpall_user"
GRANT

student$ sudo -iu postgres psql -p5432 -d db_dumpall -c "\dt+"
                    List of relations
 Schema | Name | Type  |  Owner   |  Size   | Description
```

```
--------+------+-------+----------+---------+-------------
 public | t1   | table | postgres | 0 bytes |
 public | t2   | table | postgres | 0 bytes |
 public | t3   | table | postgres | 0 bytes |
(3 rows)

student$ sudo -iu postgres psql -p5432 -d db_dumpall -U dumpall_user -c
"select * from t1"
 id
----
(0 rows)

student$ sudo -iu postgres psql -p5432 -d db_dumpall -U dumpall_user -c
"select * from t2"
id
----
(0 rows)

student$ sudo -iu postgres psql -p5432 -d db_dumpall -U dumpall_user -c
"select * from t3"
ERROR:  permission denied for table t3
```

2. Создаем резервную копию

```
student$ sudo -iu postgres pg_dumpall --globals-only -p5432 >
/tmp/global.dump
student$ sudo -iu postgres pg_dump -Cc --if-exists -d db_dumpall -p5432 >
/tmp/db_dumpall.dump
```

3. Накатываем последовательно сначала глобальные объекты, потом данные объекта базы данных на сервера на порту `5433`

```
student$ sudo -iu postgres psql -p5433 -f /tmp/global.dump
SET
SET
SET
CREATE ROLE
ALTER ROLE
psql:/tmp/global.dump:16: ERROR:  role "postgres" already exists
ALTER ROLE

student$ sudo -iu postgres psql -p5433 -f /tmp/db_dumpall.dump
SET
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
DROP DATABASE
CREATE DATABASE
ALTER DATABASE
You are now connected to database "db_dumpall" as user "postgres".
SET
SET
```

```
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE TABLE
ALTER TABLE
COPY 0
COPY 0
COPY 0
ALTER TABLE
ALTER TABLE
ALTER TABLE
GRANT
GRANT

student$ sudo -iu postgres psql -p5433 -d db_dumpall -U dumpall_user -c
"select * from t1"
id
----
(0 rows)

student$ sudo -iu postgres psql -p5433 -d db_dumpall -U dumpall_user -c
"select * from t2"
id
----
(0 rows)

student$ sudo -iu postgres psql -p5433 -d db_dumpall -U dumpall_user -c
"select * from t3"
ERROR:  permission denied for table t3
```