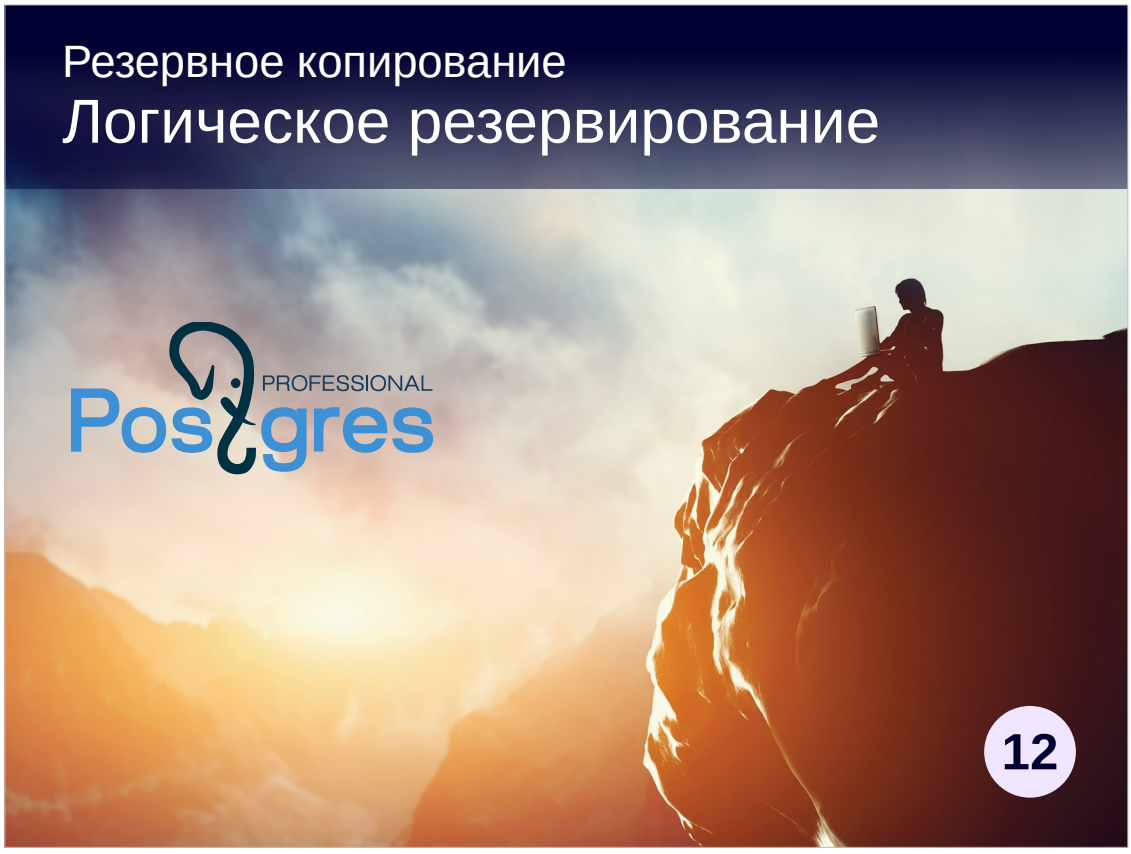


Авторы: Егор Рогов, Павел Лузанов

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

Темы

Логические и физические резервные копии

Копирование и восстановление отдельных таблиц

Копирование и восстановление отдельных баз данных

Копирование и восстановление кластера

Логическая копия

Команды SQL для создания объектов и наполнения данными

+ можно сделать копию отдельного объекта или отдельной базы
+ можно восстановиться на другой версии или архитектуре (не требуется двоичная совместимость)
+ простота использования
− невысокая скорость работы
− восстановление только на момент создания резервной копии

Логическая резервная копия — набор команд SQL, восстанавливающих кластер (или отдельную базу данных, или отдельную таблицу) с нуля: создаются необходимые объекты и наполняются данными.

Команды можно выполнить на другой версии СУБД (при наличии совместимости на уровне команд) или на другой платформе и архитектуре (не требуется двоичная совместимость).

В частности, логическую резервную копию можно использовать для долговременного хранения: ее можно будет восстановить и после обновления сервера на новую версию.

Создание логической резервной копии — относительно несложный процесс. Обычно он требует выполнения одной команды или запуск одной утилиты.

Однако для большой базы команды могут выполняться очень долго. Восстановить систему из логической копии можно ровно на момент начала резервного копирования.

https://postgrespro.ru/docs/postgresql/12/backup-dump

Физическая копия

Копия файловой системы кластера баз данных

+ быстрее, чем логическое резервирование
+ восстанавливается статистика
− можно восстановиться только на совместимой системе и на той же самой основной версии PostgreSQL
− выборочная копия невозможна, копируется весь кластер

Архив журнала предзаписи

+ возможность восстановления на определенный момент времени

Физическое резервирование подразумевает копирование всех файлов, относящихся к кластеру БД, то есть создание полной двоичной копии.

Копирование файлов работает быстрее, чем выгрузка SQL-команд при логическом резервировании; запустить сервер из созданной физической копии — дело нескольких минут, в отличие от восстановления из логической копии. Кроме того, нет необходимости заново собирать статистику — она также восстанавливается из копии.

Но есть и минусы. Из физической резервной копии можно восстановить систему только на совместимой платформе (та же ОС, та же разрядность, тот же порядок байтов в представлении чисел и т. п.) и только на той же основной версии PostgreSQL. Кроме того, невозможно сделать физическую копию отдельных баз данных кластера, возможно копирование только всего кластера целиком.

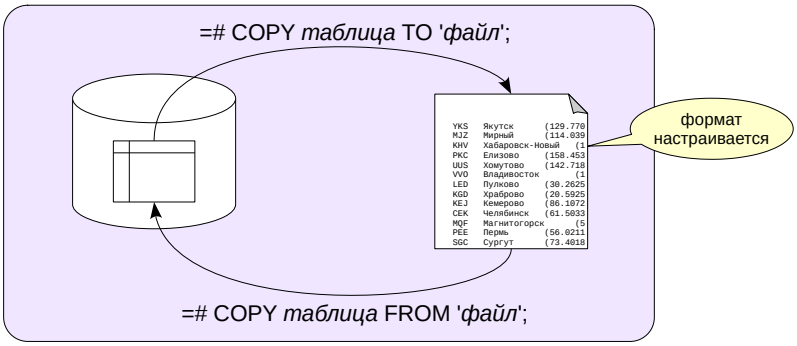
Как правило физические резервные копии используют совместно с архивом журнала предзаписи. Это позволяет восстанавливать систему не только на момент создания резервной копии, но и на произвольный момент времени.

https://postgrespro.ru/docs/postgresql/12/backup-file

https://postgrespro.ru/docs/postgresql/12/continuous-archiving

Физическое резервирование — обычный способ создания периодических резервных копий любой сколько-нибудь серьезной системы. Создание таких копий — ответственность администратора баз данных.

Копия таблицы в SQL



файл в ФС сервера и доступен владельцу экземпляра PostgreSQL
можно ограничить столбцы (или использовать произвольный запрос)
при восстановлении строки добавляются к имеющимся в таблице

Если требуется сохранить только содержимое одной таблицы, можно воспользоваться командой COPY.

Вариант COPY TO позволяет записать таблицу (или часть столбцов таблицы, или даже результат произвольного запроса) либо в файл, либо на консоль, либо на вход какой-либо программе. При этом можно указать ряд параметров, таких как формат (текстовый, CSV или двоичный), разделитель полей, текстовое представление NULL и т. п.

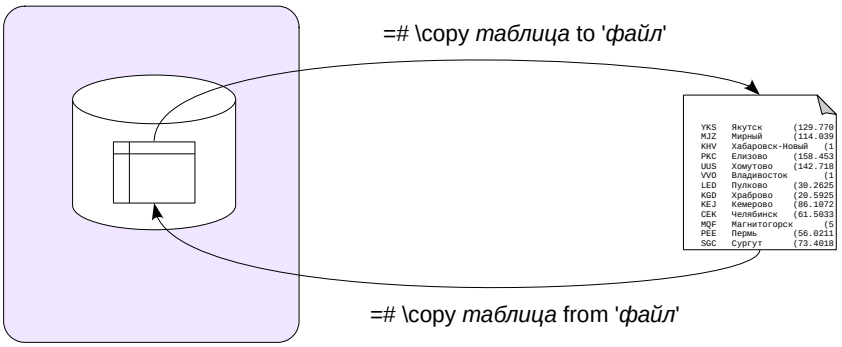
Вариант COPY FROM, наоборот, считывает из файла или из консоли строки и записывает их в таблицу. Таблица при этом не очищается, новые строки добавляются к уже существующим.

Команда COPY работает существенно быстрее, чем аналогичные команды INSERT — клиенту не нужно много раз обращаться к серверу, а серверу не нужно много раз анализировать команды.

Тонкость: при выполнении команды COPY FROM не применяются правила (rules), хотя ограничения целостности и триггеры выполняются.

https://postgrespro.ru/docs/postgresql/12/sql-copy

Копия таблицы в psql



файл в ФС клиента и доступен пользователю ОС, запустившему psql
происходит пересылка данных между клиентом и сервером
синтаксис и возможности аналогичны команде COPY

В psql существует клиентский вариант команды COPY с аналогичным синтаксисом.

Имя файла, указанное в SQL-команде COPY, соответствует файлу на сервере БД. У пользователя, под которым работает PostgreSQL (обычно postgres), должен быть доступ к этому файлу.

В клиентском же варианте команды файл располагается на клиенте, что позволяет сохранить локальную копию данных, даже не имея доступа к файловой системе сервера. Содержимое таблиц автоматически пересылается между клиентом и сервером.

https://postgrespro.ru/docs/postgresql/12/app-psql

Команда COPY

Создадим базу данных и таблицу.

```
=> CREATE DATABASE db1;
```

```
CREATE DATABASE
```

```
=> \c db1
```

```
You are now connected to database "db1" as user "student".
```

```
=> CREATE TABLE t(
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    s text
);
```

```
CREATE TABLE
```

```
=> INSERT INTO t(s) VALUES ('Привет, мир!'), (''), (NULL);
```

```
INSERT 0 3
```

Вот что показывает команда COPY (выдаем на консоль, а не в файл):

```
=> COPY t TO stdout;
```

```
1       Привет, мир!
2
3       \N
```

Видно, как различаются в выводе пустые строки и неопределенные значения.

Формат вывода настраивается достаточно гибко. Можно изменить разделитель, представление неопределенных значений и т. п. Например:

```
=> COPY t TO stdout WITH (null '<NULL>', delimiter ',');
```

```
1,Привет\, мир!
2,
3,<NULL>
```

Обратите внимание, что символ-разделитель внутри строки был экранирован (символ для экранирования тоже настраивается).

Вместо таблицы можно указать произвольный запрос.

```
=> COPY (SELECT * FROM t WHERE s IS NOT NULL) TO stdout;
```

```
1       Привет, мир!
2
```

Таким образом можно сохранить результат запроса, данные представления и т. п.

Команда поддерживает вывод в формате CSV, который поддерживается множеством программ.

```
=> COPY t TO stdout WITH (format csv);
```

```
1,"Привет, мир!"
2,""
3,
```

Аналогично работает и ввод данных из файла или с консоли.

При вводе с консоли требуется маркер конца файла - обратная косая черта с точкой. В обычном файле он не нужен.

Все параметры при вводе должны совпадать с теми, что были указаны при выводе.

```
=> TRUNCATE TABLE t;
```

```
TRUNCATE TABLE
```

```
=> COPY t FROM stdin;
1       Привет, мир!
2
3       \N
\.
```

```
COPY 3
```

Вот что загрузилось в таблицу (для наглядности настроим в psql вывод неопределенных значений):

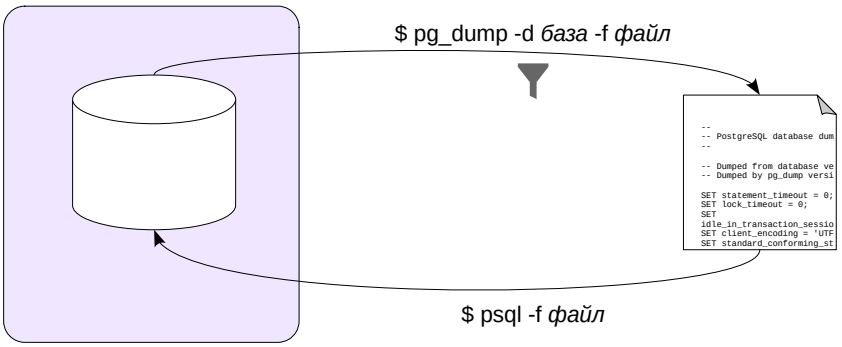
```
=> \pset null '\\N'

Null display is "\N".

=> SELECT * FROM t;

 id |      s
----+--------------
  1 | Привет, мир!
  2 |
  3 | \N
(3 rows)
```

Копия базы данных



формат: команды SQL
при выгрузке можно выбрать отдельные объекты базы данных
новая база должна быть создана из шаблона template0
заранее должны быть созданы роли и табличные пространства
после загрузки имеет смысл выполнить ANALYZE

Для создания полноценной резервной копии базы данных используется утилита pg_dump.

Если не указать имя файла (-f, --file), то утилита выведет результат на консоль. А результатом является скрипт, предназначенный для psql, который содержит команды для создания необходимых объектов и наполнения их данными.

Дополнительными ключами утилиты можно ограничить набор объектов: выбрать указанные таблицы, или все объекты в указанных схемах, или наложить другие фильтры.

Чтобы восстановить объекты из резервной копии, достаточно выполнить полученный скрипт в psql.

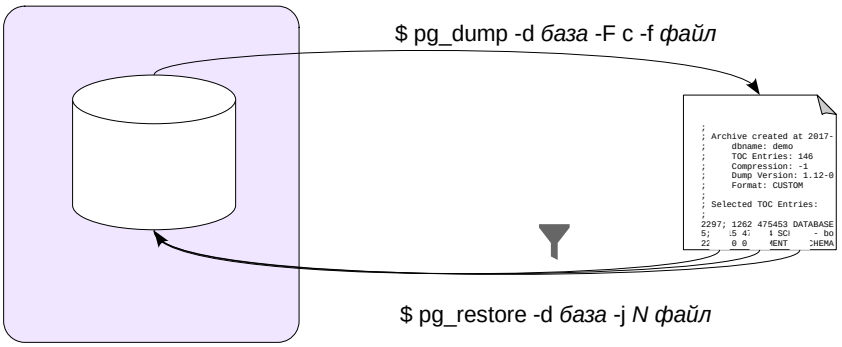
Следует иметь в виду, что базу данных для восстановления надо создавать из шаблона template0, так как все изменения, сделанные в template1, также попадут в резервную копию.

Кроме того, заранее должны быть созданы необходимые роли и табличные пространства. Поскольку эти объекты не относятся к конкретной БД, они не будут выгружены в резервную копию.

После восстановления базы имеет смысл выполнить команду ANALYZE: она соберет статистику, необходимую оптимизатору для планирования запросов.

https://postgrespro.ru/docs/postgresql/12/app-pgdump

Формат custom



внутренний формат с оглавлением

отдельные объекты базы данных можно выбрать на этапе восстановления

возможна загрузка в несколько параллельных потоков

Утилита pg_dump позволяет указать формат резервной копии. По умолчанию это plain — простые команды для psql.

Формат custom (-F c, --format=custom) создает резервную копию в специальном формате, содержащем не только объекты, но и оглавление. Наличие оглавления позволяет выбирать объекты для восстановления не при создании копии, а непосредственно при восстановлении.

Файл формата custom по умолчанию сжат.

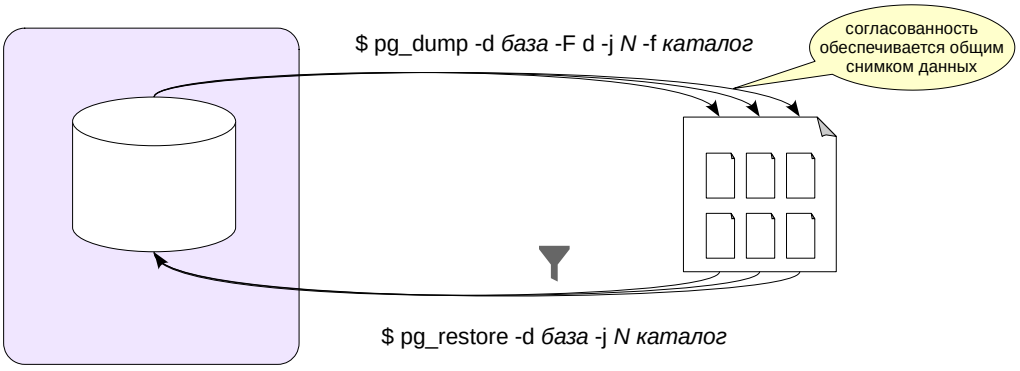
Для восстановления потребуется другая утилита — pg_restore. Она читает файл и преобразует его в команды psql. Если не указать явно имя базы данных (в ключе -d), то команды будут выведены на консоль. Если же база данных указана — утилита соединится с этой БД и выполнит команды; запускать psql не потребуется.

Чтобы восстановить только часть объектов, можно воспользоваться одним из двух подходов. Во-первых, можно ограничить объекты аналогично тому, как они ограничиваются в pg_dump. Вообще, утилита pg_restore понимает многие параметры из репертуара pg_dump.

Во-вторых, можно получить из оглавления список объектов, содержащихся в резервной копии (ключ --list). Затем этот список можно отредактировать вручную, удалив ненужное и передать измененный список на вход pg_restore (ключ --use-list).

https://postgrespro.ru/docs/postgresql/12/app-pgrestore.html

Формат directory



каталог с оглавлением и отдельными файлами на каждый объект базы
отдельные объекты базы данных можно выбрать на этапе восстановления
и выгрузка, и загрузка возможны в несколько параллельных потоков

Еще один формат резервной копии — directory. В таком случае будет создан не один файл, а каталог, содержащий объекты и оглавление. По умолчанию файлы внутри каталога будут сжаты.

Преимущество перед форматом custom состоит в том, что такая резервная копия может создаваться параллельно в несколько потоков (количество указывается в ключе -j, --jobs).

Разумеется, несмотря на параллельное выполнение, копия будет содержать согласованные данные. Это обеспечивается общим снимком данных для всех параллельно работающих процессов.

https://postgrespro.ru/docs/postgresql/12/functions-admin.html#FUNCTIONS-SNAPSHOT-SYNCHRONIZATION

Восстановление также возможно в несколько потоков (это работает и для формата custom).

В остальном возможности по работе с форматом directory не отличаются от ранее рассмотренных: поддерживаются те же ключи и подходы.

Сравнение форматов

	plain	custom	directory	tar
утилита для восстановления	psql	pg_restore		
сжатие	zlib			
выборочное восстановление		да	да	да
параллельное резервирование			да	
параллельное восстановление		да	да	

В приведенной таблице разные форматы сравниваются с точки зрения предоставляемых ими возможностей.

Отметим, что имеется и четвертый формат — tar. Он не рассматривался, так как не привносит ничего нового и не дает преимуществ перед другими форматами. Фактически он соответствует созданию tar-файла из каталога в формате directory, но не поддерживает сжатие и параллелизм.

Утилита pg_dump

При запуске без дополнительных параметров утилита pg_dump выдает команды SQL, создающие все объекты в базе данных:

```
student$ pg_dump -d db1

--
-- PostgreSQL database dump
--

-- Dumped from database version 12.13 (Ubuntu 12.13-1.pgdg20.04+1)
-- Dumped by pg_dump version 12.5 (Ubuntu 12.5-1.pgdg20.04+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: t; Type: TABLE; Schema: public; Owner: student
--

CREATE TABLE public.t (
    id integer NOT NULL,
    s text
);


ALTER TABLE public.t OWNER TO student;

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

ALTER TABLE public.t ALTER COLUMN id ADD GENERATED ALWAYS AS IDENTITY (
    SEQUENCE NAME public.t_id_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


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

COPY public.t (id, s) FROM stdin;
1       Привет, мир!
2
3       \N
.


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

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


--
-- Name: t t_pkey; Type: CONSTRAINT; Schema: public; Owner: student
--

ALTER TABLE ONLY public.t
```

```
    ADD CONSTRAINT t_pkey PRIMARY KEY (id);


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

Видно, что pg_dump создал таблицу t, добавил автоматическую генерацию идентификатора, заполнил таблицу с помощью уже рассмотренной нами команды COPY, и наконец добавил ограничение целостности для первичного ключа. Ключ --column-inserts позволяет использовать команды INSERT, но загрузка будет работать существенно дольше.

Рассмотрим некоторые полезные ключи.

Могут пригодиться при восстановлении копии на системе с другим набором ролей:

- -O, --no-owner - не генерировать команды для установки владельца объектов;
- -x, --no-acl - не генерировать команды для установки привилегий.

Полезны для выгрузки и загрузки данных частями:

- -s, --schema-only - выгрузить только определения объектов без данных;
- -a, --data-only - выгрузить только данные, без создания объектов.

Первый ключ удобен, если восстанавливать копию на системе, в которой уже есть данные, а второй - наоборот, на чистой системе:

- -c, --clean - генерировать команды DROP для создаваемых объектов;
- -C, --create - генерировать команды создания БД и подключения к ней.

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:

```
student$ pg_dump --create -d db1 | grep 'CREATE DATABASE'

CREATE DATABASE db1 WITH TEMPLATE = template0 ENCODING = 'UTF8' LC_COLLATE = 'en_US.UTF-8' LC_CTYPE = 'en_US.UTF-8';
```

Существуют ключи для выбора объектов, которые должны попасть в резервную копию:

- -n, --schema - шаблон для имен схем;
- -t, --table - шаблон для имен таблиц.

И наоборот, включить в копию все, кроме указанного:

- -N, --exclude-schema - шаблон для имен схем;
- -T, --exclude-table - шаблон для имен таблиц.

Например, восстановим таблицу t в другой базе данных.

```
=> CREATE DATABASE db2;

CREATE DATABASE

student$ pg_dump --table=t -d db1 | psql -d db2

SET
SET
SET
SET
SET
 set_config
------------

(1 row)

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

ALTER TABLE
```

Подключимся к базе данных db2 и проверим:

```
=> \c db2

You are now connected to database "db2" as user "student".

=> SELECT * FROM t;

 id |      s
----+--------------
  1 | Привет, мир!
  2 |
  3 | \N
(3 rows)
```

Утилита pg_dump - формат custom

Серьезное ограничение обычного формата (plain) состоит в том, что выбирать объекты нужно в момент выгрузки. Формат custom позволяет сначала сделать полную копию, а выбирать объекты уже при загрузке.

```
student$ pg_dump --format=custom -d db1 -f /home/student/db1.custom
```

Для восстановления объектов из такой копии предназначена утилита pg_restore. Повторим восстановление таблицы t.

```
=> DROP TABLE t;

DROP TABLE
```

Формат резервной копии указывать не обязательно - утилита распознает его сама.

Утилита pg_restore понимает те же ключи для фильтрации объектов, что и pg_dump, и даже больше:

- -I, --index - загрузить определенные индексы;
- -P, --function - загрузить определенные функции;
- -T, --trigger - загрузить определенные триггеры.

```
student$ pg_restore --table=t -d db2 /home/student/db1.custom

=> SELECT * FROM t;

 id |      s
----+--------------
  1 | Привет, мир!
  2 |
  3 | \N
(3 rows)
```

Еще один пример: восстановим целиком исходную базу данных db1.

```
=> DROP DATABASE db1;

DROP DATABASE
```

В ключе -d мы указываем любую существующую базу данных; с ключом --create утилита сама создаст базу, указанную в архиве, и тут же переключится в нее.

```
student$ pg_restore --create -d student /home/student/db1.custom
```

Проверим:

```
=> \c db1

You are now connected to database "db1" as user "student".

=> SELECT * FROM t;

 id |      s
----+--------------
  1 | Привет, мир!
  2 |
  3 | \N
(3 rows)
```

Резервную копию в обычном (plain) формате при необходимости можно изменить в текстовом редакторе. Резервная копия формата custom хранится в двоичном виде, но и для нее доступны более широкие возможности фильтрации объектов, чем предоставляемые рассмотренными ключами. Утилита pg_restore может сформировать список объектов - оглавление резервной копии:

```
student$ pg_restore --list /home/student/db1.custom

;
; Archive created at 2022-11-17 11:20:16 MSK
;     dbname: db1
;     TOC Entries: 9
;     Compression: -1
```

```
;     Dump Version: 1.14-0
;     Format: CUSTOM
;     Integer: 4 bytes
;     Offset: 8 bytes
;     Dumped from database version: 12.13 (Ubuntu 12.13-1.pgdg20.04+1)
;     Dumped by pg_dump version: 12.5 (Ubuntu 12.5-1.pgdg20.04+1)
;
;
; Selected TOC Entries:
;
203; 1259 17006 TABLE public t student
202; 1259 17004 SEQUENCE public t_id_seq student
2962; 0 17006 TABLE DATA public t student
2969; 0 0 SEQUENCE SET public t_id_seq student
2834; 2606 17013 CONSTRAINT public t t_pkey student
```

Такой список можно записать в файл, отредактировать и использовать его для восстановления с помощью ключа --use-list.

Утилита pg_dump - формат directory

Формат directory интересен тем, что позволяет выгружать данные в несколько параллельных потоков. При этом гарантируется согласованность данных: все потоки будут использовать один и тот же снимок данных.

```
student$ pg_dump --format=directory --jobs=2 -d db1 -f /home/student/db1.directory
```

Заглянем внутрь каталога:

```
student$ ls -l /home/student/db1.directory
```

```
total 8
-rw-rw-r-- 1 student student   60 ноя 17 11:20 2961.dat.gz
-rw-rw-r-- 1 student student 1997 ноя 17 11:20 toc.dat
```

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

```
student$ zcat /home/student/db1.directory/2961.dat.gz
```

```
1       Привет, мир!
2
3       \N
.
```

Восстановим базу db1 из резервной копии, используя два потока.

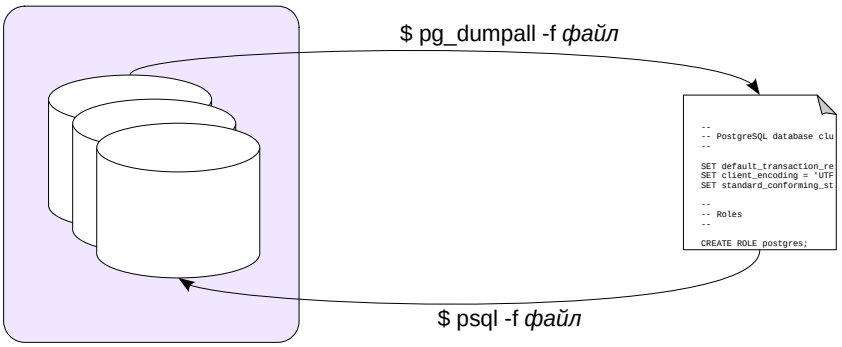
Здесь мы добавляем ключ --clean, который генерирует команду удаления БД, поскольку db1 существует. И предварительно надо отключиться от db1:

```
=> \c db2
```

```
You are now connected to database "db2" as user "student".
```

```
student$ pg_restore --clean --create --jobs=2 -d student /home/student/db1.custom
```

Копия кластера БД



формат: команды SQL
выгружает весь кластер, включая роли и табличные пространства
пользователь должен иметь доступ ко всем объектам кластера
не поддерживает параллельную выгрузку

Чтобы создать резервную копию всего кластера, включая роли и табличные пространства, можно воспользоваться утилитой pg_dumpall.

Поскольку pg_dumpall требуется доступ ко всем объектам всех БД, имеет смысл запускать ее от имени суперпользователя. Утилита по очереди подключается к каждой БД кластера и выгружает информацию с помощью pg_dump. Кроме того, она сохраняет и данные, относящиеся к кластеру в целом.

Чтобы начать работу, утилите требуется подключиться хотя бы к какой-то базе данных. По умолчанию выбирается postgres или template1, но можно указать и другую.

Результатом работы pg_dumpall является скрипт для psql. Другие форматы не поддерживаются. Это означает, что pg_dumpall не поддерживает параллельную выгрузку данных, что может оказаться проблемой при больших объемах данных. В таком случае можно воспользоваться ключом --globals-only, чтобы выгрузить только роли и табличные пространства, а сами базы данных выгрузить отдельно с помощью pg_dump в параллельном режиме.

https://postgrespro.ru/docs/postgresql/12/app-pg-dumpall

Утилита pg_dumpall

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.

Все рассматриваемые в этой теме утилиты не требуют каких-то отдельных привилегий, но у выполняющей их роли должны быть привилегии на чтение (создание) всех затронутых объектов. Утилитой pg_dump может, например, пользоваться владелец базы данных. Но поскольку для копирования кластера утилите pg_dumpall надо иметь доступ ко всем базам данных, следует использовать роль с атрибутом суперпользователя.

```
student$ pg_dumpall --clean -f /home/student/main.sql
```

В копию кластера дополнительно попадают такие команды, как:

```
student$ grep 'ROLE' main.sql
```

```
DROP ROLE postgres;
DROP ROLE student;
CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION BYPASSRLS PASSWORD 'md53175bce1d3201d16594cebf9d7eb3f9d';
CREATE ROLE student;
ALTER ROLE student WITH SUPERUSER INHERIT NOCREATEROLE NOCREATEDB LOGIN NOREPLICATION NOBYPASSRLS PASSWORD 'md550d9482e20934ce6df0bf28941f885bc';
```

Восстановление выполняется с помощью psql - никакой другой формат не поддерживается. Команда для восстановления (мы не будет ее выполнять):

```
student$ psql -f main.sql
```

В процессе восстановления могут возникать ошибки из-за существующих объектов - обычно это нормально и не мешает процессу, хотя все сообщения стоит проанализировать.

Итоги

Логическое резервирование позволяет сделать копию всего кластера, базы данных или отдельных объектов

Хорошо подходит

- для данных небольшого объема
- для длительного хранения, за время которого меняется версия сервера
- для миграции на другую платформу

Плохо подходит

- для восстановления после сбоя с минимальной потерей данных

Практика

1. Создайте резервную копию базы данных bookstore в формате custom.

   «Случайно» удалите все записи из таблицы authorship. Проверьте, что приложение перестало отображать названия книг на вкладках «Магазин», «Книги», «Каталог».

   Используйте резервную копию для восстановления потерянных данных в таблице.

   Проверьте, что нормальная работа книжного магазина восстановилась.

1. При восстановлении используйте ключ --data-only, чтобы избежать ошибки при попытке создания таблицы.

1. Восстановление потерянных данных

Создание резервной копии:

```
student$ pg_dump --format=custom -d bookstore > /home/student/bookstore.custom
```

Удаляем строки:

```
=> DELETE FROM authorship;
```

```
DELETE 9
```

Восстановление:

```
student$ pg_restore -t authorship --data-only -d bookstore /home/student/bookstore.custom
```

```
=> SELECT count(*) FROM authorship;
```

```
 count
-------
     9
(1 row)
```

Практика

1. Создайте таблицу с политикой, разрешающей чтение только части строк. Создайте непривилегированного пользователя для Алисы и предоставьте ей доступ к таблице.

   В обязанности Алисы входит резервное копирование таблицы. Сможет ли она выполнять их, не являясь суперпользователем? Проверьте.
2. Команда psql \copy позволяет направить результат на вход произвольной программы. Воспользуйтесь этим, чтобы открыть результаты какого-нибудь запроса в электронной таблице LibreOffice Calc.

1. К роли с правами суперпользователя не применяются политики защиты строк. Однако Алиса, как непривилегированный пользователь, сможет прочитать лишь часть строк таблицы и даже не узнает о том, что это не все данные.

Параметр *row_security* позволит Алисе хотя бы узнать о том, что не удалось прочитать все данные. А атрибут роли BYPASSRLS решит задачу.

2. Команда должна перенаправить результат в файл, а затем запустить libreoffice, указав этот файл в качестве параметра. Файл должен быть записан в формате CSV.

Конечно, такой способ зависит от платформы и без модификации не будет работать, например, в Windows.

1. Политики защиты строк

Создаем таблицу с политикой и роль.

```
=> CREATE DATABASE backup_logical;
```

```
CREATE DATABASE
```

```
=> \c backup_logical
```

```
You are now connected to database "backup_logical" as user "student".
```

```
=> CREATE TABLE t(
    id integer PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
    s text
);
```

```
CREATE TABLE
```

```
=> INSERT INTO t(s) VALUES ('foo'), ('bar'), ('baz');
```

```
INSERT 0 3
```

```
=> CREATE POLICY odd ON t USING (mod(id,2) = 1);
```

```
CREATE POLICY
```

```
=> ALTER TABLE t ENABLE ROW LEVEL SECURITY;
```

```
ALTER TABLE
```

```
=> CREATE ROLE alice LOGIN PASSWORD 'alicepass';
```

```
CREATE ROLE
```

```
=> GRANT SELECT ON t TO alice;
```

```
GRANT
```

Алиса пытается прочитать таблицу:

```
student$ psql "host=localhost user=alice dbname=backup_logical password=alicepass"
```

```
alice=> COPY t TO stdout; -- или SELECT * FROM t;
```

```
1       foo
3       baz
```

С выключенным параметром row_security Алиса получит ошибку, если политики запрещают видеть часть строк:

```
alice=> SET row_security = off;
```

```
SET
```

```
alice=> COPY t TO stdout;
```

```
ERROR:  query would be affected by row-level security policy for table "t"
```

Для того чтобы Алиса могла обходить политики защиты строк, не являясь суперпользователем, к ее роли надо добавить атрибут BYPASSRLS:

```
=> ALTER ROLE alice BYPASSRLS;
```

```
ALTER ROLE
```

```
alice=> COPY t TO stdout;
```

```
1       foo
2       bar
3       baz
```

2. Открытие результата запроса в LibreOffice

Попробуйте такую команду:

```
\copy t TO PROGRAM 'cat > /home/student/t.csv; libreoffice /home/student/t.csv' WITH (format csv);
```

Если вместо \copy использовать SQL-команду COPY, программа будет запущена на сервере СУБД, что, конечно, неправильно.