

Резервное копирование Логическое резервирование



Авторские права

© Postgres Professional, 2018–2022

Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

Использование материалов курса

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

Обратная связь

Отзывы, замечания и предложения направляйте по адресу:

edu@postgrespro.ru

Отказ от ответственности

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Понятие логической резервной копии

Копирование и восстановление отдельных таблиц

Копирование и восстановление баз данных

Копирование и восстановление кластера

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)
- невысокая скорость работы
- восстановление только на момент создания резервной копии

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

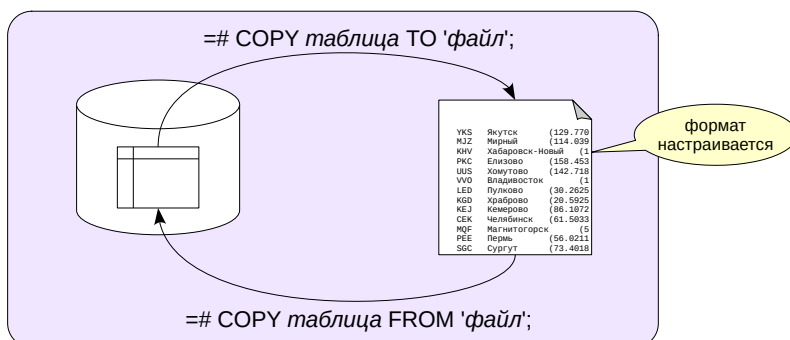
Команды можно выполнить на другой версии СУБД (при наличии совместимости на уровне команд) или на другой платформе и архитектуре (не требуется двоичная совместимость).

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

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/13/backup-dump.html>

SQL-команда COPY



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

4

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

Команда 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.

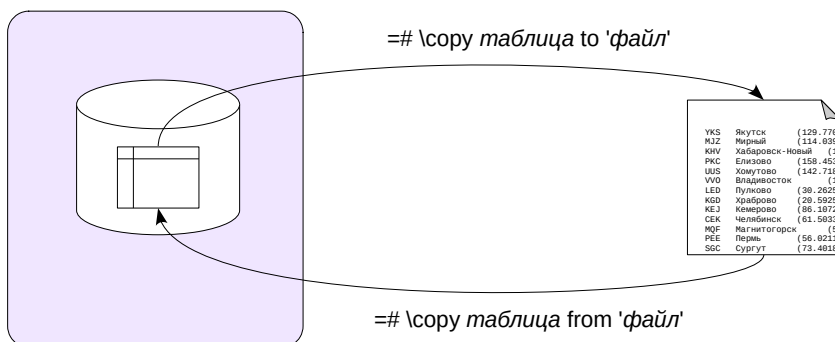
Другой вариант команды, наоборот, считывает из файла или из консоли строки с полями и записывает их в таблицу. Таблица при этом не очищается, новые строки добавляются к уже существующим.

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/13/sql-copy>

Команда psql \copy



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.

В отличие от серверного варианта COPY, который является командой SQL, клиентский вариант — это команда psql.

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

В клиентском же варианте обращение к файлу происходит на клиенте, а на сервер передается только содержимое.

<https://postgrespro.ru/docs/postgresql/13/app-psql#APP-PSQL-META-COMMANDS-COPY>

Команда COPY

Создадим базу данных и таблицу.

```
α=> CREATE DATABASE db1;
```

CREATE DATABASE

```
α=> \c db1
```

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

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

CREATE TABLE

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

INSERT 0 3

```
α=> SELECT * FROM t;
```

id	s
1	Привет, мир!
2	
3	

(3 rows)

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

```
α=> 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:

```
α=> 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.

```
α=> 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.

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

```
1,"Привет, мир!"
2,""
3,
```

Аналогично работает и ввод данных из файла или с консоли.

```
α=> TRUNCATE TABLE t;
```

TRUNCATE TABLE

Но при вводе с консоли требуется маркер конца файла — обратная косая черта с точкой; в обычном файле он не нужен.

Чтобы данные загрузились, при вводе надо указать те же параметры, что были указаны при выводе.

При загрузке также можно указать условие:

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

COPY 2

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

```
α=> \pset null '\\N'
```

Null display is "\\N".

```
α=> SELECT * FROM t;
```

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

Загрузим все строки:

```
α=> TRUNCATE TABLE t;
```

TRUNCATE TABLE

```
α=> COPY t FROM stdin;
```

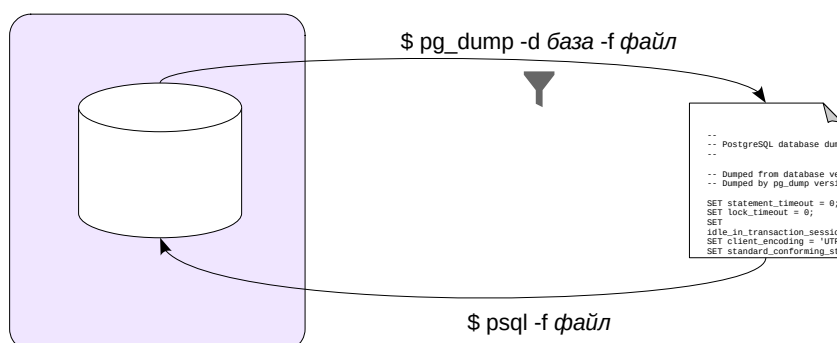
```
1      Привет, мир!  
2  
3      \N  
\.
```

COPY 3

```
α=> 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

7

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/13/app-pgdump>

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 13.6 (Ubuntu 13.6-1.pgdg22.04+1+b1)
-- Dumped by pg_dump version 13.6 (Ubuntu 13.6-1.pgdg22.04+1+b1)

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` и заполнил ее с помощью уже рассмотренной нами команды `COPY`. 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;
- `--no-comments` — не генерировать комментарии.

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.

Удобны, если восстанавливать копию на системе, в которой уже есть данные (и 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' LOCALE = '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 на другом сервере.

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433
```

```
| β=> CREATE DATABASE db2;
| CREATE DATABASE
```

```
student$ pg_dump --table=t -d db1 | psql -p 5433 -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
-----
```

(1 row)

ALTER TABLE

| `β=> \c db2`

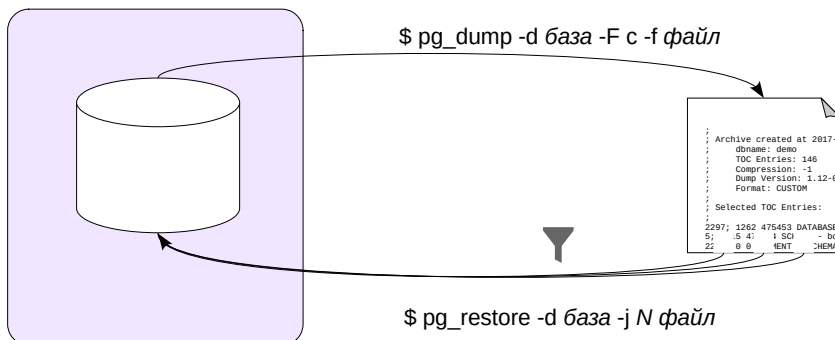
| You are now connected to database "db2" as user "student".

| `β=> SELECT * FROM t;`

id	s
1	Привет, мир!
2	
3	

(3 rows)

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 без участия `psql`.

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/13/app-pgrestore>

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`:

```
| β=> DROP TABLE t;  
| DROP TABLE
```

```
student$ pg_restore --table=t -p 5433 -d db2 /home/student/db1.custom
```

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.

Проверим, как восстановилась таблица:

```
| β=> SELECT * FROM t;  
  
| id | s  
|----+-----  
| 1 | Привет, мир!  
| 2 |  
| 3 |  
| (3 rows)
```

Еще один пример: восстановим целиком исходную базу данных `db1` на другом сервере.

```
student$ pg_restore --create -p 5433 -d postgres /home/student/db1.custom
```

Здесь мы указали БД `postgres`, но могли указать любую — утилита сама создаст нужную БД и тут же переключится в нее.

Проверим:

```
| β=> \c db1  
| You are now connected to database "db1" as user "student".  
  
| β=> SELECT * FROM t;  
  
| id | s  
|----+-----  
| 1 | Привет, мир!  
| 2 |  
| 3 |  
| (3 rows)
```

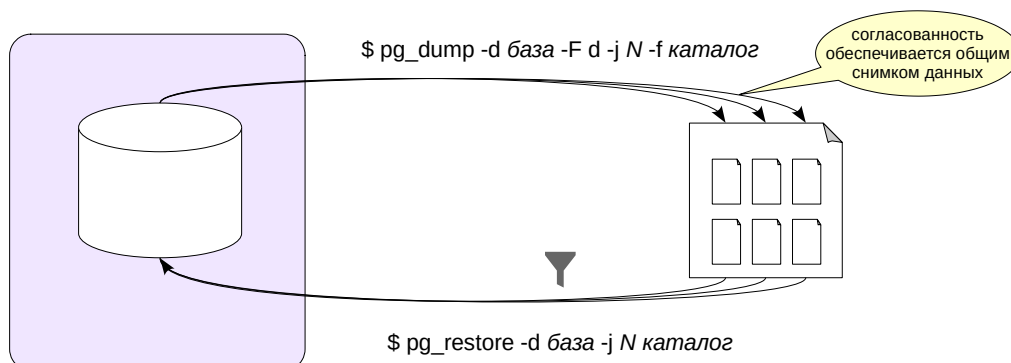
Резервную копию в обычном (`plain`) формате при необходимости можно изменить в текстовом редакторе. Резервная копия формата `custom` хранится в двоичном виде, но и для нее доступны более широкие возможности фильтрации объектов, чем рассмотренные ключи. Утилита `pg_restore` может сформировать список объектов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

```
student$ pg_restore --list /home/student/db1.custom  
  
;  
; Archive created at 2022-05-07 01:05:11 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: 13.6 (Ubuntu 13.6-1.pgdg22.04+1+b1)  
; Dumped by pg_dump version: 13.6 (Ubuntu 13.6-1.pgdg22.04+1+b1)  
;  
;
```

```
; Selected TOC Entries:
;
201; 1259 24579 TABLE public t student
200; 1259 24577 SEQUENCE public t_id_seq student
3027; 0 24579 TABLE DATA public t student
3034; 0 0 SEQUENCE SET public t_id_seq student
2895; 2606 24586 CONSTRAINT public t_t_pkey student
```

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

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

11

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/13/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 отличается от ранее рассмотренных: поддерживаются те же ключи и подходы.

Утилита `pg_dump` — формат `directory`

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

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

При этом гарантируется согласованность данных: все параллельные потоки будут использовать один и тот же снимок данных.

Заглянем внутрь каталога:

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

```
total 8
-rw-rw-r-- 1 student student 60 мая 7 01:05 3027.dat.gz
-rw-rw-r-- 1 student student 1972 мая 7 01:05 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/3027.dat.gz
```

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

Для восстановления из резервной копии предварительно отключимся от базы данных `db1`:

```
| β=> \q
```

В команду восстановления добавляем ключ `--clean`, который генерирует команду удаления существующей БД:

```
student$ pg_restore --clean --create --jobs=2 -p 5433 -d postgres /home/student/db1.directory
```

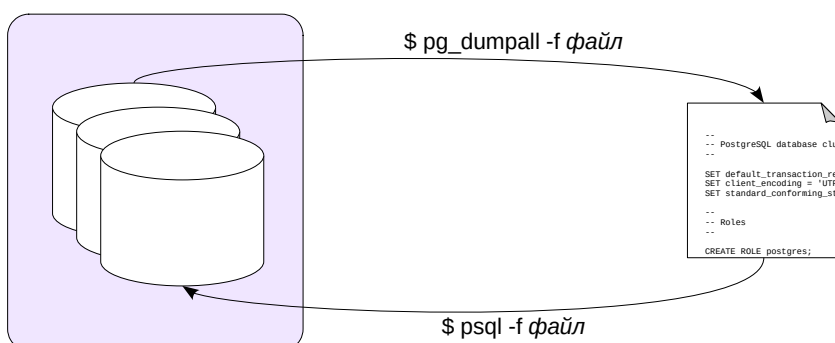

Сравнение форматов

	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.

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

14

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/13/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.

Утилиты pg_dumpall, pg_dump и pg_restore 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 надо иметь доступ ко всем БД, мы выполняем pg_dumpall под суперпользовательской ролью.

```
student$ pg_dumpall --clean -U postgres -f /home/student/alpha.sql
```

В копию кластера попадают:

- команды, которые выгружает pg_dump для каждой базы;
- команды для общих объектов кластера (ролей и табличных пространств).

Команды для общих объектов можно получить отдельно:

```
student$ pg_dumpall --clean --globals-only -U postgres
```

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

SET default_transaction_read_only = off;

SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;


--
-- Drop roles
--

DROP ROLE postgres;
DROP ROLE student;


--
-- Roles
--

CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION BYPASSRLS;
CREATE ROLE student;
ALTER ROLE student WITH NOSUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION NOBYPASSRLS PASSWORD 'md550d9482e20934ce6df0bf28941f885bc';


--
-- Role memberships
--

GRANT pg_read_all_stats TO student GRANTED BY postgres;


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

Восстановление выполняется с помощью psql — никакой другой формат не поддерживается.

```
student$ psql -p 5433 -U postgres -f /home/student/alpha.sql
```

```
SET
SET
SET
DROP DATABASE
DROP DATABASE
psql:/home/student/alpha.sql:24: ERROR:  current user cannot be dropped
psql:/home/student/alpha.sql:25: ERROR:  role "student" cannot be dropped because some objects depend on it
DETAIL:  owner of database db2
1 object in database db2
psql:/home/student/alpha.sql:32: ERROR:  role "postgres" already exists
ALTER ROLE
psql:/home/student/alpha.sql:34: ERROR:  role "student" already exists
ALTER ROLE
psql:/home/student/alpha.sql:42: NOTICE:  role "student" is already a member of role "pg_read_all_stats"
GRANT 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_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_config
-----

(1 row)

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

(1 row)

SET
SET
SET
SET
CREATE DATABASE
ALTER DATABASE
You are now connected to database "db1" as user "postgres".
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
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
  set_config
-----

(1 row)

SET
SET
SET
SET
COMMENT
SET
SET
SET
SET
SET
  set_config
-----

(1 row)

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

(1 row)

SET
SET
SET
SET

```

В процессе восстановления могут возникать ошибки из-за невозможности удалить некоторые объекты или создать уже существующие, в данном случае это нормально и не мешает процессу.

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433
```

```

|  β=> SELECT datname FROM pg_database;
|
|      datname
|      -----
|      template0
|      db2
|      template1
|      db1
|      postgres
|      student
|      (6 rows)
|
|  β=> \c db1
|
|  You are now connected to database "db1" as user "student".
|
|  β=> SELECT * FROM t;
|
|  id |      s
|  ---+-----
|  1 | Привет, мир!
|  2 |
|  3 |
|  (3 rows)

```

Влияние политик защиты строк

Если на таблицах определены политики защиты строк, то есть опасность выгрузить неполные данные и даже не узнать об этом. Чтобы этого не произошло, перед выполнением команды COPY можно установить параметр row_security в значение off — в этом случае применение политики приведет к явной ошибке.

Простой пример. Настроим политику так, чтобы не выводились пустые строки, и включим ее для владельца таблицы:

```
α=> CREATE POLICY t_s_not_null ON t USING (s IS NOT NULL);
```

```
CREATE POLICY
```

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

```
ALTER TABLE
```

```
α=> ALTER TABLE t FORCE ROW LEVEL SECURITY;
```

```
ALTER TABLE
```

Теперь запрос покажет только две строки:

```
α=> COPY t TO stdout;
```

```
1      Привет, мир!  
2
```

Но с параметром, установленным в off, будет зафиксирована ошибка:

```
α=> SET row_security = off;
```

SET

```
α=> COPY t TO stdout;
```

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

HINT: To disable the policy for the table's owner, use ALTER TABLE NO FORCE ROW LEVEL SECURITY.

Утилиты pg_dump и pg_dumpall автоматически используют этот параметр, так что дополнительные действия предпринимать не нужно:

```
student$ pg_dump -d db1 > /dev/null
```

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

HINT: To disable the policy for the table's owner, use ALTER TABLE NO FORCE ROW LEVEL SECURITY.

pg_dump: error: query was: COPY public.t (id, s) TO stdout;

Чтобы обойти политики защиты строк и выгрузить все данные, пользователь должен иметь атрибут роли BYPASSRLS:

```
α=> \c - postgres
```

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

```
α=> ALTER USER student BYPASSRLS;
```

ALTER ROLE

Теперь запросы от роли student будут игнорировать политики защиты строк:

```
student$ pg_dump -d db1 -t t --data-only
```

```
--
```

```
-- PostgreSQL database dump
```

```
--
```

```
-- Dumped from database version 13.6 (Ubuntu 13.6-1.pgdg22.04+1+b1)
```

```
-- Dumped by pg_dump version 13.6 (Ubuntu 13.6-1.pgdg22.04+1+b1)
```

```
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;
```

```
--
```

```
-- 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);
```

```
--
```

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

```
--
```

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

1. На первом сервере создайте несколько баз данных. В них создайте различные объекты (например, таблицы, представления, индексы).
2. Сделайте копию только глобальных объектов кластера с помощью утилиты `pg_dumpall`.
3. Сделайте копии каждой базы данных кластера с помощью утилиты `pg_dump` в параллельном режиме.
4. Полностью восстановите кластер на другом сервере, используя созданные резервные копии.
5. Попробуйте подобрать такие данные и параметры команды `COPY`, чтобы созданную копию таблицы невозможно было загрузить.

1. Базы данных и объекты

```
α=> CREATE DATABASE db1;
CREATE DATABASE
α=> \c db1
You are now connected to database "db1" as user "student".
α=> CREATE TABLE t1(n integer);
CREATE TABLE
α=> INSERT INTO t1 VALUES (1), (2), (3);
INSERT 0 3
α=> CREATE VIEW v1 AS SELECT * FROM t1;
CREATE VIEW
α=> CREATE DATABASE db2;
CREATE DATABASE
α=> \c db2
You are now connected to database "db2" as user "student".
α=> CREATE TABLE t2(n integer);
CREATE TABLE
α=> INSERT INTO t2 VALUES (1), (2), (3);
INSERT 0 3
α=> CREATE VIEW v2 AS SELECT * FROM t2;
CREATE VIEW
```

2. Копия глобальных объектов

```
student$ pg_dumpall --clean --globals-only -U postgres -f /home/student/alpha_globals.sql
```

3. Копии баз данных

Здесь мы ограничимся теми базами данных, которые создали сами.

```
student$ pg_dump --jobs=2 --format=directory -d db1 -f /home/student/db1.directory
student$ pg_dump --jobs=2 --format=directory -d db2 -f /home/student/db2.directory
```

4. Восстановление кластера

Сначала восстанавливаем глобальные объекты:

```
student$ psql -p 5433 -U postgres -f alpha_globals.sql
SET
SET
SET
psql:alpha_globals.sql:16: ERROR:  current user cannot be dropped
psql:alpha_globals.sql:17: ERROR:  role "student" cannot be dropped because some objects depend on it
DETAIL:  owner of database student
psql:alpha_globals.sql:24: ERROR:  role "postgres" already exists
ALTER ROLE
psql:alpha_globals.sql:26: ERROR:  role "student" already exists
ALTER ROLE
psql:alpha_globals.sql:34: NOTICE:  role "student" is already a member of role "pg_read_all_stats"
GRANT ROLE
```

Затем восстанавливаем базы данных:

```
student$ pg_restore -p 5433 -d postgres --create --jobs=2 /home/student/db1.directory
student$ pg_restore -p 5433 -d postgres --create --jobs=2 /home/student/db2.directory
```

Проверим:

```
student$ /usr/lib/postgresql/13/bin/psql -p 5433
```

```
| β=> \c db1
```

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

```
| β=> \d
```

```
|          List of relations
|  Schema | Name | Type  | Owner
|-----+-----+-----+-----
| public  | t1   | table | student
| public  | v1   | view  | student
| (2 rows)
```

```
| β=> \c db2
```

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

```
| β=> \d
```

```
|          List of relations
|  Schema | Name | Type  | Owner
|-----+-----+-----+-----
| public  | t2   | table | student
| public  | v2   | view  | student
| (2 rows)
```

5. Ломаем COPY

Например, можно установить отображение неопределенных значений, совпадающее с какими-либо данными:

```
α=> CREATE TABLE anticopy(s text);
```

```
CREATE TABLE
```

```
α=> INSERT INTO anticopy(s) VALUES ('N'), (NULL);
```

```
INSERT 0 2
```

```
α=> COPY anticopy TO stdout WITH (NULL 'N');
```

```
N
```

```
N
```

Вывод двух разных значений теперь неотличим друг от друга.

Осторожнее с изменением формата по умолчанию!