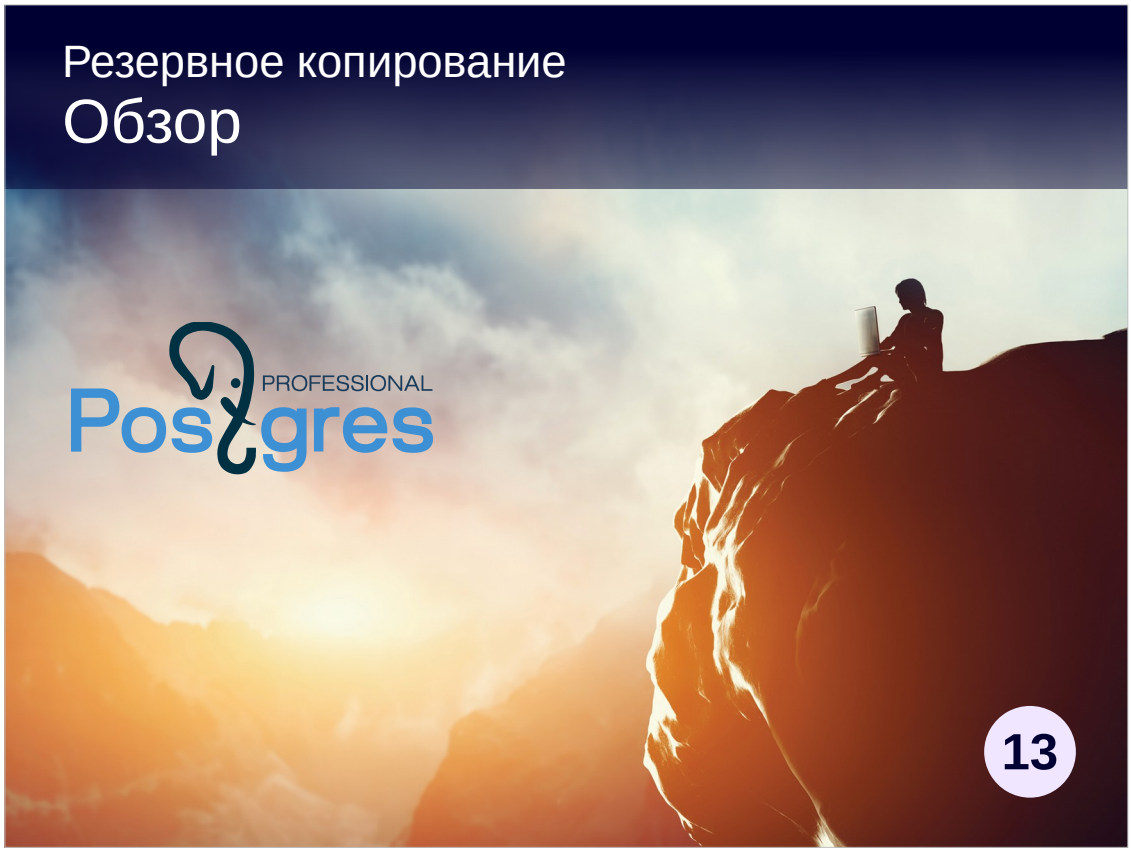


Авторы: Егор Рогов, Павел Лузанов, Илья Баштанов

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

Темы

Логическое резервное копирование

Физическое резервное копирование

Логическое копирование

Что такое логическое резервное копирование

Копия таблицы

Копия базы данных

Копия кластера

Логическая копия

Команды SQL для восстановления данных с нуля

+ можно сделать копию отдельного объекта или базы
+ можно восстановиться на кластере другой основной версии
+ можно восстановиться на другой архитектуре
− невысокая скорость

Существует два вида резервирования: логическое и физическое.

Логическое резервирование — это набор команд SQL, восстанавливающая кластер (или базу данных, или отдельный объект) с нуля.

Такая копия представляет собой, по сути, обычный текстовый файл, что дает известную гибкость. Например, можно сделать копию только тех объектов, которые нужны; можно отредактировать файл, изменив имена или типы данных и т. п.

Кроме того, команды SQL можно выполнить на другой версии СУБД (при наличии совместимости на уровне команд) или на другой архитектуре (то есть не требуется двоичная совместимость).

Однако для большой базы этот механизм неэффективен, поскольку выполнение команд займет много времени. К тому же восстановить систему из такой резервной копии можно только на тот момент, в который она была сделана.

https://postgrespro.ru/docs/postgresql/13/backup-dump

COPY: копия таблицы

Резервное копирование

вывод таблицы или результатов запроса в файл, на консоль или в программу

Восстановление

добавление строк из файла или с консоли к существующей таблице

Серверный вариант

команда SQL COPY

файл должен быть доступен пользователю postgres на сервере

Клиентский вариант

команда psql \COPY

файл должен быть доступен запустившему psql на клиенте

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

Команда позволяет записать таблицу (или результат произвольного запроса) либо в файл, либо на консоль, либо на вход произвольной программе. При этом можно указать ряд параметров, таких как формат (текстовый, csv или двоичный), разделитель полей, текстовое представление NULL и др.

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.

https://postgrespro.ru/docs/postgresql/13/sql-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 варианте обращение к файлу происходит на клиенте, а на сервер передается только содержимое.

https://postgrespro.ru/docs/postgresql/13/app-psql

COPY

Создадим базу данных и таблицу в ней.

```
=> CREATE DATABASE backup_overview;

CREATE DATABASE

=> \c backup_overview

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

=> CREATE TABLE t(id numeric, s text);

CREATE TABLE

=> INSERT INTO t VALUES (1, 'Привет!'), (2, ''), (3, NULL);

INSERT 0 3

=> SELECT * FROM t;

 id |    s
----+---------
  1 | Привет!
  2 |
  3 |
(3 rows)
```

Вот как выглядит таблица в выводе команды COPY:

```
=> COPY t TO STDOUT;

1       Привет!
2
3       \N
```

Обратите внимание на то, что пустая строка и NULL — разные значения, хотя, выполняя запрос, этого и не заметно.

Аналогично можно вводить данные:

```
=> TRUNCATE TABLE t;

TRUNCATE TABLE

=> COPY t FROM STDIN;
1       Hi there!
2
3       \N
\.

COPY 3
```

Проверим:

```
=> \pset null '<null>'

Null display is "<null>".

=> SELECT * FROM t;

 id |     s
----+-----------
  1 | Hi there!
  2 |
  3 | <null>
(3 rows)
```

pg_dump: копия базы

Резервное копирование

- выдает на консоль или в файл либо SQL-скрипт, либо архив в специальном формате с оглавлением
- поддерживает параллельное выполнение
- позволяет ограничить набор выгружаемых объектов (таблицы, схемы, только DML или только DDL и т. п.)

Восстановление

- SQL-скрипт — psql
- формат с оглавлением — pg_restore (позволяет ограничить набор объектов при восстановлении и поддерживает параллельное выполнение)
- новая база должна быть создана из шаблона template0
- заранее должны быть созданы роли и табличные пространства

Для создания полноценной резервной копии базы данных используется утилита pg_dump. В зависимости от указанных параметров, результатом работы является либо SQL-скрипт, содержащий команды, создающие выбранные объекты, либо файл в специальном формате с оглавлением.

Чтобы восстановить объекты из SQL-скрипта, достаточно прогнать его через psql.

https://postgrespro.ru/docs/postgresql/13/app-pgdump

Для восстановления резервной копии в специальном формате требуется другая утилита — pg_restore. Она читает файл и преобразует его в обычные команды psql. Преимущество в том, что набор объектов можно ограничить не при создании резервной копии, а уже при восстановлении. Кроме того, создание резервной копии в специальном формате и восстановление из нее может выполняться параллельно.

https://postgrespro.ru/docs/postgresql/13/app-pgrestore

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 относятся ко всему кластеру. После восстановления базы имеет смысл выполнить команду ANALYZE, которая соберет статистику.

pg_dumpall: копия кластера

Резервирование

- сохраняет весь кластер, включая роли и табличные пространства
- выдает на консоль или в файл SQL-скрипт
- параллельное выполнение не поддерживается, но можно выгрузить только глобальные объекты и воспользоваться pg_dump

Восстановление

- с помощью psql

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.

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 с помощью pg_dump.

https://postgrespro.ru/docs/postgresql/13/app-pg-dumpall

Утилита pg_dump

Посмотрим на результат работы утилиты pg_dump в простом формате (plain). Обратите внимание на то, в каком виде сохранены данные из таблицы.

Если в шаблон template1 вносились какие-либо изменения, они также попадут в резервную копию. Поэтому при восстановлении базы данных имеет смысл предварительно создать ее из шаблона template0 (указанный ключ --create добавляет нужные команды автоматически).

```
student$ pg_dump -d backup_overview --create

--
-- PostgreSQL database dump
--

-- Dumped from database version 13.7 (Ubuntu 13.7-1.pgdg22.04+1)
-- Dumped by pg_dump version 13.7 (Ubuntu 13.7-1.pgdg22.04+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;

--
-- Name: backup_overview; Type: DATABASE; Schema: -; Owner: student
--

CREATE DATABASE backup_overview WITH TEMPLATE = template0 ENCODING = 'UTF8' LOCALE = 'en_US.UTF-8';


ALTER DATABASE backup_overview OWNER TO student;

\connect backup_overview

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 numeric,
    s text
);


ALTER TABLE public.t OWNER TO student;

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

COPY public.t (id, s) FROM stdin;
1       Hi there!
2
3       \N
\.
```

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

В качестве примера использования скопируем таблицу в другую базу.

```
=> CREATE DATABASE backup_overview2;

CREATE DATABASE

student$ pg_dump -d backup_overview --table=t | psql -d backup_overview2

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

(1 row)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
COPY 3

student$ psql -d backup_overview2

  => SELECT * FROM t;

   id |     s
  ----+-----------
    1 | Hi there!
    2 |
    3 |
  (3 rows)
```

Физическое копирование

Что такое физическое резервное копирование

Холодное и горячее резервное копирование

Протокол репликации

Автономные резервные копии

Непрерывная архивация журналов предзаписи

Физическая копия

Используется механизм восстановления после сбоя:
копия данных и журналы предзаписи

+ скорость восстановления
+ можно восстановить кластер на определенный момент времени
− нельзя восстановить отдельную базу данных, только весь кластер
− восстановление только на той же основной версии и архитектуре

Физическое резервирование использует механизм восстановления после сбоев. Для этого требуются:

- копия файлов кластера (базовая резервная копия);

- набор журналов предзаписи, необходимых для восстановления согласованности.

Если файловая система уже согласована (копия снималась при корректно остановленном сервере), то журналы не требуются.

Однако наличие архива журналов позволяет из базовой резервной копии получить состояние кластера на любой момент времени. Таким образом можно восстановить резервную копию практически на момент сбоя (либо сознательно восстановить систему на некоторый момент в прошлом).

Высокая скорость восстановления и возможность создавать копию «на лету», не выключая сервер, делает физическое резервирование основным инструментом периодического резервного копирования.

https://postgrespro.ru/docs/postgresql/13/backup-file

https://postgrespro.ru/docs/postgresql/13/continuous-archiving

Горячо или холодно?

	Холодное резервирование		Горячее резервирование
Файловая система копируется при...	выключенном сервере	неаккуратно выключенном сервере	работающем сервере нужны специальные средства
Журналы предварительной записи...	не нужны	нужны с последней контрольной точки находятся в файловой системе	нужны за время копирования ФС сервер не должен удалить WAL раньше времени

Физическое резервирование так или иначе предполагает создание копии файловой системы.

Если копия создается при выключенном сервере, она называется «холодной». Такая копия либо содержит согласованные данные (если сервер был выключен аккуратно), либо содержит все необходимые для восстановления журналы (например, если используется снимок данных средствами операционной системы). Это упрощает восстановление, но требует останова сервера.

Если копия создается при работающем сервере (что требует определенных действий — просто так копировать файлы нельзя), она называется «горячей». В этом случае процедура сложнее, но позволяет обойтись без останова.

При горячем резервировании копия ФС будет содержать несогласованные данные. Однако механизм восстановления после сбоев можно успешно применить и к восстановлению из резервной копии. Для этого потребуются журналы предзаписи как минимум за время копирования файлов.

Автономная копия

Автономная копия содержит и файлы данных, и WAL

Резервное копирование — утилита pg_basebackup

- подключается к серверу по протоколу репликации
- выполняет контрольную точку
- копирует файловую систему в указанный каталог
- сохраняет все сегменты WAL, сгенерированные за время копирования

Восстановление

- разворачиваем созданную автономную копию
- запускаем сервер

Для создания горячей резервной копии существует утилита pg_basebackup.

Вначале утилита выполняет контрольную точку. Затем копируется файловая система кластера.

Все файлы WAL, сгенерированные сервером за время от контрольной точки до окончания копирования файлов, также копируются в резервную копию. Такая копия называется автономной, поскольку содержит в себе все необходимое для восстановления.

Для восстановления достаточно развернуть резервную копию и запустить сервер. При необходимости он выполнит восстановление согласованности с помощью имеющихся файлов WAL и будет готов к работе.

https://postgrespro.ru/docs/postgresql/13/app-pgbasebackup

Протокол репликации

Протокол

- получение потока журнальных записей
- команды управления резервным копированием и репликацией

Обслуживается процессом wal_sender

Параметр *wal_level* = replica

Слот репликации

- серверный объект для получения журнальных записей
- помнит, какая запись была считана последней
- сегмент WAL не удаляется, пока он полностью не прочитан через слот

Чтобы сохранить все необходимые для восстановления файлы WAL, сгенерированные сервером за время копирования файлов, утилита подключается к серверу по специальному протоколу репликации. Несмотря на название, это протокол используется не только для репликации (о которой пойдет речь в следующей теме), но и для резервного копирования. Протокол позволяет получать поток журнальных записей параллельно с копированием файлом.

Чтобы сервер не удалил необходимые файлы WAL преждевременно, может использоваться слот репликации.

Для того, чтобы подключение было возможно, необходим ряд настроек.

Во-первых, роль должна обладать атрибутом REPLICATION (или быть суперпользователем). Кроме того, этой роли должно быть выдано разрешение в конфигурационном файле pg_hba.conf.

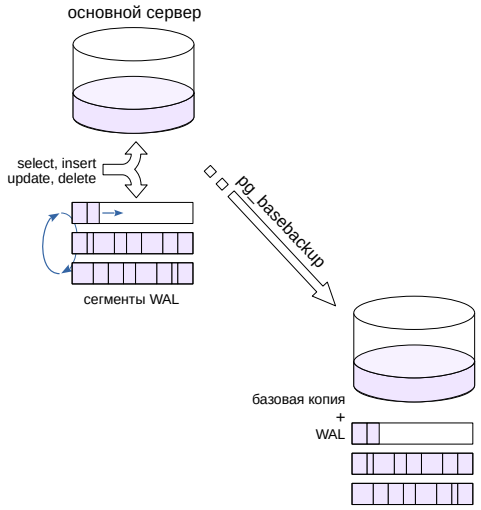
Во-вторых, параметр *max_wal_senders* должен быть установлен в достаточно большое значение. Этот параметр ограничивает число одновременно работающих процессов wal_sender, обслуживающих подключения по протоколу репликации.

В-третьих, параметр *wal_leve*l, определяющий количество информации в журнале, должен быть установлен в значение replica.

Начиная с версии 10, настройки по умолчанию уже включают все эти требования (при локальном подключении).

https://postgrespro.ru/docs/postgresql/13/protocol-replication

Автономная копия



На этом рисунке слева представлен основной сервер. Он обрабатывает поступающие запросы; при этом формируются записи WAL и изменяется состояние баз данных (сначала в буферном кеше, потом на диске). Сегменты WAL циклически перезаписываются (точнее, удаляются старые сегменты, так как имена файлов уникальны).

Внизу рисунка (а в реальной жизни — обычно на другом сервере) изображена созданная резервная копия — базовая копия данных и набор файлов WAL.

Автономная резервная копия

Значения параметров по умолчанию позволяют использовать репликацию:

```
=> SELECT name, setting
FROM pg_settings
WHERE name IN ('wal_level','max_wal_senders');
```

```
      name       | setting
-----------------+---------
 max_wal_senders | 10
 wal_level       | replica
(2 rows)
```

Разрешение на локальное подключение по протоколу репликации в pg_hba.conf также прописано по умолчанию (хотя это и зависит от конкретной пакетной сборки):

```
=> SELECT type, database, user_name, address, auth_method
FROM pg_hba_file_rules()
WHERE 'replication' = ANY(database);
```

```
 type  |   database    | user_name |  address  | auth_method
-------+---------------+-----------+-----------+-------------
 local | {replication} | {all}     | <null>    | trust
 host  | {replication} | {all}     | 127.0.0.1 | md5
 host  | {replication} | {all}     | ::1       | md5
(3 rows)
```

Еще один кластер баз данных replica был предварительно инициализирован на порту 5433. Убедимся, что кластер остановлен, с помощью утилиты пакета для Ubuntu:

```
student$ pg_lsclusters
```

```
Ver Cluster Port Status Owner    Data directory                 Log file
13  main    5432 online postgres /var/lib/postgresql/13/main    /var/log/postgresql/postgresql-13-main.log
13  replica 5433 down   postgres /var/lib/postgresql/13/replica /var/log/postgresql/postgresql-13-replica.log
```

Создадим резервную копию. Используем формат по умолчанию (plain):

```
student$ sudo rm -rf /home/student/basebackup

student$ pg_basebackup --pgdata=/home/student/basebackup
```

Заменим каталог кластера replica созданной копией, предварительно убедившись, что кластер остановлен:

```
student$ sudo pg_ctlcluster 13 replica status

pg_ctl: no server running

student$ sudo rm -rf /var/lib/postgresql/13/replica

student$ sudo mv /home/student/basebackup/ /var/lib/postgresql/13/replica
```

Файлы кластера должны принадлежать пользователю postgres.

```
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/replica
```

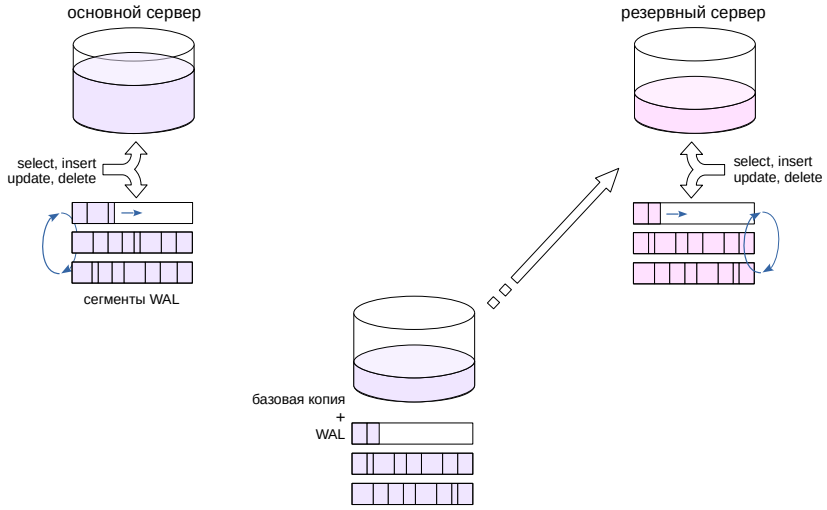
Проверим содержимое каталога:

```
student$ sudo ls -l /var/lib/postgresql/13/replica
```

```
total 336
-rw------- 1 postgres postgres    224 дек 20 11:29 backup_label
-rw------- 1 postgres postgres 262059 дек 20 11:29 backup_manifest
drwx------ 8 postgres postgres   4096 дек 20 11:29 base
drwx------ 2 postgres postgres   4096 дек 20 11:29 global
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_commit_ts
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_dynshmem
drwx------ 4 postgres postgres   4096 дек 20 11:29 pg_logical
drwx------ 4 postgres postgres   4096 дек 20 11:29 pg_multixact
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_notify
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_replslot
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_serial
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_snapshots
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_stat
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_stat_tmp
```

```
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_subtrans
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_tblspc
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_twophase
-rw------- 1 postgres postgres      3 дек 20 11:29 PG_VERSION
drwx------ 3 postgres postgres   4096 дек 20 11:29 pg_wal
drwx------ 2 postgres postgres   4096 дек 20 11:29 pg_xact
-rw------- 1 postgres postgres     88 дек 20 11:29 postgresql.auto.conf
```

Восстановление



При восстановлении базовая резервная копия, включающая необходимые файлы WAL, разворачивается, например, на другом сервере (справа на рисунке).

После старта сервера он восстанавливает согласованность и приступает к работе. Восстановление происходит на тот момент времени, на который была сделана резервная копия. Разумеется, за прошедшее время основной сервер может уйти далеко вперед.

Восстановление из автономной резервной копии

В процессе запуска произойдет восстановление из резервной копии.

```
student$ sudo pg_ctlcluster 13 replica start
```

Теперь оба сервера работают одновременно и независимо.

Основной сервер:

```
=> INSERT INTO t VALUES (4, 'Основной сервер');
```

```
INSERT 0 1
```

```
=> SELECT * FROM t;
```

```
 id |        s
----+-----------------
  1 | Hi there!
  2 |
  3 | <null>
  4 | Основной сервер
(4 rows)
```

Сервер, восстановленный из резервной копии:

```
student$ psql -p 5433 -d backup_overview
```

```
=> INSERT INTO t VALUES (4, 'Резервная копия');
```

```
INSERT 0 1
```

```
=> SELECT * FROM t;
```

```
 id |        s
----+-----------------
  1 | Hi there!
  2 |
  3 |
  4 | Резервная копия
(4 rows)
```

Архив журналов

Файловый архив

- сегменты WAL копируются в архив по мере заполнения
- механизм работает под управлением сервера
- неизбежны задержки попадания данных в архив

Потоковый архив

- в архив постоянно записывается поток журнальных записей
- требуются внешние средства
- задержки минимальны

Дальнейшее развитие идеи горячей резервной копии: раз у нас есть копия файловой системы и журналы упреждающей записи, то, постоянно сохраняя новые журналы, мы сможем восстановить систему не только на момент копирования файлов, но и вообще на произвольный момент.

Для этого есть два способа. Первый состоит в том, чтобы не просто перезаписывать файлы WAL, а предварительно откладывать их куда-то в архив. Это можно реализовать специальными настройками сервера. К сожалению, при таком варианте файл WAL не попадет в архив, пока сервер не переключится на запись в другой файл.

Второй способ — постоянно читать журнальные записи по протоколу репликации и записывать их в тот же архив. При таком варианте задержки будут минимальны, но потребуется настроить отдельную от сервера утилиту для получения данных из потока.

Файловый архив журналов

Процесс archiver

Параметры

archive_mode = on

archive_command — команда shell для копирования сегмента WAL в отдельное хранилище

Алгоритм

при переключении сегмента WAL вызывается команда *archive_command*

если команда завершается со статусом 0, сегмент удаляется

если команда возвращает не 0 (в частности, если команда не задана), сегмент остается до тех пор, пока попытка не будет успешной

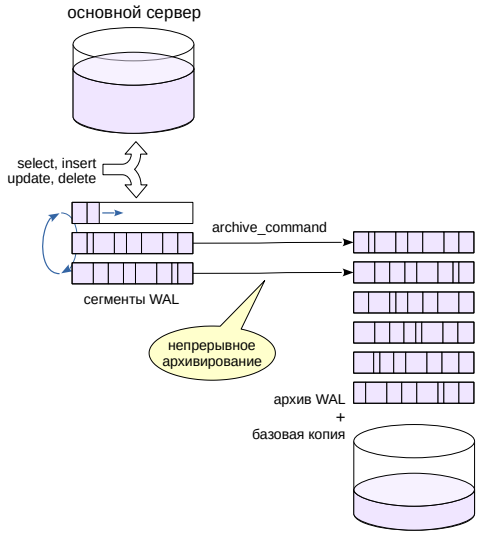
Файловый архив реализуется фоновым процессом archiver.

PostgreSQL позволяет определить для копирования произвольную команду shell в параметре *archive_command*. Сам механизм включается параметром *archive_mode* = on.

Общий алгоритм таков. При заполнении очередного сегмента WAL вызывается команда копирования. Если она завершается с нулевым статусом, то сегмент может быть удален. Если же нет, то сегмент (и следующие за ним) не будет удален, а сервер будет периодически пытаться выполнить команду, пока не получит 0.

https://postgrespro.ru/docs/postgresql/13/continuous-archiving

Файловый архив журналов



На этом рисунке изображен основной сервер с настроенным непрерывным архивированием: заполненные сегменты WAL копируются в отдельный архив с помощью команды в параметре *archive_command*. Обычно такой архив расположен на другом сервере.

Там же создается базовая резервная копия (или несколько копий, сделанных в разные моменты времени).

Потоковый архив журналов

Утилита pg_receivewal

- подключается по протоколу репликации (можно использовать слот) и направляет поток записей WAL в файлы-сегменты
- стартовая позиция — начало сегмента, следующего за последним заполненным сегментом, найденным в каталоге, или начало текущего сегмента сервера, если каталог пустой
- в отличие от файлового архива, записи пишутся постоянно
- при переходе на новый сервер надо перенастраивать параметры

Другое решение — использование утилиты pg_receivewal для записи сегментов в архив по протоколу потоковой репликации.

Обычно утилита запускается на отдельном «архивном» сервере и подключается к мастеру с параметрами, указанными в ключах. Утилита может (и должна) использовать слот репликации, чтобы гарантированно не потерять записи.

Утилита формирует файлы аналогично тому, как это делает сервер, и записывает их в указанный каталог. Еще не заполненные сегменты записываются с префиксом .partial.

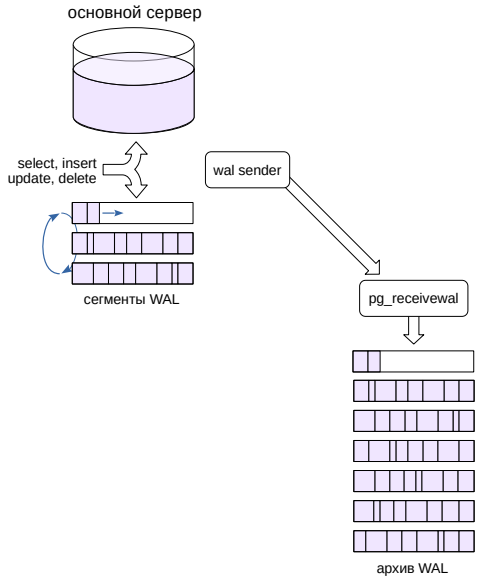
Архивирование всегда начинается с начала сегмента, следующего за последним заполненным сегментом архива. Если архив пуст (первый запуск), архивирование начинается с начала текущего сегмента.

При переходе на новый сервер утилиту требуется остановить и запустить заново с соответствующими параметрами.

Требуется учесть, что сама по себе утилита не запускается автоматически (как сервис) и не демонизируется.

https://postgrespro.ru/docs/postgresql/13/app-pgreceivewal

Потоковый архив журналов



Утилита pg_receivewal подключается к серверу по протоколу потоковой репликации. Подключение обрабатывается отдельным процессом wal sender (это необходимо учесть при установке параметра *max_wal_senders*).

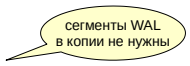
Утилита записывает данные, не дожидаясь получения всего сегмента.

Резервная копия + архив

Настроенное непрерывное архивирование журналов

Резервное копирование — pg_basebackup

- подключается к серверу по протоколу репликации
- выполняет контрольную точку
- копирует файловую систему в указанный каталог



Восстановление

- разворачиваем резервную копию
- задаем конфигурационные параметры
(чтение WAL из архива, указание целевой точки восстановления)
- создаем сигнальный файл recovery.signal
- запускаем сервер

Для создания резервной копии при настроенном непрерывном архивировании используется та же утилита pg_basebackup, только с другим набором параметров. Единственное отличие состоит в том, что в резервную копию не сохраняются файлы WAL, поскольку они уже есть в архиве.

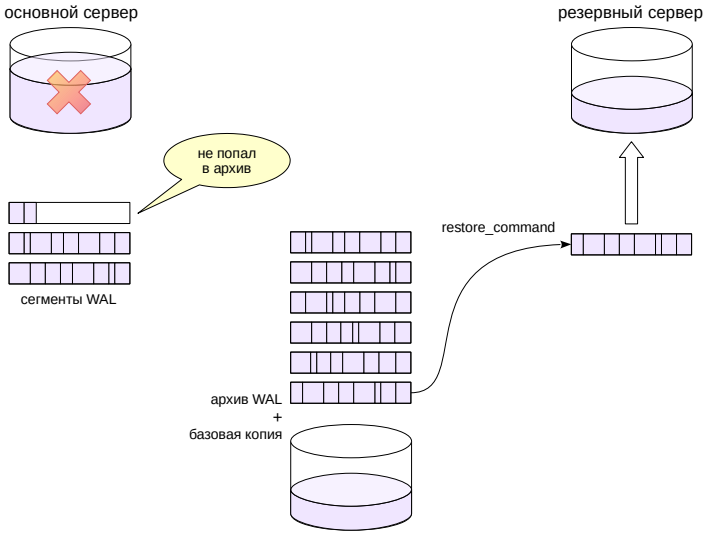
Восстановление в таком случае выполняется более сложно. Помимо разворачивания базовой резервной копии, требуется задать настройки восстановления:

- команду *restore_command* (обратную *archive_command* — она копирует нужные файлы из архива в каталог сервера);

- целевую точку восстановления.

Кроме этого, нужен сигнальный файл recovery.signal, наличие которого при старте сервера означает указание войти в режим управляемого восстановления (содержимое файла игнорируется).

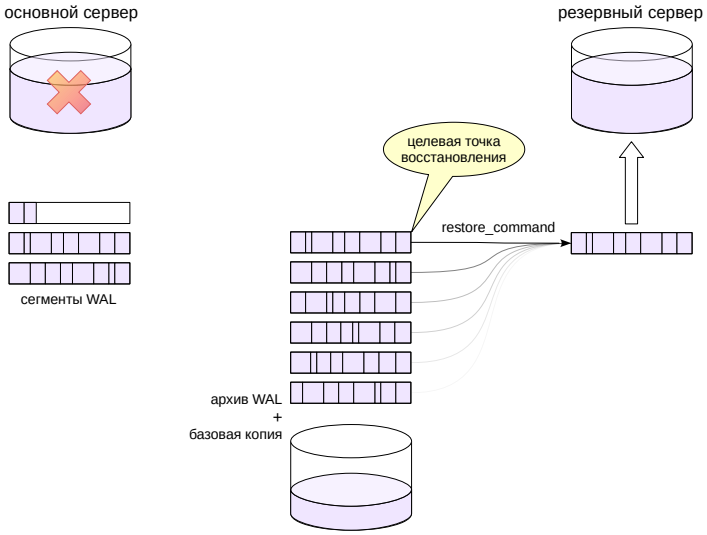
Восстановление



Процедура восстановления (например, при выходе сервера из строя основного сервера), выполняется следующим образом. На другом (или на том же) сервере разворачивается базовая резервная копия и создается файл recovery.signal. Сервер запускается и начинает читать сегменты WAL из архива, используя restore_command, и применять их.

Следует обратить внимание, что при файловой архивации последний незаполненный сегмент WAL основного сервера не попадет в архив. Однако сегмент можно вручную подложить резервному серверу в каталог pg_wal, если есть такая возможность. Конечно, таких сегментов может оказаться и несколько, но только в случае какого-то сбоя при архивировании.

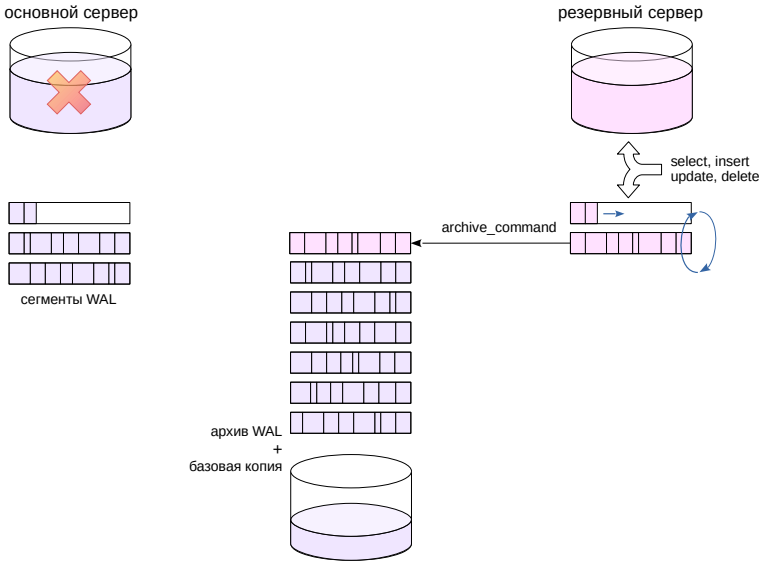
Восстановление



Резервный сервер применяет все доступные записи WAL, читая сегменты из каталога pg_wal (при отсутствии сегмента делая попытку скопировать его из архива), тем самым доводя состояние баз данных до актуального. Максимально возможная потеря — незаполненный сегмент WAL, не попавший в архив, если его по каким-то причинам невозможно скопировать.

По умолчанию применяются все доступные журнальные записи, а указание целевой точки восстановления позволяет остановить их применение в желаемый момент.

Восстановление



После этого резервный сервер переходит в обычный режим работы: принимает запросы, записывает сегменты WAL в архив и так далее, выступая в качестве нового полноценного основного сервера.

Если развернутый сервер предполагается использовать вместо основного, то его имеет смысл располагать на таком же или, по крайней мере, сравнимом по характеристикам аппаратуры сервере, чтобы избежать снижения производительности.

Итоги

Логическая резервная копия —
команды SQL для восстановления состояния объектов

- команда copy, утилиты pg_dump и pg_dumpall

Физическая резервная копия —
копия файлов кластера + набор файлов WAL

- утилита pg_basebackup

Архив журнальных файлов

- файловый или потоковый
- позволяет восстановить систему на произвольный момент времени

Практика

1. Создайте базу данных и таблицу в ней с несколькими строками.
2. Сделайте логическую копию базы данных с помощью утилиты pg_dump.

   Удалите базу данных и восстановите ее из сделанной копии.
3. Сделайте автономную физическую резервную копию кластера с помощью утилиты pg_basebackup.

   Измените таблицу.

   Восстановите новый кластер из сделанной резервной копии и проверьте, что база данных не содержит более поздних изменений.

3. В виртуальной машине курса уже создан кластер replica на порту 5433. Используйте этот кластер, для того чтобы восстановить резервную копию.

Каталог кластера располагается в /var/lib/postgresql/13/replica.

Для подключения укажите номер порта: psql -p 5433

1. База данных и таблица

```
=> CREATE DATABASE backup_overview;

CREATE DATABASE

=> \c backup_overview

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

=> CREATE TABLE t(n integer);

CREATE TABLE

=> INSERT INTO t VALUES (1), (2), (3);

INSERT 0 3
```

2. Логическая резервная копия

Создаем резервную копию:

```
student$ pg_dump -f ~/backup_overview.dump -d backup_overview --create
```

Удаляем базу данных и восстанавливаем ее из копии:

```
=> \c postgres

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

=> DROP DATABASE backup_overview;

DROP DATABASE

student$ psql -f ~/backup_overview.dump

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

(1 row)

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

(1 row)

SET
SET
SET
SET
SET
SET
CREATE TABLE
ALTER TABLE
COPY 3

=> \c backup_overview

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

=> SELECT * FROM t;
```

```
 n
---
 1
 2
 3
(3 rows)
```

3. Физическая автономная резервная копия

Создаем резервную копию:

```
student$ sudo rm -rf /home/student/basebackup
```

```
student$ pg_basebackup --pgdata=/home/student/basebackup
```

Убеждаемся, что второй сервер остановлен, и выкладываем резервную копию:

```
student$ sudo pg_ctlcluster 13 replica status
```

```
pg_ctl: no server running
```

```
student$ sudo rm -rf /var/lib/postgresql/13/replica
```

```
student$ sudo mv /home/student/basebackup/ /var/lib/postgresql/13/replica
```

```
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/replica
```

Изменяем таблицу:

```
=> DELETE FROM t;
```

```
DELETE 3
```

Запускаем сервер из резервной копии:

```
student$ sudo pg_ctlcluster 13 replica start
```

```
student$ psql -p 5433 -d backup_overview
```

```
=> SELECT * FROM t;
```

```
 n
---
 1
 2
 3
(3 rows)
```

Практика+

1. Организуйте потоковую архивацию кластера main с помощью утилиты pg_receivewal.
2. Сделайте автономную резервную копию кластера main (без WAL) с помощью утилиты pg_basebackup.
3. В кластере main создайте базу данных и таблицу в ней.
4. Выполните восстановление кластера replica из базовой копии с использованием архива. Убедитесь, что база и таблица тоже восстановились.

Кластер replica находится в /var/lib/postgresql/13/replica.

Текущий файл, записываемый утилитой pg_receivewal, имеет суффикс.partial, по окончании записи файл переименовывается. При восстановлении наряду с обычными сегментами нужно использовать и последний недозаписанный файл.

Для подключения к кластеру replica укажите номер порта: psql -p 5433.

1. Потоковый архив

Обратите внимание, что часть команд выполняется от имени пользователя postgres, а часть — от имени student.

Создаем каталог для архива WAL:

```
postgres$ mkdir /var/lib/postgresql/archive
```

Создаем слот, чтобы в архиве не было пропусков:

```
postgres$ pg_receivewal --create-slot --slot=archive
```

Запускаем утилиту pg_receivewal в фоновом режиме. Для этого следующую команду нужно выполнить в отдельном окне терминала или добавить в конце командной строки символ &.

```
postgres$ pg_receivewal -D /var/lib/postgresql/archive --slot=archive
```

```
student$ sudo ls -l /var/lib/postgresql/archive
```

```
total 16384
-rw------- 1 postgres postgres 16777216 дек 20 11:32 000000010000000000000010.partial
```

2. Базовая физическая копия без журнала

```
student$ pg_basebackup --wal-method=none --pgdata=/home/student/basebackup
```

```
NOTICE:  WAL archiving is not enabled; you must ensure that all required WAL segments are copied through other means to complete the backup
```

3. Новые база данных и таблица

```
=> CREATE DATABASE backup_overview;
```

```
CREATE DATABASE
```

```
=> \c backup_overview
```

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

```
=> CREATE TABLE t(n integer);
```

```
CREATE TABLE
```

```
=> INSERT INTO t VALUES (1), (2), (3);
```

```
INSERT 0 3
```

4. Настройка восстановления

Убеждаемся, что второй сервер остановлен, и выкладываем резервную копию:

```
student$ sudo pg_ctlcluster 13 replica status
```

```
pg_ctl: no server running
```

```
student$ sudo rm -rf /var/lib/postgresql/13/replica
```

```
student$ sudo mv /home/student/basebackup/ /var/lib/postgresql/13/replica
```

При восстановлении также используем частично записанный сегмент:

```
student$ echo "restore_command = 'cp /var/lib/postgresql/archive/%f %p || cp /var/lib/postgresql/archive/%f.partial %p'" | sudo tee /var/lib/postgresql/13/replica/postgresql.auto.conf
```

```
restore_command = 'cp /var/lib/postgresql/archive/%f %p || cp /var/lib/postgresql/archive/%f.partial %p'
```

```
student$ touch /var/lib/postgresql/13/replica/recovery.signal
```

```
student$ sudo chown -R postgres:postgres /var/lib/postgresql/13/replica
```

Запустим сервер и проверим результат:

```
student$ sudo pg_ctlcluster 13 replica start
```

```
student$ psql -p 5433 -d backup_overview
```

```
=> SELECT * FROM t;
```

```
 n
---
 1
 2
 3
(3 rows)
```

Архивация больше не нужна. Остановим утилиту и удалим слот, чтобы он не мешал очистке WAL.

```
student$ sudo killall -9 pg_receivewal
```

```
postgres$ pg_receivewal --drop-slot --slot=archive
```