Postgres PROFESSIONAL

Логическое резервирование

Темы

Логическое и физическое резервирование

Копирование отдельных таблиц

Резервирование и восстановление баз данных и кластера

Виды резервирования

Логическое резервирование

команды SQL для восстановления данных с нуля

\+ можно сделать копию отдельного объекта или базы

\+ можно восстановиться на другой версии или архитектуре

− невысокая скорость

Физическое резервирование

используется механизм восстановления после сбоя:
копия файловой системы и журналы упреждающей записи

\+ скорость восстановления

\+ можно восстановиться на определенный момент времени

− только весь кластер

− требуется много дискового пространства

COPY: копия таблицы

Резервирование

```
copy tbl to 'file';
copy (select * from tbl) to stdout
  (format text,
   delimiter ',',
   null '<null>');
```

вывод таблицы в файл
вывод запроса на консоль
формат (text, csv, binary)
разделитель полей
представление null

Восстановление

строки добавляются к таблице

```
copy tbl from 'file';
copy tbl from stdin;
```

прочитать из файла
прочитать с консоли
до EOF или `\.`

…

\COPY: копия таблицы

Клиентский вариант

тот же синтаксис, только с обратной косой чертой

```
\copy tbl to 'file';
\copy tbl from 'file';
```

Отличия

серверный вариант	*клиентский вариант*
команда SQL	команда psql
файл должен быть доступен пользователю postgres на сервере	файл должен быть доступен запустившему psql на клиенте

pg_dump: копия объектов

Резервирование

выдает на консоль скрипт для psql

`pg_dump --table=`*`tbl`* `-d` *`db`*	таблицы по шаблону
`pg_dump --schema=`*`scm`* `-d` *`db`*	схемы со всеми объектами

дополнительные параметры

`--data-only`	только DML
`--schema-only`	только DDL
`--clean`	предварительное удаление
`--compress=`*`N`*	сжатие (нужен zlib), *N*=0..9
`--file=`*`dump`* `(-f)`	сохранение скрипта в файл

Восстановление

`psql -f` *`dump`*

pg_dump: копия базы

Резервирование

выдает скрипт для psql

`pg_dump -d` *db*

дополнительные параметры

`--clean`	включить команды удаления объектов
`--create`	включить команду создания базы

Восстановление

`psql -f` *dump*

новая база должна быть создана из template0 (`--create` это учитывает)

заранее должны быть созданы роли и табличные пространства

после восстановления имеет смысл выполнить ANALYZE

pg_dump: формат custom

Резервирование

сохранение в промежуточный формат с оглавлением

можно выбрать объекты не при резервировании, а при восстановлении

сжатие (zlib) включено по умолчанию

```
pg_dump --format=custom -f dump -d db
```

Восстановление всей базы

`pg_restore -d db dump`	прямое соединение с базой db
`pg_restore dump | psql`	генерация команд и передача их psql

дополнительные параметры

`--clean`	включает удаление объектов из базы
`--create`	предварительное создание базы (той, что резервировалась, а не из `-d`)

pg_dump: формат custom

Ограничение части объектов при восстановлении

дополнительные параметры, аналогичные pg_dump

`--table=`*`tbl`*	таблицы по шаблону
`--index=`*`idx`*	индексы
`--function=`*`fun(arg,...)`*	функции
`--trigger=`*`trg`*	триггеры
`--schema=`*`scm`*	схемы со всеми объектами

Восстановление объектов по списку

получаем список объектов (оглавление резервной копии)

```
pg_restore --list dump >db.list
```

восстанавливаем, используя отредактированный вручную файл

```
pg_restore --use-list=db.list -d db dump
```

pg_dump: формат directory

Резервирование

- каталог в качестве промежуточного формата
- можно выбрать объекты не при резервировании, а при восстановлении
- сжатие (zlib) включено по умолчанию
- допускается параллельная работа в несколько потоков

```
pg_dump --format=directory --jobs=N -f dump -d db
```

Восстановление

- параллельно в несколько потоков (работает также для формата custom)

```
pg_restore -d db --jobs=N dump
```

- в остальном аналогично ранее рассмотренному

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	—

pg_dumpall: копия кластера

Резервирование

сохраняет весь кластер, включая роли и табличные пространства

запуск от имени суперпользователя

генерирует скрипт для psql (только один формат)

`pg_dumpall -f` *`dump`*

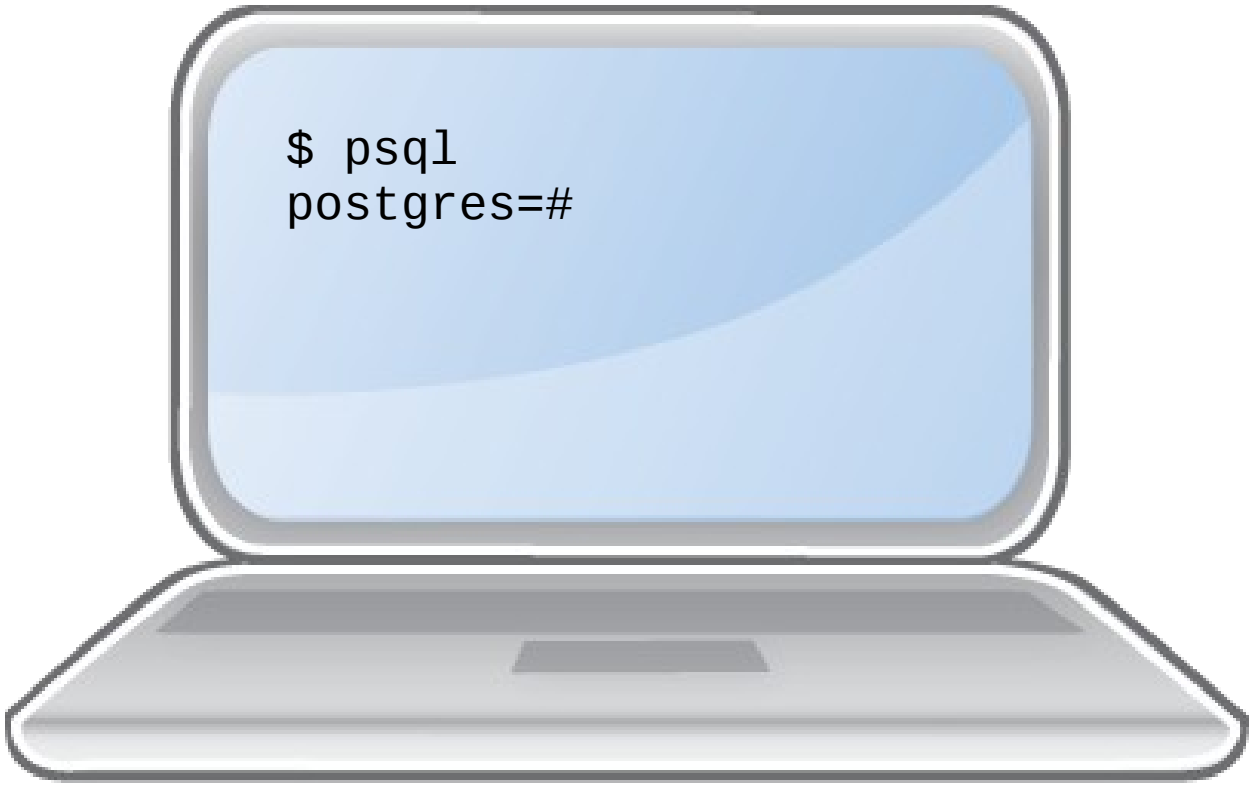
дополнительные параметры

`--clean`	включает удаление баз данных, ролей и табличных пространств
`--globals-only`	выгружает только роли и табличные пространства

Восстановление

`psql -f` *`dump`*

Демонстрация



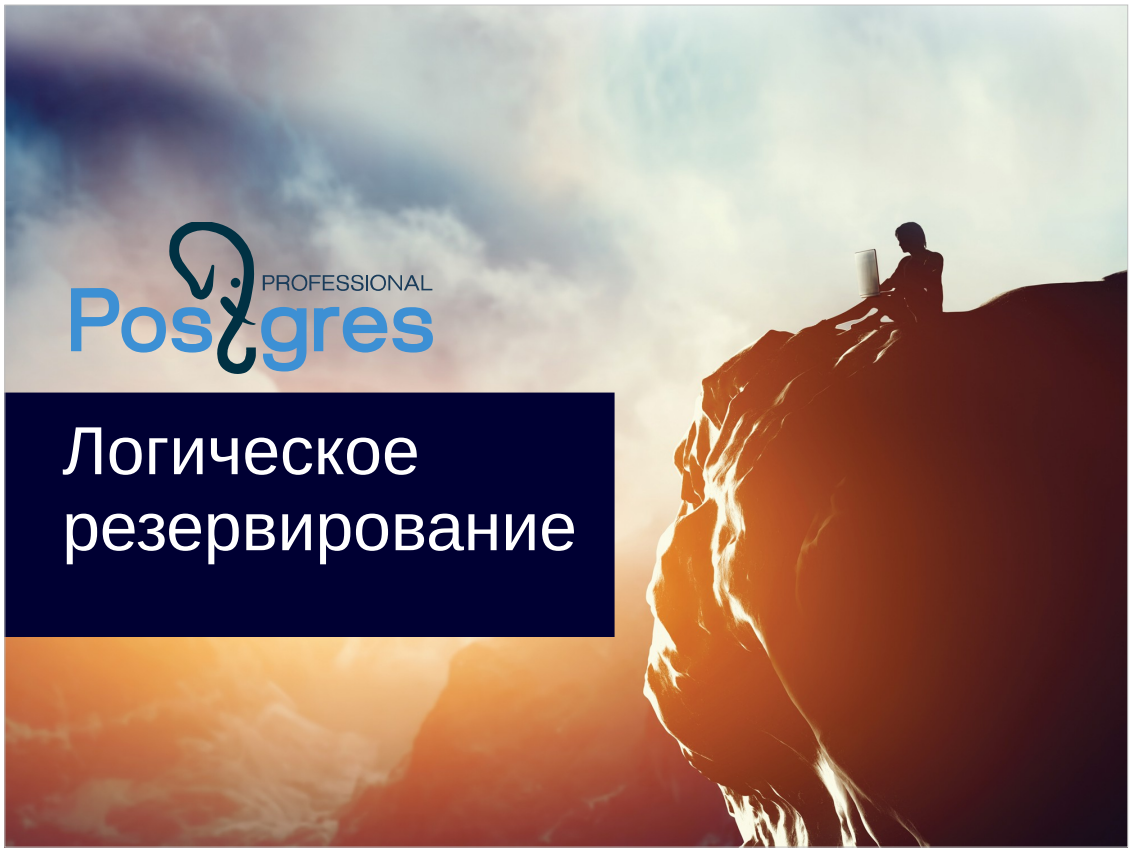
Итоги

Узнали про виды резервирования

Рассмотрели, как сделать логическую резервную копию кластера, базы данных или отдельных объектов,
и как восстановить данные из этой копии

Практика

1. Создайте базу данных DB16.
2. Создайте таблицу с несколькими строками в этой БД.
3. Создайте резервную копию базы данных.
4. Удалите таблицу.
5. Восстановите таблицу из резервной копии в БД postgres.
6. Удалите базу данных.
7. Восстановите базу данных из резервной копии.



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

Темы

Логическое и физическое резервирование

Копирование отдельных таблиц

Резервирование и восстановление баз данных и кластера

Виды резервирования

Логическое резервирование

команды SQL для восстановления данных с нуля

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

Физическое резервирование

используется механизм восстановления после сбоя:
копия файловой системы и журналы упреждающей записи

\+ скорость восстановления
\+ можно восстановиться на определенный момент времени
− только весь кластер
− требуется много дискового пространства

Существует два вида резервирования: логическое и физическое.

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

Команды можно выполнить на другой версии СУБД (при наличии совместимости на уровне команд) или на другой архитектуре (не требуется двоичная совместимость). Однако для большой базы выполняться они будут долго.

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

- копия файловой системы

- набор журналов упреждающей записи, необходимых для восстановления согласованности

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

Однако наличие журналов позволяет «догнать» состояние кластера на любой момент. Таким образом можно иметь резервную копию практически на момент сбоя (либо сознательно восстановить систему на некоторый момент в прошлом).

Восстановление происходит быстро, однако для хранения копии файловой системы и журналов требуется много дополнительного места.

http://www.postgresql.org/docs/current/static/backup.html

COPY: копия таблицы

Резервирование

```
copy tbl to 'file';                    вывод таблицы в файл
copy (select * from tbl) to stdout     вывод запроса на консоль
  (format text,                        формат (text, csv, binary)
   delimiter ',',                      разделитель полей
   null '<null>');                     представление null
```

Восстановление

строки добавляются к таблице

```
copy tbl from 'file';                  прочитать из файла
copy tbl from stdin;                   прочитать с консоли
                                       до EOF или \.
```

…

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. 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, новый 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: команда copy from не приводит к выполнению правил (rules), хотя ограничения целостности и триггеры выполняются.

http://www.postgresql.org/docs/current/static/sql-copy.html

\COPY: копия таблицы

Клиентский вариант

тот же синтаксис, только с обратной косой чертой

```
\copy tbl to 'file';
\copy tbl from 'file';
```

Отличия

серверный вариант	*клиентский вариант*
команда SQL	команда psql
файл должен быть доступен пользователю postgres на сервере	файл должен быть доступен запустившему psql на клиенте

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.

http://www.postgresql.org/docs/current/static/app-psql.html

pg_dump: копия объектов

Резервирование

выдает на консоль скрипт для psql

`pg_dump --table=`*`tbl`* `-d` *`db`*	таблицы по шаблону
`pg_dump --schema=`*`scm`* `-d` *`db`*	схемы со всеми объектами

дополнительные параметры

`--data-only`	только DML
`--schema-only`	только DDL
`--clean`	предварительное удаление
`--compress=`*`N`*	сжатие (нужен zlib), *N*=0..9
`--file=`*`dump`* `(-f)`	сохранение скрипта в файл

Восстановление

`psql -f` *`dump`*

Для создания полноценной резервной копии базы данных используется утилита pg_dump.

Если не указать имя файла (--file), то утилита выведет результат на консоль. А результатом является скрипт, предназначенный для psql, и содержащий команды, создающие указанные объекты.

Сначала рассмотрим простой пример с резервированием отдельных таблиц или объектов в схеме.

Шаблон имени таблицы указывается в ключе --table. Шаблон для схемы указывается в ключе --schema (в этом случае будут сохранены все объекты в этой схеме).

Ключи --data-only и --schema-only являются взаимоисключающими: первый сохраняет только данные, а второй — только определение объектов.

Можно указать сжатие (--compress), которое по умолчанию выключено.

Чтобы восстановить объекты из резервной копии, достаточно прогнать полученный скрипт через psql.

Если восстанавливаемые объекты присутствуют в базе, удобно задать ключ --clean, который добавит в скрипт команды по предварительному удалению объектов.

http://www.postgresql.org/docs/current/static/app-pgdump.html

pg_dump: копия базы

Резервирование

выдает скрипт для psql

```
pg_dump -d db
```

дополнительные параметры

`--clean`	включить команды удаления объектов
`--create`	включить команду создания базы

Восстановление

```
psql -f dump
```

новая база должна быть создана из template0 (`--create` это учитывает)

заранее должны быть созданы роли и табличные пространства

после восстановления имеет смысл выполнить ANALYZE

Чтобы сделать резервную копию всей базы, не надо ограничивать список объектов с помощью --table или --schema.

Полезны ключи --clean (добавляет в скрипт команды для удаления базы данных) и --create (добавляет команды для создания базы данных).

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 будет выгружены в резервную копию.

После восстановления базы имеет смысл выполнить команду ANALYZE, которая соберет статистику.

pg_dump: формат custom

Резервирование

сохранение в промежуточный формат с оглавлением

можно выбрать объекты не при резервировании, а при восстановлении

сжатие (zlib) включено по умолчанию

```
pg_dump --format=custom -f dump -d db
```

Восстановление всей базы

`pg_restore -d db dump`	прямое соединение с базой db
`pg_restore dump | psql`	генерация команд и передача их psql

дополнительные параметры

`--clean`	включает удаление объектов из базы
`--create`	предварительное создание базы (той, что резервировалась, а не из `-d`)

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

Формат custom (--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.

Утилита pg_restore понимает многие параметры из репертуара pg_dump. Например, при указании ключа --clean она сгенерирует команды для удаления объектов. Ключ --create предварительно создаст базу данных (не ту, которая указана в -d, а ту, имя которой указано в резервной копии — и восстановление будет происходить в ней же).

http://www.postgresql.org/docs/current/static/app-pgrestore.html

pg_dump: формат custom

Ограничение части объектов при восстановлении

дополнительные параметры, аналогичные pg_dump

`--table=`*`tbl`*	таблицы по шаблону
`--index=`*`idx`*	индексы
`--function=`*`fun(arg,...)`*	функции
`--trigger=`*`trg`*	триггеры
`--schema=`*`scm`*	схемы со всеми объектами

Восстановление объектов по списку

получаем список объектов (оглавление резервной копии)

```
pg_restore --list dump >db.list
```

восстанавливаем, используя отредактированный вручную файл

```
pg_restore --use-list=db.list -d db dump
```


Чтобы восстановить только часть объектов, можно воспользоваться одним из двух подходов.

Во-первых, можно ограничить объекты аналогично тому, как они ограничиваются в pg_dump. Например, ключ --table задает шаблон имени таблицы, ключ --schema задает шаблон имени схемы.

Имеются и дополнительные ключи, такие как --index (восстановление указанных индексов), --function (функции) и --trigger (триггеры).

Во-вторых, можно получить из оглавления список объектов, содержащихся в резервной копии (ключ --list). Затем этот список можно отредактировав вручную, удалив ненужное и, возможно, изменив порядок строк.

Далее при восстановлении отредактированный список подается на вход pg_restore (ключ --use-list).

pg_dump: формат directory

Резервирование

- каталог в качестве промежуточного формата
- можно выбрать объекты не при резервировании, а при восстановлении
- сжатие (zlib) включено по умолчанию
- допускается параллельная работа в несколько потоков

```
pg_dump --format=directory --jobs=N -f dump -d db
```

Восстановление

- параллельно в несколько потоков (работает также для формата custom)

```
pg_restore -d db --jobs=N dump
```

- в остальном аналогично ранее рассмотренному

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 --jobs).

Разумеется, несмотря на параллельное выполнение, копия будет содержать согласованные данные. Это достигается тем, что все параллельно работающие процессы будут разделять общий снимок данных.

Восстановление также возможно в несколько потоков, причем 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.

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.

pg_dumpall: копия кластера

Резервирование

сохраняет весь кластер, включая роли и табличные пространства

запуск от имени суперпользователя

генерирует скрипт для psql (только один формат)

```
pg_dumpall -f dump
```

дополнительные параметры

`--clean`	включает удаление баз данных, ролей и табличных пространств
`--globals-only`	выгружает только роли и табличные пространства

Восстановление

```
psql -f dump
```


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

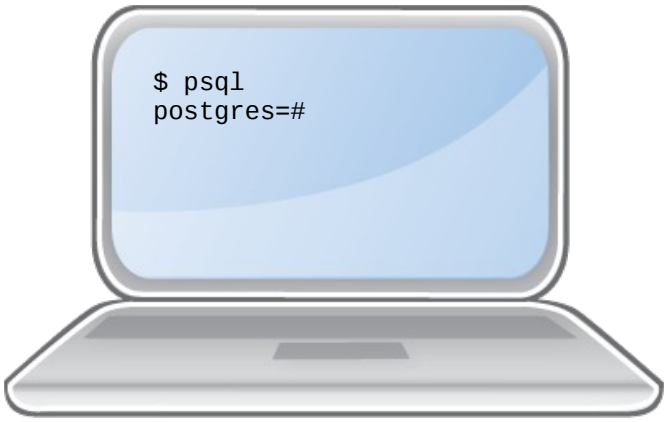
Поскольку pg_dumpall требуется доступ ко всем объектам всех БД, имеет смысл запускать ее от имени суперпользователя. Утилита по очереди подключается к каждой БД кластера и выгружает информацию с помощью pg_dump. Кроме того, она сохраняет и данные, относящиеся к кластеру в целом.

Ключ --clean позволяет включить в скрипт команды для предварительного удаления всех баз данных, ролей и табличных пространств.

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.

http://www.postgresql.org/docs/current/static/app-pg-dumpall.html

Демонстрация



Итоги

Узнали про виды резервирования

Рассмотрели, как сделать логическую резервную копию кластера, базы данных или отдельных объектов,
и как восстановить данные из этой копии

Практика

1. Создайте базу данных DB16.
2. Создайте таблицу с несколькими строками в этой БД.
3. Создайте резервную копию базы данных.
4. Удалите таблицу.
5. Восстановите таблицу из резервной копии в БД postgres.
6. Удалите базу данных.
7. Восстановите базу данных из резервной копии.


```
# create database db16;

# \c db16
# create table t(n numeric);
# insert into t values (1), (2), (3);

$ pg_dump --format=custom -f db16.dump -d db16
-- подойдет и формат directory

# drop table t;

$ pg_restore --table t -d postgres db16.dump

# \c postgres
# drop database db16;

$ pg_restore --create -d postgres db16.dump
```