Министерство науки и образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский государственный технический университет имени   
Н.Э. Баумана

(национальный исследовательский университет)»

(МГТУ им. Н.Э. Баумана)



**УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ К ЛАБОРАТОРНОЙ РАБОТЕ**

**ПО КУРСУ «БАЗЫ ДАННЫХ»**

**Лабораторная работа №2** **«Копирование и восстановление баз данных в PostgreSQL»**

Авторы:

Кудрявцев А.П., [kudryavtsevap@bmstu.ru](mailto:kudryavtsevap@bmstu.ru)

Фомин М.М.

Москва, 2025

1. **Логическое резервирование**

Логическое резервирование — набор команд SQL, восстанавливающих кластер или базу данных или отдельную таблицу. Создаются необходимые объекты и наполняются данными.

Восстановление 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.

Небольшое отступление. Команда CREATE DATABASE выполняет копирование одной из существующих системных баз данных – template1, либо template0. По умолчанию копируется стандартная системная база template1. Таким образом, template1 это - шаблон, на основе которого создаются новые базы. Если добавить объекты в template1, то впоследствии они будут копироваться в новые базы данных. Это позволяет внести изменения в стандартный набор объектов.

Вторая системная база template0. При инициализации она содержит те же самые объекты, что и template1, предопределённые в рамках устанавливаемой версии PostgreSQL. В template0 не следует вносить никакие изменения после инициализации кластера. Если в команде CREATE DATABASE указать в качестве шаблона template0 вместо template1, вы сможете получить «чистую» пользовательскую базу данных (в которой никаких пользовательских объектов нет, есть только системные объекты в первозданном виде), не содержащую ничего, что могло быть добавлено на месте в template1. Это особенно полезно при восстановлении дампа pg\_dump: скрипт выгруженного дампа должен восстанавливаться в чистую базу, чтобы он мог воссоздать нужное содержимое базы, избежав конфликтов с объектами, которые могли быть добавлены в template1.

1.1. Утилита pg\_dump

Для создания полноценной резервной копии базы данных используется утилита pg\_dump. Эта программа генерирует файл с командами SQL, которые при выполнении на сервере пересоздадут объекты базы данных в том же самом состоянии, в котором они были на момент выгрузки. Она создаёт целостные копии, даже если база параллельно используется. Программа pg\_dump не препятствует доступу других пользователей к базе данных (ни для чтения, ни для записи). Программа pg\_dump выгружает только одну базу данных. Выгружаемые данные могут быть сохранены в виде скрипта, либо в одном из архивных форматов. Скрипты представляют собой текстовые файлы, содержащие SQL-команды, необходимые для воссоздания базы данных до состояния на момент создания скрипта. Для восстановления из скрипта его содержимое можно передать psql. Скрипты можно использовать для восстановления базы на других машинах, в том числе с иной архитектурой, а с некоторыми коррективами даже в других СУБД.

Синтаксис команды имеет следующий вид:

pg\_dump [*параметр-подключения...*] [*параметр...*] [*имя\_бд*]

***параметр-подключения***

-d *имя\_бд* или --dbname=*имя\_бд* – указывает имя базы данных для подключения. Равнозначно указанию *имя\_бд* в первом аргументе, не являющемся ключом в командной строке.

-h *сервер* или --host=*сервер* – указывает имя компьютера, на котором работает сервер. Если значение начинается с косой черты, оно определяет каталог Unix-сокета. Значение по умолчанию берётся из переменной окружения PGHOST, если она установлена. В противном случае выполняется подключение к Unix-сокету.

-p *порт* или --port=*порт* – указывает TCP-порт или расширение файла локального Unix-сокета, через который сервер принимает подключения. Значение по умолчанию определяется переменной окружения PGPORT, если она установлена, либо числом, заданным при компиляции.

-U *имя\_пользователя* или --username=*имя*\_*пользователя* – имя пользователя, под которым производится подключение.

-w или --no-password – не выдавать запрос на ввод пароля. Если сервер требует аутентификацию по паролю и пароль не доступен с помощью других средств, таких как файл .pgpass, попытка соединения не удастся. Этот параметр может быть полезен в пакетных заданиях и скриптах, где нет пользователя, который вводит пароль.

-W или –password – принудительно запрашивать пароль перед подключением к базе данных. Это - несущественный параметр, так как pg\_dump запрашивает пароль автоматически, если сервер проверяет подлинность по паролю. Однако, чтобы понять это, pg\_dump лишний раз подключается к серверу. Поэтому иногда имеет смысл ввести -W, чтобы исключить эту ненужную попытку подключения.

--role=*имя роли* – задаёт имя роли, которая будет осуществлять выгрузку. Получив это имя, pg\_dump выполнит SET ROLE *имя\_роли* после подключения к базе данных. Это полезно, когда проходящий проверку пользователь (указанный в -U) не имеет прав, необходимых для pg\_dump, но может переключиться на роль, наделённую этими правами. В некоторых окружениях правила запрещают подключаться к серверу непосредственно суперпользователю, и этот параметр позволяет выполнить выгрузку, не нарушая их.

**Список основных параметров**

-f *файл* или --file=*файл* – отправить вывод в указанный файл. Параметр можно не указывать, если используется формат с выводом в файл. В этом случае будет использован стандартный вывод. Однако для формата с выводом в каталог параметр является обязательным и должен задавать путь к каталогу. В этом случае целевой каталог будет создан командой pg\_dump и не должен существовать заранее.

-F *format* или --format=*format* – указывает формат вывода копии. format может принимать следующие значения:

p или plain – сформировать текстовый SQL-скрипт. Это поведение по умолчанию.

c или custom – выгрузить данные в специальном архивном формате, пригодном для дальнейшего использования утилитой pg\_restore. Наряду с форматом directory является наиболее гибким форматом, позволяющим вручную выбирать и сортировать восстанавливаемые объекты. Вывод в этом формате по умолчанию сжимается.

t или tar – выгрузить данные в формате tar, для дальнейшего использования с утилитой pg\_restore. Этот формат совместим с форматом вывода в каталог: если архив распаковать, получится корректная копия в формате каталога. Однако формат tar не поддерживает сжатие. Также, применяя формат tar, при восстановлении нельзя изменить относительный порядок элементов данных.

-C или --create – сформировать в начале вывода команду для создания базы данных и затем подключения к ней. В этом случае не важно, какая база указана в параметрах подключения перед выполнением скрипта. Также, если указан ключ --clean, то скрипт сначала удалит, а затем пересоздаст базу данных перед подключением к ней.

С ключом –create в выходной файл также включается комментарий к базе данных (если он задан) и все назначения переменных конфигурации, связанные с базой данных, то есть все команды ALTER DATABASE ... SET . и ALTER ROLE ... IN DATABASE ... SET ..., ссылающиеся на эту базу данных. Также выгружаются права доступа к самой базе данных, если не добавлен ключ --no-acl.

Этот параметр игнорируется, когда данные выгружаются в архивных форматах (не в текстовом). Для таких форматов данный параметр можно указать при вызове pg\_restore.

Сделаем резервную копию БД demo. Сначала создадим каталог Backup в домашней папке, куда будем помещать файл.

Откроем терминал в Linux/Mac или командную строку в Windows. Если путь к каталогу bin PostgreSQL не добавлен в системную переменную PATH, то для перехода в него будет использована команда cd. Этот каталог обычно находится в каталоге установки PostgreSQL. Введем команду (рис. 1.1.1):

**Windows**

C:\>pg\_dump -d demo -U postgres -f C:\Backup\demo.sql

**Linux**

~$ pg\_dump -d demo -U postgres -f ~/Backup/demo.sql

Рисунок 1.1.1

В командной строке, если не указывать host и port, то выберутся значения по умолчанию – *localhost* и *5432*, соответственно. После ввода команды pg\_dump затребует пароль. После подтверждения создастся резервная копия БД demo в каталоге Backup.

Откроем demo.sql (рис. 1.1.2):

...

GRANT USAGE ON SCHEMA bookings TO readers;

GRANT USAGE ON SCHEMA bookings TO writer\_aircrafts;

--

-- Name: TABLE aircrafts\_data; Type: ACL; Schema: bookings; Owner: postgres

--

GRANT SELECT ON TABLE bookings.aircrafts\_data TO readers;

GRANT INSERT,UPDATE ON TABLE bookings.aircrafts\_data TO writer\_aircrafts;

...

--

-- Name: airports\_data; Type: TABLE; Schema: bookings; Owner: postgres

--

CREATE TABLE bookings.airports\_data (

airport\_code character(3) NOT NULL,

airport\_name jsonb NOT NULL,

city jsonb NOT NULL,

coordinates point NOT NULL,

timezone text NOT NULL

);

ALTER TABLE bookings.airports\_data OWNER TO postgres;

--

-- Name: TABLE airports\_data; Type: COMMENT; Schema: bookings; Owner: postgres

--

COMMENT ON TABLE bookings.airports\_data IS 'Airports (internal data)';

--

-- Name: COLUMN airports\_data.airport\_code; Type: COMMENT; Schema: bookings; Owner: postgres

--

COMMENT ON COLUMN bookings.airports\_data.airport\_code IS 'Airport code';

--

-- Name: COLUMN airports\_data.airport\_name; Type: COMMENT; Schema: bookings; Owner: postgres

--

COMMENT ON COLUMN bookings.airports\_data.airport\_name IS 'Airport name';

--

-- Name: COLUMN airports\_data.city; Type: COMMENT; Schema: bookings; Owner: postgres

--

COMMENT ON COLUMN bookings.airports\_data.city IS 'City';

...

COPY bookings.aircrafts\_data (aircraft\_code, model, range) FROM stdin;

773 {"en": "Boeing 777-300", "ru": "Боинг 777-300"} 11100

763 {"en": "Boeing 767-300", "ru": "Боинг 767-300"} 7900

SU9 {"en": "Sukhoi Superjet-100", "ru": "Сухой Суперджет-100"} 3000

320 {"en": "Airbus A320-200", "ru": "Аэробус A320-200"} 5700

321 {"en": "Airbus A321-200", "ru": "Аэробус A321-200"} 5600

319 {"en": "Airbus A319-100", "ru": "Аэробус A319-100"} 6700

733 {"en": "Boeing 737-300", "ru": "Боинг 737-300"} 4200

CN1 {"en": "Cessna 208 Caravan", "ru": "Сессна 208 Караван"} 1200

CR2 {"en": "Bombardier CRJ-200", "ru": "Бомбардье CRJ-200"} 2700

\.

Рисунок 1.1.2

И увидим знакомые нам скрипты.

Важно отметить, что при таком использовании pg\_dump скрипт по созданию БД demo не создается. Для того, чтобы скрипт формировался надо указать параметр -C или --create (рис. 1.1.3):

**Windows**

C:\>pg\_dump -d demo -U postgres --create -f C:\Backup\demo.sql

**Linux**

~$ pg\_dump -d demo -U postgres --create -f ~/Backup/demo.sql

Рисунок 1.1.3

Как видим, в скрипте БД формируется с использованием шаблона template0 (рис. 1.1.4):

...

CREATE DATABASE demo WITH TEMPLATE = template0 ENCODING = 'UTF8'   
LOCALE\_PROVIDER = libc LOCALE = 'ru-RU';

...

Рисунок 1.1.4

Проведем краш-тест БД demo – удалим ее, а потом восстановим из файла demo.sql, созданного с ключом --create. Запустим psql от имени суперпользователя postgres и выполним команду DROP DATABASE  
(рис. 1.1.5):

Server [localhost]:

Database [postgres]:

Port [5432]:

Username [postgres]:

Пароль пользователя postgres:

psql (16.8)

ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной

страницы Windows (1251).

8-битовые (русские) символы могут отображаться некорректно.

Подробнее об этом смотрите документацию psql, раздел

"Notes for Windows users".

Введите "help", чтобы получить справку.

postgres=# \! chcp 1251

Текущая кодовая страница: 1251

postgres=# DROP DATABASE demo;

DROP DATABASE

postgres=# \l

Список баз данных

Имя | Владелец | Кодировка | Провайдер локали | LC\_COLLATE | LC\_CTYPE |

-----------+----------+-----------+------------------+------------+----------+-----

postgres | postgres | UTF8 | libc | ru-RU | ru-RU |

template0 | postgres | UTF8 | libc | ru-RU | ru-RU |

template1 | postgres | UTF8 | libc | ru-RU | ru-RU |

| | | | | |

(3 строки)

Рисунок 1.1.5

Восстановим БД demo из файла demo.sql (рис. 1.1.6):

**Windows**

psql -U postgres -d postgres -f C:\Backup\demo.sql

**Linux**

~$ psql -U postgres -d postgres -f ~/Backup/demo.sql

Рисунок 1.1.6

Запустим psql и проверим (рис. 1.1.7 и рис. 1.1.8):

**postgres=# \l**

Список баз данных

Имя | Владелец | Кодировка | Провайдер локали | LC\_COLLATE | LC\_CTYPE | локаль ICU | Правила ICU | Права доступа

-----------+----------+-----------+------------------+------------+----------+------------+-------------+-----------------------------

demo | postgres | UTF8 | libc | ru-RU | ru-RU | | | postgres=CTc/postgres +

| | | | | | | | readers=c/postgres +

| | | | | | | | writer\_aircrafts=c/postgres

postgres | postgres | UTF8 | libc | ru-RU | ru-RU | | |

template0 | postgres | UTF8 | libc | ru-RU | ru-RU | | | =c/postgres +

| | | | | | | | postgres=CTc/postgres

template1 | postgres | UTF8 | libc | ru-RU | ru-RU | | | =c/postgres +

| | | | | | | | postgres=CTc/postgres

(4 строки)

Рисунок 1.1.7

**demo=# \z**

bookings | aircrafts | представление | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | aircrafts\_data | таблица | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres +| |

| | | writer\_aircrafts=aw/postgres | |

bookings | airports | представление | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | airports\_data | таблица | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | boarding\_passes | таблица | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | bookings | таблица | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | flights | таблица | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | flights\_flight\_id\_seq | последовательность | | |

bookings | flights\_v | представление | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | routes | представление | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | seats | таблица | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | ticket\_flights | таблица | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

bookings | tickets | таблица | postgres=arwdDxt/postgres +| |

| | | readers=r/postgres | |

public | test | таблица |

Рисунок 1.1.8

Как можем убедиться, что БД восстановлена и введенные нами ранее роли присутствуют.

Мы рассмотрели использование pg\_dump для создания текстовых файлов. Проблема в том, что текстовый файл можно восстановить только целиком. Если мы сохранили копию всей базы, то и восстановить сможем только всю базу. Но обычно нам нужно не только это. Поэтому PostgreSQL предлагает другие формы, обладающие дополнительной функциональностью.

Рассмотрим пользовательский формат custom (-F *c*, --format=*custom*), создающий 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 и занимает намного меньше места, чем исходные данные. Для восстановления потребуется другая утилита – pg\_restore. Синтаксис команды имеет следующий вид:

pg\_restore [*параметр-подключения...*] [*параметр...*] [*имя\_файла*]

Список *параметр-подключения* такой же, как у pg\_dump.

**Список основных параметров**

*имя\_файла*– указывает расположение восстанавливаемого файла архива (или каталога, для архива в формате каталога).

-a или --data-only – восстанавливать только данные, без схемы (определений данных). При этом восстанавливаются данные таблиц, большие объекты и значения последовательностей, имеющиеся в архиве.

-c или --clean – прежде чем восстанавливать объекты базы данных, выполните команды DROP для всех восстанавливаемых объектов. Этот параметр полезен для перезаписи существующей базы данных. Без дополнительного указания --if-exists при этом могут выдаваться безвредные сообщения об ошибках, если таких объектов не окажется в целевой базе данных.

-C или --create – создать базу данных, прежде чем восстанавливать данные. Если также указан параметр --clean, удалить и пересоздать целевую базу данных перед подключением к ней.

-d *имя\_бд* или --dbname=*имя\_бд* – подключиться к базе данных *имя\_базы.* В аргументе -d можно указать любую базу данных, существующую в кластере, если восстанавливаем БД; pg\_restore использует её только, чтобы выполнить команду CREATE DATABASE. Либо БД, для восстановления в ней объекта.

-f *имя\_файла* или --file=*имя\_файла* – задаёт файл для вывода сгенерированного скрипта или списка объектов, получаемого с параметром   
-l. Чтобы выбрать stdout, используйте - .

-l или --list – вывести оглавление архива.

-L *файл-список* или --use-list=*файл-список* – восстановить из архива только элементы, перечисленные в файле-списке, и в том порядке, в каком они идут в этом файле.

-t *таблица* или --table=*таблица* – восстановить определение и/или данные только указанной таблицы. В этом контексте под «таблицей» подразумеваются также представления, материализованные представления, последовательности и сторонние таблицы. Чтобы выбрать несколько таблиц, ключ -t можно указать несколько раз.

Создадим резервную копию с пользовательским форматом (рис. 1.1.9):

**Windows**

C:\>pg\_dump -d demo -U postgres --create -Fc -f C:\Backup\demo.cst

**Linux**

~$ pg\_dump -d demo -U postgres --create -Fc -f ~/Backup/demo.cst

Рисунок 1.1.9

Сравним размеры архивных файлов demo.sql (plain) и demo.cst (custom) (рис. 1.1.10):

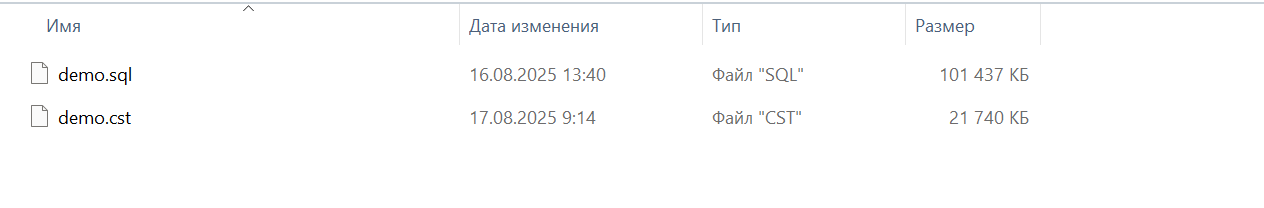


Рисунок 1.1.10

Очевидное преимущество использования пользовательского формата.

Создав копию, мы можем просмотреть содержимое файла (рис. 1.1.11):

C:\Users\Alex>pg\_restore --list C:\Backup\demo.cst

;

; Archive created at 2025-08-17 09:13:35

; dbname: demo

; TOC Entries: 156

; Compression: gzip

; Dump Version: 1.15-0

; Format: CUSTOM

; Integer: 4 bytes

; Offset: 8 bytes

; Dumped from database version: 16.8

; Dumped by pg\_dump version: 16.8

;

;

; Selected TOC Entries:

;

6; 2615 16869 SCHEMA - bookings postgres

4989; 0 0 COMMENT - SCHEMA bookings postgres

4990; 0 0 ACL - SCHEMA bookings postgres

230; 1255 16870 FUNCTION bookings lang() postgres

231; 1255 16871 FUNCTION bookings now() postgres

...

4971; 0 16872 TABLE DATA bookings aircrafts\_data postgres

4972; 0 16882 TABLE DATA bookings airports\_data postgres

4973; 0 16891 TABLE DATA bookings boarding\_passes postgres

4974; 0 16894 TABLE DATA bookings bookings postgres

4975; 0 16897 TABLE DATA bookings flights postgres

4977; 0 16914 TABLE DATA bookings seats postgres

4978; 0 16918 TABLE DATA bookings ticket\_flights postgres

4979; 0 16923 TABLE DATA bookings tickets postgres

4980; 0 16928 TABLE DATA public test postgres

...

Рисунок 1.1.11

Повторим тест по восстановлению БД demo, но уже из архива demo.cst. Удалим БД demo:

DROP DATABASE demo;

И восстановим (рис. 1.1.12):

**Windows**

C:\> pg\_restore -d postgres -U postgres -C С:\Backup\demo.cst

**Linux**

~$ pg\_restore -d postgres -U postgres -C ~/Backup/demo.cst

Рисунок 1.1.12

Здесь указали для подключения БД postgres, пользователя postgres и создать БД – параметр -C. Необходимо подключиться к любой БД, чтобы выполнить команду CREATE DATABASE.

У pg\_restore есть много дополнительных возможностей, например восстановление части базы данных. Выполним удаление таблицы test из схемы public БД demo (рис. 1.1.13):

demo=# \dt

bookings | aircrafts\_data | таблица | postgres

bookings | airports\_data | таблица | postgres

bookings | boarding\_passes | таблица | postgres

bookings | bookings | таблица | postgres

bookings | flights | таблица | postgres

bookings | seats | таблица | postgres

bookings | ticket\_flights | таблица | postgres

bookings | tickets | таблица | postgres

public | test | таблица | postgres

demo=# DROP TABLE test;

DROP TABLE

demo=# \dt

bookings | aircrafts\_data | таблица | postgres

bookings | airports\_data | таблица | postgres

bookings | boarding\_passes | таблица | postgres

bookings | bookings | таблица | postgres

bookings | flights | таблица | postgres

bookings | seats | таблица | postgres

bookings | ticket\_flights | таблица | postgres

bookings | tickets | таблица | postgres

Рисунок 1.1.13

Восстановим таблицу test (рис. 1.1.14 и рис. 1.1.15):

**Windows**

C:>pg\_restore -d demo -U postgres -t test C:\Backup\demo.cst

**Linux**

~$ pg\_restore -d demo -U postgres -t test ~/Backup/demo.cst

Рисунок 1.1.14

В команде указали БД demo, где должна быть восстановлена таблица test.

demo=# \dt

bookings | aircrafts\_data | таблица | postgres

bookings | airports\_data | таблица | postgres

bookings | boarding\_passes | таблица | postgres

bookings | bookings | таблица | postgres

bookings | flights | таблица | postgres

bookings | seats | таблица | postgres

bookings | ticket\_flights | таблица | postgres

bookings | tickets | таблица | postgres

public | test | таблица | postgres

Рисунок 1.1.15

Мы убедились, что таблица восстановлена.

Обратимся к pgAdmin, и выполнив резервное копирование и восстановление с помощью этой графической оболочки. pgAdmin для этих операций использует утилиты pg\_dump и pg\_restore. И в этом кроется одно «но». А именно, версии этих утилит, используемых pgAdmin и сервером PostgreSQL должны совпадать. Если вы обновляли pgAdmin, необходимо скопировать pg\_dump и pg\_restore из каталога bin сервера PostgreSQL в соответствующий каталог pgAdmin.

Щелкнем ПКМ по demo на дереве объектов и выберем команду *Backup…* из контекстного меню (рис. 1.1.16):

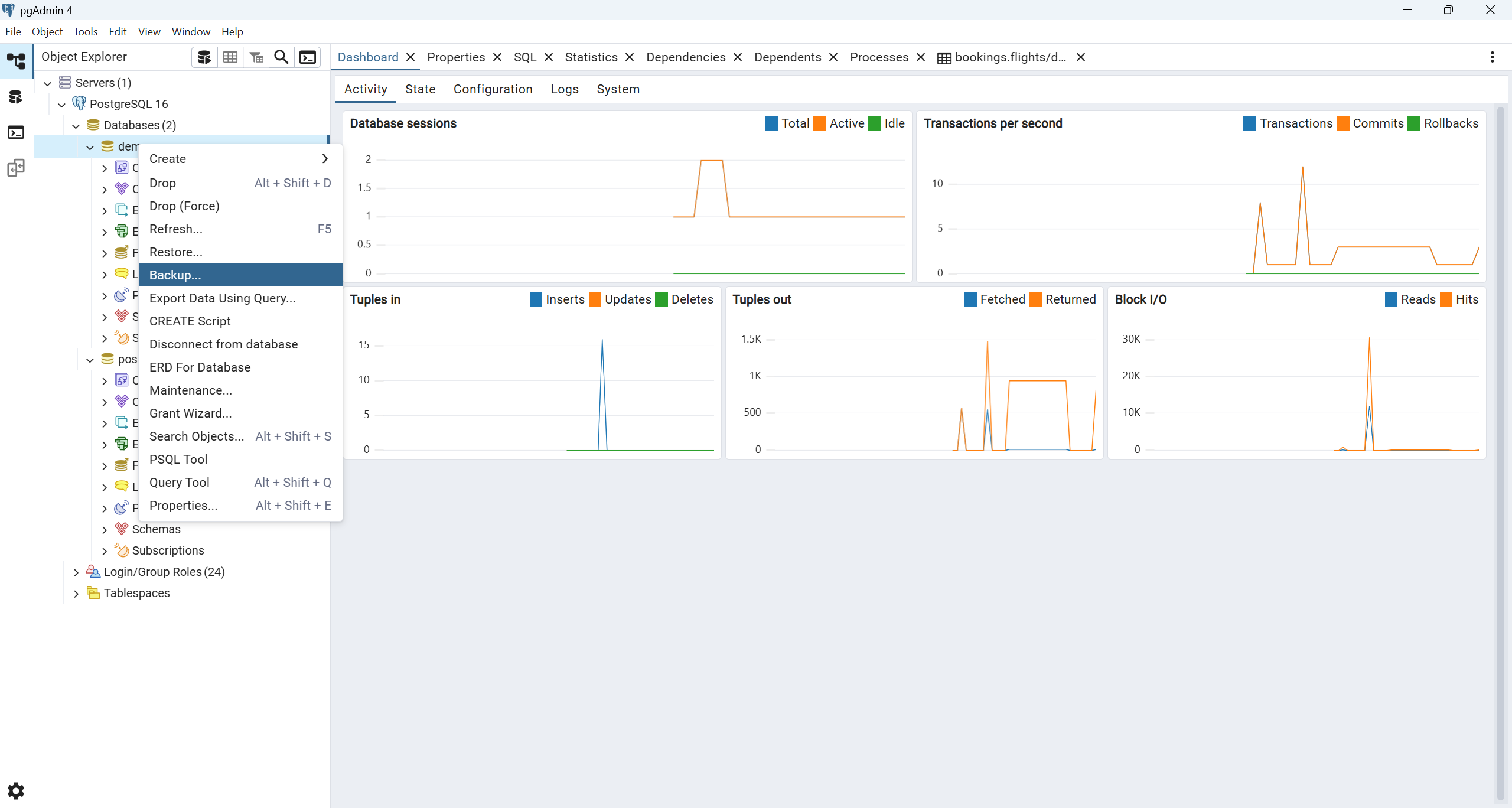


Рисунок 1.1.16

Откроется диалог *Backup* (рис. 1.1.17), все переключатели и поля соответствуют параметрам pg\_dump. Введем имя файла, формат архива и степень сжатия, которая может изменяться от 0 до 9. 0 означает, что сжатия не будет выполняться, а 9 – максимальное сжатие.

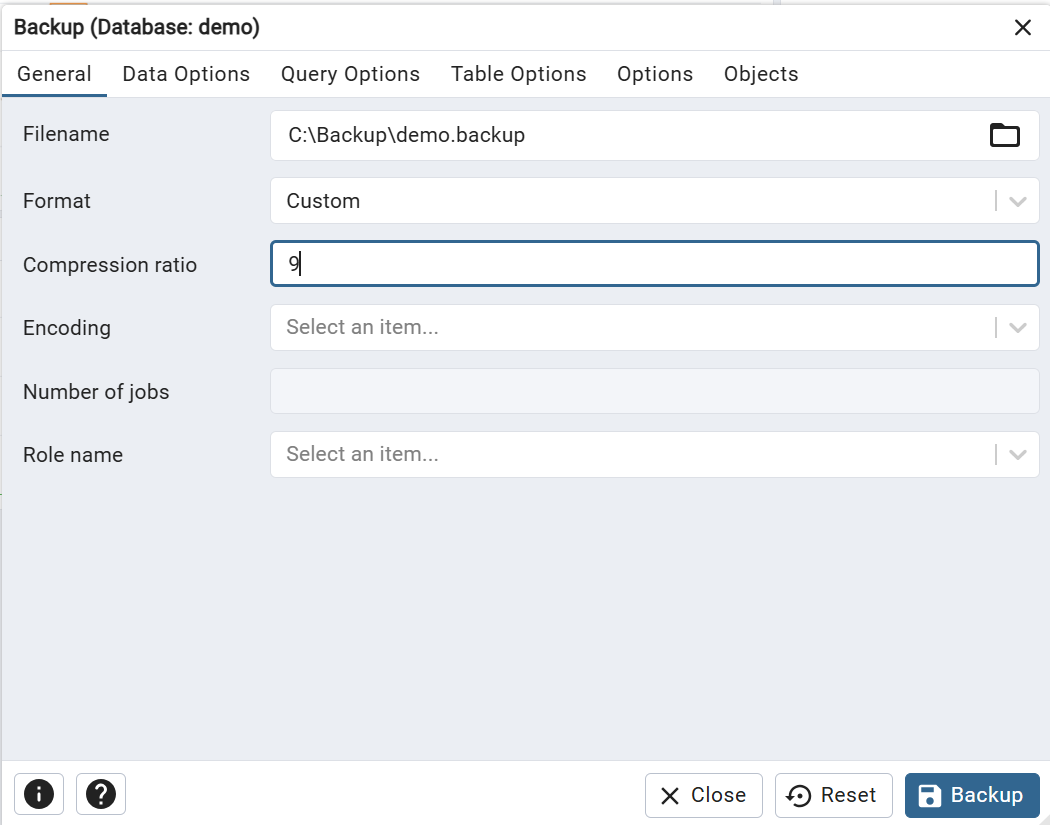


Рисунок 1.1.17

Мы выполняем резервное копирования всей БД, поэтому пропустим часть вкладок, которые позволяют архивировать определенный тип данных, и остановимся на *Query Option*, в ней активируем переключатель *Include CREATE DATABASE statement* (рис. 1.1.18):

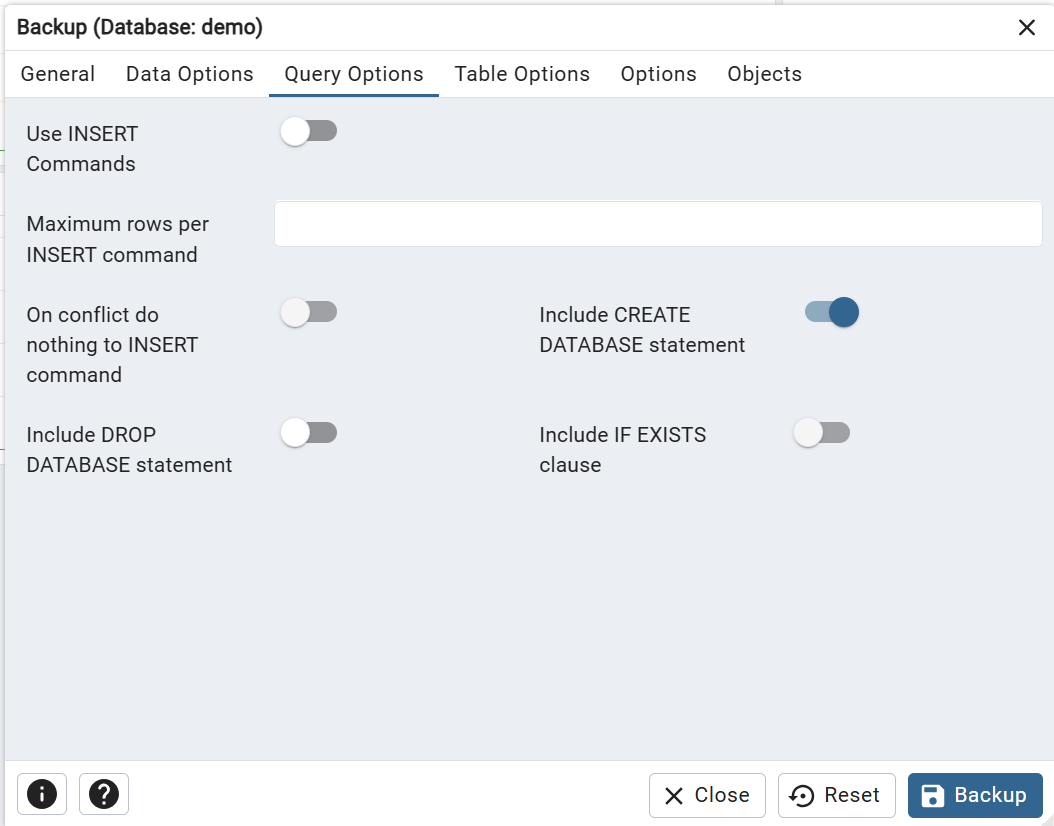


Рисунок 1.1.18

Далее выбираем вкладку *Objects* (рис. 1.1.19):

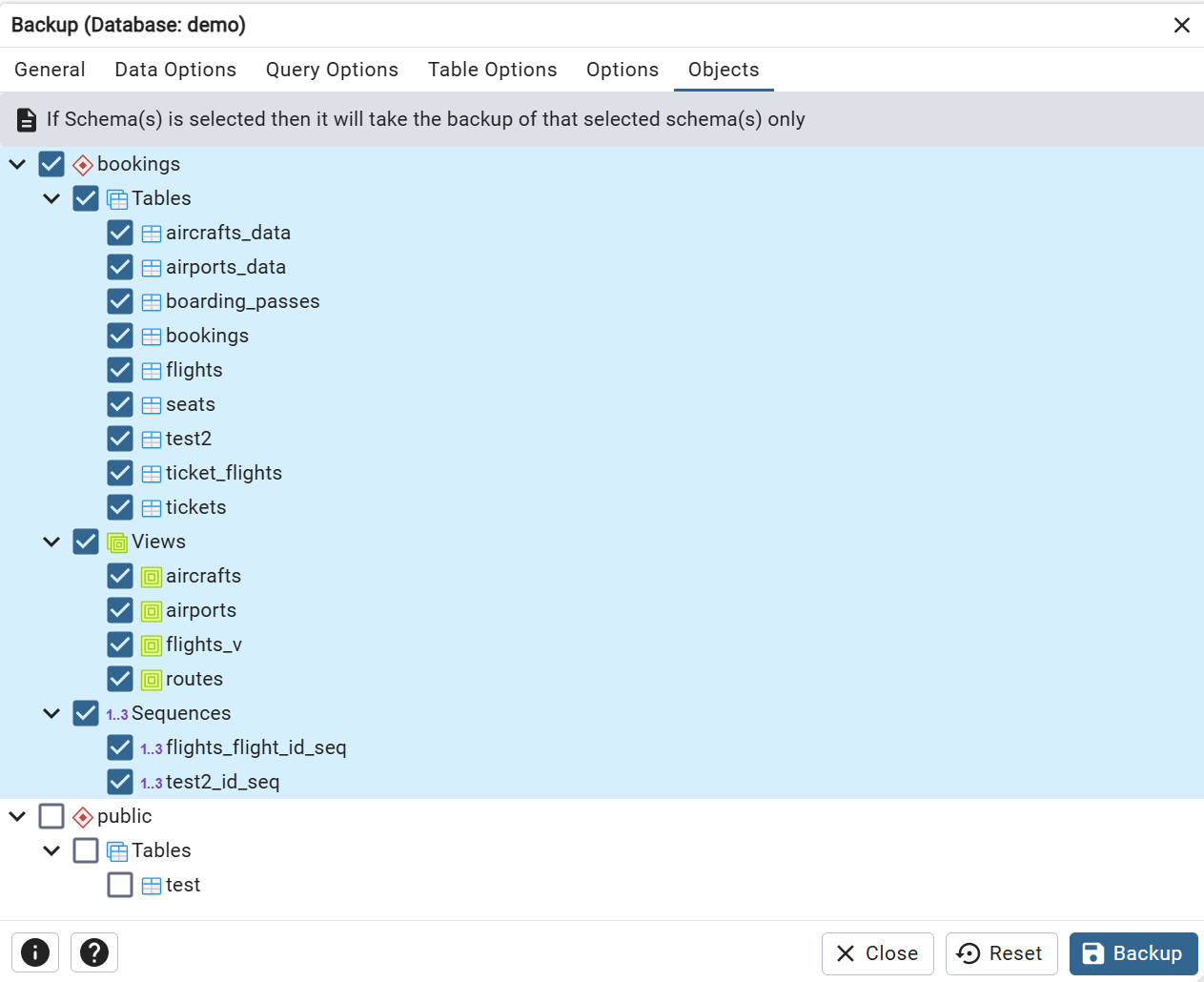


Рисунок 1.1.19

Выбираем все объекты БД demo и нажимаем на кнопку *Backup*.

Результат об выполнении резервного копирования отобразится в pgAdmin (рис. 1.1.20):

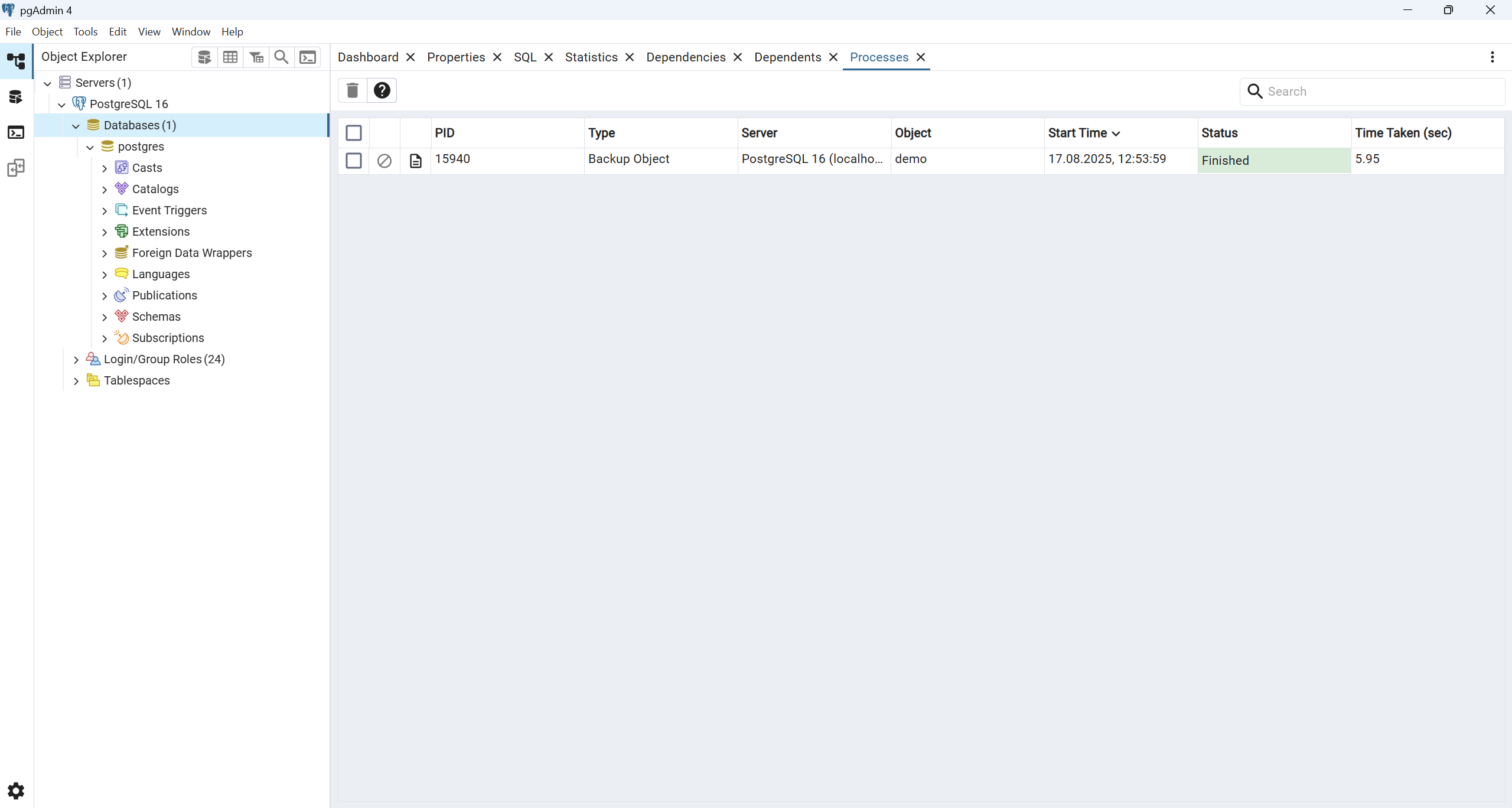


Рисунок 1.1.20

В папке Backup появится еще один архивный файл (рис. 1.1.21):

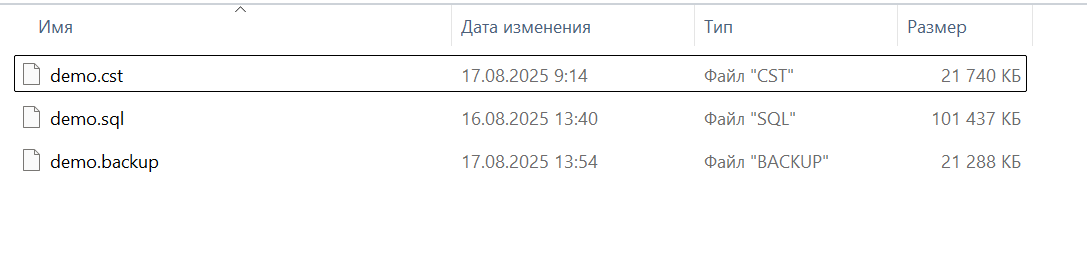


Рисунок 1.1.21

Перед тем, как восстанавливать БД demo, удалим ее.

Щелкаем ПКМ по БД postgres, выбираем команду *Restore…*. Откроется диалог Restore (рис. 1.1.22):

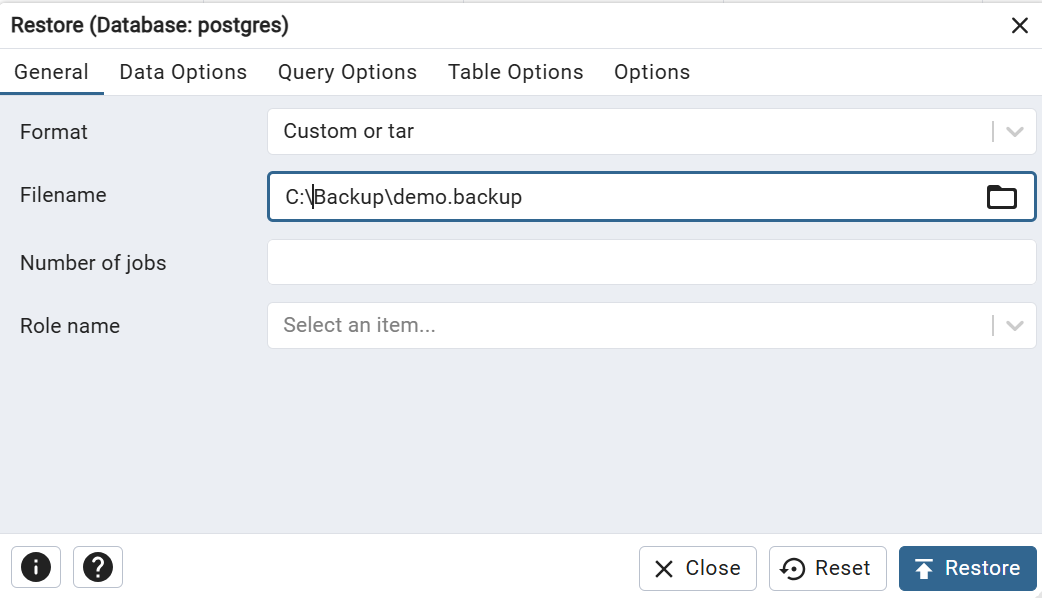


Рисунок 1.1.22

Напомню, что все поля и переключатели этого диалога, соответствуют параметрам утилиты pg\_restore.

В диалоге выбираем формат и файл архива. Далее переходим на вкладку Query Options (рис. 1.1.23):



Рисунок 1.1.23

Перед тем, как восстанавливать БД demo, мы ее удалили, поэтому нам достаточно выбрать опцию создать БД. Но если выполняется восстановление БД по верх существующей, то надо еще выбрать опции *Clean before restore* и *Include IF EXISTS clause*. Последняя опция позволяет избежать сообщений, что неких объектов нет в БД. pgAdmin воспринимает эти безобидные сообщения, как ошибки.

И наконец, жмем на кнопку *Restore*. Результат выполнения утилиты можно увидеть в pgAdmin (рис. 1.1.24):

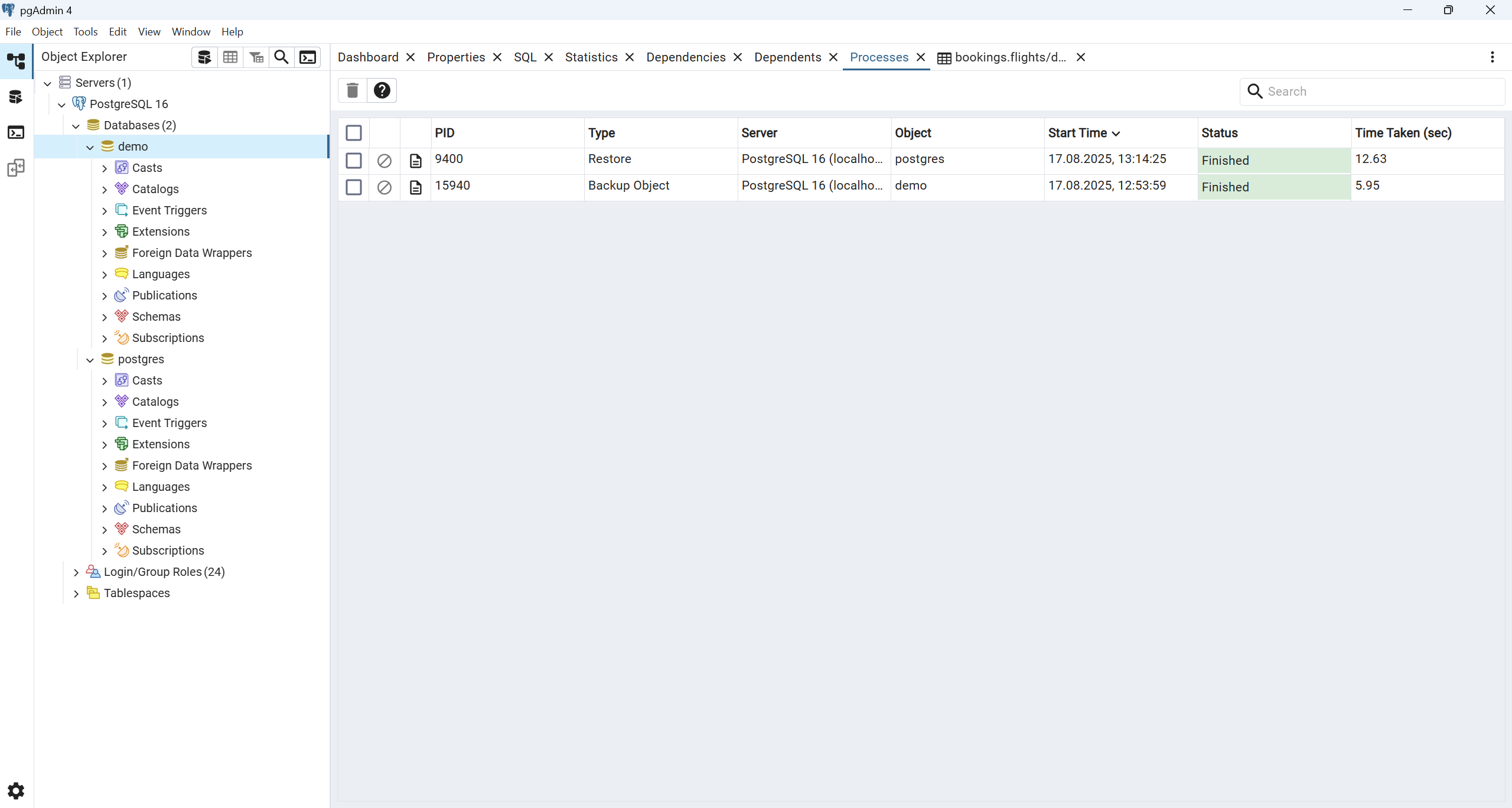


Рисунок 1.1.24

Если щелкнуть по значку *View Details* отобразятся логи выполнения утилиты (рис. 1.1.25):

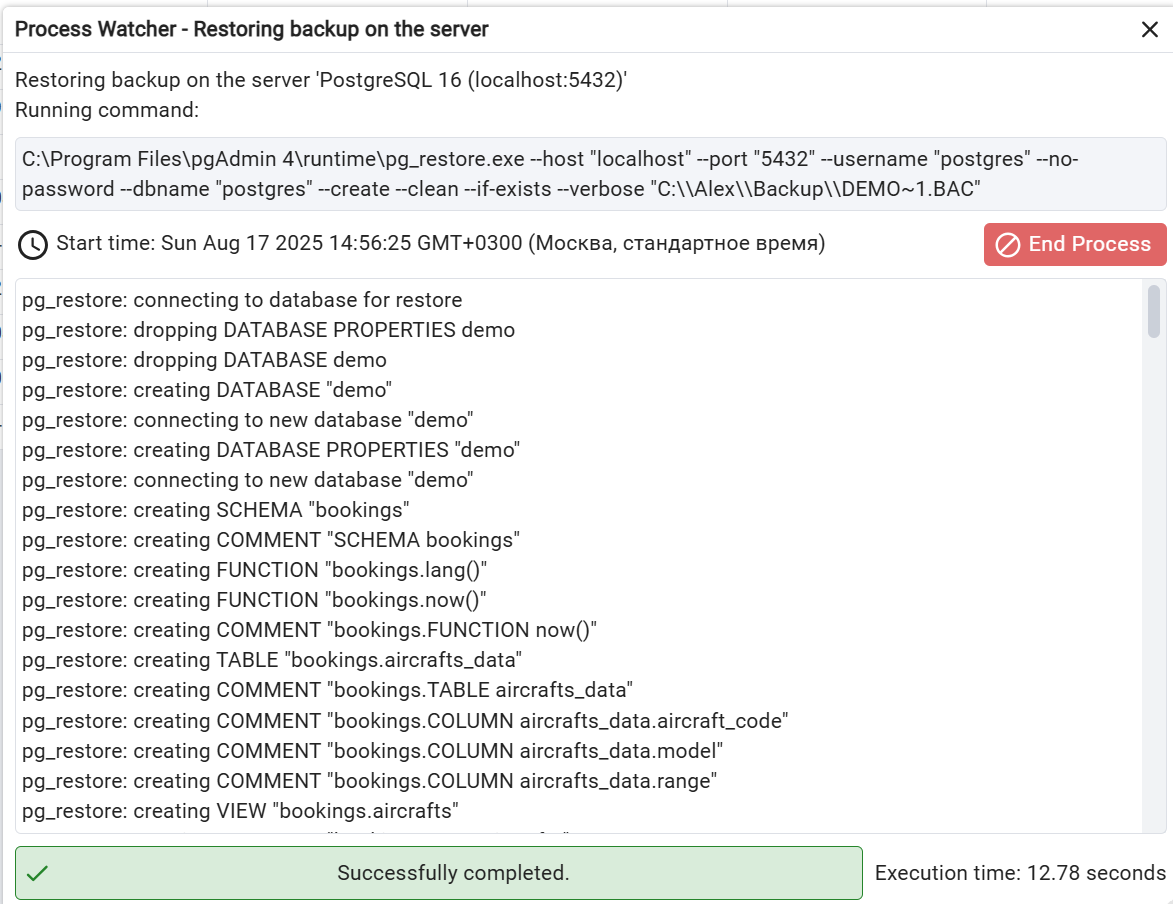


Рисунок 1.1.25

Логи очень полезны, если по каким-то причинам восстановление БД завершилось с ошибкой.

1.2. Утилита pg\_dumpall

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 БД postgres, но можно указать и другую. Результатом работы pg\_dumpall является скрипт для psql.

При создании обычной копии с помощью pg\_dump базы данных сохраняются все права, но не команды CREATE USER. Эти глобальные данные не включаются в обычную копию, их может извлечь только pg\_dumpall. Если нужны только глобальные данные, запустите pg\_dumpall с ключом -g (--globals-only).

Но поскольку для копирования кластера надо иметь доступ ко всем БД, мы должны выполнять pg\_dumpall от имени суперпользователя (рис. 1.2.1):

**Windows**

C:\> pg\_dumpall -l demo -U postgres -f C:\Backup\demo\_all.sql

Пароль:

Пароль:

Пароль:

Пароль:

**Linux**

~$ pg\_dumpall -l demo -U postgres -f ~/Backup/demo\_all.sql

Пароль:

Пароль:

Пароль:

Пароль:

Рисунок 1.2.1

В процессе выполнения, будет предлагаться несколько раз набрать пароль.

Восстановить базы данных из этого файла можно так:

psql -X -f C:\Backup\demo\_all.sql -d postgres

1. Практическое задание
2. Выполнить логическое резервное копирование и восстановление БД.
3. Выполнить восстановление только одной таблицы.
4. Выполнить логическое резервное копирование и восстановление всего кластера.

В отчете привести скрипты и скрины практических заданий.

3. Список использованных источников

1. Документация PostgreSQL https://postgrespro.ru/docs
2. PostgreSQL 11. Мастерство разработки. Ганс-Юрген Шениг / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2019.