

Организация данных Логическая структура



Авторские права

© Postgres Professional, 2017–2021

Авторы: Егор Рогов, Павел Лузанов

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.

Базы данных и шаблоны

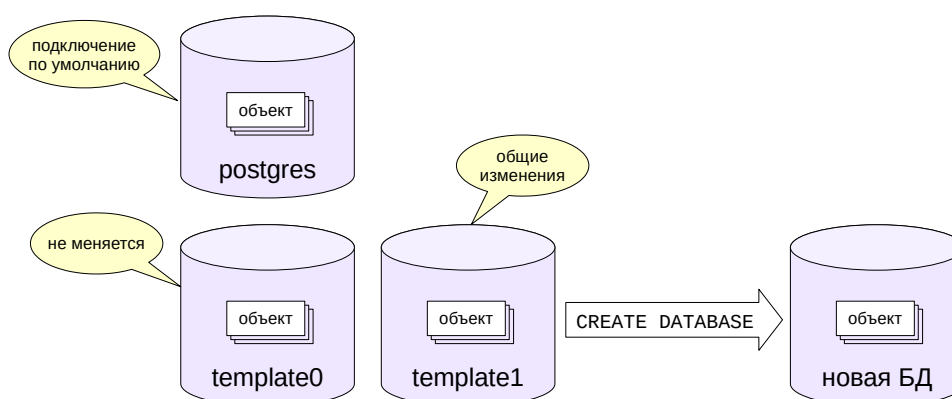
Схемы и путь поиска

Специальные схемы

Системный каталог

Инициализация кластера создает три базы данных

Новая база всегда клонируется из существующей



3

Экземпляр PostgreSQL управляет несколькими базами данных — кластером. При инициализации кластера (автоматически при установке PostgreSQL либо вручную командой `initdb`) создаются три одинаковые базы данных. Все остальные БД, создаваемые пользователем, клонируются из какой-либо существующей.

Шаблонная БД `template1` используется по умолчанию для создания новых баз данных. В нее можно добавить объекты и расширения, которые будут копироваться в каждую новую базу данных.

Шаблон `template0` не должен изменяться. Он нужен как минимум в двух ситуациях. Во-первых, для восстановления БД из резервной копии, выполненной утилитой `pg_dump` (как рассматривается в теме «Резервное копирование. Логическое резервирование»). Во-вторых, при создании новой БД с кодировкой, отличной от указанной при инициализации кластера (подробнее обсуждается в курсе DBA2).

База данных `postgres` используется при подключении по умолчанию пользователем `postgres`. Она не является обязательной, но некоторые утилиты предполагают ее наличие, поэтому ее не рекомендуется удалять, даже если она не нужна.

<https://postgrespro.ru/docs/postgresql/12/manage-ag-templatedbs>

Базы данных

Список баз данных можно получить в psql такой командой:

```
=> \l
```

List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
postgres	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	
student	student	UTF8	en_US.UTF-8	en_US.UTF-8	
template0	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres +
					postgres=CTc/postgres
template1	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres +
					postgres=CTc/postgres

(4 rows)

Здесь выводится много полей, которые нас сейчас не интересуют.

Когда мы создаем новую базу данных, она (по умолчанию) копируется из шаблона template1.

```
=> CREATE DATABASE data_logical;
```

```
CREATE DATABASE
```

```
=> \c data_logical
```

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

```
=> \l
```

List of databases					
Name	Owner	Encoding	Collate	Ctype	Access privileges
data_logical	student	UTF8	en_US.UTF-8	en_US.UTF-8	
postgres	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	
student	student	UTF8	en_US.UTF-8	en_US.UTF-8	
template0	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres +
					postgres=CTc/postgres
template1	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres +
					postgres=CTc/postgres

(5 rows)

Пространство имен для объектов

- разделение объектов на логические группы
- предотвращение конфликта имен между приложениями

Схема и пользователь — разные сущности

Специальные схемы

- public — по умолчанию в ней создаются все объекты
- pg_catalog — системный каталог
- information_schema — вариант системного каталога
- pg_temp — для временных таблиц
- ...

Схемы представляют собой пространства имен для объектов БД. Они позволяют разделить объекты на логические группы для управления ими, предотвратить конфликты имен при работе нескольких пользователей или приложений.

В PostgreSQL *схема* и *пользователь* — разные сущности (хотя настройки по умолчанию упрощают работу с одноименными схемами).

Существует несколько специальных схем, обычно присутствующих в каждой базе данных.

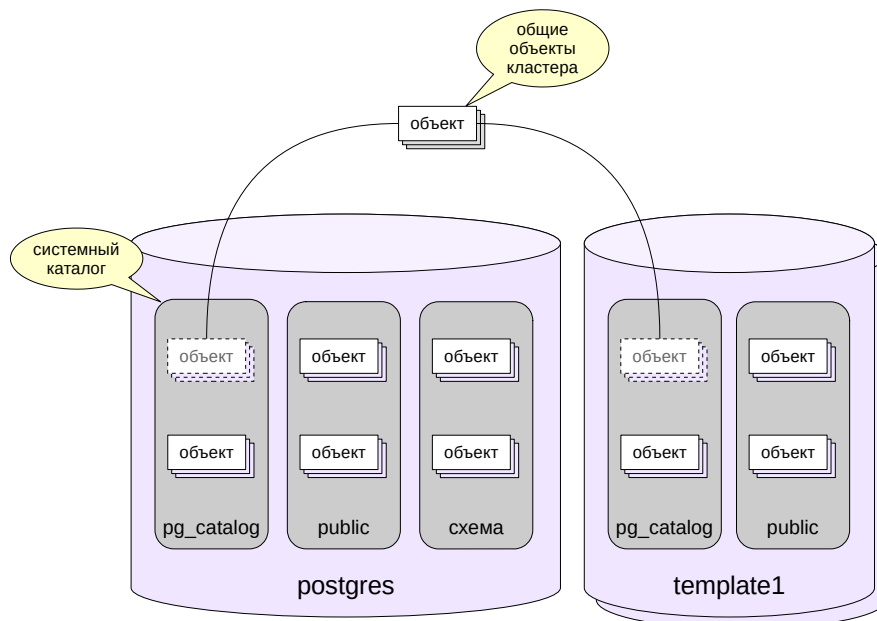
Схема public используется по умолчанию для хранения объектов, если не выполнены иные настройки.

Схема pg_catalog хранит объекты *системного каталога*. Системный каталог — это метаданные об объектах, принадлежащих кластеру, которая хранится в самом кластере в виде таблиц. Альтернативное представление системного каталога (определенное в стандарте SQL) дает схема information_schema.

Схема pg_temp служит для хранения временных таблиц. (На самом деле таблицы создаются в схемах pg_temp_1, pg_temp_2 и т. п. — у каждого пользователя своя схема. Но обращаются все пользователи к ней как к pg_temp.)

Есть и другие схемы, но они носят технический характер.

<https://postgrespro.ru/docs/postgresql/12/ddl-schemas>



Схемы принадлежат базам данных, все объекты БД распределены по каким-либо схемам.

Однако несколько таблиц системного каталога хранят информацию, общую для всего кластера. Это список баз данных, список пользователей и некоторые другие сведения. Эти таблицы хранятся вне какой-либо базы данных, но при этом одинаково видны из каждой БД.

Таким образом, клиент, подключенный к какой-либо базе данных, видит в системном каталоге описание объектов не только данной базы, но и общих объектов кластера. Описание объектов других баз данных можно получить, только подключившись к ним.

Схемы

Для списка схем в psql есть специальная команда (\dn = describe namespace):

```
=> \dn
```

```
List of schemas
Name | Owner
-----+-----
public | postgres
(1 row)
```

Эта команда не показывает служебные схемы. Чтобы увидеть их, нужно добавить модификатор S (он работает аналогичным образом и для многих других команд):

```
=> \dnS
```

```
List of schemas
Name | Owner
-----+-----
information_schema | postgres
pg_catalog | postgres
pg_temp_1 | postgres
pg_toast | postgres
pg_toast_temp_1 | postgres
public | postgres
(6 rows)
```

Про некоторые из этих схем (public, pg_catalog, information_schema) мы уже говорили; про остальные поговорим позже в других темах.

Еще один полезный модификатор — знак «плюс», который выводит дополнительную информацию:

```
=> \dn+
```

```
List of schemas
Name | Owner | Access privileges | Description
-----+-----+-----+-----
public | postgres | postgres=UC/postgres+ | standard public schema
(1 row)
```

Создадим новую схему:

```
=> CREATE SCHEMA special;
```

```
CREATE SCHEMA
```

```
=> \dn
```

```
List of schemas
Name | Owner
-----+-----
public | postgres
special | student
(2 rows)
```

Создадим таблицу:

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

```
CREATE TABLE
```

По умолчанию таблица будет создана в схеме public. Список таблиц в этой схеме можно получить командой \dt с указанием шаблона для имен схем и таблиц:

```
=> \dt public.*
```

```
List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | t | table | student
(1 row)
```

Таблицу (как и другие объекты) можно перемещать между схемами. Поскольку речь идет о логической организации, перемещение происходит только в системном каталоге; сами данные физически остаются на месте.

```
=> ALTER TABLE t SET SCHEMA special;
```

```
ALTER TABLE
```

Что останется в схеме public?

```
=> \dt public.*
```

Did not find any relation named "public.*".

Ничего. А в special?

```
=> \dt special.*
```

```
          List of relations
 Schema | Name | Type | Owner
-----+-----+-----+-----
 special | t    | table | student
(1 row)
```

Таблица переместилась. Теперь к ней можно обращаться с явным указанием схемы:

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

```
 n
---
(0 rows)
```

Но если опустить имя схемы, таблица не будет найдена:

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

```
ERROR:  relation "t" does not exist
LINE 1: SELECT * FROM t;
                        ^
```


Определение схемы объекта

квалифицированное имя (*схема.имя*) явно определяет схему
имя без квалификатора проверяется в схемах, указанных в пути поиска

Путь поиска

определяется параметром *search_path*,
реальное значение — функция *current_schemas*
не включаются несуществующие схемы и схемы, к которым нет доступа
схемы *pg_temp* и *pg_catalog* неявно включены первыми,
если не указаны в *search_path*
первая явно указанная в пути схема используется для создания объектов

При указании объекта надо определить, о какой схеме идет речь, поскольку в разных схемах могут храниться объекты с одинаковыми именами.

Если имя объекта квалифицировано именем схемы, то все просто — используется явно указанная схема. Если имя использовано без квалификатора, PostgreSQL пытается найти имя в одной из схем, перечисленных в пути поиска, который определяется конфигурационным параметром *search_path*.

Путь поиска может отличаться от значения параметра *search_path*. В путь поиска не включаются несуществующие схемы из *search_path*, а также схемы, к которым у пользователя нет доступа (разграничению доступа посвящена одна из последующих тем курса). Кроме того, в начало пути поиска неявно добавляются:

- схема *pg_catalog*, чтобы всегда иметь доступ к системному каталогу
- схема *pg_temp*, если пользователь создавал временные объекты

Реальный путь поиска, включая неявные схемы, возвращает вызов функции: *current_schemas(true)*. Схемы перебираются в указанном в пути поиска порядке, слева направо. Если в схеме нет объекта с нужным именем, поиск продолжается в следующей схеме. Можно провести аналогию между путем поиска *search_path* и путем PATH в операционных системах.

При создании нового объекта с именем без квалификатора он попадает в первую явно указанную в пути схему (не считая несуществующих и тех, на которые недостаточно прав).

<https://postgrespro.ru/docs/postgresql/12/runtime-config-client#GUC-SEARCH-PATH>

Путь поиска

Путь поиска по умолчанию имеет такое значение:

```
=> SHOW search_path;
```

```
search_path
-----
"$user", public
(1 row)
```

Конструкция «\$user» обозначает схему с тем же именем, что и имя текущего пользователя (в нашем случае — student). Поскольку такой схемы нет, она игнорируется.

Чтобы не думать над тем, какие схемы есть, каких нет, и какие не указаны явно, можно воспользоваться функцией:

```
=> SELECT current_schemas(true);
```

```
current_schemas
-----
{pg_catalog,public}
(1 row)
```

Видим, что была убрана несуществующая схема и неявно была добавлена схема системного каталога.

Установим путь поиска, например, так:

```
=> SET search_path = public, special;
```

SET

Теперь таблица будет найдена.

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

```
n
---
(0 rows)
```

Здесь мы установили конфигурационный параметр на уровне сеанса (при переподключении значение пропадет). Устанавливать такое значение на уровне всего кластера тоже не правильно — возможно, этот путь нужен не всегда и не всем.

Но параметр можно установить и на уровне отдельной базы данных:

```
=> ALTER DATABASE data_logical SET search_path = public, special;
```

ALTER DATABASE

Теперь он будет устанавливаться для всех новых подключений к БД data_logical. Проверим:

```
=> \c data_logical
```

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

```
=> SHOW search_path;
```

```
search_path
-----
public, special
(1 row)
```

Описание всех объектов кластера

набор таблиц в каждой базе данных (схема `pg_catalog`)
и несколько глобальных объектов кластера
набор представлений для удобства

Доступ

запросы SQL, специальные команды `psql`

Правила организации

названия таблиц начинаются с «`pg_`»
имена столбцов содержат трехбуквенный префикс
в качестве ключа используется столбец `oid` типа `oid`
названия объектов хранятся в нижнем регистре

Системный каталог хранит метainформацию об объектах кластера. В каждой базе данных имеется собственный набор таблиц, описывающих объекты этой конкретной базы. Также есть несколько таблиц, общих для всего кластера и не принадлежащих никакой отдельной БД. Для удобства над таблицами также определены несколько представлений.

<https://postgrespro.ru/docs/postgresql/12/catalogs>

К системному каталогу можно обращаться с помощью обычных запросов SQL, а `psql` имеет целый ряд команд для удобства просмотра. Таблицы системного каталога не обновляют напрямую; они изменяются автоматически при выполнении команд DDL.

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

Все имена таблиц системного каталога начинаются с `pg_`, например, `pg_database`. Столбцы таблиц начинаются с префикса, обычно соответствующего имени таблицы, например, `datname`. Имена объектов хранятся в нижнем регистре, например, `'postgres'`.

В таблицах системного каталога первичные ключи не определены явным образом, но в большинстве случаев в качестве первичного ключа выступает столбец `oid`, имеющий специальный тип `oid` — `object identifier` (целое 32-битное число). До версии PostgreSQL 12 этот столбец был скрытым (его можно было увидеть, только явно указав имя в списке `SELECT`).

<https://postgrespro.ru/docs/postgresql/12/datatype-oid>

Системный каталог

Для того, чтобы вывести информацию о любых объектах, psql (как и любые графические среды) обращается к таблицам системного каталога.

Например, команда `\l` для получения списка баз данных кластера, обращается к таблице:

```
=> SELECT datname FROM pg_database;
```

```
datname
-----
postgres
student
template1
template0
data_logical
(5 rows)
```

Мы всегда можем посмотреть, какие запросы выполняет команда:

```
=> \set ECHO_HIDDEN on
```

```
=> \l
```

```
***** QUERY *****
```

```
SELECT d.datname as "Name",
       pg_catalog.pg_get_userbyid(d.datdba) as "Owner",
       pg_catalog.pg_encoding_to_char(d.encoding) as "Encoding",
       d.datcollate as "Collate",
       d.datctype as "Ctype",
       pg_catalog.array_to_string(d.datacl, E'\n') AS "Access privileges"
FROM pg_catalog.pg_database d
ORDER BY 1;
*****
```

```

                                List of databases
   Name   | Owner   | Encoding | Collate | Ctype   | Access privileges
-----+-----+-----+-----+-----+-----
data_logical | student | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
postgres   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
student     | student | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
template0   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres      +
            |         |          |             |             | postgres=CTc/postgres
template1   | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/postgres      +
            |         |          |             |             | postgres=CTc/postgres
(5 rows)
```

Таким образом можно исследовать системный каталог.

Отключим вывод команд.

```
=> \set ECHO_HIDDEN off
```

Список схем находится в таблице:

```
=> SELECT nsprname FROM pg_namespaces;
```

```
nsprname
-----
pg_toast
pg_temp_1
pg_toast_temp_1
pg_catalog
public
information_schema
special
(7 rows)
```

А такие объекты, как таблицы и индексы, можно посмотреть так:

```
=> SELECT relname, relkind, relnamespace
FROM pg_class WHERE relname = 't';
```

```

relname | relkind | relnamespace
-----+-----+-----
t       | r       |          16407
(1 row)

```

Все столбцы здесь начинаются на rel (relation, отношение).

- relkind — тип объекта (r — таблица, i — индекс и т. п.);
- relnamespace — схема.

Поле relnamespace имеет тип oid; вот соответствующая строка таблицы pg_namespace:

```
=> SELECT oid, nspname FROM pg_namespace WHERE nspname = 'special';
```

```

oid | nspname
-----+-----
16407 | special
(1 row)

```

Для удобства преобразования между текстовым представлением и oid можно воспользоваться приведением к типу regnamespace:

```
=> SELECT relname, relkind, relnamespace::regnamespace::text
FROM pg_class WHERE relname = 't';
```

```

relname | relkind | relnamespace
-----+-----+-----
t       | r       | special
(1 row)

```

А вот как можно получить список объектов в схеме, например, pg_catalog:

```
=> SELECT relname, relkind FROM pg_class
WHERE relnamespace = 'pg_catalog'::regnamespace LIMIT 5;
```

```

          relname          | relkind
-----+-----
pg_statistic              | r
pg_type                   | r
pg_foreign_server          | r
pg_aggregate_fnoid_index  | i
pg_am_name_index          | i
(5 rows)

```

Аналогичные reg-типы определены и для некоторых других таблиц системного каталога. Они позволяют сократить запросы и обойтись без явного соединения таблиц.

Удаление объектов

Можно ли удалить схему special?

```
=> DROP SCHEMA special;
```

```

ERROR:  cannot drop schema special because other objects depend on it
DETAIL:  table t depends on schema special
HINT:   Use DROP ... CASCADE to drop the dependent objects too.

```

Схему нельзя удалить, если в ней находятся какие-либо объекты. Их надо перенести или удалить.

Но можно удалить схему сразу вместе со всеми ее объектами:

```
=> DROP SCHEMA special CASCADE;
```

```

NOTICE:  drop cascades to table t
DROP SCHEMA

```

Базу данных можно удалить, если к ней нет активных подключений.

```
=> \conninfo
```

```
You are connected to database "data_logical" as user "student" via socket in "/var/run/postgresql" at port "5432".
```

```
=> \c postgres
```

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

```
=> DROP DATABASE data_logical;
```

```
DROP DATABASE
```


Логически

кластер содержит базы данных,
базы данных — схемы,
схемы — конкретные объекты (таблицы, индексы и т. п.)

Базы данных создаются клонированием существующих

Схема объекта определяется по пути поиска

Полное описание содержимого кластера баз данных
хранится в системном каталоге

1. В новой базе данных создайте схему, названную так же, как и пользователь. Создайте схему app. Создайте несколько таблиц в обеих схемах.
2. Получите в `rsql` описание созданных схем и список всех таблиц в них.
3. Установите путь поиска так, чтобы при подключении к базе данных таблицы из обеих схем были доступны по неквалифицированному имени; приоритет должна иметь «пользовательская» схема. Проверьте правильность настройки.

1. База данных, схемы, таблицы

Создаем базу данных:

```
=> CREATE DATABASE data_logical;
```

CREATE DATABASE

```
=> \c data_logical
```

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

Схемы:

```
=> CREATE SCHEMA student;
```

CREATE SCHEMA

```
=> CREATE SCHEMA app;
```

CREATE SCHEMA

Таблицы для схемы student:

```
=> CREATE TABLE a(s text);
```

CREATE TABLE

```
=> INSERT INTO a VALUES ('student');
```

INSERT 0 1

```
=> CREATE TABLE b(s text);
```

CREATE TABLE

```
=> INSERT INTO b VALUES ('student');
```

INSERT 0 1

Таблицы для схемы app:

```
=> CREATE TABLE app.a(s text);
```

CREATE TABLE

```
=> INSERT INTO app.a VALUES ('app');
```

INSERT 0 1

```
=> CREATE TABLE app.c(s text);
```

CREATE TABLE

```
=> INSERT INTO app.c VALUES ('app');
```

INSERT 0 1

2. Описание схем и таблиц

Описание схем:

```
=> \dn
```

```
List of schemas
Name  | Owner
-----+-----
app    | student
public | postgres
student | student
(3 rows)
```

Описание таблиц:

```
=> \dt student.*
```

```

      List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
student | a    | table | student
student | b    | table | student
(2 rows)

```

```
=> \dt app.*
```

```

      List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
app    | a    | table | student
app    | c    | table | student
(2 rows)

```

3. Путь поиска

С текущими настройками пути поиска видны таблицы только схемы student:

```
=> SELECT * FROM a;
```

```

      s
-----
student
(1 row)

```

```
=> SELECT * FROM b;
```

```

      s
-----
student
(1 row)

```

```
=> SELECT * FROM c;
```

```

ERROR:  relation "c" does not exist
LINE 1: SELECT * FROM c;
                        ^

```

Изменим путь поиска.

```
=> ALTER DATABASE data_logical SET search_path = "$user",app,public;
```

```
ALTER DATABASE
```

```
=> \c
```

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

```
=> SHOW search_path;
```

```

      search_path
-----
"$user", app, public
(1 row)

```

Теперь видны таблицы из обеих схем, но приоритет остается за student:

```
=> SELECT * FROM a;
```

```

      s
-----
student
(1 row)

```

```
=> SELECT * FROM b;
```

```

      s
-----
student
(1 row)

```

```
=> SELECT * FROM c;
```

```
s
-----
app
(1 row)
```