

Data Organization Logical Structure



Copyright

© Postgres Professional, 2017–2025

Authors: Egor Rogov, Pavel Luzanov, Ilya Bashtanov, Igor Gnatyuk

Translated by: Liudmila Mantrova, Alexander Meleshko, Elena Sharafutdinova

Photo by: Oleg Bartunov (Phu monastery, Bhrikuti summit, Nepal)

Use of Course Materials

Non-commercial use of course materials (presentations, demonstrations) is allowed without restrictions. Commercial use is possible only with the written permission of Postgres Professional. It is prohibited to make changes to the course materials.

Feedback

Please send your feedback, comments and suggestions to:

edu@postgrespro.ru

Disclaimer

Postgres Professional assumes no responsibility for any damages and losses, including loss of income, caused by direct or indirect, intentional or accidental use of course materials. Postgres Professional company specifically disclaims any warranties on course materials. Course materials are provided “as is,” and Postgres Professional company has no obligations to provide maintenance, support, updates, enhancements, or modifications.

Databases and Templates

Schemas and Search Path

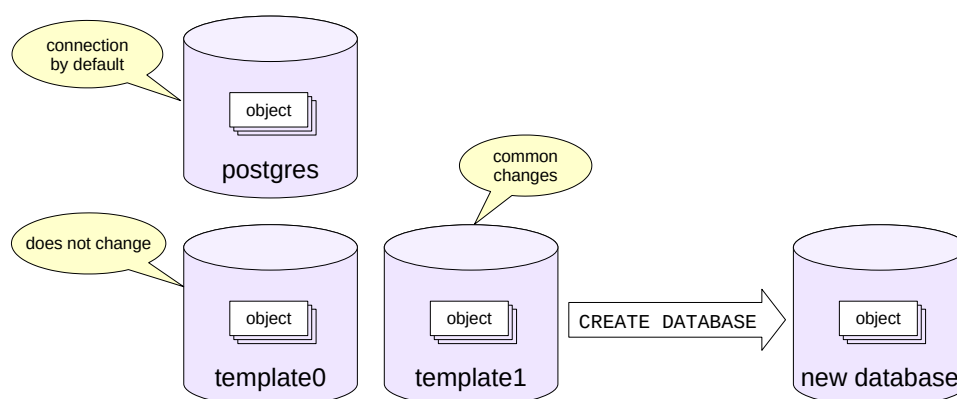
Special Schemas

System Catalog

Database Cluster

Cluster initialization creates three databases

A new database is always cloned from an existing one



3

A PostgreSQL instance manages multiple databases. This is known as a database cluster. During cluster initialization (which is performed either automatically during installation or manually using the `initdb` command), three identical databases are created. All other databases created by the user are cloned from an existing one.

By default, the *template1* DB is used as the source for creating new databases. Any objects and extensions added to the template will be copied into any database created from it.

The *template0* database must never be modified. It is required in at least two scenarios. First, to restore the database from a backup made by the `pg_dump` utility (this is discussed in the Logical Backup lesson). Secondly, it is used when creating a new database with an encoding different from the one specified during cluster initialization (this is discussed in the DBA2 course).

The *postgres* database is used to connect to by default by the *postgres* user. It is not a hard requirement to have it, but some utilities expect the *postgres* database to be there, so removing it is not a good idea, even if you never use the database directly.

<https://postgrespro.com/docs/postgresql/16/manage-ag-templatedbs>

Databases

To obtain the list of databases, open psql and run:

```
=> \l
```

```

                                List of databases
   Name   | Owner   | Encoding | Locale Provider | Collate  | Ctype    | ICU
-----+-----+-----+-----+-----+-----+-----
postgres | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
student  | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
template0 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
          |          | =c/postgres |                |            |            |
          |          | postgres=CTc/postgres |                |            |            |
template1 | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
          |          | =c/postgres |                |            |            |
          |          | postgres=CTc/postgres |                |            |            |
(4 rows)
```

The student database is there for the student user to connect to more easily. There are some columns in the output that are of no interest for us for now.

When a database is created, the template1 template is used by default.

```
=> 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 | Locale Provider | Collate  | Ctype    | ICU
-----+-----+-----+-----+-----+-----+-----
data_logical | student | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
postgres    | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
student     | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
template0   | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
          |          | =c/postgres |                |            |            |
          |          | postgres=CTc/postgres |                |            |            |
template1   | postgres | UTF8     | libc            | en_US.UTF-8 | en_US.UTF-8 |
          |          | =c/postgres |                |            |            |
          |          | postgres=CTc/postgres |                |            |            |
(5 rows)
```

Namespaces for objects

- divide objects into logical groups
- prevent name conflicts between applications

Schemas and users are different entities

Special schemas

- `public` — all objects are created here by default
- `pg_catalog` — system catalog
- `information_schema` — an alternative variant of the system catalog
- `pg_temp` — a storage for temporary tables
- ...

Schemas are namespaces for database objects. They separate objects into groups for easier management and serve to prevent name conflicts when multiple users access the same database, or when multiple applications or extensions are installed.

In PostgreSQL, *schema* and *user* are different entities (although the default settings allow users to seamlessly operate schemas of the same name).

There are several special schemas usually present in any database.

The *public* schema is used by default for storing objects, unless intentionally configured otherwise.

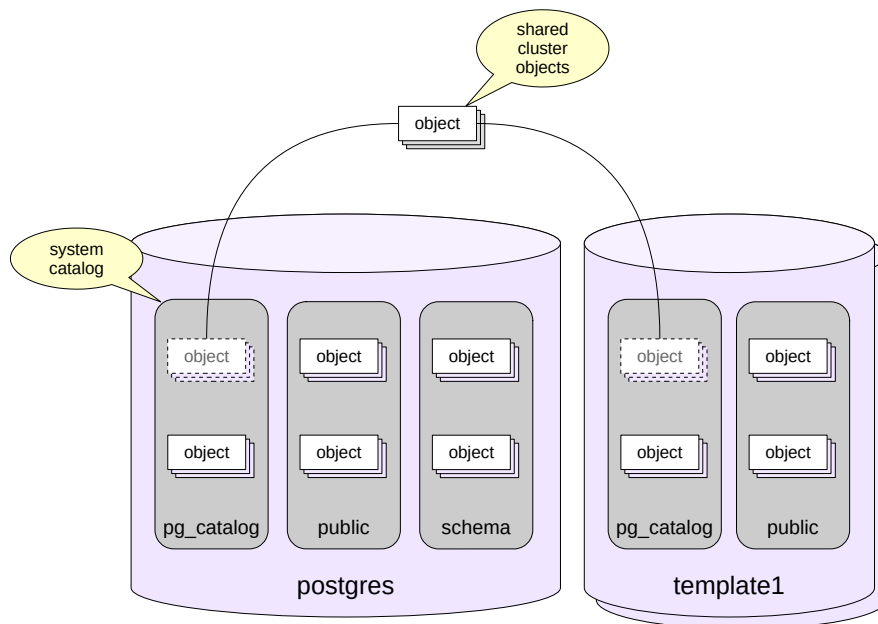
The *pg_catalog* schema stores *system catalog* objects. The system catalog is a collection of tables containing metadata about objects belonging to the cluster. It is itself stored in the cluster. The *information_schema* is another one with an alternative representation of the system catalog (as defined in the SQL standard).

The *pg_temp* schema stores temporary tables. (In fact, temporary tables are created in schemas called `pg_temp_1`, `pg_temp_2`, and so on: each user has its own schema, but they are all referred to as `pg_temp`.)

There are other special schemas, more technical in nature.

<https://postgrespro.com/docs/postgresql/16/ddl-schemas>

Cluster Databases and Schemas



6

Schemas belong to databases, and all database objects are distributed between schemas.

However, several system catalog tables store information that is shared by the entire cluster: the list of databases, the list of users, and some other data. These tables are stored outside of any single database, but at the same time they are accessible from any database within the cluster.

This way, a client connected to a database can see descriptions of not only objects that belong to the database but also of cluster-wide objects. Descriptions of objects in other databases, however, cannot be accessed without connecting to the databases first.

Schemas

To display schemas in psql, use \dn command (= describe namespace):

```
=> \dn
```

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

It does not show any system schemas, though. To list them, add the 'S' flag (it works for many other commands too):

```
=> \dnS
```

```
      List of schemas
Name | Owner
-----+-----
information_schema | postgres
pg_catalog | postgres
pg_toast | postgres
public | pg_database_owner
(4 rows)
```

We have already talked about some of these schemas (public, pg_catalog, information_schema); we will discuss the rest in other lectures.

Another useful flag is the plus sign, which displays additional information:

```
=> \dn+
```

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

Create a new schema:

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

```
CREATE SCHEMA
```

```
=> \dn
```

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

Create a table:

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

```
CREATE TABLE
```

By default, the table will be created in the public schema. To list all the tables in this schema, use the \dt command, specifying a template for schema and table names:

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

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

Tables (as well as other objects) can be moved between schemas. Since this is the logical level, the moving takes place only in the system catalog; physically, the data remains in place.

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

ALTER TABLE

What will be left in the public schema?

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

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

Nothing. What about special?

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

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

The table has been moved. Now you can access it, as long as you specify the schema.

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

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

Without the schema name, no table is found.

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

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


Determining the object's schema

schema is explicitly defined by a qualified name (*schema.name*)
name without a qualifier is looked up in the schemas specified in the search path

Search path

defined by the *search_path* parameter,
the real value is shown by the *current_schemas* function
non-existent and inaccessible schemas are excluded
objects are created in the first schema explicitly specified in the path
pg_temp and *pg_catalog* schemas are implicitly included first
unless they are already specified in the *search_path* parameter

When specifying an object, the schema it belongs to must be specified as well, since different schemas may contain objects with the same name.

If the object name is qualified by a schema, the explicitly specified schema is used. Otherwise, the schema is determined by the *search_path* configuration parameter. A search path is a list of schemas that is scanned sequentially from left to right, while excluding non-existent schemas and any schemas inaccessible to the current user.

When a new object is created with an unqualified name, the first viable schema from the search path is selected as the target schema for the object to be stored in. During object search, two schemas are implicitly added to the front of the search path:

- the *pg_catalog* schema for access to the system catalog,
- the *pg_temp* schema if the user has created any temporary objects.

The actual search path, including implicit schemas, can be obtained using the *current_schemas(true)* function.

The concept of *search_path* is similar to the *PATH* variable in operating systems.

<https://postgrespro.com/docs/postgresql/16/runtime-config-client#GUC-SEARCH-PATH>

Search Path

This is the default search path value:

```
=> SHOW search_path;
```

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

"\$user" stands for the schema with the same name as the current user (in our case, student). Since there is no student schema, it is ignored.

Instead of guessing which schemas actually exist, which do not, and which are currently unavailable, you can use the function:

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

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

The logical parameter defines if the system schemas implicitly added to the path are displayed or not. Here, in addition to excluding the non-existent schema, PostgreSQL has implicitly included the system catalog schema at the top of the list:

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

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

We can define the search path, for example:

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

```
SET
```

The table will now be found.

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

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

We have just set a configuration parameter at the session level. If we restart the session, it will have reverted to default. Setting the parameter at the cluster level is also ill-advised. Perhaps this path does not always work for everyone, and besides, different databases may have different sets of schemas.

Thankfully, you can set the parameter for a specific database:

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

```
ALTER DATABASE
```

Now, it should apply for all connections to data_logical. Let's try:

```
=> \c data_logical
```

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

```
=> SHOW search_path;
```

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

Description of all cluster objects

- the list of tables in each database (the `pg_catalog` schema) and several cluster-wide objects
- several quality-of-life views

Access

- SQL queries, special `psql` commands

Conventions

- table names start with `pg_`
- column names contain a three-letter prefix
- the column `oid` of the `oid` type is used as the primary key
- object names are always lowercase

The system catalog stores metadata about cluster objects. In each database, you can find a separate set of tables that describe the objects of this particular database, as well as several tables common to the whole cluster. There are several convenient views defined for these tables.

<https://postgrespro.com/docs/postgresql/16/catalogs>

You can access the system catalog using regular SQL queries. DDL commands can be used to modify data in the system catalog. In addition, `psql` offers a number of commands to browse the system catalog content.

<https://postgrespro.com/docs/postgresql/16/app-psql>

All system catalog table names start with `pg_`, for example, `pg_database`. Table columns start with a prefix, usually corresponding to the table name, for example, `datname`. Object names are always in lowercase, for example, `postgres`.

System catalog tables have primary keys. As a rule, these are columns with the name `oid` and of the `oid` type (object identifier, a 32-bit integer). These identifiers are also found in other columns as individual values or arrays, linking tables together logically. Foreign keys in the system catalog are not explicitly defined.

<https://postgrespro.com/docs/postgresql/16/datatype-oid>

System Catalog

To display information about an object, psql (like other interactive user tools) accesses system catalog tables.

For example, the `\l` command (list cluster databases) reads from `pg_database` table:

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

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

You can always check what queries a command performs:

```
=> \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",
  CASE d.datlocprovider WHEN 'c' THEN 'libc' WHEN 'i' THEN 'icu' END AS "Locale Provider",
  d.datcollate as "Collate",
  d.datctype as "Ctype",
  d.daticulocale as "ICU Locale",
  d.daticurules as "ICU Rules",
  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	Locale Provider	Collate	Ctype	ICU
Locale	ICU Rules	Access privileges				
data_logical	student	UTF8	libc	en_US.UTF-8	en_US.UTF-8	
postgres	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8	
student	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8	
template0	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8	
	=c/postgres		+			
	postgres=CTc/postgres					
template1	postgres	UTF8	libc	en_US.UTF-8	en_US.UTF-8	
	=c/postgres		+			
	postgres=CTc/postgres					

(5 rows)

This is one way to explore the system catalog.

Disable extended command output.

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

The schema list is stored in the table:

```
=> SELECT nspname FROM pg_namespace;
```

```

      nspname
-----
pg_toast
pg_catalog
public
information_schema
special
(5 rows)

```

You can get lists of objects such as tables or indexes like this:

```

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

 relname | relkind | relnamespace 
-----+-----+-----
t        | r       |          16387
(1 row)

```

All columns here start with rel (relation).

- relkind — object type (r - table, i - index, etc.)
- relnamespace — schema

The relnamespace field is of the oid type; here is the corresponding row in the pg_namespace table:

```

=> SELECT oid, nspname FROM pg_namespace WHERE oid = 16387;

 oid | nspname 
-----+-----
16387 | special
(1 row)

```

To easily translate between text and oid formats, a special regnamespace type is used:

```

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

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

```

For example, let's retrieve the list of objects from 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_table   | r
pg_proc_oid_index  | i
pg_proc_prname_args_nsp_index | i
(5 rows)

```

Similar reg types are defined for some other tables in the system catalog. They simplify queries and avoid unnecessary table joins.

Deleting Objects

Can you delete the special schema?

```

=> 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.

```

A schema cannot be deleted while there are any objects in it. You need to delete or move them elsewhere first.

Or you can delete a schema along with all its objects:

```

=> DROP SCHEMA special CASCADE;

```

```

NOTICE: drop cascades to table t
DROP SCHEMA

```

What about deleting the entire database? Firstly, you cannot delete the database that you are currently connected to, so let's disconnect.

```
=> \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".
```

Secondly, the database also cannot be deleted while there are any active connections to it. Let's create a connection in a separate session and try to delete the database:

```
| => \c data_logical
```

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

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

```
ERROR: database "data_logical" is being accessed by other users  
DETAIL: There is 1 other session using the database.
```

An expected error. However, there is the FORCE option that will try to forcibly terminate all database connections, and then proceed to delete it:

```
=> DROP DATABASE data_logical WITH (FORCE);
```

```
DROP DATABASE
```

On the logical level

- a cluster contains databases
- a database contains schemas
- a schema contains specific objects (tables, indexes etc.)

New databases are created by cloning existing ones

Object schemas are determined by the search path

The system catalog stores a full description of the contents of the database cluster

1. In a new database, create a schema with the same name as the current user. Create a schema named *app*. Create several tables in both schemas.
2. Use `psql` to obtain descriptions of the schemas and a list of tables within them.
3. Modify the search path parameter so that when connecting to the database, tables from both schemas are accessible by an unqualified name. The “username” schema from task 1 should have priority.
Verify that the new configuration works as intended.

1. Database, Schemas, Tables

Create a database:

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

CREATE DATABASE

```
=> \c data_logical
```

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

Schemas:

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

CREATE SCHEMA

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

CREATE SCHEMA

Tables for the student schema:

```
=> 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

Tables for the app schema:

```
=> 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. Description of Schemas and Tables

Description of schemas:

```
=> \dn
```

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

Description of tables:

```
=> \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. Search Path

The current search path settings show only the tables inside 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;
                        ^

```

Change the search path at the database level.

```
=> 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)

```

Tables from both schemas are visible now, with student taking precedence:

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

```

      s
-----
 student
(1 row)

```

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

```

      s
-----
 student
(1 row)

```

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

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

```
=> select username, application_name from pg_stat_activity where datname = 'data_logical';
```

```
username | application_name
-----+-----
student  | psql
(1 row)
```

```
=> \c postgres
```

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

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

DROP DATABASE