

Data Organization System Catalog



16

Copyright

© Postgres Professional, 2017–2025

Authors: Egor Rogov, Pavel Luzanov, Ilya Bashtanov, Alexey Beresnev

Translated by: Liudmila Mantrova, Alexander Meleshko, Elena Sharafutdinova

Photo: Oleg Bartunov (Phu Monastery and Bhrikuti Peak, 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.

What Is the System Catalog and how to Access It
System Catalog Objects and their Location
Object Naming Rules
Special Data Types

A set of tables and views
describing all objects in a database cluster

Schemas

primary schema: `pg_catalog`

alternative representation: `information_schema` (SQL standard)

SQL access

view: `SELECT`

change: `CREATE`, `ALTER`, `DROP`

psql access

commands for convenient data visualization

The system catalog is a collection of tables and views that describe all database objects. It is metadata for the contents of the cluster:

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

Starting from version 14 of PostgreSQL, primary keys and unique constraints have been added for most system catalog tables.

You can access this metadata using regular SQL queries. `SELECT` commands can give you a description of an object, and DDL (Data Definition Language) commands let you add and modify objects.

All system catalog tables and views are located in the `pg_catalog` schema.

There is another schema, as defined by the SQL standard:

`information_schema`. It is more stable and portable than `pg_catalog`, but does not reflect specific features of PostgreSQL.

Client programs can read the contents of the system catalog and display it to the user in a convenient way. For example, GUI-based development and management environments usually come with a hierarchical object navigation tool.

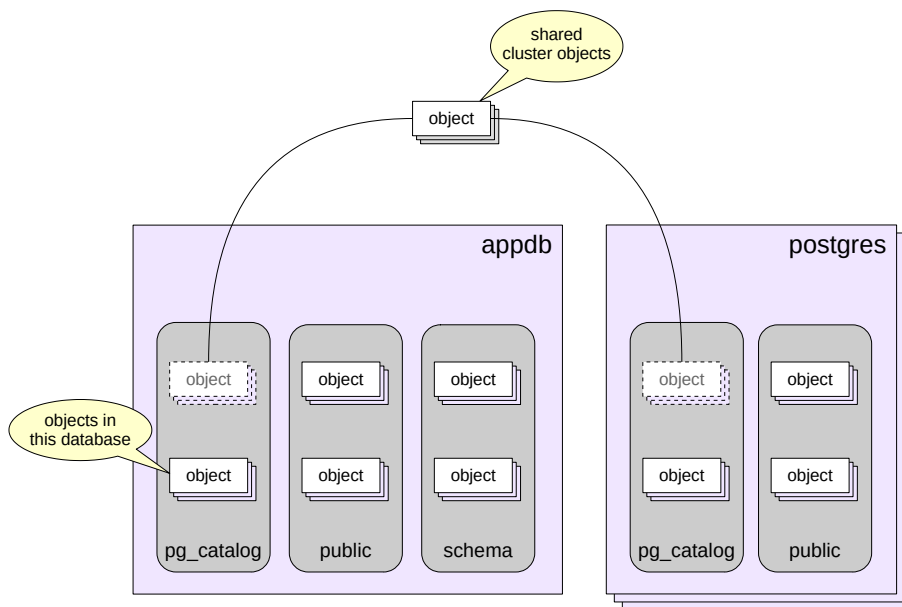
The `psql` client also offers a number of convenient built-in commands specifically designed for working with the system catalog. Most of these commands start with `\d` (as in “describe”). For the full list of commands and their descriptions, see:

<https://postgrespro.com/docs/postgresql/16/app-psql#APP-PSQL-META-COMMANDS>

We will look at the most commonly used ones in the demo.

The course materials also include the `catalogs.pdf` file that features a diagram of the main system catalog tables and related `psql` commands.

Cluster-Level Objects

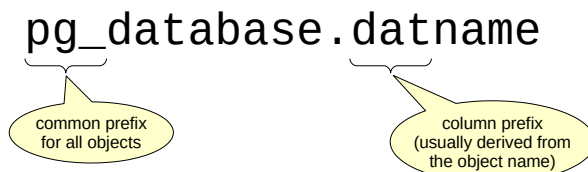


In a database cluster, each database has its own set of system catalog tables. However, there are several system catalog objects that are shared between all cluster databases. The most obvious example is the list of the databases themselves.

These tables are stored outside of any single database, but at the same time they are accessible from any database within the cluster.

Naming Rules

Object (table, view) and column name prefixes



Object names are always lowercase

All system catalog tables and views begin with the prefix `pg_`. In order to avoid potential conflicts, it is not recommended to create your own objects starting with `pg_`.

Column names have a three-letter prefix, which is usually derived from the name of the table. There is no underscore after the prefix. There are some exceptions to this rule, such as the `oid` column and others.

Object names are always stored in lowercase.

Specific pg_catalog Objects

Create a database and some test objects:

```
=> CREATE DATABASE data_catalog;
```

CREATE DATABASE

```
=> \c data_catalog
```

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

```
=> CREATE TABLE employees(  
  id integer GENERATED ALWAYS AS IDENTITY PRIMARY KEY,  
  name text,  
  manager integer  
);
```

CREATE TABLE

```
=> CREATE VIEW top_managers AS  
  SELECT * FROM employees WHERE manager IS NULL;
```

CREATE VIEW

```
=> CREATE TEMP TABLE emp_salaries(  
  employee integer,  
  salary numeric  
);
```

CREATE TABLE

We are familiar with some of the system catalog tables from the previous lesson. This is databases:

```
=> SELECT * FROM pg_database WHERE datname = 'data_catalog' \gx
```

```
-[ RECORD 1 ]--+-  
oid          | 16386  
datname      | data_catalog  
datdba       | 16384  
encoding     | 6  
datlocprovider | c  
datistemplate | f  
dataallowconn | t  
datconnlimit  | -1  
datfrozensxid | 722  
datminmxid   | 1  
dattablespace | 1663  
datcollate   | en_US.UTF-8  
datctype     | en_US.UTF-8  
daticulocale |  
daticurules  |  
datcollversion | 2.39  
datacl       |
```

And schemas:

```
=> SELECT * FROM pg_namespace WHERE nspname = 'public' \gx
```

```
-[ RECORD 1 ]--+-  
oid          | 2200  
nspname      | public  
nspowner     | 6171  
nspacl       | {pg_database_owner=UC/pg_database_owner,=U/pg_database_owner}
```

pg_class is an important table that stores descriptions for multiple types of objects: tables, views, indexes, sequences. All these objects in PostgreSQL are called relations, thus the prefix "rel" in the column names:

```
=> SELECT relname, relkind, relnamespace, relfilenode, relowner, relpersistence  
FROM pg_class WHERE relname ~ '^(emp|top)';
```

relname	relkind	relnamespace	relfilenode	relowner	relpersistence
employees_id_seq	S	2200	16387	16384	p
employees	r	2200	16388	16384	p
employees_pkey	i	2200	16393	16384	p
top_managers	v	2200	0	16384	p
emp_salaries	r	16399	16401	16384	t

(5 rows)

The object type is determined by the relkind column, while relpersistence distinguishes temporary objects from permanent ones.

When actively using temporary objects, the system catalog tables accumulate numerous obsolete row versions, which may degrade query performance at all query processing stages. In such cases, timely catalog table vacuuming becomes essential.

Naturally, only a subset of columns in pg_class are relevant for each object type. Moreover, examining object names (rather than identifiers like relnamespace, relowner, etc.) is more practical. For this purpose, there are specialized system views such as:

```
=> SELECT schemaname, tablename, tableowner
FROM pg_tables WHERE schemaname ~ '(public|pg_temp.+);
```

schemaname	tablename	tableowner
public	employees	student
pg_temp_4	emp_salaries	student

(2 rows)

```
=> SELECT *
FROM pg_views WHERE schemaname = 'public';
```

schemaname	viewname	viewowner	definition
public	top_managers	student	SELECT id, name, manager FROM employees WHERE (manager IS NULL);

(1 row)

Using psql

psql has a set of built-in commands for obtaining information from the system catalog. These short commands are more convenient than making direct queries to system tables and views.

A list of all tables is obtained with the command:

```
=> \dt
```

List of relations			
Schema	Name	Type	Owner
pg_temp_4	emp_salaries	table	student
public	employees	table	student

(2 rows)

This command returns a list of all views in the public schema:

```
=> \dv public.*
```

List of relations			
Schema	Name	Type	Owner
public	top_managers	view	student

(1 row)

List of tables, views, indexes, and sequences:

```
=> \dtvis
```

List of relations				
Schema	Name	Type	Owner	Table
pg_temp_4	emp_salaries	table	student	employees
public	employees	table	student	
public	employees_id_seq	sequence	student	
public	employees_pkey	index	student	
public	top_managers	view	student	

(5 rows)

Appended with the + modifier, these commands will return more detailed data:

=> \dt+

List of relations							
Schema	Name	Type	Owner	Persistence	Access method	Size	Description
pg_temp_4	emp_salaries	table	student	temporary	heap	8192 bytes	
public	employees	table	student	permanent	heap	8192 bytes	

(2 rows)

To get detailed information about a specific object, use the \d command (without any additional letters):

=> \d top_managers

View "public.top_managers"				
Column	Type	Collation	Nullable	Default
id	integer			
name	text			
manager	integer			

The + modifier still works:

=> \d+ top_managers

View "public.top_managers"						
Column	Type	Collation	Nullable	Default	Storage	Description
id	integer				plain	
name	text				extended	
manager	integer				plain	

View definition:

```
SELECT id,
       name,
       manager
FROM employees
WHERE manager IS NULL;
```

You can use the command not only on relations, but other objects as well, such as schemas (\dn) and functions (\df).

The S modifier makes the command display system objects in addition to user ones. You can use wildcard patterns to filter the output:

=> \dfs pg*size

List of functions				
Schema	Name	Result data type	Argument data types	Type
pg_catalog	pg_column_size	integer	"any"	func
pg_catalog	pg_database_size	bigint	name	func
pg_catalog	pg_database_size	bigint	oid	func
pg_catalog	pg_indexes_size	bigint	regclass	func
pg_catalog	pg_relation_size	bigint	regclass	func
pg_catalog	pg_relation_size	bigint	regclass, text	func
pg_catalog	pg_table_size	bigint	regclass	func
pg_catalog	pg_tablespace_size	bigint	name	func
pg_catalog	pg_tablespace_size	bigint	oid	func
pg_catalog	pg_total_relation_size	bigint	regclass	func

(10 rows)

Usually, such psql commands have mnemonic names. For example, \df is describe function, \sf is show function:

```
=> \sf pg_catalog.pg_database_size(oid)

CREATE OR REPLACE FUNCTION pg_catalog.pg_database_size(oid)
  RETURNS bigint
  LANGUAGE internal
  PARALLEL SAFE STRICT
AS $function$pg_database_size_oid$function$
```

You can get the full list of commands from the documentation or with the `psql \?` command.

Exploring the System Catalog Structure

All `psql` commands that describe objects query system catalog tables. To view these queries, set the `psql` variable `ECHO_HIDDEN`. For example, to examine the `employees` table structure:

```
=> \set ECHO_HIDDEN on

=> \dt employees

***** QUERY *****
SELECT n.nspname as "Schema",
       c.relname as "Name",
       CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN 'materialized
view' WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 't' THEN 'TOAST table' WHEN 'f'
THEN 'foreign table' WHEN 'p' THEN 'partitioned table' WHEN 'I' THEN 'partitioned index'
END as "Type",
       pg_catalog.pg_get_userbyid(c.relowner) as "Owner"
FROM pg_catalog.pg_class c
     LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
     LEFT JOIN pg_catalog.pg_am am ON am.oid = c.relam
WHERE c.relkind IN ('r','p','t','s','')
     AND c.relname OPERATOR(pg_catalog.~) '^(employees)$' COLLATE pg_catalog.default
     AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
*****
```

```

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

```
=> \unset ECHO_HIDDEN
```

Most system catalog tables feature primary keys (typically the `oid` column) and uniqueness constraints. As an example, the `pg_attribute` table (containing relation attribute information) implements these constraints:

```
=> \d pg_attribute
```

Table "pg_catalog.pg_attribute"				
Column	Type	Collation	Nullable	Default
attrelid	oid		not null	
attname	name		not null	
atttypid	oid		not null	
attlen	smallint		not null	
attnum	smallint		not null	
attcacheoff	integer		not null	
atttypmod	integer		not null	
attndims	smallint		not null	
attbyval	boolean		not null	
attalign	"char"		not null	
attstorage	"char"		not null	
attcompression	"char"		not null	
attnotnull	boolean		not null	
atthasdef	boolean		not null	
atthasmissing	boolean		not null	
attidentity	"char"		not null	
attgenerated	"char"		not null	
attisdropped	boolean		not null	
attislocal	boolean		not null	
attinhcount	smallint		not null	
attstattarget	smallint		not null	
attcollation	oid		not null	
attacl	aclitem[]			
attoptions	text[]	C		
attfdwoptions	text[]	C		
attmissingval	anyarray			

Indexes:

"pg_attribute_relid_attnum_index" PRIMARY KEY, btree (attrelid, attnum)

"pg_attribute_relid_attnam_index" UNIQUE CONSTRAINT, btree (attrelid, attname)

Referential integrity is maintained through foreign-key-like constraints with additional complexity: the referencing column may be an array of elements, zero may represent undefined references. The `pg_get_catalog_foreign_keys()` function lists these pseudo-foreign keys. For instance, `pg_attribute` references:

```
=> SELECT *
FROM   pg_get_catalog_foreign_keys()
WHERE  fktable = 'pg_attribute'::regclass;
```

fktable	fkcols	pktable	pkcols	is_array	is_opt
pg_attribute	{attrelid}	pg_class	{oid}	f	f
pg_attribute	{atttypid}	pg_type	{oid}	f	t
pg_attribute	{attcollation}	pg_collation	{oid}	f	t

(3 rows)

- `fktable`, `fkcols` — referencing table and its columns
- `pktable`, `pkcols` — key referenced
- `is_array` — whether the referencing column is an array
- `is_opt` — whether the referencing column can contain 0

oid type – object identifier

- oid column ensures object uniqueness in system catalog tables
- integer with an auto increment

reg* types

- oid aliases for *some* system catalog tables (regclass for pg_class, etc.)
- converting the text name of an object to the oid type and vice versa

Most system catalog tables use a column with oid name and data type of the same name as a primary key.

The oid (Object Identifier) type is a 32 bit integer (about 4 billion possible values) with an auto increment.

There are several special data types (in fact, oid aliases) starting with reg that are used to convert object names to oids and back.

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

oid and reg* Data Types

As shown before, table and view descriptions are stored in pg_class table, and column descriptions are in a separate pg_attribute table. So, to get a list of columns in a specific table, you need to join pg_class and pg_attribute:

```
=> SELECT a.attname, a.atttypid
FROM pg_attribute a
WHERE a.attrelid = (
    SELECT oid FROM pg_class WHERE relname = 'employees'
)
AND a.attnum > 0;
```

attname	atttypid
id	23
name	25
manager	23

(3 rows)

Using reg* types, the query can be simplified by omitting the explicit access to pg_class:

```
=> SELECT a.attname, a.atttypid
FROM pg_attribute a
WHERE a.attrelid = 'employees'::regclass
AND a.attnum > 0;
```

attname	atttypid
id	23
name	25
manager	23

(3 rows)

Here, the string “employees” was converted to the oid type. Similarly, oid can be displayed as a text value:

```
=> SELECT a.attname, a.atttypid::regtype
FROM pg_attribute a
WHERE a.attrelid = 'employees'::regclass
AND a.attnum > 0;
```

attname	atttypid
id	integer
name	text
manager	integer

(3 rows)

A list of all reg* types:

```
=> \dT reg*
```

List of data types		
Schema	Name	Description
pg_catalog	regclass	registered class
pg_catalog	regcollation	registered collation
pg_catalog	regconfig	registered text search configuration
pg_catalog	regdictionary	registered text search dictionary
pg_catalog	regnamespace	registered namespace
pg_catalog	regoper	registered operator
pg_catalog	regoperator	registered operator (with args)
pg_catalog	regproc	registered procedure
pg_catalog	regprocedure	registered procedure (with args)
pg_catalog	regrole	registered role
pg_catalog	regtype	registered type

(11 rows)

Takeaways



The system catalog contains cluster metadata stored within the cluster itself

SQL access and additional psql commands

Some system catalog tables are stored in databases, some are shared across the entire cluster

The system catalog uses special data types

1. Get a description of the `pg_class` table.
2. Get a *detailed* description of the `pg_tables` view.
3. Create a database and a temporary table in it.
Get a complete list of schemas in the database, including system schemas.
4. Get a list of views in the `information_schema`.
5. What queries does the following `psql` command perform?
`\d+ pg_views`

1. Description of pg_class

=> \d pg_class

Table "pg_catalog.pg_class"				
Column	Type	Collation	Nullable	Default
oid	oid		not null	
relname	name		not null	
relnamespace	oid		not null	
reltype	oid		not null	
reloftype	oid		not null	
relowner	oid		not null	
relam	oid		not null	
relfilenode	oid		not null	
reltablespace	oid		not null	
relpages	integer		not null	
reltuples	real		not null	
relallvisible	integer		not null	
reltoastrelid	oid		not null	
relhasindex	boolean		not null	
relisshared	boolean		not null	
relpersistence	"char"		not null	
relkind	"char"		not null	
relnatts	smallint		not null	
relchecks	smallint		not null	
relhasrules	boolean		not null	
relhastriggers	boolean		not null	
relhassubclass	boolean		not null	
relrowsecurity	boolean		not null	
relforcerowsecurity	boolean		not null	
relispopulated	boolean		not null	
relreplident	"char"		not null	
relispartition	boolean		not null	
relrewrite	oid		not null	
relfrozenxid	xid		not null	
relminmxid	xid		not null	
relacl	aclitem[]			
reloptions	text[]	C		
relpartbound	pg_node_tree	C		

Indexes:

"pg_class_oid_index" PRIMARY KEY, btree (oid)

"pg_class_relname_nsp_index" UNIQUE CONSTRAINT, btree (relname, relnamespace)

"pg_class_tblspc_relfilenode_index" btree (reltablespace, relfilenode)

2. Detailed Description of pg_tables

=> \d+ pg_tables

View "pg_catalog.pg_tables"						
Column	Type	Collation	Nullable	Default	Storage	Description
schemaname	name				plain	
tablename	name				plain	
tableowner	name				plain	
tablespace	name				plain	
hasindexes	boolean				plain	
hasrules	boolean				plain	
hastriggers	boolean				plain	
rowsecurity	boolean				plain	

View definition:

```
SELECT n.nspname AS schemaname,
       c.relname AS tablename,
       pg_get_userbyid(c.relowner) AS tableowner,
       t.spcname AS tablespace,
       c.relhasindex AS hasindexes,
       c.relhasrules AS hasrules,
       c.relhastriggers AS hastriggers,
       c.relrowsecurity AS rowsecurity
FROM pg_class c
     LEFT JOIN pg_namespace n ON n.oid = c.relnamespace
     LEFT JOIN pg_tablespace t ON t.oid = c.reltablespace
WHERE c.relkind = ANY (ARRAY['r'::"char", 'p'::"char"]);
```

3. List of All Schemas

```
=> CREATE DATABASE data_catalog;
```

```
CREATE DATABASE
```

```
=> \c data_catalog
```

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

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

```
CREATE TABLE
```

```
=> \dnS
```

```

      List of schemas
  Name                | Owner
-----+-----
information_schema    | postgres
pg_catalog             | postgres
pg_temp_4              | postgres
pg_toast              | postgres
pg_toast_temp_4       | postgres
public                | pg_database_owner
(6 rows)
```

Temporary tables are stored in schemas named pg_temp_N, where N is a number. Such schemas are created for each session where temporary objects appear, so there can be multiple schemas. To get the name of the schema for the current session, use the following system function:

```
=> SELECT pg_my_temp_schema()::regnamespace;
```

```

pg_my_temp_schema
-----
pg_temp_4
(1 row)
```

In general, the exact name of the schema is not required: you can access temporary objects in your session by using just pg_temp:

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

```

n
--
(0 rows)
```

We already know what some of the schemas are there for, and we will learn more about the rest (pg_toast*) in a later lesson.

4. Get a List of Views in the information_schema

Use the template:

```
=> \dv information_schema.*
```

List of relations		Type	Owner
Schema	Name		
information_schema	_pg_foreign_data_wrappers	view	postgres
information_schema	_pg_foreign_servers	view	postgres
information_schema	_pg_foreign_table_columns	view	postgres
information_schema	_pg_foreign_tables	view	postgres
information_schema	_pg_user_mappings	view	postgres
information_schema	administrable_role_authorizations	view	postgres
information_schema	applicable_roles	view	postgres
information_schema	attributes	view	postgres
information_schema	character_sets	view	postgres
information_schema	check_constraint_routine_usage	view	postgres
information_schema	check_constraints	view	postgres
information_schema	collation_character_set_applicability	view	postgres
information_schema	collations	view	postgres
information_schema	column_column_usage	view	postgres
information_schema	column_domain_usage	view	postgres
information_schema	column_options	view	postgres
information_schema	column_privileges	view	postgres
information_schema	column_udt_usage	view	postgres
information_schema	columns	view	postgres
information_schema	constraint_column_usage	view	postgres
information_schema	constraint_table_usage	view	postgres
information_schema	data_type_privileges	view	postgres
information_schema	domain_constraints	view	postgres
information_schema	domain_udt_usage	view	postgres
information_schema	domains	view	postgres
information_schema	element_types	view	postgres
information_schema	enabled_roles	view	postgres
information_schema	foreign_data_wrapper_options	view	postgres
information_schema	foreign_data_wrappers	view	postgres
information_schema	foreign_server_options	view	postgres
information_schema	foreign_servers	view	postgres
information_schema	foreign_table_options	view	postgres
information_schema	foreign_tables	view	postgres
information_schema	information_schema_catalog_name	view	postgres
information_schema	key_column_usage	view	postgres
information_schema	parameters	view	postgres
information_schema	referential_constraints	view	postgres
information_schema	role_column_grants	view	postgres
information_schema	role_routine_grants	view	postgres
information_schema	role_table_grants	view	postgres
information_schema	role_udt_grants	view	postgres
information_schema	role_usage_grants	view	postgres
information_schema	routine_column_usage	view	postgres
information_schema	routine_privileges	view	postgres
information_schema	routine_routine_usage	view	postgres
information_schema	routine_sequence_usage	view	postgres
information_schema	routine_table_usage	view	postgres
information_schema	routines	view	postgres
information_schema	schemata	view	postgres
information_schema	sequences	view	postgres
information_schema	table_constraints	view	postgres
information_schema	table_privileges	view	postgres
information_schema	tables	view	postgres
information_schema	transforms	view	postgres
information_schema	triggered_update_columns	view	postgres
information_schema	triggers	view	postgres
information_schema	udt_privileges	view	postgres
information_schema	usage_privileges	view	postgres
information_schema	user_defined_types	view	postgres
information_schema	user_mapping_options	view	postgres
information_schema	user_mappings	view	postgres
information_schema	view_column_usage	view	postgres
information_schema	view_routine_usage	view	postgres
information_schema	view_table_usage	view	postgres
information_schema	views	view	postgres

(65 rows)

5. Queries to the System Catalog

To see the queries psql runs, use the ECHO_HIDDEN variable:

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

```
=> \d+ pg_views
```

***** QUERY *****

```
SELECT c.oid,
       n.nspname,
       c.relname
FROM pg_catalog.pg_class c
     LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relname OPERATOR(pg_catalog.~) '^(pg_views)$' COLLATE pg_catalog.default
     AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 2, 3;
*****
```

***** QUERY *****

```
SELECT c.relchecks, c.relkind, c.relhasindex, c.relhasrules, c.relhastriggers,
       c.relrowsecurity, c.relforcerowsecurity, false AS relhasoids, c.relispartition,
       pg_catalog.array_to_string(c.reloptions || array(select 'toast.' || x from
pg_catalog.unnest(tc.reloptions) x), ', ')
, c.reltablespace, CASE WHEN c.reloftype = 0 THEN '' ELSE
c.reloftype::pg_catalog.regtype::pg_catalog.text END, c.relpersistence, c.relreplident,
am.amname
FROM pg_catalog.pg_class c
     LEFT JOIN pg_catalog.pg_class tc ON (c.reltoastrelid = tc.oid)
     LEFT JOIN pg_catalog.pg_am am ON (c.relam = am.oid)
WHERE c.oid = '12028';
*****
```

***** QUERY *****

```
SELECT a.attname,
       pg_catalog.format_type(a.atttypid, a.atttypmod),
       (SELECT pg_catalog.pg_get_expr(d.adbin, d.adrelid, true)
        FROM pg_catalog.pg_attrdef d
        WHERE d.adrelid = a.attrelid AND d.adnum = a.attnum AND a.attahasdef),
       a.attnotnull,
       (SELECT c.collname FROM pg_catalog.pg_collation c, pg_catalog.pg_type t
        WHERE c.oid = a.attcollation AND t.oid = a.atttypid AND a.attcollation <>
t.typcollation) AS attcollation,
       a.attidentity,
       a.attgenerated,
       a.attstorage,
       pg_catalog.col_description(a.attrelid, a.attnum)
FROM pg_catalog.pg_attribute a
WHERE a.attrelid = '12028' AND a.attnum > 0 AND NOT a.attisdropped
ORDER BY a.attnum;
*****
```

***** QUERY *****

```
SELECT pg_catalog.pg_get_viewdef('12028'::pg_catalog.oid, true);
*****
```

***** QUERY *****

```
SELECT r.rulename, trim(trailing ';' from pg_catalog.pg_get_ruledef(r.oid, true))
FROM pg_catalog.pg_rewrite r
WHERE r.ev_class = '12028' AND r.rulename != '_RETURN' ORDER BY 1;
*****
```

View "pg_catalog.pg_views"

Column	Type	Collation	Nullable	Default	Storage	Description
-----+-----+-----+-----+-----+-----+-----						
schemaname	name				plain	
viewname	name				plain	
viewowner	name				plain	
definition	text				extended	

View definition:

```
SELECT n.nspname AS schemaname,
       c.relname AS viewname,
       pg_get_userbyid(c.relowner) AS viewowner,
       pg_get_viewdef(c.oid) AS definition
FROM pg_class c
     LEFT JOIN pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind = 'v'::"char";
```

psql ran five queries to display this result.

=> \set ECHO_HIDDEN off