Visiting The Catalog

A Stroll Through The PostgreSQL Catalog

Charles Clavadetscher

Swiss PostgreSQL Users Group

SwissPUG Series, 19.09.2019, Zurich, Switzerland

Outline

In Short About Me

- Senior DB Engineer at KOF ETH Zurich
  - KOF is the Center of Economic Research of the
  - ETHZ the Swiss Institute of Technology in Zurich, Switzerland
  - Independent economic research on business cycle tendencies for almost all sectors
  - Maintenance of all databases at KOF: PostgreSQL, Oracle, MySQL and MSSQL Server. Focus on migrating to PostgreSQL
  - Support in business process re-engineering
- Co-founder and treasurer of the SwissPUG, the Swiss PostgreSQL Users Group
- Member of the board of the Swiss PGDay

Outline

Introduction

PostgreSQL: The Catalog

What Are The Catalog And The Information Schema ?

Source: Wikipedia - https://en.wikipedia.org/wiki/Database_catalog

The database catalog of a database instance consists of metadata in which definitions of database objects such as base tables, views (virtual tables), synonyms, value ranges, indexes, users, and user groups are stored.

The SQL standard specifies a uniform means to access the catalog, called the INFORMATION_SCHEMA, but not all databases follow this, even if they implement other aspects of the SQL standard. For an example of database-specific metadata access methods, see Oracle metadata.

Introduction

PostgreSQL: The Catalog

The Catalog

- Is a set of tables in the schema `pg_catalog`
- As such the table definitions are registered in the catalog itself.
- Contains all required information about all objects existing in the database.
- Is a central feature of all relational databases.
- Is used by the DB management system for operations.
- Is accessible for analysis on the DB structure, either directly or through a standard set of views, the `information_schema`.
- Can change between major versions.

Catalog or catalogs ? The documentation names each table in pg_catalog as a system catalog. So actually there are as many catalogs as tables. In this presentation we use the word catalog to refer to the complete set of catalogs.

Introduction

PostgreSQL: The Catalog

Catalog tables are defined in the catalog tables.

```
charles@db.localhost=# SELECT *
                       FROM pg_catalog.pg_class
                       WHERE oid = 'pg_catalog.pg_class'::regclass;

-[ RECORD 1 ]-------+----------------------------------------
relname             | pg_class
relnamespace        | 11
reltype             | 83
reloftype           | 0
relowner            | 10
relam               | 0
relfilenode         | 0
reltablespace       | 0
relpages            | 13
reltuples           | 514
relallvisible       | 2
reltoastrelid       | 0
relhasindex         | t
relisshared         | f
[...]
```

Introduction

PostgreSQL: The Catalog

The catalog can change between major versions.

The introduction of new features can lead to changes in the catalog.

```
charles@db.localhost=# \d pg_catalog.pg_class
                    Table "pg_catalog.pg_class"
       Column        |     Type     | Collation | Nullable | Default
---------------------+--------------+-----------+----------+---------
 relname             | name         |           | not null |
 relnamespace        | oid          |           | not null |
 [...]
 relrowsecurity      | boolean      |           | not null |
 relforcerowsecurity | boolean      |           | not null |
 [...]
```

Row level security is a feature introduced in Version 9.5. The pg_catalog.pg_class table prior to that version did not have attributes for it.

Introduction

Warning

pg_catalog tables are not constrained

- No foreign keys.
- No checks.
- **Consistency is not enforced as with user tables.**

Outline

Exploring The Catalog

A complex landscape

How big is the catalog ?

```
charles@db.localhost=# SELECT c.relkind, count(c.*)
                       FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n
                       WHERE n.nspname = 'pg_catalog'
                       AND c.relnamespace = n.oid
                       GROUP BY c.relkind ;
 relkind | count
---------+-------
 r       |    62  -- Relations (ordinary tables)
 v       |    59  -- Views
 i       |   115  -- Indexes

charles@db.localhost=# SELECT pg_size_pretty(sum(pg_total_relation_size(c.oid)))
                       FROM pg_catalog.pg_class c, pg_catalog.pg_namespace n
                       WHERE n.nspname = 'pg_catalog'
                       AND c.relnamespace = n.oid ;
 pg_size_pretty
----------------
 14 MB  -- Size of "empty" database
```

The size obviously changes when adding and dropping user objects.

Exploring The Catalog

A complex landscape

Catalog views are not the standardized information_schema views.

They are a human readable version of some catalog tables, partially protected, along with a set of helpful information on operations, such as statistics.

```
charles@db.localhost=# SELECT schemaname, tablename, tableowner
                       FROM pg_catalog.pg_tables  -- From a catalog view
                       WHERE schemaname = 'pg_catalog'
                       AND tablename = 'pg_class';
schemaname  | pg_catalog
tablename   | pg_class
tableowner  | postgres

charles@db.localhost=# SELECT c.relnamespace, c.relname, c.relowner
                       FROM pg_catalog.pg_class c,  -- From a catalog table
                            pg_catalog.pg_namespace n
                       WHERE c.relname = 'pg_class'
                       AND n.nspname = 'pg_catalog'
                       AND n.oid = c.relnamespace;
relnamespace   | 11
relname        | pg_class
relowner       | 10
```

Exploring The Catalog

A complex landscape

Where to find information about catalog contents ?

Descriptions of object in the database are stored (where else ?) in the catalog in table pg_catalog.pg_description. Unfortunately system catalog tables and views do not have comments associated with them.

```
charles@db.localhost=# SELECT count(d.*)
                       FROM pg_catalog.pg_description d,
                            pg_catalog.pg_class c,
                            pg_catalog.pg_namespace n
                       WHERE d.objoid = c.oid
                       AND c.relnamespace = n.oid
                       AND n.nspname = 'pg_catalog';
 count
-------
     0
```

So, what can you do ?

- Official documentation
- Mailing list archives
- Internet search

Exploring The Catalog

Helpful tricks

psql with -E option.

```
$ psql -E -h localhost -U charles db
psql (10.5)

charles@db.localhost=# \d pg_catalog.pg_class
********* 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_class)$'
  AND n.nspname OPERATOR(pg_catalog.~) '^(pg_catalog)$'
ORDER BY 2, 3;
**************************

[...]
```

All queries required to produce the display of the table definition are listed one after the other.

Exploring The Catalog

Object Identifiers

Object identifiers, short OID are used in system catalog tables to identify objects uniquely and to set up dependencies between them, althought these are not enforced through constraints.

- Basically an OID is a number (more exactly an unsigned 4-Byte integer).
- There is a set of alias types for the most common oid classes that map a human readable name to the oid and vice versa.

```
charles@db.localhost=# SELECT oid, oid::regtype, typname, typnamespace::regnamespace,
                             typowner::regrole, typlen, typarray::regtype,
                             typinput::regprocedure
                      FROM pg_catalog.pg_type WHERE oid = 'oid'::regtype;

 oid | oid | typname | typnamespace | typowner | typlen | typarray |    typinput
-----+-----+---------+--------------+----------+--------+----------+----------------
  26 | oid | oid     | pg_catalog   | postgres |      4 | oid[]    | oidin(cstring)
```

Usually these types are names beginning with "reg". So you can find its list in the documentation or using:

```
SELECT typname FROM pg_catalog.pg_type WHERE typname ~ '^reg' order by typname;
```

Exploring The Catalog

Tables 1/12

Let's have a look at a very simple statement.

```
charles@db.localhost=# CREATE TABLE public.test
                       (
                         testid SERIAL PRIMARY KEY
                       );
CREATE TABLE
```

What happened in the catalog ?

Exploring The Catalog

Tables 2/12

We have a table.

```
charles@db.localhost=# SELECT relname,
                             relnamespace::regnamespace,
                             reltype::regtype,
                             relhasindex,
                             relkind,
                             relnatts,
                             relhaspkey
                       FROM pg_catalog.pg_class
                       WHERE oid = 'public.test'::regclass;

 relname | relnamespace | reltype | relhasindex | relkind | relnatts | relhaspkey
---------+--------------+---------+-------------+---------+----------+------------
 test    | public       | test    | t           | r       |        1 | t
```

With a type "test", one attribute, at least one index and a primary key.

Exploring The Catalog

Tables 3/12

Looking closer we have not just created one type, but two. We also have the array of type test as own datatype.

```
charles@db.localhost=# SELECT typname,
                             typnamespace::regnamespace,
                             typcategory,
                             typarray::regtype
                       FROM pg_catalog.pg_type
                       WHERE typrelid = 'public.test'::regclass;

 typname | typnamespace | typcategory | typarray
---------+--------------+-------------+----------
 test    | public       | C           | test[]

charles@db.localhost=# SELECT typname,
                             typnamespace::regnamespace,
                             typcategory,
                             typelem::regtype
                       FROM pg_catalog.pg_type
                       WHERE oid = 'test[]'::regtype;

 typname | typnamespace | typcategory | typelem
---------+--------------+-------------+---------
 _test   | public       | A           | test
```

Exploring The Catalog

Tables 4/12

Our table has attributes.

```
charles@db.localhost=# SELECT attname,
                              atttypid::regtype,
                              attnum,
                              attnotnull,
                              atthasdef,
                              attisdropped
                       FROM pg_catalog.pg_attribute
                       WHERE attrelid = 'public.test'::regclass
                       ORDER BY attnum DESC;
```

attname	atttypid	attnum	attnotnull	atthasdef	attisdropped
testid	**integer**	**1**	**t**	**t**	**f**
ctid	tid	-1	t	f	f
xmin	xid	-3	t	f	f
cmin	cid	-4	t	f	f
xmax	xid	-5	t	f	f
cmax	cid	-6	t	f	f
tableoid	oid	-7	t	f	f

User defined attributes have attnum > 0. As we can see the attribute cannot be null and has a default value.

Exploring The Catalog

Tables 5/12

Let's look for the default value.

```
charles@db.localhost=# SELECT adrelid::regclass,
                              adnum,
                              adbin,
                              pg_catalog.pg_get_expr(adbin, adrelid)
                       FROM pg_catalog.pg_attrdef
                       WHERE adrelid = 'public.test'::regclass;
-[ RECORD 1 ]----------------------------------------------------------------
adrelid     | test
adnum       | 1
adbin       | {FUNCEXPR :funcid 480 :funcresulttype 23 :funcretset false :func[...]
pg_get_expr | nextval('test_testid_seq'::regclass)
```

The first parameter of our table was defined as serial. This implicitely created a sequence, now used in the default value.

Exploring The Catalog

Tables 6/12

Well the sequence must be in the catalog, as well.

```
charles@db.localhost=# SELECT seqrelid::regclass,
                              seqtypid::regtype,
                              seqstart,
                              seqincrement, '[...]' as other_stuff
                       FROM pg_catalog.pg_sequence
                       WHERE seqrelid = 'test_testid_seq'::regclass;

    seqrelid     | seqtypid | seqstart | seqincrement | other_stuff
-----------------+----------+----------+--------------+-------------
 test_testid_seq | integer  |        1 |            1 | [...]
```

And well, yes, the sequence is a table...

```
charles@db.localhost=# SELECT relname,
                              reltype::regtype,
                              relkind,
                              relnatts
                       FROM pg_catalog.pg_class
                       WHERE oid = 'test_testid_seq'::regclass;

     relname     |     reltype     | relkind | relnatts
-----------------+-----------------+---------+----------
 test_testid_seq | test_testid_seq | S       |        3
```

Exploring The Catalog

Tables 7/12

With its own type and 3 user defined attributes.

```
charles@db.localhost=# SELECT typname FROM pg_catalog.pg_type
                       WHERE typrelid = 'test_testid_seq'::regclass;
     typname
-----------------
 test_testid_seq
(1 row)

charles@db.localhost=# SELECT attname FROM pg_catalog.pg_attribute
                       WHERE attrelid = 'test_testid_seq'::regclass ORDER BY attnum DESC;
  attname
------------
 is_called
 log_cnt
 last_value
 ctid
 xmin
 cmin
 xmax
 cmax
 tableoid
```

Exploring The Catalog

Tables 8/12

And we have an index.

```
charles@db.localhost=# SELECT indexrelid::regclass,
                              indrelid::regclass,
                              indnatts,
                              indisunique,
                              indisprimary,
                              indclass
                       FROM pg_catalog.pg_index
                       WHERE indrelid = 'public.test'::regclass;

 indexrelid | indrelid | indnatts | indisunique | indisprimary | indclass
------------+----------+----------+-------------+--------------+----------
 test_pkey  | test     |        1 | t           | t            | 1978
```

As we see the index is a table with an entry in pg_class.

```
charles@db.localhost=# SELECT relname, relnamespace::regnamespace,
                              reltype::regtype, relkind, relnatts
                       FROM pg_catalog.pg_class
                       WHERE oid = 'test_pkey'::regclass;

  relname  | relnamespace | reltype | relkind | relnatts
-----------+--------------+---------+---------+----------
 test_pkey | public       | -       | i       |        1
```

As it looks the index does not have its own type. This is probably due to internals of the index implementation.

Exploring The Catalog

Tables 9/12

But an index also has operations that act on it. In the previous slide we saw the misterious indclass 1978. What is it ?

```
charles@db.localhost=# SELECT am.amname, opc.opcmethod, opc.opcname, opc.opcintype::regtype
                       FROM pg_catalog.pg_opclass opc,
                            pg_catalog.pg_am am
                       WHERE opc.oid = 1978
                       AND am.oid = opc.opcmethod;

 amname | opcmethod | opcname  | opcintype
--------+-----------+----------+-----------
 btree  |       403 | int4_ops | integer
```

Ok, a binary tree... makes sense, but wait, if the index is a table it has attributes...

```
charles@db.localhost=# SELECT attname,
                              atttypid::regtype,
                              attnum,
                              attnotnull,
                              atthasdef,
                              attisdropped
                       FROM pg_catalog.pg_attribute
                       WHERE attrelid = 'public.test_pkey'::regclass;

 attname | atttypid | attnum | attnotnull | atthasdef | attisdropped
---------+----------+--------+------------+-----------+--------------
 testid  | integer  |      1 | f          | f         | f
```

Exploring The Catalog

Tables 10/12

Anything else ? Let's check if there are dependencies somewhere.

```
charles@db.localhost=# SELECT [...]
                       FROM pg_catalog.pg_depend
                       LEFT JOIN [...]
                       WHERE (objid = 'public.test'::regclass
                              OR refobjid = 'public.test'::regclass);
```

classid	objid	refclassid	refobjid	refobjsubid	deptype	relkind
pg_class	test_testid_seq	pg_class	test	testid	a	S
pg_class	test	pg_namespace	public		n	r
pg_type		pg_class	test		i	
pg_attrdef		pg_class	test	testid	a	
pg_constraint		pg_class	test	testid	a	

The sequence and type that we saw before, the namespace, the default value for column testid and a constraint on the same column.

Exploring The Catalog

Tables 11/12

The complete statement of the previous slide, just for documentation.

```
SELECT d.classid::regclass,
       CASE d.classid
         WHEN 'pg_class'::regclass THEN d.objid::regclass::text
         ELSE NULL::text
       END AS objid,
       d.refclassid::regclass,
       CASE d.refclassid
         WHEN 'pg_class'::regclass THEN d.refobjid::regclass::text
         WHEN 'pg_namespace'::regclass THEN d.refobjid::regnamespace::text
         ELSE NULL::text
       END AS refobjid,
       CASE WHEN d.refobjsubid > 0 THEN a.attname
            ELSE NULL::text
       END AS refobjsubid,
       d.deptype,
       c.relkind
FROM pg_catalog.pg_depend d
LEFT JOIN pg_catalog.pg_attribute a ON (a.attrelid = refobjid AND a.attnum = d.refobjsubid)
LEFT JOIN pg_catalog.pg_class c ON (c.oid = d.objid)
WHERE (d.objid = 'public.test'::regclass
       OR d.refobjid = 'public.test'::regclass);
```

Exploring The Catalog

Tables 12/12

Finally, let's take a look at the constraint that we found in the dependency table.

```
charles@db.localhost=# SELECT conname,
                              contype,
                              conrelid::regclass,
                              conindid::regclass,
                              conkey
                       FROM pg_catalog.pg_constraint
                       WHERE conrelid = 'public.test'::regclass;

  conname   | contype | conrelid | conindid   | conkey
------------+---------+----------+------------+--------
 test_pkey | p       | test     | test_pkey | 1
```

Mmh... the first parameter of our table is a primary key.

Exploring The Catalog

Tables Summary

New entries were created.

```
pg_class         3 (table, index, sequence)
pg_type          3 (test, test[] and sequence type)
pg_index         1 (the index for the primary key)
pg_sequence      1 (the sequence information)
pg_attribute    17 (the user defined attribute and the system attributes
                    plus the attributes for index and sequence)
pg_attrdef       1 (the definition of the default for the user defined attribute)
pg_depend        5 (for schema, sequence, type, default value and contraint)
pg_constraint    1 (for the primary key)
```

Not bad for a very simple DDL statement.

Questions, so far ?

Exploring The Catalog

Other Interesting Catalog Tables

- Functions: pg_catalog.pg_proc stores how functions are defined, including the complete source code.
- Views: pg_catalog.pg_views

The information about a view includes its definition and dependencies such as the rewrite rules that apply to it.

```
charles@db.localhost=# CREATE VIEW public.v_test AS
                       SELECT now() AS tstamp,
                              * FROM public.test;
CREATE VIEW

charles@db.localhost=# select * from pg_catalog.pg_views
                       WHERE viewname = 'v_test';
 schemaname | viewname | viewowner |          definition
------------+----------+-----------+-------------------------
 public     | v_test   | charles   |  SELECT now() AS tstamp,+
            |          |           |     test.testid,        +
            |          |           |     test.txt            +
            |          |           |    FROM test;
```

Exploring The Catalog

Statistics

The catalog views contain lots of statistics.

```
charles@db.localhost=# SELECT relname, relkind
                       FROM pg_catalog.pg_class
                       WHERE relname ~ '^pg_stat'
                       AND relnamespace = 'pg_catalog'::regnamespace;
           relname            | relkind
------------------------------+---------
 pg_stat_activity             | v
 pg_stat_all_indexes          | v
 pg_stat_all_tables           | v
 pg_stat_sys_indexes          | v
 pg_stat_sys_tables           | v
 pg_stat_user_functions       | v
 pg_stat_user_indexes         | v
 pg_stat_user_tables          | v
 pg_stat_wal_receiver         | v
 pg_stat_xact_all_tables      | v
[...]
(33 rows)
```

Exploring The Catalog

Statistics

Most of the statistics are views. You can look up their definition.

```
charles@db.localhost=# SELECT definition
                       FROM pg_catalog.pg_views
                       WHERE viewname = 'pg_stat_user_tables';

  SELECT pg_stat_all_tables.relid,
    pg_stat_all_tables.schemaname,
    pg_stat_all_tables.relname,
    [...]
   FROM pg_stat_all_tables
  WHERE [...];

charles@db.localhost=# SELECT definition
                       FROM pg_catalog.pg_views
                       WHERE viewname = 'pg_stat_all_tables';
 SELECT c.oid AS relid,
    n.nspname AS schemaname,
    c.relname,
    pg_stat_get_numscans(c.oid) AS seq_scan,
    pg_stat_get_tuples_returned(c.oid) AS seq_tup_read,
    (sum(pg_stat_get_numscans(i.indexrelid)))::bigint AS idx_scan,
    ((sum(pg_stat_get_tuples_fetched(i.indexrelid)))::bigint + pg_stat_get_tuples_fetched(c.o
    [...]
   FROM ((pg_class c
     LEFT JOIN pg_index i ON ((c.oid = i.indrelid)))
     LEFT JOIN pg_namespace n ON ((n.oid = c.relnamespace)))
  WHERE (c.relkind = ANY (ARRAY['r'::"char", 't'::"char", 'm'::"char"]))
  GROUP BY c.oid, n.nspname, c.relname;
```

Exploring The Catalog

Statistics

The statistics collector, as well as functions and system views that display its results are a very wide topic that deserve its own lecture.

The first starting point if you plan to dig deeper into PostgreSQL statistics is the official documentation, in particular chapter 28.2, The Statistics Collector.

- List of the views.
- List of the functions.

In this context we will only look at a simple example.

Exploring The Catalog

Statistics

Let's look what statistics we have on our table.

```
charles@db.localhost=# SELECT * FROM pg_stat_user_tables
                       WHERE relid = 'public.test'::regclass ;
-[ RECORD 1 ]-------+--------
relid               | 5314183
schemaname          | public
relname             | test
seq_scan            | 0
seq_tup_read        | 0
idx_scan            | 0
idx_tup_fetch       | 0
n_tup_ins           | 0
n_tup_upd           | 0
n_tup_del           | 0
n_tup_hot_upd       | 0
n_live_tup          | 0
n_dead_tup          | 0
n_mod_since_analyze | 0
last_vacuum         |
last_autovacuum     |
last_analyze        |
[...]
```

Nothing happened so far in our table.

Exploring The Catalog

Statistics

```
charles@db.localhost=# INSERT INTO public.test (txt) VALUES ('Bla');
charles@db.localhost=# UPDATE public.test SET txt = 'Blu' WHERE testid = 1;
charles@db.localhost=# SELECT * FROM public.test;
 testid | txt
--------+-----
      1 | Blu

-[ RECORD 1 ]-------+--------
relid               | 5314183
schemaname          | public
relname             | test
seq_scan            | 1  -- from SELECT
seq_tup_read        | 1  -- from SELECT
idx_scan            | 1  -- from UPDATE
idx_tup_fetch       | 1  -- from UPDATE
n_tup_ins           | 1  -- from INSERT
n_tup_upd           | 1  -- from UPDATE
n_tup_del           | 0
n_tup_hot_upd       | 1  -- from UPDATE: No index update
n_live_tup          | 1  -- from UPDATE: new row
n_dead_tup          | 1  -- from UPDATE: old row
n_mod_since_analyze | 2  -- 2 changes since last analyzed
last_vacuum         |
[...]
```

Statistics are updated with every action.

Exploring The Catalog

Statistics

```
charles@db.localhost=# VACUUM ANALYZE public.test;

-[ RECORD 1 ]-------+------------------------------
relid               | 5314183
schemaname          | public
relname             | test
[...]
n_live_tup          | 1
n_dead_tup          | 0
n_mod_since_analyze | 0
last_vacuum         | 2019-02-12 13:04:10.639841+01
last_autovacuum     |
last_analyze        | 2019-02-12 13:04:10.640651+01
[...]
```

Vacuum "cleans" the entries and analyze updates the entry in pg_catalog.pg_class.

```
charles@db.localhost=# select * from pg_class where oid = 'public.test'::regclass;
-[ RECORD 1 ]-------+--------
relname             | test
relnamespace        | 2200
[...]
relpages            | 1
reltuples           | 1
relallvisible       | 1
[...]
```

Exploring The Catalog

Cluster-Wide Catalog Tables

Roles (pg_catalog.pg_authid) is a good example for a cluster-wide catalog table. There is a single instance of the table containing this information shared by all databases in the cluster. Since the catalog table contains sensitive information such as password hashes, it cannot be accessed directly (unless you are a superuser). The access is implemented through a system catalog view that masks the passwords.

Another example is pg_catalog.pg_database that contains information about the databases available in the cluster.

Outline

The Information Schema

Definition

The information schema is covered in the ISO/IEC standard 9075, part 4, chapter 19 of 2016.

The PostgreSQL documentation describes it as:

- The information schema consists of a set of views that contain information about the objects defined in the current database.
- The information schema **is defined in the SQL standard and can therefore be expected to be portable and remain stable - unlike the system catalogs, which are specific to PostgreSQL and are modeled after implementation concerns**.
- The information schema views do not, however, contain information about PostgreSQL-specific features ; to inquire about those you need to query the system catalogs or other PostgreSQL-specific views".

Usually, queries against the information schema only deliver information on objects if the caller is the owner or has specific privileges on them.

The Information Schema

Example: Table Constraints

Querying the system catalog.

```
charles@db.localhost=# SELECT con.connamespace::regnamespace,
                             con.conname,
                             c.relnamespace::regnamespace,
                             con.conrelid::regclass,
                             con.contype
                      FROM pg_constraint con, pg_class c
                      WHERE con.conrelid = 'public.test'::regclass
                      AND c.oid = con.conrelid;

 connamespace |  conname   | relnamespace | conrelid | contype
--------------+------------+--------------+----------+---------
 public       | test_pkey | public       | test     | p
```

Querying the information schema.

```
charles@db.localhost=# SELECT constraint_schema,
                             constraint_name,
                             table_schema,
                             table_name,
                             constraint_type
                      FROM information_schema.table_constraints
                      WHERE table_schema = 'public' AND table_name = 'test';
 constraint_schema |      constraint_name      | table_schema | table_name | constraint_type
-------------------+---------------------------+--------------+------------+-----------------
 public            | test_pkey                 | public       | test       | PRIMARY KEY
 public            | 2200_5314183_1_not_null | public       | test       | CHECK
```

The Information Schema

Example: Table Constraints

Where does the CHECK constraint come from ?

```
charles@db.localhost=# SELECT definition
                       FROM pg_catalog.pg_views
                       WHERE schemaname = 'information_schema'
                       AND viewname = 'table_constraints';
```

```
SELECT (current_database())::information_schema.sql_identifier AS constraint_catalog,
[...]
  FROM [...], pg_constraint c, [...]
UNION ALL
SELECT [...],
   ((((((nr.oid)::text || '_'::text) || (r.oid)::text) ||
     '_'::text) || (a.attnum)::text) ||
     '_not_null'::text))::information_schema.sql_identifier AS constraint_name,
   [...]
   ('CHECK'::character varying)::information_schema.character_data AS constraint_type,
   [...]
  FROM pg_namespace nr,
   pg_class r,
   pg_attribute a
 WHERE ((nr.oid = r.relnamespace) AND (r.oid = a.attrelid) AND a.attnotnull AND [...];
```

Outline

Some usages

Corner Case - Support Case

From: https://www.postgresql.org/message-id/2613.1548440336@sss.pgh.pa.us

```
[...] st_envelope_in and st_envelope_out are not mentioned here?
That explains your problem.  You'd need to add those two rows to pg_depend,
which could go something like

insert into pg_depend (classid, objid, objsubid,
                       refclassid, refobjid, refobjsubid, deptype)
values (
  'pg_proc'::regclass,
  'sde.st_envelope_in(cstring)'::regprocedure,
  0,
  'pg_type'::regclass,
  'sde.st_envelope'::regtype,
  0,
  'n');
```

Of course core developers such as Tom Lane, who offered these instructions are very well aware of the internals.

Some usages

Refactoring

You realize that your naming of columns was not always very consistent. Now you want to find out which tables have an attribute with a specific name or part of it.

```
kofadmin@kofdb.archivedb=> SELECT  n.nspname||'.'||c.oid::regclass AS tablename,
                                   a.attname,
                                   a.attnum
                           FROM pg_catalog.pg_class c,
                                pg_catalog.pg_attribute a,
                                pg_catalog.pg_namespace n
                           WHERE a.attrelid = c.oid
                           AND a.attnum > 0
                           AND a.attname ~ 'validity'
                           AND c.relkind IN ('r', 'v')
                           AND c.relnamespace = n.oid
                           AND n.nspname = 'operations';
                tablename                 |      attname       | attnum
------------------------------------------+--------------------+--------
 operations.survey                        | validity           |      9
 operations.survey_form                   | validity           |      8
 operations.mail_templates                | overwrite_validity |      8
 operations.web_form_templates            | validity           |      4
 operations.company_weights_history       | validity           |     16
 operations.survey_participants_history | validity           |      2
 [...]
```

Some usages

Developers' Request

My developer says "What is hstore ? Can't you deliver json ?"

```
kofadmin@kofdb.archivedb=> SELECT n.nspname||'.'||c.oid::regclass AS tablename,
                                  a.attname,
                                  a.attnum
                           FROM pg_catalog.pg_class c,
                                pg_catalog.pg_attribute a,
                                pg_catalog.pg_namespace n,
                                pg_catalog.pg_type t
                           WHERE a.attrelid = c.oid
                           AND a.attnum > 0
                           AND a.atttypid = t.oid
                           AND t.oid::regtype = 'hstore'::regtype
                           AND c.relkind IN ('r', 'v')
                           AND c.relnamespace = n.oid
                           AND n.nspname IN ('operations','kofdata');
                tablename                |      attname       | attnum
-----------------------------------------+--------------------+--------
 kofdata.survey_data                     | data               |     16
 kofdata.survey_data_archive             | data               |     16
 operations.company_weights              | specials           |     13
 operations.company_weights_history      | specials           |     13
[...]
```

Some usages

Developers' Error

I try to update data on the TEST database but I get this error.

```
Error in postgresqlExecStatement(conn, statement, ...) :
  RS-DBI driver: (could not Retrieve the result : ERROR:  This table cannot be modified.
CONTEXT:  PL/pgSQL function protect_table() line 4 at RAISE
)
```

What is this function ?

```
SELECT oid::regprocedure, prorettype::regtype
FROM pg_catalog.pg_proc WHERE prosrc ~ 'This table cannot be modified.';
       oid       | prorettype
-----------------+------------
 protect_table() | trigger
```

Oh, a trigger function... mmh... but on what tables is it used ?

```
SELECT c.relnamespace::regnamespace AS schemaname,
       t.tgrelid::regclass AS tablename
FROM pg_catalog.pg_trigger t,
     pg_catalog.pg_class c
WHERE c.oid = t.tgrelid
AND t.tgfoid = 'protect_table()'::regprocedure
ORDER BY c.relnamespace::regnamespace,
       t.tgrelid::regclass;

 schemaname | tablename
------------+-----------
(0 rows)
```

None... that's weird...

Some usages

Developers' Error

Hey guy, are you sure that you are trying to update stuff on the TEST DB ?

Time passes...

Sorry, I forgot to change the server parameter in my connection and was connecting to the PROD DB.

I love you, too...

Outline

Wrap Up

- If you plan to contribute to the code you will need to get in touch with at least those system catalog tables that require management.
- If you plan to build a tool for PostgreSQL you most probably will need to use queries to the system catalog tables.
- You may be confronted with questions that you cannot answer without searching for its answers in the system catalog tables.
- So it makes sense, when you have some spare time, to take a look...
- ... and a look is not a touch.

Resources

These slides

- http://www.artesano.ch/documents/04-publications/visiting_the_catalog_pdfa.pdf

Official PostgreSQL documentation

- Catalog: https://www.postgresql.org/docs/current/catalogs.html
- Structure and initial setup: https://www.postgresql.org/docs/current/bki.html
- Information schema: https://www.postgresql.org/docs/current/information-schema.html
- Object identifiers: https://www.postgresql.org/docs/current/datatype-oid.html
- System functions: https://www.postgresql.org/docs/current/functions-info.html

Contact

- Work: clavadetscher@kof.ethz.ch
  http://www.kof.ethz.ch
- SwissPUG: clavadetscher@swisspug.org
  http://www.swisspug.org
- Private: charles@artesano.ch
  http://www.artesano.ch

Thank you

Thank you very much for your attention !

Q&A