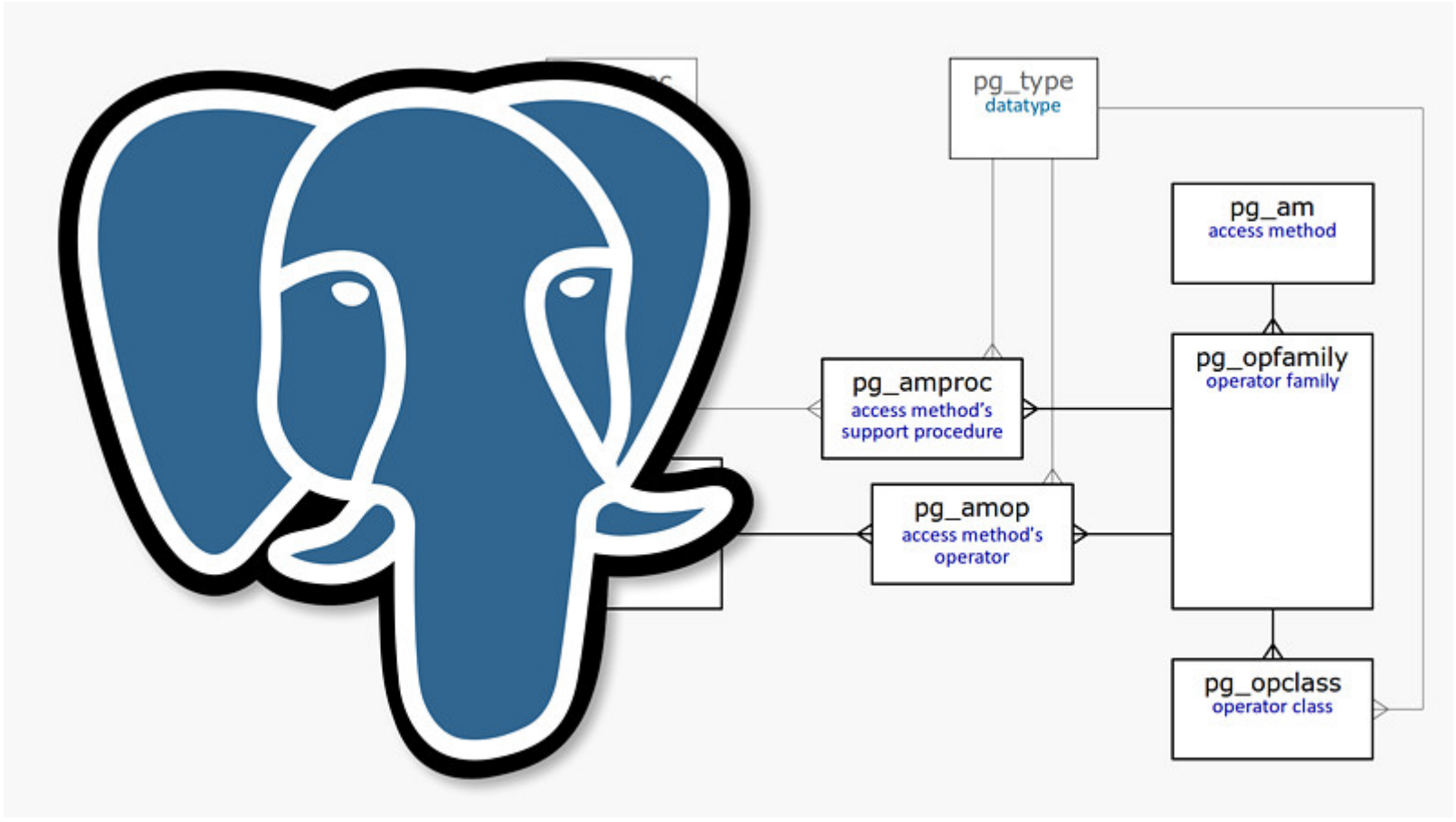
August 13, 2020 · PostgreSQL · By Egor Rogov

Indexes in PostgreSQL – 2



Habr Article

Interface

In the first article, we've mentioned that an access method must provide information about itself. Let's look into the structure of the access method interface.

Properties

All properties of access methods are stored in the "pg_am" table ("am" stands for access method). We can also get a list of available methods from this same table:

```
postgres=# select amname from pg_am;
```

```
 amname
--------
 btree
 hash
 gist
 gin
 spgist
 brin
(6 rows)
```

Although sequential scan can rightfully be referred to access methods, it is not on this list for historical reasons.

In PostgreSQL versions 9.5 and lower, each property was represented with a separate field of the "pg_am" table. Starting with version 9.6, properties are queried with special functions and are separated into several layers:

- Access method properties - "pg_indexam_has_property"

- Properties of a specific index - "pg_index_has_property"
- Properties of individual columns of the index - "pg_index_column_has_property"

The access method layer and index layer are separated with an eye towards the future: as of now, all indexes based on one access method will always have the same properties.

The following four properties are those of the access method (by an example of "btree"):

```
postgres=# select a.amname, p.name, pg_indexam_has_property(a.oid,p.name)
from pg_am a,
     unnest(array['can_order','can_unique','can_multi_col','can_exclude']) p(name)
where a.amname = 'btree'
order by a.amname;
```

```
 amname |     name      | pg_indexam_has_property
--------+---------------+-------------------------
 btree  | can_order     | t
 btree  | can_unique    | t
 btree  | can_multi_col | t
 btree  | can_exclude   | t
(4 rows)
```

- can_order.
  The access method enables us to specify the sort order for values when an index is created (only applicable to "btree" so far).
- can_unique.
  Support of the unique constraint and primary key (only applicable to "btree").
- can_multi_col.
  An index can be built on several columns.
- can_exclude.
  Support of the exclusion constraint EXCLUDE.

The following properties pertain to an index (let's consider an existing one for example):

```
postgres=# select p.name, pg_index_has_property('t_a_idx'::regclass,p.name)
from unnest(array[
       'clusterable','index_scan','bitmap_scan','backward_scan'
     ]) p(name);
```

```
     name      | pg_index_has_property
---------------+-----------------------
 clusterable   | t
 index_scan    | t
 bitmap_scan   | t
 backward_scan | t
(4 rows)
```

- clusterable.
  A possibility to reorder rows according to the index (clustering with the same-name command CLUSTER).
- index_scan.
  Support of index scan. Although this property may seem odd, not all indexes can return TIDs one by one - some return results all at once and support only bitmap scan.
- bitmap_scan.
  Support of bitmap scan.
- backward_scan.
  The result can be returned in the reverse order of the one specified when building the index.

Finally, the following are column properties:

```
postgres=# select p.name,
     pg_index_column_has_property('t_a_idx'::regclass,1,p.name)
from unnest(array[
       'asc','desc','nulls_first','nulls_last','orderable','distance_orderable',
       'returnable','search_array','search_nulls'
     ]) p(name);
```

```
        name         | pg_index_column_has_property
---------------------+------------------------------
 asc                 | t
 desc                | f
 nulls_first         | f
 nulls_last          | t
 orderable           | t
 distance_orderable  | f
 returnable          | t
 search_array        | t
 search_nulls        | t
(9 rows)
```

- asc, desc, nulls_first, nulls_last, orderable.
  These properties are related to ordering the values (we'll discuss them when we reach a description of "btree" indexes).
- distance_orderable.
  The result can be returned in the sort order determined by the operation (only applicable to GiST and RUM indexes so far).
- returnable.
  A possibility to use the index without accessing the table, that is, support of index-only scans.
- search_array.
  Support of search for several values with the expression «*indexed-field* IN (*list_of_constants*)», which is the same as «*indexed-field* = ANY(*array_of_constants*)».
- search_nulls.
  A possibility to search by IS NULL and IS NOT NULL conditions.

We've already discussed some of the properties in detail. Some properties are specific to certain access methods. We will discuss such properties when considering these specific methods.

Operator classes and families

In addition to properties of an access method exposed by the described interface, information is needed to know which data types and which operators the access method accepts. To this end, PostgreSQL introduces *operator class* and *operator family* concepts.

An operator class contains a minimal set of operators (and maybe, auxiliary functions) for an index to manipulate a certain data type.

An operator class is included in some operator family. Moreover, one common operator family can contain several operator classes if they have the same semantics. For example, "integer_ops" family includes "int8_ops", "int4_ops", and "int2_ops" classes for types "bigint", "integer", and "smallint", having different sizes but the same meaning:

```
postgres=# select opfname, opcname, opcintype::regtype
from pg_opclass opc, pg_opfamily opf
where opf.opfname = 'integer_ops'
and opc.opcfamily = opf.oid
and opf.opfmethod = (
      select oid from pg_am where amname = 'btree'
    );
```

```
   opfname   | opcname  | opcintype
-------------+----------+-----------
 integer_ops | int2_ops | smallint
 integer_ops | int4_ops | integer
 integer_ops | int8_ops | bigint
(3 rows)
```

Another example: "datetime_ops" family includes operator classes to manipulate dates (both with and without time):

```
postgres=# select opfname, opcname, opcintype::regtype
from pg_opclass opc, pg_opfamily opf
where opf.opfname = 'datetime_ops'
and opc.opcfamily = opf.oid
and opf.opfmethod = (
      select oid from pg_am where amname = 'btree'
    );
```

```
   opfname    |     opcname     |          opcintype
--------------+-----------------+----------------------------
 datetime_ops | date_ops        | date
 datetime_ops | timestamptz_ops | timestamp with time zone
 datetime_ops | timestamp_ops   | timestamp without time zone
(3 rows)
```

An operator family can also include additional operators to compare values of different types. Grouping into families enables the planner to use an index for predicates with values of different types. A family can also contain other auxiliary functions.

In most cases, we do not need to know anything about operator families and classes. Usually we just create an index, using a certain operator class by default.

However, we can explicitly specify the operator class. This is a simple example of when the explicit specification is necessary: in a database with the collation different from C, a regular index does not support the LIKE operation:

```
postgres=# show lc_collate;
```

```
 lc_collate
-------------
 en_US.UTF-8
(1 row)
```

```
postgres=# explain (costs off) select * from t where b like 'A%';
```

```
         QUERY PLAN
----------------------------
 Seq Scan on t
   Filter: (b ~~ 'A%'::text)
(2 rows)
```

We can overcome this limitation by creating an index with the operator class "text_pattern_ops" (notice how the condition in the plan has changed):

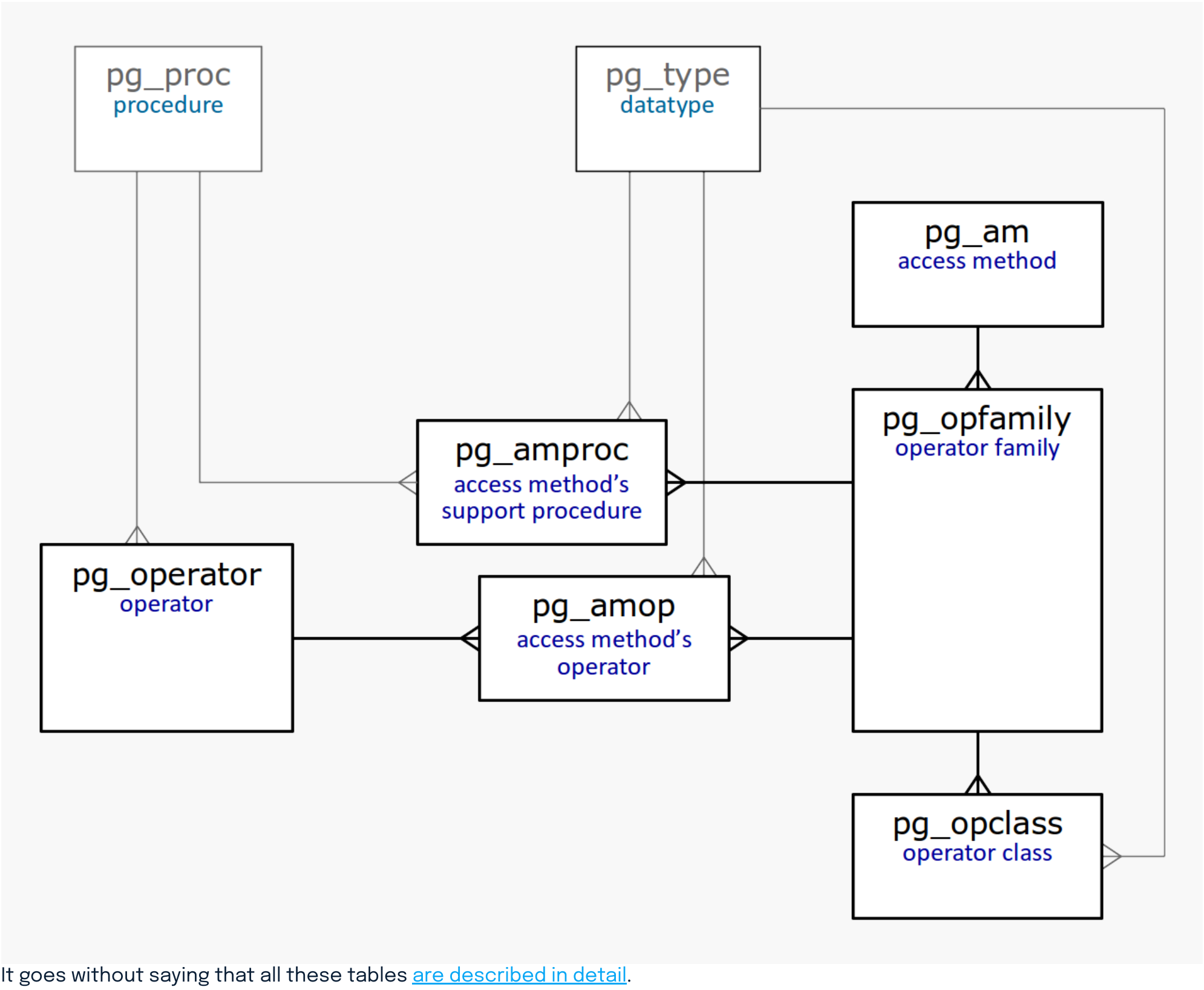
```
postgres=# create index on t(b text_pattern_ops);

postgres=# explain (costs off) select * from t where b like 'A%';
```

```
                          QUERY PLAN
---------------------------------------------------------------
 Bitmap Heap Scan on t
   Filter: (b ~~ 'A%'::text)
   ->  Bitmap Index Scan on t_b_idx1
         Index Cond: ((b ~>=~ 'A'::text) AND (b ~<~ 'B'::text))
(4 rows)
```

System catalog

In conclusion of this article, we provide a simplified diagram of tables in the system catalog that are directly related to operator classes and families.



It goes without saying that all these tables are described in detail.

The system catalog enables us to find answers to a number of questions without looking into the documentation. For example, which data types can a certain access method manipulate?

```
postgres=# select opcname, opcintype::regtype
from pg_opclass
where opcmethod = (select oid from pg_am where amname = 'btree')
order by opcintype::regtype::text;
```

```
       opcname       |          opcintype
---------------------+-----------------------------
 abstime_ops         | abstime
 array_ops           | anyarray
 enum_ops            | anyenum
...
```

Which operators does an operator class contain (and therefore, index access can be used for a condition that includes such an operator)?

```
postgres=# select amop.amopopr::regoperator
from pg_opclass opc, pg_opfamily opf, pg_am am, pg_amop amop
where opc.opcname = 'array_ops'
and opf.oid = opc.opcfamily
and am.oid = opf.opfmethod
and amop.amopfamily = opc.opcfamily
and am.amname = 'btree'
and amop.amoplefttype = opc.opcintype;
```

```
       amopopr
----------------------
 <(anyarray,anyarray)
 <=(anyarray,anyarray)
 =(anyarray,anyarray)
 >=(anyarray,anyarray)
 >(anyarray,anyarray)
(5 rows)
```

Previous article

Next article

Egor Rogov

← Back to all articles

Products

Postgres Pro Enterprise
Postgres Pro Standard
Cloud Solutions
Postgres Extensions

Services

24×7×365 Technical Support
Migration to Postgres
High Availability Deployment
Database Audit
Remote DBA for PostgreSQL

About

Leadership team
Partners
Customers
In the News
Press Releases
Press Info

Resources

Blog
Documentation
Webinars
Videos
Presentations

Community

Events
Training Courses
Intro Book
Demo Database
Mailing List Archives

Contacts

Neptune House, Marina Bay, office 207, Gibraltar, GX11 1AA
info@postgrespro.com

Get in touch!

Your First and Last Name

Company

E-mail

Message

I confirm that I have read and accepted PostgresPro's Privacy Policy.

I agree to get Postgres Pro discount offers and other marketing communications.

Send a message

EULA

EULA for Cloud Environments

Privacy Policy