Basic Tools

Installation and Management; psql

Authors: Egor Rogov, Pavel Luzanov
Translated by Liudmila Mantrova

Contact Us
Please send your feedback to: edu@postgrespro.ru

Agenda

PostgreSQL installation types

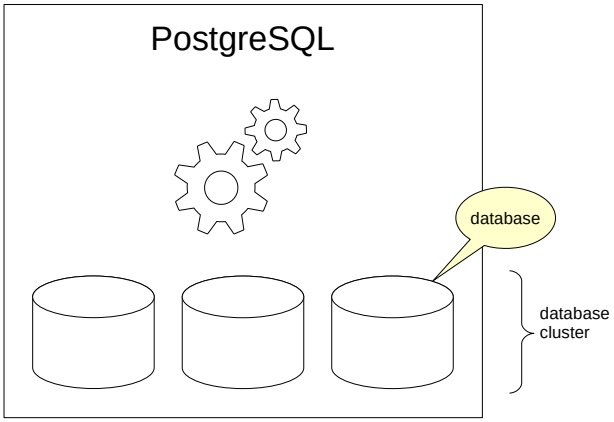
Managing the server

Server log file

Setting up configuration parameters

Using psql

Database Cluster and Server



Let’s start with the main concepts.

PostgreSQL is a program that belongs to the class of database management systems.

When this program is running, we call it a PostgreSQL server, or server instance. A server is still a “black box” for us, but soon we’ll learn how it works.

All data managed by PostgreSQL is stored in databases. A single PostgreSQL instance can work with several databases at the same time. This set of databases is called a database cluster. We'll discuss databases in more detail in lecture “Data organization. Logical structure”.

To sum it up: a database cluster stores data in files; a server or a server instance is a program that manages the database cluster.

Installation

Installation Types

- pre-built packages (preferred)
- installation from source code
- managed databases (cloud services)

Extensions

- provide additional features
- are installed separately
- are shipped with the server as modules and programs (~50 extensions)

The preferable way to install PostgreSQL is via package managers (such as rpm or dpkg) using pre-built packages. In this case, you get a comprehensive installation that is easy to support and upgrade. Pre-built packages are available for most operating systems.

Another option is to build PostgreSQL from source code. It can be necessary if you would like to use PostgreSQL in a non-standard configuration or run it on an exotic platform.

Pre-built packages and source code are available at http://www.postgresql.org/download/

Besides, you can work with cloud-based managed databases that do not require any installation at all. Such ready-to-use services are provided by all major cloud platforms (Amazon RDS, Google Cloud SQL, Microsoft Azure), as well as Yandex.Cloud and Mail.ru Cloud Solutions.

In this course, we are going to use a virtual machine with Xubuntu OS; PostgreSQL is installed from the package for this OS.

There are a lot of PostgreSQL extensions that add new database functionality “on the fly,” without modifying the system core. About 50 extensions are included into the PostgreSQL distribution itself.

https://postgrespro.com/docs/postgresql/12/contrib

https://postgrespro.com/docs/postgresql/12/contrib-prog

You can look up the list of available extensions and check if they are already installed in the `pg_available_extensions` view.

Managing the Server

Server management utility

pg_ctlcluster | pg_ctl

Primary tasks

- start the server
- stop the server
- reload server configuration

The main server management tasks are: initializing the database cluster, starting and stopping the server, reloading configuration parameters, and a couple of others. To perform these actions, use the pg_ctl utility, which is provided together with PostgreSQL.

For Ubuntu OS, the package distribution has no direct access to pg_ctl: it provides the pg_ctlcluster wrapper for this utility. To view reference documentation for pg_ctlcluster, run the following command:

```
$ man pg_ctlcluster
```

For more information about managing the server that can be useful for DBAs, see:

https://postgrespro.com/docs/postgresql/12/app-pg-ctl

https://postgrespro.com/docs/postgresql/12/runtime

Installation and Management

In the course VM, PostgreSQL is installed from a pre-built package. Let's take a look at the installation directory:

```
student$ ls -l /usr/lib/postgresql/12
```

```
total 8
drwxr-xr-x 2 root root 4096 Sep 27 19:36 bin
drwxr-xr-x 4 root root 4096 Sep 27 19:37 lib
```

The server is owned by root.

The database cluster is initialized automatically during the package installation and is located in /var/lib/postgresql/12/main.

In the topics that follow, we are going to refer to this directory as PGDATA, which is the name of the environment variable that can be set for use in some server utilities.

The PGDATA directory is owned by the postgres user. Here is its contents:

```
student$ sudo ls -l /var/lib/postgresql/12/main
```

```
total 84
drwx------ 8 postgres postgres 4096 Oct 19 17:00 base
drwx------ 2 postgres postgres 4096 Oct 19 17:00 global
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_commit_ts
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_dynshmem
drwx------ 4 postgres postgres 4096 Oct 19 17:00 pg_logical
drwx------ 4 postgres postgres 4096 Oct 19 17:00 pg_multixact
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_notify
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_replslot
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_serial
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_snapshots
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_stat
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_stat_tmp
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_subtrans
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_tblspc
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_twophase
-rw------- 1 postgres postgres    3 Oct 19 17:00 PG_VERSION
drwx------ 3 postgres postgres 4096 Oct 19 17:00 pg_wal
drwx------ 2 postgres postgres 4096 Oct 19 17:00 pg_xact
-rw------- 1 postgres postgres   88 Oct 19 17:00 postgresql.auto.conf
-rw------- 1 postgres postgres  130 Oct 19 17:00 postmaster.opts
-rw------- 1 postgres postgres  108 Oct 19 17:00 postmaster.pid
```

Package installation enables PostgreSQL autostart, so you don't have to launch PostgreSQL after loading the operating system.

To explicitly manage the server, you can run the following commands either on behalf of the postgres OS user, or using sudo:

Stop the server:

```
student$ sudo pg_ctlcluster 12 main stop
```

Start the server:

```
student$ sudo pg_ctlcluster 12 main start
```

Restart the server:

```
student$ sudo pg_ctlcluster 12 main restart
```

Reload server configuration:

```
student$ sudo pg_ctlcluster 12 main reload
```

Server Log File

A log file can contain:

- server signal messages
- user session messages
- application messages

You can configure:

- log file location
- message format
- events to log

Database operation is tracked in the server log. It records the information about starting and stopping the server, as well as various signal messages about possible issues.

Log files can also contain the information about the executed commands, their execution time, locks, etc. It can be used for tracing user sessions.

Application developers can direct their own messages to the server log.

PostgreSQL settings enable you to fine-tune the scope and format of logged error messages.

For example, the CSV output format is convenient for automating log analysis.

https://postgrespro.com/docs/postgresql/12/runtime-config-logging

Server Log

The server log is located here:

```
student$ ls -l /var/log/postgresql/postgresql-12-main.log
```

```
-rw-r----- 1 postgres adm 1791 Oct 19 17:00 /var/log/postgresql/postgresql-12-main.log
```

Let's take a look at the last lines in the log file:

```
student$ tail -n 10 /var/log/postgresql/postgresql-12-main.log
```

```
2021-10-19 17:00:12.887 MSK [3382] LOG:  shutting down
2021-10-19 17:00:12.901 MSK [3380] LOG:  database system is shut down
2021-10-19 17:00:13.103 MSK [3457] LOG:  starting PostgreSQL 12.8 (Ubuntu 12.8-1.pgdg20.04+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0, 64-bit
2021-10-19 17:00:13.104 MSK [3457] LOG:  listening on IPv4 address "127.0.0.1", port 5432
2021-10-19 17:00:13.107 MSK [3457] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2021-10-19 17:00:13.128 MSK [3458] LOG:  database system was shut down at 2021-10-19 17:00:12 MSK
2021-10-19 17:00:13.134 MSK [3457] LOG:  database system is ready to accept connections
2021-10-19 17:00:15.535 MSK [3457] LOG:  received SIGHUP, reloading configuration files
2021-10-19 17:00:15.597 MSK [3544] student@student ERROR:  database "tools_overview" does not exist
2021-10-19 17:00:15.597 MSK [3544] student@student STATEMENT:  DROP DATABASE tools_overview;
```

Configuration Parameters

For the whole instance:

- the main configuration file is postgresql.conf
- `ALTER SYSTEM` creates postgresql.auto.conf

For the current session:

- `SET/RESET`
- `set_config()`

To view the current setting:

- `SHOW`
- `current_setting()`
- `pg_settings`

PostgreSQL server is set up using various configuration parameters. They define how to manage resource consumption, tune system processes and user sessions, manage the server log, and handle many other tasks. We will deal with some of these parameters later in this course. But now it's important to figure out how to check and update the current settings.

Server settings are usually defined in configuration files. The main configuration file is postgresql.conf; it has to be edited manually. Another configuration file is postgresql.auto.conf; it is updated by the `ALTER SYSTEM` command. Parameter values set via `ALTER SYSTEM` override those that are defined in postgresql.conf.

Most settings can be changed in user sessions without a server restart.

Different ways of setting and updating parameters are described here:

https://postgrespro.com/docs/postgresql/12/config-setting

Current parameter values are displayed in the `pg_settings` view:

https://postgrespro.com/docs/postgresql/12/view-pg-settings

Configuration Parameters

The main configuration file is postgresql.conf. It is located in the following directory:

```
student$ ls -l /etc/postgresql/12/main
```

```
total 56
drwxr-xr-x 2 postgres postgres  4096 Oct 19 17:00 conf.d
-rw-r--r-- 1 postgres postgres   315 Oct 19 17:00 environment
-rw-r--r-- 1 postgres postgres   143 Oct 19 17:00 pg_ctl.conf
-rw-r----- 1 postgres postgres  4933 Oct 19 17:00 pg_hba.conf
-rw-r----- 1 postgres postgres  1636 Oct 19 17:00 pg_ident.conf
-rw-r--r-- 1 postgres postgres 27019 Oct 19 17:00 postgresql.conf
-rw-r--r-- 1 postgres postgres   317 Oct 19 17:00 start.conf
```

Other configuration files are also located in this directory.

Let's check the value of the work_mem parameter:

```
=> SHOW work_mem;
```

```
 work_mem
----------
 4MB
(1 row)
```

This parameter defines the amount of memory that can be used by internal operations of sorting and building hash tables before they start utilizing temporary files on disk.

The default value is 4MB, but it is too small. Suppose we would like to increase it to 16MB for the whole instance. There are two ways to do it.

The first option is to modify postgresql.conf by uncommenting and editing the line that defines this parameter:

```
student$ grep '#work_mem' /etc/postgresql/12/main/postgresql.conf
```

```
#work_mem = 4MB                         # min 64kB
```

Another option is to use an SQL command. Let's try it out:

```
=> ALTER SYSTEM SET work_mem TO '16MB';
```

```
ALTER SYSTEM
```

This change does not make it into postgresql.conf; it gets registered in another file (located in the PGDATA directory):

```
student$ cat /var/lib/postgresql/12/main/postgresql.auto.conf
```

```
cat: /var/lib/postgresql/12/main/postgresql.auto.conf: Permission denied
```

For the change to take effect, configuration files have to be reloaded. We can either use pg_ctlcluster, or call the following SQL function:

```
=> SELECT pg_reload_conf();
```

```
 pg_reload_conf
----------------
 t
(1 row)
```

Let's check that the new value has been applied. Instead of the SHOW command, we can use the following query:

```
=> SELECT current_setting('work_mem');
```

```
 current_setting
-----------------
 16MB
(1 row)
```

To restore the default value, simply use RESET instead of the SET command (and don't forget to reload configuration files):

```
=> ALTER SYSTEM RESET work_mem;
```

```
ALTER SYSTEM
```

```
=> SELECT pg_reload_conf();
```

```
 pg_reload_conf
----------------
 t
(1 row)
```

Most parameters can be modified for the current session on the fly. For example, if we are going to run a query that sorts a lot of data, we can increase work_mem for this session:

```
=> SET work_mem = '64MB';
```

```
SET
```

Let’s use another way to check the current setting. Here we’ll query the pg_settings view:

```
=> SELECT name, setting, unit FROM pg_settings WHERE name = 'work_mem';
```

```
   name   | setting | unit
----------+---------+------
 work_mem | 65536   | kB
(1 row)
```

The new value applies only to the current session (or to the current transaction if SET LOCAL was specified).

psql Client

A command-line client for working with PostgreSQL

Shipped together with the database system

Used by DBAs and developers for running scripts and interacting with the server

There are various third-party clients for PostgreSQL, but their evaluation is beyond the scope of this course.

We are going to use psql, a command-line client:

- It is the only client shipped with the server.

- The experience of working in psql will be useful to both DBAs and developers, regardless of which tool they are going to use in the future.

To enable the interactive mode, psql provides built-in support for readline and pager programs (such as less). Using psql, you can interact with the operating system, browse through the system catalog, and write scripts to automate routine tasks.

https://postgrespro.com/docs/postgresql/12/app-psql

Connection

When starting psql, you have to specify connection parameters. The mandatory parameters are:

- database name, coincides with the username by default;
- username (role), coincides with the name of the current OS user by default;
- host, local connection is used by default;
- port, 5432 is usually used by default.

Parameters are specified like this:

```
student$ psql -d database -U role -h host -p port
```

VM settings allow connecting to PostgreSQL without specifying any parameters:

```
student$ psql
```

Let's check the current connection:

```
=> \conninfo
```

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

The \connect command establishes a new connection from within psql. It can be shortened to \c. Here we are going to indicate an optional part of the command in square brackets: \c[onnect].

Reference Information

Reference information on psql is available both in documentation and directly from the command line. The command

```
student$ psql --help
```

explains how to launch psql. If documentation is installed, you can also get man page reference by running

```
student$ man psql
```

The psql utility can run two types of commands: SQL commands and its own commands that start with a backslash, like \conninfo.

You can get the list of these commands with their short descriptions from within psql:

- \h[elp] displays the list of SQL commands supported by the server or the syntax of a particular SQL command specified as its argument.
- \? displays the list of psql commands.

Formatting Output

The psql client can display query results in different output formats:

- a format with aligned values;
- unaligned format;
- expanded format.

By default, the aligned output format is used:

```
=> SELECT name, setting, unit FROM pg_settings LIMIT 7;
```

```
          name           |  setting   | unit
-------------------------+------------+------
 allow_system_table_mods | off        |
 application_name        | psql       |
 archive_cleanup_command |            |
 archive_command         | (disabled) |
 archive_mode            | off        |
 archive_timeout         | 0          | s
 array_nulls             | on         |
(7 rows)
```

In this format, column headings and the row count footer are displayed, and column width is aligned by values.

The following psql commands are used to switch between output formats:

- \a toggles between unaligned and aligned output formats.
- \t toggles the display of column headings and row count footer.

Let's switch off the alignment and hide the header and footer:

```
=> \a \t
```

```
Output format is unaligned.
Tuples only is on.
```

```
=> SELECT name, setting, unit FROM pg_settings LIMIT 7;
```

```
allow_system_table_mods|off|
application_name|psql|
archive_cleanup_command||
archive_command|(disabled)|
archive_mode|off|
archive_timeout|0|s
array_nulls|on|
```

```
=> \a \t
```

```
Output format is aligned.
Tuples only is off.
```

This format is hard to view, but can be quite useful for automated output processing.

The expanded format is convenient if you have to display many columns for just a couple of entries. To turn it on, specify \gx at the end of the command instead of the semicolon:

```
=> SELECT name, setting, unit, category, context, vartype,
    min_val, max_val, boot_val, reset_val
FROM pg_settings
WHERE name = 'work_mem' \gx
```

```
-[ RECORD 1 ]----------------------
name      | work_mem
setting   | 4096
unit      | kB
category  | Resource Usage / Memory
context   | user
vartype   | integer
min_val   | 64
max_val   | 2147483647
boot_val  | 4096
reset_val | 4096
```

If the expanded format is required on a regular basis (and not just for one command), you can turn it on using the \x toggle.

All formatting capabilities can be configured with the \pset command.

Interacting with OS and Running Scripts

Shell commands can be run from within psql:

```
=> \! pwd
```

```
/home/student
```

You can create an SQL query that produces several other SQL queries, and write them into a file using the \o[ut] command:

```
=> \a \t
```

```
Output format is unaligned.
Tuples only is on.
```

```
=> \pset fieldsep ''
```

```
Field separator is "".
```

```
=> \o dev1_psql.log
```

```
=> SELECT format('SELECT %L AS tbl, count(*) FROM %I;', tablename, tablename)
FROM pg_tables LIMIT 3;
```

Nothing is displayed in the terminal. Let's take a look into the file:

```
=> \! cat dev1_psql.log
```

```
SELECT 'pg_statistic' AS tbl, count(*) FROM pg_statistic;
SELECT 'pg_type' AS tbl, count(*) FROM pg_type;
SELECT 'pg_foreign_server' AS tbl, count(*) FROM pg_foreign_server;
```

Let's direct the output back to the terminal and restore the default formatting.

```
=> \o \t \a
```

```
Tuples only is off.
Output format is aligned.
```

Now run these commands from a file using \i[nclude]:

```
=> \i dev1_psql.log
```

```
     tbl      | count
--------------+-------
 pg_statistic |   422
(1 row)

   tbl   | count
---------+-------
 pg_type |   406
(1 row)

        tbl        | count
-------------------+-------
 pg_foreign_server |     0
(1 row)
```

Note that you can achieve the same result in a single step using the \gexec command:

```
=> SELECT format('SELECT %L AS tbl, count(*) FROM %I;', tablename, tablename)
FROM pg_tables LIMIT 3 \gexec
```

```
     tbl      | count
--------------+-------
 pg_statistic |   422
(1 row)

   tbl   | count
---------+-------
 pg_type |   406
(1 row)

        tbl        | count
-------------------+-------
 pg_foreign_server |     0
(1 row)
```

Here are some other ways to run commands from a file:

- psql < filename
- psql -f filename
- psql -c 'command' (works for a single command only)

---

psql Variables

Just like shell, psql provides its own variables.

Let's set a variable:

```
=> \set TEST Hi!
```

To display the assigned value, put a colon before the variable name:

```
=> \echo :TEST
```

```
Hi!
```

You can unset the variable value as follows:

```
=> \unset TEST
```

```
=> \echo :TEST
```

```
:TEST
```

Variables can be used for things like storing the text of frequently used queries. Here is a query that returns top five biggest tables:

```
=> \set top5 'SELECT tablename, pg_total_relation_size(schemaname||''.''||tablename) AS bytes FROM pg_tables ORDER BY bytes DESC LIMIT 5;'
```

To run the query, it is enough to enter:

```
=> :top5
```

```
   tablename    |  bytes
----------------+---------
 pg_depend      | 1130496
 pg_proc        | 1015808
 pg_attribute   |  688128
 pg_rewrite     |  679936
 pg_description |  573440
(5 rows)
```

It is convenient to add the command that assigns the top5 variable into the .psqlrc startup file located in the user's home directory. The commands written in .psqlrc will be run automatically each time psql is started.

You can assign a query result to a variable using \gset:

```
=> SELECT current_setting('work_mem') AS current_work_mem \gset
```

```
=> \echo work_mem value: :current_work_mem
```

```
work_mem value: 4MB
```

When run without parameters, \set returns values of all variables, including the built-in ones. To get help on built-in variables only, run:

```
\? variables
```

Summary

Installing PostgreSQL from pre-built packages is the preferred installation type

Pre-packaged distributions bring some OS specifics that you should know:

- how to start and stop the server
- where configuration files are located
- where to find the server log

psql is a client application for working with PostgreSQL

Practice

1. Set the *work_mem* parameter to 8 MB in the postgresql.conf file. Reload the server configuration and check that the changes have come into effect.
2. Create a file called ddl.sql. In this file, write the `CREATE TABLE` command that creates an arbitrary table.
   Create another file called populate.sql; it should contain some commands that insert rows into this table.
   Start psql, run both scripts, and check that the table is created and contains the specified rows.
3. Find today's entries in the log file.

To do practical assignments, you have to log in to the operating system on behalf of the `student` user (the password is `student`).

Start psql in the terminal by typing psql, without parameters. The connection will be established with the default settings.

```
student:~$ psql
```

It is convenient to have separate databases for tasks that are related to different topics. Let's create one:

```
student/student=# CREATE DATABASE tools_overview;
CREATE DATABASE

student/student=# \c tools_overview
You are now connected to database "tools_overview" as
user "student".

student/tools_overview=#
```

Task 1. You can use any text editor. The virtual machine provides mousepad, gedit, vim, nano.

Note: if you launch an editor from the GUI environment instead of the terminal, it will be started on behalf of the `student` OS user.

Task 1. Configuration Parameters

Add the following line to the end of the configuration file:

```
student$ echo 'work_mem = 8MB' | sudo tee -a /etc/postgresql/12/main/postgresql.conf
```

```
work_mem = 8MB
```

You can do it in any text editor.

Reload the server configuration:

```
student$ sudo pg_ctlcluster 12 main reload
```

Check the result:

```
student$ psql
```

```
=> SELECT current_setting('work_mem') AS work_mem;
```

```
 work_mem
----------
 8MB
(1 row)
```

Task 2. Running Scripts in psql

In the ddl.sql file, write a command that creates a table with PostgreSQL keywords (you can use any text editor):

```
student$ cat >ddl.sql <<EOF
CREATE TABLE keywords (
word text,
category text,
description text
);
EOF
```

In the populate.sql file, write commands that fill the keywords table:

```
student$ cat >populate.sql <<EOF
INSERT INTO keywords
SELECT * FROM pg_get_keywords();
EOF
```

Create a database and connect to this database:

```
=> CREATE DATABASE tools_overview;
```

```
CREATE DATABASE
```

```
=> \c tools_overview
```

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

Run the scripts and check the table entries:

```
=> \i ddl.sql
```

```
CREATE TABLE
```

```
=> \i populate.sql
```

```
INSERT 0 442
```

```
=> SELECT * FROM keywords LIMIT 10;
```

```
   word    | category | description
-----------+----------+-------------
 abort     | U        | unreserved
 absolute  | U        | unreserved
 access    | U        | unreserved
 action    | U        | unreserved
 add       | U        | unreserved
 admin     | U        | unreserved
 after     | U        | unreserved
 aggregate | U        | unreserved
 all       | R        | reserved
 also      | U        | unreserved
(10 rows)
```

Task 3. Viewing the Log File

You can open the log in any text editor. Each log entry starts with the date (this is the default setting in package installations) and can take several lines. Today's entries will appear at the end of the file.

```
student$ tail /var/log/postgresql/postgresql-12-main.log
```

```
2021-10-19 17:04:37.604 MSK [31255] bob@access_overview CONTEXT:  SQL function "foo" statement 1
2021-10-19 17:04:37.604 MSK [31255] bob@access_overview STATEMENT:  SELECT foo();
2021-10-19 17:04:37.976 MSK [31255] bob@access_overview ERROR:  permission denied for function foo
2021-10-19 17:04:37.976 MSK [31255] bob@access_overview STATEMENT:  SELECT foo();
2021-10-19 17:04:38.339 MSK [31255] bob@access_overview ERROR:  permission denied for function baz
2021-10-19 17:04:38.339 MSK [31255] bob@access_overview STATEMENT:  SELECT baz();
2021-10-19 17:05:00.786 MSK [34215] student@student ERROR:  database "tools_overview" does not exist
2021-10-19 17:05:00.786 MSK [34215] student@student STATEMENT:  DROP DATABASE tools_overview;
2021-10-19 17:05:00.969 MSK [7687] LOG:  received SIGHUP, reloading configuration files
2021-10-19 17:05:00.972 MSK [7687] LOG:  parameter "work_mem" changed to "8MB"
```