

What is pg_cron?

pg_cron is a simple cron-based job scheduler for PostgreSQL (10 or higher) that runs inside the database as an extension.

Contents

- How pg_cron works
 - Cron syntax
 - Managing and creating jobs
 - Creating a cron job
 - Creating a cron job in a different database
 - Removing a cron job
 - Altering a cron job
 - Installing pg_cron
 - Setting up pg_cron
 - Monitoring jobs
 - Example use cases
 - Managed services
 - Code of Conduct
-

How pg_cron works

The extension creates a background worker that tracks jobs in the `cron.job` table.

```
CREATE TABLE cron.job (  
    jobid bigint primary key default pg_catalog.nextval('cron.jobid_seq'),  
    schedule text not null,  
    command text not null,  
    nodename text not null default 'localhost',  
    nodeport int not null default pg_catalog.inet_server_port(),  
    database text not null default pg_catalog.current_database(),  
    username text not null default current_user  
);
```

Based on your configurations, to execute a job, the extension establishes a PostgreSQL connection or spawns a database worker.

pg_cron can run multiple jobs in parallel, but only one instance of each specific job at a time. If a second instance is triggered before the first finishes, it's queued and starts as soon as the first one completes.

Cron syntax

The code in `pg_cron` that handles parsing and scheduling comes directly from the cron source code by Paul Vixie, hence the same options are supported.

```
+----- min (0 - 59)
| +----- hour (0 - 23)
| | +----- day of month (1 - 31) or last day of the month ($)
| | | +----- month (1 - 12)
| | | | +----- day of week (0 - 6) (0 to 6 are Sunday to
| | | | | Saturday, or use names; 7 is also Sunday)
| | | | |
| | | | |
| | | | |
* * * * *
```

An easy way to create a cron schedule is: crontab.guru.

`pg_cron` also allows you: - to use `$` to indicate last day of the month. - to use `[1-59]` seconds to schedule a job based on an interval. Note, you cannot use seconds with the other time units.

Example cron schedules:

```
'10 seconds' # every 10 seconds
* * * * * # every minute
*/5 * * * * # every 5 minutes
0 * * * * # every hour
0 0 * * * # daily at 12AM
0 0 * * 1-5 # 12AM every weekday
0 1 * * 0 # 1AM every Sunday
0 13 2 6 * # 1PM on the 2nd of June
```

Managing and creating jobs

Cron jobs can be managed by directly interacting with the `cron.job` table if you have the required permissions. However, it is recommended to use the cron functions: - `cron.schedule` - `cron.schedule_in_database` - `cron.unschedule` - `cron.alter_job`

Note, an RLS policy ensures that jobs can only be seen and modified by the user that created them, unless the user is a superuser or has the `bypassrls` attribute.

Creating a cron job

`cron.schedule` signatures

```
-- create job, return jobid
CREATE OR REPLACE FUNCTION cron.schedule(schedule text, command text)
```

```
RETURNS bigint;
```

```
-- create named job, return jobid  
CREATE OR REPLACE FUNCTION cron.schedule(job_name text, schedule text, command text)  
RETURNS bigint
```

Examples

Create a cron job

```
-- Delete old data on Saturday at 3:30am (GMT)  
SELECT cron.schedule(  
    '30 3 * * 6',  
    $$DELETE FROM events WHERE event_time < now() - interval '1 week'$$  
);  
-- returns cron id
```

Create a named cron job

```
-- Vacuum every day at 10:00am (GMT)  
SELECT cron.schedule(  
    'nightly-vacuum',  
    '0 10 * * *',  
    'VACUUM'  
);  
-- returns cron id
```

Create a job that runs every 30 seconds

```
-- run SELECT 1 every 30 seconds  
SELECT cron.schedule(  
    'run_every_30_seconds',  
    '30 seconds',  
    'SELECT 1'  
);  
-- returns cron id
```

Create a job that calls a stored procedure every 5 seconds

```
-- Call a stored procedure every 5 seconds  
SELECT cron.schedule(  
    'process-updates',  
    '5 seconds',  
    'CALL process_updates()'  
);  
-- returns cron id
```

Create a job that processes payroll at 12:00 of the last day of each month

```
-- Process payroll at 12:00 of the last day of each month
SELECT cron.schedule(
    'process-payroll',
    '0 12 $ * *',
    'CALL process_payroll()'
);
-- returns cron id
```

Creating a cron job in a different database

`cron.schedule_in_database` signature

```
-- create job, return jobid
CREATE OR REPLACE FUNCTION cron.schedule_in_database(
    job_name text,
    schedule text,
    command text,
    database text,
    username text DEFAULT NULL::text,
    active boolean DEFAULT true
)
RETURNS bigint
```

Example

Create a cron job in a different database

```
-- Delete old data on Saturday at 3:30am (GMT)
SELECT cron.schedule_in_database(
    'delete_old_data',
    '30 3 * * 6',
    $$DELETE FROM events WHERE event_time < now() - interval '1 week'$$,
    'some_other_database'
);
-- returns cron id
```

Removing a cron job

`cron.unschedule` signatures

```
-- remove job by name, return true if job was removed
CREATE OR REPLACE FUNCTION cron.unschedule(job_name text)
RETURNS boolean

-- remove job by id, return true if job was removed
```

```
CREATE OR REPLACE FUNCTION cron.unschedule(job_id bigint)
RETURNS boolean
```

Examples

Remove a named cron job

```
-- delete job by name
SELECT cron.unschedule('nightly-vacuum');
-- returns true if job was removed
```

Remove a cron job by id

```
-- delete job by id
SELECT cron.unschedule(42);
-- returns true if job was removed
```

Altering a cron job

cron.alter_job signature

```
CREATE OR REPLACE FUNCTION cron.alter_job(
    job_id bigint,
    schedule text DEFAULT NULL::text,
    command text DEFAULT NULL::text,
    database text DEFAULT NULL::text,
    username text DEFAULT NULL::text,
    active boolean DEFAULT NULL::boolean
)
RETURNS void
```

Examples

Change a job's schedule

```
-- change job's schedule
SELECT cron.alter_job(42, '0 10 * * *');
-- returns void
```

Change a job's, schedule, command, and username

```
-- change job's command
SELECT cron.alter_job(
    42,
    '0 10 * * *',
    'VACUUM',
    username := 'some_other_user'
```

```
);  
-- returns void
```

Deactivate a job

```
-- deactivate job  
SELECT cron.alter_job(42, active := false);  
-- returns void
```

Installing pg_cron

Install on Red Hat, CentOS, Fedora, Amazon Linux with PostgreSQL 18 using PGDG:

```
# Install the pg_cron extension  
sudo yum install -y pg_cron_18
```

Install on Debian, Ubuntu with PostgreSQL 18 using apt.postgresql.org:

```
# Install the pg_cron extension  
sudo apt-get -y install postgresql-18-cron
```

You can also install pg_cron by building it from source:

```
git clone https://github.com/citusdata/pg_cron.git  
cd pg_cron  
# Ensure pg_config is in your path, e.g.  
export PATH=/usr/pgsql-18/bin:$PATH  
make && sudo PATH=$PATH make install
```

Windows

Ensure C++ support in Visual Studio is installed and run x64 Native Tools Command Prompt for VS [version] as administrator. Then use nmake to build:

```
set "PGROOT=C:\Program Files\PostgreSQL\18"  
cd %TMP%  
git clone https://github.com/citusdata/pg_cron.git  
cd pg_cron  
nmake /F Makefile.win  
nmake /F Makefile.win install
```

Installation Notes - Windows

Missing Header If compilation fails with Cannot open include file: 'postgres.h': No such file or directory, make sure PGROOT is correct.

Permissions If installation fails with `Access is denied`, re-run the installation instructions as an administrator.

Setting up pg_cron

To start the `pg_cron` background worker, you need to add `pg_cron` to `shared_preload_libraries` in `postgresql.conf`. Note that `pg_cron` does not run any jobs as long as a server is in hot standby mode, but it automatically starts when the server is promoted.

```
# add to postgresql.conf
```

```
# required to load pg_cron background worker on start-up
shared_preload_libraries = 'pg_cron'
```

By default, the `pg_cron` background worker expects its metadata tables to be created in the “postgres” database. However, you can configure this by setting the `cron.database_name` configuration parameter in `postgresql.conf`.

```
# add to postgresql.conf
```

```
# optionally, specify the database in which the pg_cron background worker should run (default)
cron.database_name = 'postgres'
```

`pg_cron` may only be installed to one database in a cluster. If you need to run jobs in multiple databases, use `cron.schedule_in_database()`.

Previously `pg_cron` could only use GMT time, but now you can adapt your time by setting `cron.timezone` in `postgresql.conf`.

```
# add to postgresql.conf
```

```
# optionally, specify the timezone in which the pg_cron background worker should run (default)
cron.timezone = 'PRC'
```

After restarting PostgreSQL, you can create the `pg_cron` functions and metadata tables using `CREATE EXTENSION pg_cron`.

```
-- run as superuser:
CREATE EXTENSION pg_cron;
```

```
-- optionally, grant usage to regular users:
GRANT USAGE ON SCHEMA cron TO marco;
```

Ensuring pg_cron can start jobs

Important: By default, `pg_cron` uses `libpq` to open a new connection to the local database, which needs to be allowed by `pg_hba.conf`. It may be necessary to enable `trust` authentication for connections coming from localhost in for the

user running the cron job, or you can add the password to a .pgpass file, which libpq will use when opening a connection.

You can also use a unix domain socket directory as the hostname and enable **trust** authentication for local connections in pg_hba.conf, which is normally safe:

```
# Connect via a unix domain socket:
cron.host = '/tmp'
```

```
# Can also be an empty string to look for the default directory:
cron.host = ''
```

Alternatively, pg_cron can be configured to use background workers. In that case, the number of concurrent jobs is limited by the **max_worker_processes** setting, so you may need to raise that.

```
# Schedule jobs via background workers instead of localhost connections
cron.use_background_workers = on
# Increase the number of available background workers from the default of 8
max_worker_processes = 20
```

For security, jobs are executed in the database in which the **cron.schedule** function is called with the same permissions as the current user. In addition, users are only able to see their own jobs in the **cron.job** table.

```
-- View active jobs
select * from cron.job;
```

Extension settings

The pg_cron extension supports the following configuration parameters:

Setting	Default	Description
cron.database_name	postgres	Database in which the pg_cron background worker should run.
cron.enable_superuser_jobs	off	Allow jobs to be scheduled as superusers.
cron.host	localhost	Hostname to connect to postgres.
cron.launch_active_jobs	on	When off, disables all active jobs without requiring a server restart
cron.log_min_messages	WARNING	Log_min_messages for the launcher bgworker.
cron.log_run	on	Log all run details in the cron.job_run_details table.
cron.log_statement	on	Log all cron statements prior to execution.
cron.max_running_jobs	30	Maximum number of jobs that can be running at the same time.
cron.timezone	GMT	Timezone in which the pg_cron background worker should run.

Setting	Default	Description
<code>cron.use_background_workers</code>	<code>off</code>	Use background workers instead of client connections.

Changing settings

To view setting configurations, run:

```
SELECT * FROM pg_settings WHERE name LIKE 'cron.%';
```

Setting can be changed in the `postgresql.conf` file or with the below command:

```
ALTER SYSTEM SET cron.<parameter> TO '<value>';
```

`cron.log_min_messages` and `cron.launch_active_jobs` have a setting context of `sighup`. They can be finalized by executing `SELECT pg_reload_conf();`

All the other settings have a `postmaster` context and only take effect after a server restart.

Monitoring jobs

Reviewing the `cron.job_run_details` table

You can view job activity in the `cron.job_run_details` table:

```
select * from cron.job_run_details order by start_time desc limit 5;
```

jobid	runid	job_pid	database	username	command	status	return_message
11	4328	2610	postgres	marco	<code>select pg_sleep(3)</code>	running	NULL
10	4327	2609	postgres	marco	<code>select process()</code>	succeeded	<code>SELECT 1</code>
10	4321	2603	postgres	marco	<code>select process()</code>	succeeded	<code>SELECT 1</code>
10	4320	2602	postgres	marco	<code>select process()</code>	failed	server reset
9	4320	2602	postgres	marco	<code>select do_stuff()</code>	failed	job canceled

(10 rows)

The records in the table are not cleaned automatically, but every user that can schedule cron jobs also has permission to delete their own `cron.job_run_details` records.

Especially when you have jobs that run every few seconds, it can be a good idea to clean up regularly, which can easily be done using `pg_cron` itself:

```
-- Delete old cron.job_run_details records of the current user every day at noon
SELECT cron.schedule('delete-job-run-details', '0 12 * * *', $$DELETE FROM cron.job_run_details
```

If you do not want to use `cron.job_run_details` at all, then you can add `cron.log_run = off` to `postgresql.conf`.

Other cron logging settings

If the `cron.log_statement` setting is configured, jobs will be logged before execution. The `cron.log_min_messages` setting controls the minimum level of messages that will be recorded.

Example use cases

Articles showing possible ways of using `pg_cron`:

- Auto-partitioning using `pg_partman`
- Computing rollups in an analytical dashboard
- Deleting old data, vacuum
- Feeding cats
- Routinely invoking a function
- Postgres as a cron server

Managed services

The following table keeps track of which of the major managed Postgres services support `pg_cron`.

Service	Supported
Aiven	:heavy_check_mark:
Alibaba Cloud	:heavy_check_mark:
Amazon RDS	:heavy_check_mark:
Azure	:heavy_check_mark:
Clickhouse	:heavy_check_mark:
Crunchy Bridge	:heavy_check_mark:
DigitalOcean	:heavy_check_mark:
Google Cloud	:heavy_check_mark:
Heroku	:x:
Instaclustr	:heavy_check_mark:
Lakebase	:x:
Neon	:heavy_check_mark:
PlanetScale	:heavy_check_mark:
ScaleGrid	:heavy_check_mark:
Scaleway	:heavy_check_mark:
Snowflake	:heavy_check_mark:
Supabase	:heavy_check_mark:
Ubicloud	:heavy_check_mark:
YugabyteDB	:heavy_check_mark:

Code of Conduct

This project has adopted the Microsoft Open Source Code of Conduct. For more information see the Code of Conduct FAQ or contact opencode@microsoft.com with any additional questions or comments.