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. It uses the same syntax as regular cron, but it allows you to schedule PostgreSQL commands directly from the database:

```
-- Delete old data on Saturday at 3:30am (GMT)
SELECT cron.schedule('30 3 * * 6', $$DELETE FROM events WHERE event_time < now() - interval
 schedule
----------
       42

-- Vacuum every day at 10:00am (GMT)
SELECT cron.schedule('nightly-vacuum', '0 10 * * *', 'VACUUM');
 schedule
----------
       43

-- Change to vacuum at 3:00am (GMT)
SELECT cron.schedule('nightly-vacuum', '0 3 * * *', 'VACUUM');
 schedule
----------
       43

-- Stop scheduling jobs
SELECT cron.unschedule('nightly-vacuum' );
 unschedule
------------
 t
(1 row)

SELECT cron.unschedule(42);
 unschedule
------------
          t
```

pg_cron can run multiple jobs in parallel, but it runs at most one instance of a job at a time. If a second run is supposed to start before the first one finishes, then the second run is queued and started as soon as the first run completes.

The schedule uses the standard cron syntax, in which * means "run every time period", and a specific number means "but only at this time":

```
 +------------- min (0 - 59)
 | +-------------- hour (0 - 23)
 | | +--------------- day of month (1 - 31)
 | | | +---------------- 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.

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. Be aware that pg_cron always uses GMT!

Installing pg_cron

Install on Red Hat, CentOS, Fedora, Amazon Linux with PostgreSQL 12 using PGDG:

```
# Install the pg_cron extension
sudo yum install -y pg_cron_12
```

Install on Debian, Ubuntu with PostgreSQL 12 using apt.postgresql.org:

```
# Install the pg_cron extension
sudo apt-get -y install postgresql-12-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-12/bin:$PATH
make && sudo PATH=$PATH make install
```

Setting up pg_cron

To start the pg_cron background worker when PostgreSQL starts, you need to add pg_cron to `shared_preload_libraries` in postgresql.conf. Note that pg_cron does not run any jobs as a long 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 (defau
cron.database_name = 'postgres'
```

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

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

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

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.

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
```

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'
```

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.

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:
Crunchy Bridge	:heavy_check_mark:
DigitalOcean	:heavy_check_mark:
Google Cloud	:heavy_check_mark:
Heroku	:x:
ScaleGrid	:heavy_check_mark:
Scaleway	:heavy_check_mark:
Supabase	:heavy_check_mark: