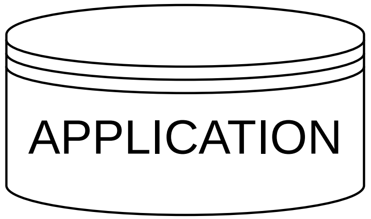

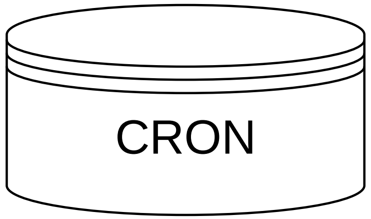

- Extensions:
    - ~~pg_cron~~ not overhere
    - pg_cron_helper
    - db_link
    - postgres_fdw
        - “cron_ctrl_server”
            - user_mapping(s)
- Schema: “’cron”

- Extensions:
    - pg_cron
    - pg_cron_helper
    - db_link
    - postgres_fdw
        - “cron_ctrl_server”
            - user_mapping(s)
- Schema: “’cron”

APPLICATION
CRON

Regulating jobs will be dealt with from the application-database
Both the maintenance actions as well as execution(s) of a job.

In the cron-database the actions triggered on the application-db will be reflected.
Actions like creating, running or disabling jobs. These actions are made available
via the extension pg_cron_helper. The job-definitions are stored in regular
tables and necessary actions from extension pg_cron are taken to trigger a job.

The cron-db *can* serve multiple applications. Applications in the sense of
a database and its logic being an application. Though the cron-db can serve
multiple applications it assures only one instance of a job will run at the same time.

The communication between the APP-db and the CRON-db is established via
a database-links to and from the databases. Hence the need for the
extension: db_link.

APPLICATION
CRON

From the extension ‘helper’ a foreign-server is created: “cron_ctrl_server”. This is used to perform the actions on the cron-db. Note also one needs to provide user-mapping(s). For every db-account on the application-db, that needs to do anything with jobs, a user-mapping must be created. This user-mapping contains the *user* and optionally the *password* of an account in the cron-db with sufficient privileges.

As the routines from the ‘helper’ are created with “security definer” the owner of the routine is the default*) user. The user-mapping used, adheres to that owner.

*) Some of the routines have a parameter that holds a user-name to look for that specific account in the cron-db.
For simplicity reasons this is for now out-of-scope.

APPLICATION

CRON

Schema: “’cron”
No tables to handle jobs

Schema: “’cron”
From extension pg_cron:
- job
- job_run_details

From extension pg_cron_helper:
- job_definition
- job_run

pg_cron_helper

Routines available:

Functions (returning output)

- get_job_state
- list_jobs

Procedures (actions/commands)

- _cron_job_execution
  For internal use only. **NOT** to used directly
- create_job
- drop_job
- enable_job
- disable_job
- run_job
- stop_job

pg_cron and its “_helper”

Lifespan of a job via pg_cron_helper:

1) Create a job (maintenance)
2) Enable job (maintenance)
3) Run a job (execution)

   In normal situations a job finishes.
   Depending on the parameters at time of creation it:
   * schedules itself again
   * drops itself

4) Stop a job (execution)
   step on the break if unforeseen/unwanted situation occurs
5) Disable a job (maintenance)
   Let the jobs’ definition reside but it is not scheduled to run
6) Drop job (maintenance)

Let’s have a more detail look at them from application-DB point of view ...

pg_cron_helper (create)

Create a job has the following parameters:

1) job_name
2) job_action (should hold the db-code to be executed on application-database)
3) start_date (default current date+time)
4) repeat_interval (default an empty string => not repeating so run once)
5) end_date (default NULL => if applicable, it will repeat till further notice)
6) enabled (default *false* => disabled by default so it does start when created)
7) auto_drop (default *true => not implemented yet*)
8) comments (default empty string)
   Holds e.g. description of what the job does or reference to the documentation or ...
9) user_name (default current-user: owner of the routine)
   As mentioned we will skip this for simplicity/complexity reason.

The combination of job-name and user-name is considered to be unique.
So you could think of it as: “a job for this DB-application is unique” …
! as we leave out the user-name !

The registration in the cron-db (job and job_definition) will contain the name of the application-db.
This assures the uniqueness of job-names across the postgres instance in case multiple application-db’s are served.

pg_cron_helper (enable)

Enable a job has the following parameters:
1) job_name
2) user_name (default current-user: owner of the routine) skipped

Makes the job eligible to run. It will take the action(s) to schedule it according its definition provided in the call of the “create-job”. So it could well be, it will run only once.
The parameters given by the create-routine will be used to create a scheduling-pattern which is than used to schedule it via the corresponding routine from the cron-extension.
A regular cron-scheduling pattern consist out of 5 elements:

```
 ┌───────────── minute (0–59)
 │ ┌───────────── hour (0–23)
 │ │ ┌───────────── day of the month (1–31)
 │ │ │ ┌───────────── month (1–12)
 │ │ │ │ ┌───────────── day of the week (0–6) (Sunday to Saturday;
 │ │ │ │ │
 │ │ │ │ │
 * * * * *
```

However it is also possible to provide a schedule that is not possible to put in these 5 elements.
For example one can not construct a schedule for every 7 minutes. But can provide like oracle way: `'FREQ=minutely;INTERVAL 7'`
The table “job_definition” does accommodate such features

pg_cron_helper (run)

Run a job has the following parameters:
1) job_name
2) user_name (default current-user: owner of the routine) skipped

If the conditions for the job (identified by the name) are met, it will be scheduled to run immediately*), if defined without repeating, or according the provided schedule (considering the start-date and repeat-interval).
Technically it takes the “job_definition” to schedule it with the cron-extension, which in turn runs it and creates entries in tables “job” and “job_run_details”.
This run-routine also takes care of the administration of the table: “job_run”. It interlinks the “job_definition” to the latest “job_run_details”

*) *Immediatly* means in this case that it is scheduled in 5 seconds time. This is needed to let the underlying cron-software have the opportunity to pick-up the job.

pg_cron_helper (stop)

Stops a job has the following parameters:
1) job_name
2) force (default *false*)
3) user_name (default current-user: owner of the routine) skipped

If the job (identified by the name) is running, it will be signaled to stop when “force” is false. Leading to a graceful stop.
If “force” is true the underlying background process is terminated. The database is taking care of the rollback and reverting to the begin of the session. Note that if within the job itself other connections have initiated (and closed) the action that were performed **won’t** be reverted.

pg_cron_helper (disable)

Disables a job has the following parameters:
1) job_name
2) user_name (default current-user: owner of the routine) skipped

A job (identified by the name) can be disabled by calling this routine. It will use the cron’s routine to remove the job from the cron-jobs and clearing the cron-pattern and job-id in table: “job_definition”

If a job is re-enabled it will calculate the next time to run and schedule it by the ”schedule”-routine of extension cron.

pg_cron_helper (drop)

Drop a job has the following parameters:
1) job_name
2) force default false
3) user_name (default current-user: owner of the routine) skipped

A job (identified by the name) will be removed from the scheduling as well as removed from the job_definition-table. The history of the runs of the jobs (table job_run) will also be purged.

pg_cron_helper

When communicating with pg_cron_helper the account used for this administration to/in the cron-db should have sufficient privileges to execute the job on the application-db side.
See https://github.com/citusdata/pg_cron#ensuring-pg_cron-can-start-jobs how this can be achieved.

When **restoring** a database the privileges of accounts **and** records in the tables “job” and “job_definition” in cron-db should be the same. Note that the job-name also contains the application-db.

A way to restore a database in another cluster:

- Create the user-account with same name as the owner of the objects from the source database in the new cluster
- Either make this user trusted or put the password in the “.pgpass”-file (see link above)
- Create a dump with owner information
- Load the file in the new database
- Execute the sql-scipt that does the “create-job” for all the jobs.
  This information can also be retrieved from the table: ”job_definition” from the source-database filtered on the column: “user_name” being user/owner and column: “database_name” being the `current_database()`

Privileges setup

Notes on privileges:

- The user in the DB-link from application-DB must have sufficient privileges to execute the routines at the cron-DB
- Likewise the other way around

So as the routines are defined with “security definer” the user in the db-link should have these privileges to execute the code.
The easiest way to achieve it in a single application-DB setup is to make the db-link-users the same as the owner of these routines.