
pg*timetable*Documentation

Release stable

Nov 23, 2022

CONTENTS:

1	Introduction	1
1.1	Main features	1
1.2	Quick Start	1
1.3	Command line options	2
1.4	Contributing	3
1.5	Support	3
1.6	Authors	3
2	Project background	5
2.1	Project feedback	5
3	Installation	7
3.1	Official release packages	7
3.2	Docker	7
3.3	Build from sources	8
4	Components	9
4.1	Command	9
4.2	Task	10
4.3	Chain	12
5	Getting started	15
5.1	Add simple job	15
5.2	Examples	16
6	Samples	17
6.1	Basic	17
6.2	Send email	17
6.3	Download, Transform and Import	19
6.4	Run tasks in autonomous transaction	21
6.5	Shutdown the scheduler and terminate the session	22
6.6	Access previous task result code and output from the next task	22
7	Migration from others schedulers	25
7.1	Migrate jobs from pg_cron to pg_timetable	25
7.2	Migrate jobs from pgAgent to pg_timetable	26
8	REST API	29
8.1	Health check endpoints	29
8.2	Chain management endpoints	29

9 Database Schema	31
9.1 Main tables and objects	31
9.2 Jobs related functions	35
9.3 ron related functions	38
9.4 ER-Diagram	42
10 Indices and tables	43

INTRODUCTION

pg_timetable is an advanced job scheduler for PostgreSQL, offering many advantages over traditional schedulers such as **cron** and others. It is completely database driven and provides a couple of advanced concepts.

1.1 Main features

- Tasks can be arranged in chains
- A chain can consist of built-in commands, SQL and executables
- Parameters can be passed to chains
- Missed tasks (possibly due to downtime) can be retried automatically
- Support for configurable repetitions
- Built-in tasks such as sending emails, etc.
- Fully database driven configuration
- Full support for database driven logging
- Cron-style scheduling at the PostgreSQL server time zone
- Optional concurrency protection
- Task and chain can have execution timeout settings

1.2 Quick Start

1. Download `pg_timetable` executable
2. Make sure your PostgreSQL server is up and running and has a role with `CREATE` privilege for a target database, e.g.

```
my_database=> CREATE ROLE scheduler PASSWORD 'somesstrong';  
my_database=> GRANT CREATE ON DATABASE my_database TO scheduler;
```

3. Create a new job, e.g. run `VACUUM` each night at 00:30 Postgres server time zone

```
my_database=> SELECT timetable.add_job('frequent-vacuum', '30 * * * *',  
↪ 'VACUUM');  
add_job  
-----
```

(continues on next page)

(continued from previous page)

```

3
(1 row)

```

4. Run the `pg_timetable`

```

# pg_timetable postgresql://scheduler:somestrong@localhost/my_database --
↪clientname=vacuummer

```

5. PROFIT!

1.3 Command line options

```

# ./pg_timetable

Application Options:
  -c, --clientname=                Unique name for application instance [
↪$PGTT_CLIENTNAME]
  --config=                        YAML configuration file
  --no-program-tasks              Disable executing of PROGRAM tasks [
↪NOPROGRAMTASKS]

Connection:
  -h, --host=                      PostgreSQL host (default: localhost) [
↪$PGTT_PGHOST]
  -p, --port=                      PostgreSQL port (default: 5432) [
↪PGPORT]
  -d, --dbname=                   PostgreSQL database name (default:
↪timetable) [$PGTT_PGDATABASE]
  -u, --user=                      PostgreSQL user (default: scheduler) [
↪$PGTT_PGUSER]
  --password=                     PostgreSQL user password [
↪$PGTT_PGPASSWORD]
  --sslmode=[disable|require]     What SSL priority use for connection
↪(default: disable)
  --pgurl=                        PostgreSQL connection URL [
↪$PGTT_URL]
  --timeout=                      PostgreSQL connection timeout in seconds
↪(default: 90) [$PGTT_TIMEOUT]

Logging:
  --log-level=[debug|info|error]   Verbosity level for stdout and log file
↪(default: info)
  --log-database-level=[debug|info|error] Verbosity level for database storing
↪(default: info)
  --log-file=                      File name to store logs
  --log-file-format=[json|text]    Format of file logs (default: json)

Start:
  -f, --file=                     SQL script file to execute during startup
  --init                          Initialize database schema to the latest
↪version and exit. Can be used
                                  with --upgrade

```

(continues on next page)

(continued from previous page)

<code>--upgrade</code>	Upgrade database to the latest version
<code>--debug</code>	Run in debug mode. Only asynchronous.
<code>↳chains will be executed</code>	
Resource:	
<code>--cron-workers=</code>	Number of parallel workers for scheduled.
<code>↳chains (default: 16)</code>	
<code>--interval-workers=</code>	Number of parallel workers for interval.
<code>↳chains (default: 16)</code>	
<code>--chain-timeout=</code>	Abort any chain that takes more than the
<code>↳specified number of milliseconds</code>	
<code>--task-timeout=</code>	Abort any task within a chain that takes
<code>↳more than the specified number</code>	
	of milliseconds
REST:	
<code>--rest-port:</code>	REST API port (default: 0) [%PGTT_RESTPORT
<code>↳%]</code>	

1.4 Contributing

If you want to contribute to **pg_timetable** and help make it better, feel free to open an [issue](#) or even consider submitting a [pull request](#). You also can give a [star](#) to **pg_timetable** project, and to tell the world about it.

1.5 Support

For professional support, please contact [Cybertec](#).

1.6 Authors

Implementation: [Pavlo Golub](#)

Initial idea and draft design: [Hans-Jürgen Schönig](#)

PROJECT BACKGROUND

The `pg_timetable` project got started back in 2019 for internal scheduling needs at Cybertec.

For more background on the project motivations and design goals see the original series of blogposts announcing the project and the following feature updates.

Cybertec also provides commercial 9-to-5 and 24/7 support for `pg_timetable`.

- [Project announcement](#)
- [v2 released](#)
- [Start-up improvements](#)
- [v3 released](#)
- [Exclusive jobs explained](#)
- [Asynchronous chain execution](#)
- [v4 released](#)
- [PostgreSQL schedulers: comparison table](#)

2.1 Project feedback

For feature requests or troubleshooting assistance please open an issue on project's [Github page](#).

INSTALLATION

pg_timetable is compatible with the latest supported PostgreSQL versions: 11, 12, 13, 14 (stable); 15 (dev).

Note:

```
CREATE OR REPLACE FUNCTION starts_with(text, text)
RETURNS bool AS
$$
SELECT
    CASE WHEN length($2) > length($1) THEN
        FALSE
    ELSE
        left($1, length($2)) = $2
    END
$$
LANGUAGE SQL
IMMUTABLE STRICT PARALLEL SAFE
COST 5;
```

3.1 Official release packages

You may find binary package for your platform on the official [Releases](#) page. Right now **Windows**, **Linux** and **macOS** packages are available.

3.2 Docker

The official docker image can be found here: https://hub.docker.com/r/cybertecpostgresql/pg_timetable

Note: The latest tag is up to date with the *master* branch thanks to [this github action](#). In production you probably want to use the latest [stable tag](#).

Run **pg_timetable** in Docker:

```
docker run --rm \
cybertecpostgresql/pg_timetable:latest \
-h 10.0.0.3 -p 54321 -c worker001
```

Run **pg_timetable** in Docker with Environment variables:

```
docker run --rm \
-e PGTT_PGHOST=10.0.0.3 \
-e PGTT_PGPORT=54321 \
cybertecpostgresql/pg_timetable:latest \
-c worker001
```

3.3 Build from sources

1. Download and install **Go** on your system.
2. Clone **pg_timetable** repo:

```
$ git clone https://github.com/cybertec-postgresql/pg_timetable.git
$ cd pg_timetable
```

3. Run **pg_timetable**:

```
$ go run main.go --dbname=dbname --clientname=worker001 --user=scheduler --
->password=strongpwd
```

4. Alternatively, build a binary and run it:

```
$ go build
$ ./pg_timetable --dbname=dbname --clientname=worker001 --user=scheduler --
->password=strongpwd
```

5. (Optional) Run tests in all sub-folders of the project:

```
$ psql --command="CREATE USER scheduler PASSWORD 'somestrong'"
$ createdb --owner=scheduler timetable
$ go test -failfast -timeout=300s -count=1 -p 1 ./...
```

COMPONENTS

The scheduling in **pg_timetable** encompasses three different abstraction levels to facilitate the reuse with other parameters or additional schedules.

Command

The base level, **command**, defines *what* to do.

Task

The second level, **task**, represents a chain element (step) to run one of the commands. With **tasks** we define order of commands, arguments passed (if any), and how errors are handled.

Chain

The third level represents a connected tasks forming a chain of tasks. **Chain** defines *if*, *when*, and *how often* a job should be executed.

4.1 Command

Currently, there are three different kinds of commands:

SQL

SQL snippet. Starting a cleanup, refreshing a materialized view or processing data.

PROGRAM

External Command. Anything that can be called as an external binary, including shells, e.g. `bash`, `pwsh`, etc. The external command will be called using `golang's exec.CommandContext`.

BUILTIN

Internal Command. A prebuilt functionality included in **pg_timetable**. These include:

- *NoOp*,
- *Sleep*,
- *Log*,
- *SendMail*,
- *Download*,
- *CopyFromFile*,
- *CopyToFile*,
- *Shutdown*.

4.2 Task

The next building block is a **task**, which simply represents a step in a list of chain commands. An example of tasks combined in a chain would be:

1. Download files from a server
2. Import files
3. Run aggregations
4. Build report
5. Remove the files from disk

Note: All tasks of the chain in **pg_timetable** are executed within one transaction. However, please, pay attention there is no opportunity to rollback PROGRAM and BUILTIN tasks.

4.2.1 Table timetable.task

chain_id bigint

Link to the chain, if NULL task considered to be disabled

task_order DOUBLE PRECISION

Indicates the order of task within a chain.

kind timetable.command_kind

The type of the command. Can be *SQL* (default), *PROGRAM* or *BUILTIN*.

command text

Contains either a SQL command, a path to application or name of the *BUILTIN* command which will be executed.

run_as text

The role as which the task should be executed as.

database_connection text

The connection string for the external database that should be used.

ignore_error boolean

Specify if the next task should proceed after encountering an error (default: *false*).

autonomous boolean

Specify if the task should be executed out of the chain transaction. Useful for *VACUUM*, *CREATE DATABASE*, *CALL* etc.

timeout integer

Abort any task within a chain that takes more than the specified number of milliseconds.

Warning: If the **task** has been configured with **ignore_error** set to **true** (the default value is **false**), the worker process will report a success on execution *even if the task within the chain fails*.

As mentioned above, **commands** are simple skeletons (e.g. *send email*, *vacuum*, etc.). In most cases, they have to be brought to live by passing input parameters to the execution.

4.2.2 Table timetable.parameter

task_id bigint

The ID of the task.

order_id integer

The order of the parameter. Several parameters are processed one by one according to the order.

value jsonb

A JSON value containing the parameters.

4.2.3 Parameter value format

Depending on the **command** kind argument can be represented by different *JSON* values.

Kind

Schema

Example

SQL

array

```
'[ "one", 2, 3.14, false ] '::jsonb
```

PROGRAM

array of strings

```
'["-x", "Latin-ASCII", "-o", "orte_ansi.txt", "orte.txt"] '::jsonb
```

BUILTIN: Sleep

integer

```
'5' :: jsonb
```

BUILTIN: Log

any

```
'"WARNING" '::jsonb
'{"Status": "WARNING"} '::jsonb
```

BUILTIN: SendMail

object

```
'{
  "username":    "user@example.com",
  "password":    "password",
  "serverhost":  "smtp.example.com",
  "serverport":  587,
  "senderaddr":  "user@example.com",
  "ccaddr":      ["recipient_cc@example.com"],
  "bccaddr":     ["recipient_bcc@example.com"],
  "toaddr":      ["recipient@example.com"],
```

(continues on next page)

(continued from previous page)

```
"subject":      "pg_timetable - No Reply",
"attachment":  ["/temp/attachments/Report.pdf","config.yaml"],
"attachmentdata": [{"name": "File.txt", "base64data": "RmlsZSBDb250ZW50"}],
"msgbody":     "<h2>Hello User,</h2> <p>check some attachments!</p>",
"contenttype": "text/html; charset=UTF-8"
}'::jsonb
```

BUILTIN: Download

object

```
'{
  "workersnum": 2,
  "fileurls": ["http://example.com/foo.gz", "https://example.com/bar.csv"],
  "destpath": "."
}'::jsonb
```

BUILTIN: CopyFromFile

object

```
'{
  "sql": "COPY location FROM STDIN",
  "filename": "download/orte_ansi.txt"
}'::jsonb
```

BUILTIN: CopyToFile

object

```
'{
  "sql": "COPY location TO STDOUT",
  "filename": "download/location.txt"
}'::jsonb
```

BUILTIN: Shutdown*value ignored***BUILTIN: NoOp***value ignored*

4.3 Chain

Once tasks have been arranged, they have to be scheduled as a **chain**. For this, **pg_timetable** builds upon the enhanced **cron**-string, all the while adding multiple configuration options.

4.3.1 Table `timetable.chain`

chain_name text

The unique name of the chain.

run_at timetable.cron

Standard *cron*-style value at Postgres server time zone or `@after`, `@every`, `@reboot` clause.

max_instances integer

The amount of instances that this chain may have running at the same time.

timeout integer

Abort any chain that takes more than the specified number of milliseconds.

live boolean

Control if the chain may be executed once it reaches its schedule.

self_destruct boolean

Self destruct the chain after successful execution. Failed chains will be executed according to the schedule one more time.

exclusive_execution boolean

Specifies whether the chain should be executed exclusively while all other chains are paused.

client_name text

Specifies which client should execute the chain. Set this to *NULL* to allow any client.

Note: All chains in `pg_timetable` are scheduled at the PostgreSQL server time zone. You can change the `timezone` for the **current session** when adding new chains, e.g.

```
SET TIME ZONE 'UTC';  
  
-- Run VACUUM at 00:05 every day in August UTC  
SELECT timetable.add_job('execute-func', '5 0 * 8 *', 'VACUUM');
```


GETTING STARTED

A variety of examples can be found in the *Samples*. If you want to migrate from a different scheduler, you can use scripts from *Migration from others schedulers* chapter.

5.1 Add simple job

In a real world usually it's enough to use simple jobs. Under this term we understand:

- job is a chain with only one **task** (step) in it;
- it doesn't use complicated logic, but rather simple **command**;
- it doesn't require complex transaction handling, since one task is implicitly executed as a single transaction.

For such a group of chains we've introduced a special function `timetable.add_job()`.

timetable.add_job(job_name, job_schedule, job_command, ...) RETURNS BIGINT

Creates a simple one-task chain

Parameters

- **job_name** (*text*) – The unique name of the **chain** and **command**.
- **job_schedule** (*timetable.cron*) – Time schedule in cron syntax at Postgres server time zone
- **job_command** (*text*) – The SQL which will be executed.
- **job_parameters** (*jsonb*) – Arguments for the chain **command**. Default: NULL.
- **job_kind** (*timetable.command_kind*) – Kind of the command: *SQL*, *PROGRAM* or *BUILTIN*. Default: *SQL*.
- **job_client_name** (*text*) – Specifies which client should execute the chain. Set this to *NULL* to allow any client. Default: NULL.
- **job_max_instances** (*integer*) – The amount of instances that this chain may have running at the same time. Default: NULL.
- **job_live** (*boolean*) – Control if the chain may be executed once it reaches its schedule. Default: TRUE.
- **job_self_destruct** (*boolean*) – Self destruct the chain after execution. Default: FALSE.
- **job_ignore_errors** (*boolean*) – Ignore error during execution. Default: TRUE.
- **job_exclusive** (*boolean*) – Execute the chain in the exclusive mode. Default: FALSE.

Returns

the ID of the created chain

Return type

integer

5.2 Examples

1. Run `public.my_func()` at 00:05 every day in August Postgres server time zone:

```
SELECT timetable.add_job('execute-func', '5 0 * 8 *', 'SELECT public.my_
↳func()');
```

2. Run `VACUUM` at minute 23 past every 2nd hour from 0 through 20 every day Postgres server time zone:

```
SELECT timetable.add_job('run-vacuum', '23 0-20/2 * * *', 'VACUUM');
```

3. Refresh materialized view every 2 hours:

```
SELECT timetable.add_job('refresh-matview', '@every 2 hours', 'REFRESH_
↳MATERIALIZED VIEW public.mat_view');
```

4. Clear log table after `pg_timetable` restart:

```
SELECT timetable.add_job('clear-log', '@reboot', 'TRUNCATE timetable.log');
```

5. Reindex at midnight Postgres server time zone on Sundays with `reindexdb` utility:

- using default database under default user (no command line arguments)

```
SELECT timetable.add_job('reindex', '0 0 * * 7', 'reindexdb', job_kind_
↳:= 'PROGRAM');
```

- specifying target database and tables, and be verbose

```
SELECT timetable.add_job('reindex', '0 0 * * 7', 'reindexdb',
↳["--table=foo", "--dbname=postgres", "--verbose"]::jsonb, 'PROGRAM
↳');
```

- passing password using environment variable through bash shell

```
SELECT timetable.add_job('reindex', '0 0 * * 7', 'bash',
↳["-c", "PGPASSWORD=5m3R7K4754p4m reindexdb -U postgres -h 192.168.0.
↳221 -v"]::jsonb,
↳'PROGRAM');
```

SAMPLES

6.1 Basic

This sample demonstrates how to create a basic one-step chain with parameters. It uses CTE to directly update the **timetable** schema tables.

```
1 SELECT timetable.add_job(  
2     job_name           => 'notify every minute',  
3     job_schedule       => '* * * * *',  
4     job_command        => 'SELECT pg_notify($1, $2)',  
5     job_parameters     => '[ "TT_CHANNEL", "Ahoj from SQL base task" ]' :: jsonb,  
6     job_kind           => 'SQL'::timetable.command_kind,  
7     job_client_name    => NULL,  
8     job_max_instances  => 1,  
9     job_live           => TRUE,  
10    job_self_destruct  => FALSE,  
11    job_ignore_errors  => TRUE  
12 ) as chain_id;
```

6.2 Send email

This sample demonstrates how to create an advanced email job. It will check if there are emails to send, will send them and log the status of the command execution. You don't need to setup anything, every parameter can be specified during the chain creation.

```
1 DO $$  
2     -- An example for using the SendMail task.  
3 DECLARE  
4     v_mail_task_id bigint;  
5     v_log_task_id  bigint;  
6     v_chain_id     bigint;  
7 BEGIN  
8     -- Get the chain id  
9     INSERT INTO timetable.chain (chain_name, max_instances, live) VALUES ('Send Mail', 1,  
10    ↪ TRUE)  
11     RETURNING chain_id INTO v_chain_id;  
12     -- Add SendMail task  
13     INSERT INTO timetable.task (chain_id, task_order, kind, command)
```

(continues on next page)

(continued from previous page)

```

14 SELECT v_chain_id, 10, 'BUILTIN', 'SendMail'
15 RETURNING task_id INTO v_mail_task_id;
16
17 -- Create the parameters for the SensMail task
18 -- "username":          The username used for authenticating on the mail_
↳server
19 -- "password":         The password used for authenticating on the mail server
20 -- "serverhost":       The IP address or hostname of the mail server
21 -- "serverport":       The port of the mail server
22 -- "senderaddr":       The email that will appear as the sender
23 -- "ccaddr":           String array of the recipients(Cc) email addresses
24 -- "bccaddr":          String array of the recipients(Bcc) email addresses
25 -- "toaddr":           String array of the recipients(To) email addresses
26 -- "subject":          Subject of the email
27 -- "attachment":       String array of the attachments (local file)
28 -- "attachmentdata":   Pairs of name and base64-encoded content
29 -- "msgbody":          The body of the email
30
31 INSERT INTO timetable.parameter (task_id, order_id, value)
32 VALUES (v_mail_task_id, 1, '{
33     "username": "user@example.com",
34     "password": "password",
35     "serverhost": "smtp.example.com",
36     "serverport": 587,
37     "senderaddr": "user@example.com",
38     "ccaddr": ["recipient_cc@example.com"],
39     "bccaddr": ["recipient_bcc@example.com"],
40     "toaddr": ["recipient@example.com"],
41     "subject": "pg_timetable - No Reply",
42     "attachment": ["D:\\Go stuff\\Books\\Concurrency in Go.pdf","report.
↳yaml"],
43     "attachmentdata": [{"name": "File.txt", "base64data": "RmlsZSBDb250ZW50"}
↳],
44     "msgbody": "<b>Hello User,</b> <p>I got some Go books for you enjoy
↳</p> <i>pg_timetable</i>!",
45     "contenttype": "text/html; charset=UTF-8"
46     }'::jsonb);
47
48 -- Add Log task and make it the last task using `task_order` column (=30)
49 INSERT INTO timetable.task (chain_id, task_order, kind, command)
50 SELECT v_chain_id, 30, 'BUILTIN', 'Log'
51 RETURNING task_id INTO v_log_task_id;
52
53 -- Add housekeeping task, that will delete sent mail and update parameter for the_
↳previous logging task
54 -- Since we're using special add_task() function we don't need to specify the `chain_
↳id`.
55 -- Function will take the same `chain_id` from the parent task, SendMail in this_
↳particular case
56 PERFORM timetable.add_task(
57     kind => 'SQL',
58     parent_id => v_mail_task_id,

```

(continues on next page)

(continued from previous page)

```

59     command => format(
60 $query$WITH sent_mail(toaddr) AS (DELETE FROM timetable.parameter WHERE task_id = %s
↳RETURNING value->>'username')
61 INSERT INTO timetable.parameter (task_id, order_id, value)
62 SELECT %s, 1, to_jsonb('Sent emails to: ' || string_agg(sent_mail.toaddr, ';'))
63 FROM sent_mail
64 ON CONFLICT (task_id, order_id) DO UPDATE SET value = EXCLUDED.value$query$,
65         v_mail_task_id, v_log_task_id
66     ),
67     order_delta => 10
68 );
69
70 -- In the end we should have something like this. Note, that even Log task was created
↳earlier it will be executed later
71 -- due to `task_order` column.
72
73 -- timetable=> SELECT task_id, chain_id, kind, left(command, 50) FROM timetable.task
↳ORDER BY task_order;
74 -- task_id | chain_id | task_order | kind | left
75 -- -----+-----+-----+-----+-----
↳-----
76 --      45 |      24 |      10 | BUILTIN | SendMail
77 --      47 |      24 |      20 | SQL     | WITH sent_mail(toaddr) AS (DELETE FROM
↳timetable.p
78 --      46 |      24 |      30 | BUILTIN | Log
79 -- (3 rows)
80
81 END;
82 $$
83 LANGUAGE PLPGSQL;

```

6.3 Download, Transform and Import

This sample demonstrates how to create enhanced three-step chain with parameters. It uses DO statement to directly update the **timetable** schema tables.

```

1 -- An enhanced example consisting of three tasks:
2 -- 1. Download text file from internet using BUILT-IN command
3 -- 2. Remove accents (diacritic signs) from letters using PROGRAM command (can be done
↳with `unaccent` PostgreSQL extension)
4 -- 3. Import text file as CSV file using BUILT-IN command (can be down with `psql -c /
↳copy`)
5 DO $$
6 DECLARE
7     v_head_id bigint;
8     v_task_id bigint;
9     v_chain_id bigint;
10 BEGIN
11     -- Create the chain with default values executed every minute (NULL == '* * * * *'
↳:: timetable.cron)

```

(continues on next page)

(continued from previous page)

```

12 INSERT INTO timetable.chain (chain_name, live)
13 VALUES ('Download locations and aggregate', TRUE)
14 RETURNING chain_id INTO v_chain_id;
15
16 -- Step 1. Download file from the server
17 -- Create the chain
18 INSERT INTO timetable.task (chain_id, task_order, kind, command, ignore_error)
19 VALUES (v_chain_id, 1, 'BUILTIN', 'Download', TRUE)
20 RETURNING task_id INTO v_task_id;
21
22 -- Create the parameters for the step 1:
23 INSERT INTO timetable.parameter (task_id, order_id, value)
24     VALUES (v_task_id, 1,
25         '{
26             "workersnum": 1,
27             "fileurls": ["https://www.cybertec-postgresql.com/secret/orte.txt"],
28             "destpath": "."
29         }'::jsonb);
30
31 RAISE NOTICE 'Step 1 completed. Chain added with ID: %; DownloadFile task added with
↳ ID: %', v_chain_id, v_task_id;
32
33 -- Step 2. Transform Unicode characters into ASCII
34 -- Create the program task to call 'uconv' and name it 'unaccent'
35 INSERT INTO timetable.task (chain_id, task_order, kind, command, ignore_error, task_
↳ name)
36 VALUES (v_chain_id, 2, 'PROGRAM', 'uconv', TRUE, 'unaccent')
37 RETURNING task_id INTO v_task_id;
38
39 -- Create the parameters for the 'unaccent' task. Input and output files in this case
40 -- Under Windows we should call PowerShell instead of "uconv" with command:
41 -- Set-content "orte_ansi.txt" ((Get-content "orte.txt").Normalize("FormD") -replace
↳ '\p{M}', '')
42 INSERT INTO timetable.parameter (task_id, order_id, value)
43     VALUES (v_task_id, 1, ["-x", "Latin-ASCII", "-o", "orte_ansi.txt", "orte.txt"]
↳ '::jsonb);
44
45 RAISE NOTICE 'Step 2 completed. Unacent task added with ID: %', v_task_id;
46
47 -- Step 3. Import ASCII file to PostgreSQL table using "CopyFromFile" built-in
↳ command
48 INSERT INTO timetable.task (chain_id, task_order, kind, command)
49     VALUES (v_chain_id, 3, 'BUILTIN', 'CopyFromFile')
50 RETURNING task_id INTO v_task_id;
51
52 -- Prepare the destination table 'location'
53 CREATE TABLE IF NOT EXISTS location(name text);
54
55 -- Add the parameters for the download task. Execute client side COPY to 'location'
↳ from 'orte_ansi.txt'
56 INSERT INTO timetable.parameter (task_id, order_id, value)
57     VALUES (v_task_id, 1, '{"sql": "COPY location FROM STDIN", "filename": "orte_

```

(continues on next page)

(continued from previous page)

```

58 ↪ansi.txt" }'::jsonb);
59     RAISE NOTICE 'Step 3 completed. Import task added with ID: %', v_task_id;
60 END;
61 $$ LANGUAGE PLPGSQL;

```

6.4 Run tasks in autonomous transaction

This sample demonstrates how to run special tasks out of chain transaction context. This is useful for special routines and/or non-transactional operations, e.g. *CREATE DATABASE*, *REINDEX*, *VACUUM*, *CREATE TABLESPACE*, etc.

```

1  -- An advanced example showing how to use atonomous tasks.
2  -- This one-task chain will execute test_proc() procedure.
3  -- Since procedure will make two commits (after f1() and f2())
4  -- we cannot use it as a regular task, because all regular tasks
5  -- must be executed in the context of a single chain transaction.
6  -- Same rule applies for some other SQL commands,
7  -- e.g. CREATE DATABASE, REINDEX, VACUUM, CREATE TABLESPACE, etc.
8  CREATE OR REPLACE FUNCTION f (msg TEXT) RETURNS void AS $$
9  BEGIN
10     RAISE notice '%', msg;
11 END;
12 $$ LANGUAGE PLPGSQL;
13
14 CREATE OR REPLACE PROCEDURE test_proc () AS $$
15 BEGIN
16     PERFORM f('hey 1');
17     COMMIT;
18     PERFORM f('hey 2');
19     COMMIT;
20 END;
21 $$
22 LANGUAGE PLPGSQL;
23
24 WITH
25     cte_chain (v_chain_id) AS (
26         INSERT INTO timetable.chain (chain_name, run_at, max_instances, live, self_
27 ↪destruct)
28         VALUES (
29             'call proc() every 10 sec', -- chain_name,
30             '@every 10 seconds',      -- run_at,
31             1,                        -- max_instances,
32             TRUE,                      -- live,
33             FALSE -- self_destruct
34         ) RETURNING chain_id
35     ),
36     cte_task(v_task_id) AS (
37         INSERT INTO timetable.task (chain_id, task_order, kind, command, ignore_error,
38 ↪autonomous)
39         SELECT v_chain_id, 10, 'SQL', 'CALL test_proc()', TRUE, TRUE

```

(continues on next page)

(continued from previous page)

```

38     FROM cte_chain
39     RETURNING task_id
40 )
41 SELECT v_chain_id, v_task_id FROM cte_task, cte_chain;

```

6.5 Shutdown the scheduler and terminate the session

This sample demonstrates how to shutdown the scheduler using special built-in task. This can be used to control maintenance windows, to restart the scheduler for update purposes, or to stop session before the database should be dropped.

```

1  -- This one-task chain (aka job) will terminate pg_timetable session.
2  -- This is useful for maintaining purposes or before database being destroyed.
3  -- One should take care of restarting pg_timetable if needed.
4
5  SELECT timetable.add_job (
6     job_name      => 'Shutdown pg_timetable session on schedule',
7     job_schedule => '* * 1 * *',
8     job_command  => 'Shutdown',
9     job_kind     => 'BUILTIN'
10 );

```

6.6 Access previous task result code and output from the next task

This sample demonstrates how to check the result code and output of a previous task. If the last task failed, that is possible only if *ignore_error boolean = true* is set for that task. Otherwise, a scheduler will stop the chain. This sample shows how to calculate failed, successful, and the total number of tasks executed. Based on these values, we can calculate the success ratio.

```

1  WITH
2     cte_chain (v_chain_id) AS ( -- let's create a new chain and add tasks to it later
3         INSERT INTO timetable.chain (chain_name, run_at, max_instances, live)
4         VALUES ('many tasks', '* * * * *', 1, true)
5         RETURNING chain_id
6     ),
7     cte_tasks(v_task_id) AS ( -- now we'll add 500 tasks to the chain, some of them will
↳fail
8         INSERT INTO timetable.task (chain_id, task_order, kind, command, ignore_error)
9         SELECT v_chain_id, g.s, 'SQL', 'SELECT 1.0 / round(random())::int4;', true
10        FROM cte_chain, generate_series(1, 500) AS g(s)
11        RETURNING task_id
12    ),
13    report_task(v_task_id) AS ( -- and the last reporting task will calculate the
↳statistic
14        INSERT INTO timetable.task (chain_id, task_order, kind, command)
15        SELECT v_chain_id, 501, 'SQL', $CMD$DO
16    $$
17 DECLARE

```

(continues on next page)

(continued from previous page)

```
18     s TEXT;
19 BEGIN
20     WITH report AS (
21         SELECT
22             count(*) FILTER (WHERE returncode = 0) AS success,
23             count(*) FILTER (WHERE returncode != 0) AS fail,
24             count(*) AS total
25         FROM timetable.execution_log
26         WHERE chain_id = current_setting('pg_timetable.current_chain_id')::bigint
27             AND txid = txid_current()
28     )
29     SELECT 'Tasks executed:' || total ||
30           '; succeeded: ' || success ||
31           '; failed: ' || fail ||
32           '; ratio: ' || 100.0*success/GREATEST(total,1)
33     INTO s
34     FROM report;
35     RAISE NOTICE '%', s;
36 END;
37 $$
38 $CMD$
39     FROM cte_chain
40     RETURNING task_id
41 )
42 SELECT v_chain_id FROM cte_chain
```


MIGRATION FROM OTHERS SCHEDULERS

7.1 Migrate jobs from `pg_cron` to `pg_timetable`

If you want to quickly export jobs scheduled from `pg_cron` to `pg_timetable`, you can use this SQL snippet:

```
1 SELECT timetable.add_job(  
2     job_name           => COALESCE(jobname, 'job: ' || command),  
3     job_schedule       => schedule,  
4     job_command        => command,  
5     job_kind           => 'SQL',  
6     job_live           => active  
7 ) FROM cron.job;
```

The `timetable.add_job()`, however, has some limitations. First of all, the function will mark the task created as **autonomous**, specifying scheduler should execute the task out of the chain transaction. It's not an error, but many autonomous chains may cause some extra connections to be used.

Secondly, database connection parameters are lost for source `pg_cron` jobs, making all jobs local. To export every information available precisely as possible, use this SQL snippet under the role they were scheduled in `pg_cron`:

```
1 SET ROLE 'scheduler'; -- set the role used by pg_cron  
2  
3 WITH cron_chain AS (  
4     SELECT  
5         nextval('timetable.chain_chain_id_seq'::regclass) AS cron_id,  
6         jobname,  
7         schedule,  
8         active,  
9         command,  
10        CASE WHEN  
11            database != current_database()  
12            OR nodename != 'localhost'  
13            OR username != CURRENT_USER  
14            OR nodeport != inet_server_port()  
15        THEN  
16            format('host=%s port=%s dbname=%s user=%s', nodename, nodeport, database,  
17            ↪username)  
18        END AS connstr  
19        FROM  
20        cron.job  
)
```

(continues on next page)

(continued from previous page)

```

21 cte_chain AS (
22     INSERT INTO timetable.chain (chain_id, chain_name, run_at, live)
23     SELECT
24         cron_id, COALESCE(jobname, 'cronjob' || cron_id), schedule, active
25     FROM
26         cron_chain
27 ),
28 cte_tasks AS (
29     INSERT INTO timetable.task (chain_id, task_order, kind, command, database_
↳ connection)
30     SELECT
31         cron_id, 1, 'SQL', command, connstr
32     FROM
33         cron_chain
34     RETURNING
35         chain_id, task_id
36 )
37 SELECT * FROM cte_tasks;

```

7.2 Migrate jobs from pgAgent to pg_timetable

To migrate jobs from **pgAgent**, please use this script. **pgAgent** doesn't have concept of *PROGRAM* task, thus to emulate *BATCH* steps, **pg_timetable** will execute them inside the shell. You may change the shell by editing *cte_shell* CTE clause.

```

1 CREATE OR REPLACE FUNCTION bool_array_to_cron(bool[], start_with int4 DEFAULT 0) RETURNS_
↳ TEXT AS
2 $$
3 WITH u AS (
4     SELECT unnest($1) e, generate_series($2, array_length($1, 1)-1+$2) AS i
5 )
6 SELECT COALESCE(string_agg(i::text, ','), '*') FROM u WHERE e
7 $$
8 LANGUAGE sql;
9
10
11 WITH
12 cte_shell(shell, cmd_param) AS (
13     VALUES ('sh', '-c') -- set the shell you want to use for batch steps, e.g. "pwsh_
↳ -c", "cmd /C"
14 ),
15 pga_schedule AS (
16     SELECT
17         s.jscjobid,
18         s.jscname,
19         format('%s %s %s %s %s',
20             bool_array_to_cron(s.jscminutes),
21             bool_array_to_cron(s.jschours),
22             bool_array_to_cron(s.jscmonthdays),
23             bool_array_to_cron(s.jscmonths, 1),

```

(continues on next page)

(continued from previous page)

```

24         bool_array_to_cron(s.jscweekdays, 1)) AS schedule
25     FROM
26         pgagent.pga_schedule s
27     WHERE s.jscenabled
28           AND now() < COALESCE(s.jscend, 'infinity'::timestamptz)
29           AND now() > s.jscstart
30 ),
31 pga_chain AS (
32     SELECT
33         nextval('timetable.chain_chain_id_seq'::regclass) AS chain_id,
34         jobid,
35         format('%s @ %s', jobname, jscname) AS jobname,
36         jobhostagent,
37         jobenabled,
38         schedule
39     FROM
40         pgagent.pga_job JOIN pga_schedule ON jobid = jscjobid
41 ),
42 cte_chain AS (
43     INSERT INTO timetable.chain (chain_id, chain_name, client_name, run_at, live)
44         SELECT
45             chain_id, jobname, jobhostagent, schedule, jobenabled
46         FROM
47             pga_chain
48 ),
49 pga_step AS (
50     SELECT
51         c.chain_id,
52         nextval('timetable.task_task_id_seq'::regclass) AS task_id,
53         rank() OVER (ORDER BY jstname) AS jstorder,
54         jstid,
55         jstname,
56         jstenabled,
57         CASE jstkind WHEN 'b' THEN 'PROGRAM' ELSE 'SQL' END AS jstkind,
58         jstcode,
59         COALESCE(
60             NULLIF(jstconnstr, ''),
61             CASE
62                 WHEN jstdbname = current_database() THEN NULL
63                 WHEN jstdbname > '' THEN 'dbname=' || jstdbname
64             END
65         ) AS jstconnstr,
66         jstonerror != 'f' AS jstignoreerror
67     FROM
68         pga_chain c JOIN pgagent.pga_jobstep js ON c.jobid = js.jstjobid
69 ),
70 cte_tasks AS (
71     INSERT INTO timetable.task(task_id, chain_id, task_name, task_order, kind,
72     ↪command, database_connection)
73     SELECT
74         task_id, chain_id, jstname, jstorder, jstkind::timetable.command_
75     ↪kind,

```

(continues on next page)

(continued from previous page)

```
74         CASE jstkind WHEN 'SQL' THEN jstcode ELSE sh.shell END,  
75         jstconnstr  
76     FROM  
77         pga_step, cte_shell sh  
78 ),  
79 cte_parameters AS (  
80     INSERT INTO timetable.parameter (task_id, order_id, value)  
81     SELECT  
82         task_id, 1, jsonb_build_array(sh.cmd_param, s.jstcode)  
83     FROM  
84         pga_step s, cte_shell sh  
85     WHERE  
86         s.jstkind = 'PROGRAM'  
87 )  
88 SELECT * FROM pga_chain;
```


REST API

pg_timetable has a rich REST API, which can be used by external tools in order to perform start/stop/reinitialize/restarts/reloads, by any kind of tools to perform HTTP health checks, and of course, could also be used for monitoring.

Below you will find the list of **pg_timetable** REST API endpoints.

8.1 Health check endpoints

GET /liveness

Always returns HTTP status code 200, indicating that **pg_timetable** is running.

GET /readiness

Returns HTTP status code 200 when the **pg_timetable** is running, and the scheduler is in the main loop processing chains. If the scheduler connects to the database, creates the database schema, or upgrades it, it will return the HTTP status code 503.

8.2 Chain management endpoints

GET /startchain?id=<chain-id>

Returns HTTP status code 200 if the chain with the given id can be added to the worker queue. It doesn't, however, mean the chain execution starts immediately. It is up to the worker to perform load and other checks before starting the chain. In the case of an error, the HTTP status code 400 followed by an error message returned.

GET /stopchain?id=<chain-id>

Returns HTTP status code 200 if the chain with the given id is working at the moment and can be stopped. If the chain is running the cancel signal would be sent immediately. In the case of an error, the HTTP status code 400 followed by an error message returned.

DATABASE SCHEMA

`pg_timetable` is a database driven application. During the first start the necessary schema is created if absent.

9.1 Main tables and objects

```

1 CREATE SCHEMA timetable;
2
3 -- define migrations you need to apply
4 -- every change to this file should populate this table.
5 -- Version value should contain issue number zero padded followed by
6 -- short description of the issue\feature\bug implemented\resolved
7 CREATE TABLE timetable.migration(
8     id INT8 NOT NULL,
9     version TEXT NOT NULL,
10    PRIMARY KEY (id)
11 );
12
13 INSERT INTO
14     timetable.migration (id, version)
15 VALUES
16     (0, '00259 Restart migrations for v4'),
17     (1, '00305 Fix timetable.is_cron_in_time'),
18     (2, '00323 Append timetable.delete_job function'),
19     (3, '00329 Migration required for some new added functions'),
20     (4, '00334 Refactor timetable.task as plain schema without tree-like dependencies'),
21     (5, '00381 Rewrite active chain handling'),
22     (6, '00394 Add started_at column to active_session and active_chain tables'),
23     (7, '00417 Rename LOG database log level to INFO'),
24     (8, '00436 Add txid column to timetable.execution_log');
25
26 CREATE DOMAIN timetable.cron AS TEXT CHECK(
27     substr(VALUE, 1, 6) IN ('@every', '@after') AND (substr(VALUE, 7) :: INTERVAL) IS_
↳ NOT NULL
28     OR VALUE = '@reboot'
29     OR VALUE ~ '^(((\d+,)+\d+|(\d+(\-|\/)\d+)|(\*(\-|\/)\d+)|\d+|\*) +){4}(((\d+,)+\d+|(\
↳ \d+(\-|\/)\d+)|(\*(\-|\/)\d+)|\d+|\*) ?)$'
30 );
31
32 COMMENT ON DOMAIN timetable.cron IS 'Extended CRON-style notation with support of_

```

(continues on next page)

(continued from previous page)

```

↪interval values';
33
34 CREATE TABLE timetable.chain (
35     chain_id          BIGSERIAL    PRIMARY KEY,
36     chain_name       TEXT         NOT NULL UNIQUE,
37     run_at           timetable.cron,
38     max_instances    INTEGER,
39     timeout          INTEGER      DEFAULT 0,
40     live             BOOLEAN      DEFAULT FALSE,
41     self_destruct    BOOLEAN      DEFAULT FALSE,
42     exclusive_execution BOOLEAN    DEFAULT FALSE,
43     client_name      TEXT
44 );
45
46 COMMENT ON TABLE timetable.chain IS
47     'Stores information about chains schedule';
48 COMMENT ON COLUMN timetable.chain.run_at IS
49     'Extended CRON-style time notation the chain has to be run at';
50 COMMENT ON COLUMN timetable.chain.max_instances IS
51     'Number of instances (clients) this chain can run in parallel';
52 COMMENT ON COLUMN timetable.chain.timeout IS
53     'Abort any chain that takes more than the specified number of milliseconds';
54 COMMENT ON COLUMN timetable.chain.live IS
55     'Indication that the chain is ready to run, set to FALSE to pause execution';
56 COMMENT ON COLUMN timetable.chain.self_destruct IS
57     'Indication that this chain will delete itself after successful run';
58 COMMENT ON COLUMN timetable.chain.exclusive_execution IS
59     'All parallel chains should be paused while executing this chain';
60 COMMENT ON COLUMN timetable.chain.client_name IS
61     'Only client with this name is allowed to run this chain, set to NULL to allow any
↪client';
62
63 CREATE TYPE timetable.command_kind AS ENUM ('SQL', 'PROGRAM', 'BUILTIN');
64
65 CREATE TABLE timetable.task (
66     task_id          BIGSERIAL          PRIMARY KEY,
67     chain_id        BIGINT             REFERENCES timetable.chain(chain_id) ON
↪UPDATE CASCADE ON DELETE CASCADE,
68     task_order      DOUBLE PRECISION    NOT NULL,
69     task_name       TEXT,
70     kind           timetable.command_kind NOT NULL DEFAULT 'SQL',
71     command         TEXT               NOT NULL,
72     run_as         TEXT,
73     database_connection TEXT,
74     ignore_error    BOOLEAN            NOT NULL DEFAULT FALSE,
75     autonomous      BOOLEAN            NOT NULL DEFAULT FALSE,
76     timeout        INTEGER             DEFAULT 0
77 );
78
79 COMMENT ON TABLE timetable.task IS
80     'Holds information about chain elements aka tasks';
81 COMMENT ON COLUMN timetable.task.chain_id IS

```

(continues on next page)

(continued from previous page)

```

82     'Link to the chain, if NULL task considered to be disabled';
83 COMMENT ON COLUMN timetable.task.task_order IS
84     'Indicates the order of task within a chain';
85 COMMENT ON COLUMN timetable.task.run_as IS
86     'Role name to run task as. Uses SET ROLE for SQL commands';
87 COMMENT ON COLUMN timetable.task.ignore_error IS
88     'Indicates whether a next task in a chain can be executed regardless of the success.
89     ↳of the current one';
90 COMMENT ON COLUMN timetable.task.kind IS
91     'Indicates whether "command" is SQL, built-in function or an external program';
92 COMMENT ON COLUMN timetable.task.command IS
93     'Contains either an SQL command, or command string to be executed';
94 COMMENT ON COLUMN timetable.task.timeout IS
95     'Abort any task within a chain that takes more than the specified number of.
96     ↳milliseconds';
97
98 -- parameter passing for a chain task
99 CREATE TABLE timetable.parameter(
100     task_id    BIGINT REFERENCES timetable.task(task_id)
101               ON UPDATE CASCADE ON DELETE CASCADE,
102     order_id  INTEGER CHECK (order_id > 0),
103     value     JSONB,
104     PRIMARY KEY (task_id, order_id)
105 );
106
107 COMMENT ON TABLE timetable.parameter IS
108     'Stores parameters passed as arguments to a chain task';
109
110 CREATE UNLOGGED TABLE timetable.active_session(
111     client_pid BIGINT NOT NULL,
112     server_pid BIGINT NOT NULL,
113     client_name TEXT NOT NULL,
114     started_at TIMESTAMPTZ DEFAULT now()
115 );
116
117 COMMENT ON TABLE timetable.active_session IS
118     'Stores information about active sessions';
119
120 CREATE TYPE timetable.log_type AS ENUM ('DEBUG', 'NOTICE', 'INFO', 'ERROR', 'PANIC',
121     ↳'USER');
122
123 CREATE OR REPLACE FUNCTION timetable.get_client_name(integer) RETURNS TEXT AS
124 $$
125     SELECT client_name FROM timetable.active_session WHERE server_pid = $1 LIMIT 1
126 $$
127 LANGUAGE sql;
128
129 CREATE TABLE timetable.log
130 (
131     ts          TIMESTAMPTZ          DEFAULT now(),
132     pid         INTEGER              NOT NULL,
133     log_level   timetable.log_type  NOT NULL,

```

(continues on next page)

(continued from previous page)

```

131     client_name    TEXT                DEFAULT timetable.get_client_name(pg_backend_
↳ pid()),
132     message       TEXT,
133     message_data  jsonb
134 );
135
136 COMMENT ON TABLE timetable.log IS
137     'Stores log entries of active sessions';
138
139 CREATE TABLE timetable.execution_log (
140     chain_id      BIGINT,
141     task_id       BIGINT,
142     txid          INTEGER NOT NULL,
143     last_run      TIMESTAMPTZ DEFAULT now(),
144     finished      TIMESTAMPTZ,
145     pid           BIGINT,
146     returncode   INTEGER,
147     kind          timetable.command_kind,
148     command       TEXT,
149     output        TEXT,
150     client_name   TEXT                NOT NULL
151 );
152
153 COMMENT ON TABLE timetable.execution_log IS
154     'Stores log entries of executed tasks and chains';
155
156 CREATE UNLOGGED TABLE timetable.active_chain(
157     chain_id      BIGINT NOT NULL,
158     client_name   TEXT NOT NULL,
159     started_at   TIMESTAMPTZ DEFAULT now()
160 );
161
162 COMMENT ON TABLE timetable.active_chain IS
163     'Stores information about active chains within session';
164
165 CREATE OR REPLACE FUNCTION timetable.try_lock_client_name(worker_pid BIGINT, worker_name_
↳ TEXT)
166 RETURNS bool AS
167 $CODE$
168 BEGIN
169     IF pg_is_in_recovery() THEN
170         RAISE NOTICE 'Cannot obtain lock on a replica. Please, use the primary node';
171         RETURN FALSE;
172     END IF;
173     -- remove disconnected sessions
174     DELETE
175     FROM timetable.active_session
176     WHERE server_pid NOT IN (
177         SELECT pid
178         FROM pg_catalog.pg_stat_activity
179         WHERE application_name = 'pg_timetable'
180     );

```

(continues on next page)

(continued from previous page)

```

181 DELETE
182     FROM timetable.active_chain
183     WHERE client_name NOT IN (
184         SELECT client_name FROM timetable.active_session
185     );
186 -- check if there any active sessions with the client name but different client pid
187 PERFORM 1
188     FROM timetable.active_session s
189     WHERE
190         s.client_pid <> worker_pid
191         AND s.client_name = worker_name
192     LIMIT 1;
193 IF FOUND THEN
194     RAISE NOTICE 'Another client is already connected to server with name: %',
195     ↪ worker_name;
196     RETURN FALSE;
197 END IF;
198 -- insert current session information
199 INSERT INTO timetable.active_session(client_pid, client_name, server_pid) VALUES
200 ↪ (worker_pid, worker_name, pg_backend_pid());
201 RETURN TRUE;
202 END;
203 $CODE$
204 STRICT
205 LANGUAGE plpgsql;

```

9.2 Jobs related functions

```

1 -- add_task() will add a task to the same chain as the task with `parent_id`
2 CREATE OR REPLACE FUNCTION timetable.add_task(
3     IN kind timetable.command_kind,
4     IN command TEXT,
5     IN parent_id BIGINT,
6     IN order_delta DOUBLE PRECISION DEFAULT 10
7 ) RETURNS BIGINT AS $$
8     INSERT INTO timetable.task (chain_id, task_order, kind, command)
9         SELECT chain_id, task_order + $4, $1, $2 FROM timetable.task WHERE task_id = $3
10         RETURNING task_id
11 $$ LANGUAGE SQL;
12
13 COMMENT ON FUNCTION timetable.add_task IS 'Add a task to the same chain as the task with
14 ↪ parent_id';
15
16 -- add_job() will add one-task chain to the system
17 CREATE OR REPLACE FUNCTION timetable.add_job(
18     job_name          TEXT,
19     job_schedule      timetable.cron,
20     job_command       TEXT,
21     job_parameters    JSONB DEFAULT NULL,

```

(continues on next page)

(continued from previous page)

```

21     job_kind          timetable.command_kind DEFAULT 'SQL'::timetable.command_kind,
22     job_client_name   TEXT DEFAULT NULL,
23     job_max_instances INTEGER DEFAULT NULL,
24     job_live          BOOLEAN DEFAULT TRUE,
25     job_self_destruct BOOLEAN DEFAULT FALSE,
26     job_ignore_errors BOOLEAN DEFAULT TRUE,
27     job_exclusive     BOOLEAN DEFAULT FALSE
28 ) RETURNS BIGINT AS $$
29     WITH
30         cte_chain (v_chain_id) AS (
31             INSERT INTO timetable.chain (chain_name, run_at, max_instances, live, self_
↳destruct, client_name, exclusive_execution)
32             VALUES (job_name, job_schedule, job_max_instances, job_live, job_self_
↳destruct, job_client_name, job_exclusive)
33             RETURNING chain_id
34         ),
35         cte_task(v_task_id) AS (
36             INSERT INTO timetable.task (chain_id, task_order, kind, command, ignore_
↳error, autonomous)
37             SELECT v_chain_id, 10, job_kind, job_command, job_ignore_errors, TRUE
38             FROM cte_chain
39             RETURNING task_id
40         ),
41         cte_param AS (
42             INSERT INTO timetable.parameter (task_id, order_id, value)
43             SELECT v_task_id, 1, job_parameters FROM cte_task, cte_chain
44         )
45     SELECT v_chain_id FROM cte_chain
46 $$ LANGUAGE SQL;
47
48 COMMENT ON FUNCTION timetable.add_job IS 'Add one-task chain (aka job) to the system';
49
50 -- notify_chain_start() will send notification to the worker to start the chain
51 CREATE OR REPLACE FUNCTION timetable.notify_chain_start(
52     chain_id BIGINT,
53     worker_name TEXT
54 ) RETURNS void AS $$
55     SELECT pg_notify(
56         worker_name,
57         format('{"ConfigID": %s, "Command": "START", "Ts": %s}',
58             chain_id,
59             EXTRACT(epoch FROM clock_timestamp())::bigint)
60     )
61 $$ LANGUAGE SQL;
62
63 COMMENT ON FUNCTION timetable.notify_chain_start IS 'Send notification to the worker to
↳start the chain';
64
65 -- notify_chain_stop() will send notification to the worker to stop the chain
66 CREATE OR REPLACE FUNCTION timetable.notify_chain_stop(
67     chain_id BIGINT,
68     worker_name TEXT

```

(continues on next page)

(continued from previous page)

```

69 ) RETURNS void AS $$
70     SELECT pg_notify(
71         worker_name,
72         format('{ "ConfigID": %s, "Command": "STOP", "Ts": %s}',
73             chain_id,
74             EXTRACT(epoch FROM clock_timestamp())::bigint)
75     )
76 $$ LANGUAGE SQL;
77
78 COMMENT ON FUNCTION timetable.notify_chain_stop IS 'Send notification to the worker to
↳ stop the chain';
79
80 -- move_task_up() will switch the order of the task execution with a previous task
↳ within the chain
81 CREATE OR REPLACE FUNCTION timetable.move_task_up(IN task_id BIGINT) RETURNS boolean AS $
↳ $
82     WITH current_task (ct_chain_id, ct_id, ct_order) AS (
83         SELECT chain_id, task_id, task_order FROM timetable.task WHERE task_id =
↳ $1
84     ),
85     tasks(t_id, t_new_order) AS (
86         SELECT task_id, COALESCE(LAG(task_order) OVER w, LEAD(task_order) OVER w)
87         FROM timetable.task t, current_task ct
88         WHERE chain_id = ct_chain_id AND (task_order < ct_order OR task_id = ct_
↳ id)
89         WINDOW w AS (PARTITION BY chain_id ORDER BY ABS(task_order - ct_order))
90         LIMIT 2
91     ),
92     upd AS (
93         UPDATE timetable.task t SET task_order = t_new_order
94         FROM tasks WHERE tasks.t_id = t.task_id AND tasks.t_new_order IS NOT NULL
95         RETURNING true
96     )
97     SELECT COUNT(*) > 0 FROM upd
98 $$ LANGUAGE SQL;
99
100 COMMENT ON FUNCTION timetable.move_task_up IS 'Switch the order of the task execution
↳ with a previous task within the chain';
101
102 -- move_task_down() will switch the order of the task execution with a following task
↳ within the chain
103 CREATE OR REPLACE FUNCTION timetable.move_task_down(IN task_id BIGINT) RETURNS boolean
↳ AS $$
104     WITH current_task (ct_chain_id, ct_id, ct_order) AS (
105         SELECT chain_id, task_id, task_order FROM timetable.task WHERE task_id =
↳ $1
106     ),
107     tasks(t_id, t_new_order) AS (
108         SELECT task_id, COALESCE(LAG(task_order) OVER w, LEAD(task_order) OVER w)
109         FROM timetable.task t, current_task ct
110         WHERE chain_id = ct_chain_id AND (task_order > ct_order OR task_id = ct_
↳ id)

```

(continues on next page)

(continued from previous page)

```

111         WINDOW w AS (PARTITION BY chain_id ORDER BY ABS(task_order - ct_order))
112         LIMIT 2
113     ),
114     upd AS (
115         UPDATE timetable.task t SET task_order = t_new_order
116         FROM tasks WHERE tasks.t_id = t.task_id AND tasks.t_new_order IS NOT NULL
117         RETURNING true
118     )
119     SELECT COUNT(*) > 0 FROM upd
120 $$ LANGUAGE SQL;
121
122 COMMENT ON FUNCTION timetable.move_task_down IS 'Switch the order of the task execution,
123 ↪with a following task within the chain';
124
125 -- delete_job() will delete the chain and its tasks from the system
126 CREATE OR REPLACE FUNCTION timetable.delete_job(IN job_name TEXT) RETURNS boolean AS $$
127     WITH del_chain AS (DELETE FROM timetable.chain WHERE chain.chain_name = $1 RETURNING
128     ↪chain_id)
129     SELECT EXISTS(SELECT 1 FROM del_chain)
130 $$ LANGUAGE SQL;
131
132 COMMENT ON FUNCTION timetable.delete_job IS 'Delete the chain and its tasks from the
133 ↪system';
134
135 -- delete_task() will delete the task from a chain
136 CREATE OR REPLACE FUNCTION timetable.delete_task(IN task_id BIGINT) RETURNS boolean AS $$
137     WITH del_task AS (DELETE FROM timetable.task WHERE task_id = $1 RETURNING task_id)
138     SELECT EXISTS(SELECT 1 FROM del_task)
139 $$ LANGUAGE SQL;
140
141 COMMENT ON FUNCTION timetable.delete_task IS 'Delete the task from a chain';

```

9.3 ron related functions

```

1 CREATE OR REPLACE FUNCTION timetable.cron_split_to_arrays(
2     cron text,
3     OUT mins integer[],
4     OUT hours integer[],
5     OUT days integer[],
6     OUT months integer[],
7     OUT dow integer[]
8 ) RETURNS record AS $$
9 DECLARE
10     a_element text[];
11     i_index integer;
12     a_tmp text[];
13     tmp_item text;
14     a_range int[];
15     a_split text[];
16     a_res integer[];

```

(continues on next page)

(continued from previous page)

```

17     allowed_range integer[];
18     max_val integer;
19     min_val integer;
20 BEGIN
21     a_element := regexp_split_to_array(cron, '\s+');
22     FOR i_index IN 1..5 LOOP
23         a_res := NULL;
24         a_tmp := string_to_array(a_element[i_index],',');
25         CASE i_index -- 1 - mins, 2 - hours, 3 - days, 4 - weeks, 5 - DOWs
26             WHEN 1 THEN allowed_range := '{0,59}';
27             WHEN 2 THEN allowed_range := '{0,23}';
28             WHEN 3 THEN allowed_range := '{1,31}';
29             WHEN 4 THEN allowed_range := '{1,12}';
30         ELSE
31             allowed_range := '{0,7}';
32         END CASE;
33         FOREACH tmp_item IN ARRAY a_tmp LOOP
34             IF tmp_item ~ '^[0-9]+$' THEN -- normal integer
35                 a_res := array_append(a_res, tmp_item::int);
36             ELSIF tmp_item ~ '^[*]+$' THEN -- '*' any value
37                 a_range := array(select generate_series(allowed_range[1], allowed_
↳range[2]));
38                 a_res := array_cat(a_res, a_range);
39             ELSIF tmp_item ~ '^[0-9]+[-][0-9]+$' THEN -- '-' range of values
40                 a_range := regexp_split_to_array(tmp_item, '-');
41                 a_range := array(select generate_series(a_range[1], a_range[2]));
42                 a_res := array_cat(a_res, a_range);
43             ELSIF tmp_item ~ '^[0-9]+[\/][0-9]+$' THEN -- '/' step values
44                 a_range := regexp_split_to_array(tmp_item, '/');
45                 a_range := array(select generate_series(a_range[1], allowed_range[2], a_
↳range[2]));
46                 a_res := array_cat(a_res, a_range);
47             ELSIF tmp_item ~ '^[0-9-]+[\/][0-9]+$' THEN -- '-' range of values and '/'
↳step values
48                 a_split := regexp_split_to_array(tmp_item, '/');
49                 a_range := regexp_split_to_array(a_split[1], '-');
50                 a_range := array(select generate_series(a_range[1], a_range[2], a_
↳split[2]::int));
51                 a_res := array_cat(a_res, a_range);
52             ELSIF tmp_item ~ '^[*]+[\/][0-9]+$' THEN -- '*' any value and '/' step values
53                 a_split := regexp_split_to_array(tmp_item, '/');
54                 a_range := array(select generate_series(allowed_range[1], allowed_
↳range[2], a_split[2]::int));
55                 a_res := array_cat(a_res, a_range);
56             ELSE
57                 RAISE EXCEPTION 'Value ("%") not recognized', a_element[i_index]
58                     USING HINT = 'fields separated by space or tab.'+
59                     'Values allowed: numbers (value list with ","), '+
60                     'any value with "*", range of value with "-" and step values with "/"
↳"!';
61             END IF;
62         END LOOP;

```

(continues on next page)

(continued from previous page)

```

63     SELECT
64         ARRAY_AGG(x.val), MIN(x.val), MAX(x.val) INTO a_res, min_val, max_val
65     FROM (
66         SELECT DISTINCT UNNEST(a_res) AS val ORDER BY val) AS x;
67     IF max_val > allowed_range[2] OR min_val < allowed_range[1] THEN
68         RAISE EXCEPTION '% is out of range: %', a_res, allowed_range;
69     END IF;
70     CASE i_index
71         WHEN 1 THEN mins := a_res;
72         WHEN 2 THEN hours := a_res;
73         WHEN 3 THEN days := a_res;
74         WHEN 4 THEN months := a_res;
75     ELSE
76         dow := a_res;
77     END CASE;
78     END LOOP;
79     RETURN;
80 END;
81 $$ LANGUAGE PLPGSQL STRICT;
82
83 CREATE OR REPLACE FUNCTION timetable.cron_months(
84     from_ts timestamptz,
85     allowed_months int[]
86 ) RETURNS SETOF timestamptz AS $$
87     WITH
88     am(am) AS (SELECT UNNEST(allowed_months)),
89     genm(ts) AS ( --generated months
90         SELECT date_trunc('month', ts)
91         FROM pg_catalog.generate_series(from_ts, from_ts + INTERVAL '1 year', INTERVAL
92         ↪ '1 month') g(ts)
93     )
94     SELECT ts FROM genm JOIN am ON date_part('month', genm.ts) = am.am
95 $$ LANGUAGE SQL STRICT;
96
97 CREATE OR REPLACE FUNCTION timetable.cron_days(
98     from_ts timestamptz,
99     allowed_months int[],
100    allowed_days int[],
101    allowed_week_days int[]
102 ) RETURNS SETOF timestamptz AS $$
103     WITH
104     ad(ad) AS (SELECT UNNEST(allowed_days)),
105     am(am) AS (SELECT * FROM timetable.cron_months(from_ts, allowed_months)),
106     gend(ts) AS ( --generated days
107         SELECT date_trunc('day', ts)
108         FROM am,
109             pg_catalog.generate_series(am.am, am.am + INTERVAL '1 month'
110             - INTERVAL '1 day', -- don't include the same day of the next month
111             INTERVAL '1 day') g(ts)
112     )
113     SELECT ts
114     FROM gend JOIN ad ON date_part('day', gend.ts) = ad.ad

```

(continues on next page)

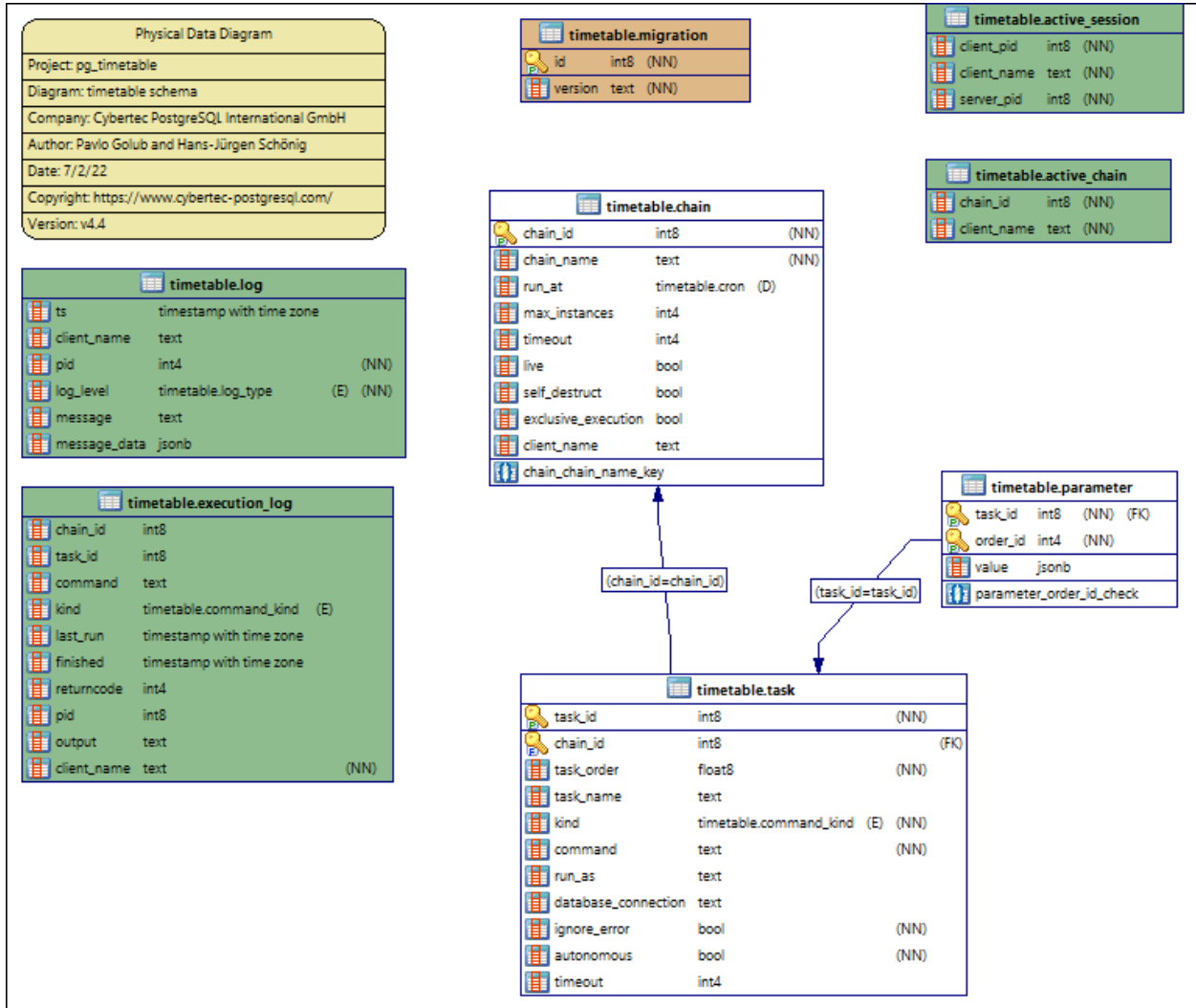
(continued from previous page)

```

114     WHERE extract(dow from ts)=ANY(allowed_week_days)
115 $$ LANGUAGE SQL STRICT;
116
117 CREATE OR REPLACE FUNCTION timetable.cron_times(
118     allowed_hours int[],
119     allowed_minutes int[]
120 ) RETURNS SETOF time AS $$
121     WITH
122     ah(ah) AS (SELECT UNNEST(allowed_hours)),
123     am(am) AS (SELECT UNNEST(allowed_minutes))
124     SELECT make_time(ah.ah, am.am, 0) FROM ah CROSS JOIN am
125 $$ LANGUAGE SQL STRICT;
126
127
128 CREATE OR REPLACE FUNCTION timetable.cron_runs(
129     from_ts timestamp with time zone,
130     cron text
131 ) RETURNS SETOF timestamptz AS $$
132     SELECT cd + ct
133     FROM
134         timetable.cron_split_to_arrays(cron) a,
135         timetable.cron_times(a.hours, a.mins) ct CROSS JOIN
136         timetable.cron_days(from_ts, a.months, a.days, a.dow) cd
137     WHERE cd + ct > from_ts
138     ORDER BY 1 ASC;
139 $$ LANGUAGE SQL STRICT;
140
141 -- is_cron_in_time returns TRUE if timestamp is listed in cron expression
142 CREATE OR REPLACE FUNCTION timetable.is_cron_in_time(
143     run_at timetable.cron,
144     ts timestamptz
145 ) RETURNS BOOLEAN AS $$
146     SELECT
147     CASE WHEN run_at IS NULL THEN
148         TRUE
149     ELSE
150         date_part('month', ts) = ANY(a.months)
151         AND (date_part('dow', ts) = ANY(a.dow) OR date_part('isodow', ts) = ANY(a.dow))
152         AND date_part('day', ts) = ANY(a.days)
153         AND date_part('hour', ts) = ANY(a.hours)
154         AND date_part('minute', ts) = ANY(a.mins)
155     END
156     FROM
157         timetable.cron_split_to_arrays(run_at) a
158 $$ LANGUAGE SQL;
159
160 CREATE OR REPLACE FUNCTION timetable.next_run(cron timetable.cron) RETURNS timestamptz
161     AS $$
162     SELECT * FROM timetable.cron_runs(now(), cron) LIMIT 1
163 $$ LANGUAGE SQL STRICT;

```

9.4 ER-Diagram



INDICES AND TABLES

- `genindex`
- `modindex`
- `search`