The PostgreSQL Job Scheduler You Always Wanted (But Be Careful What You Ask For)

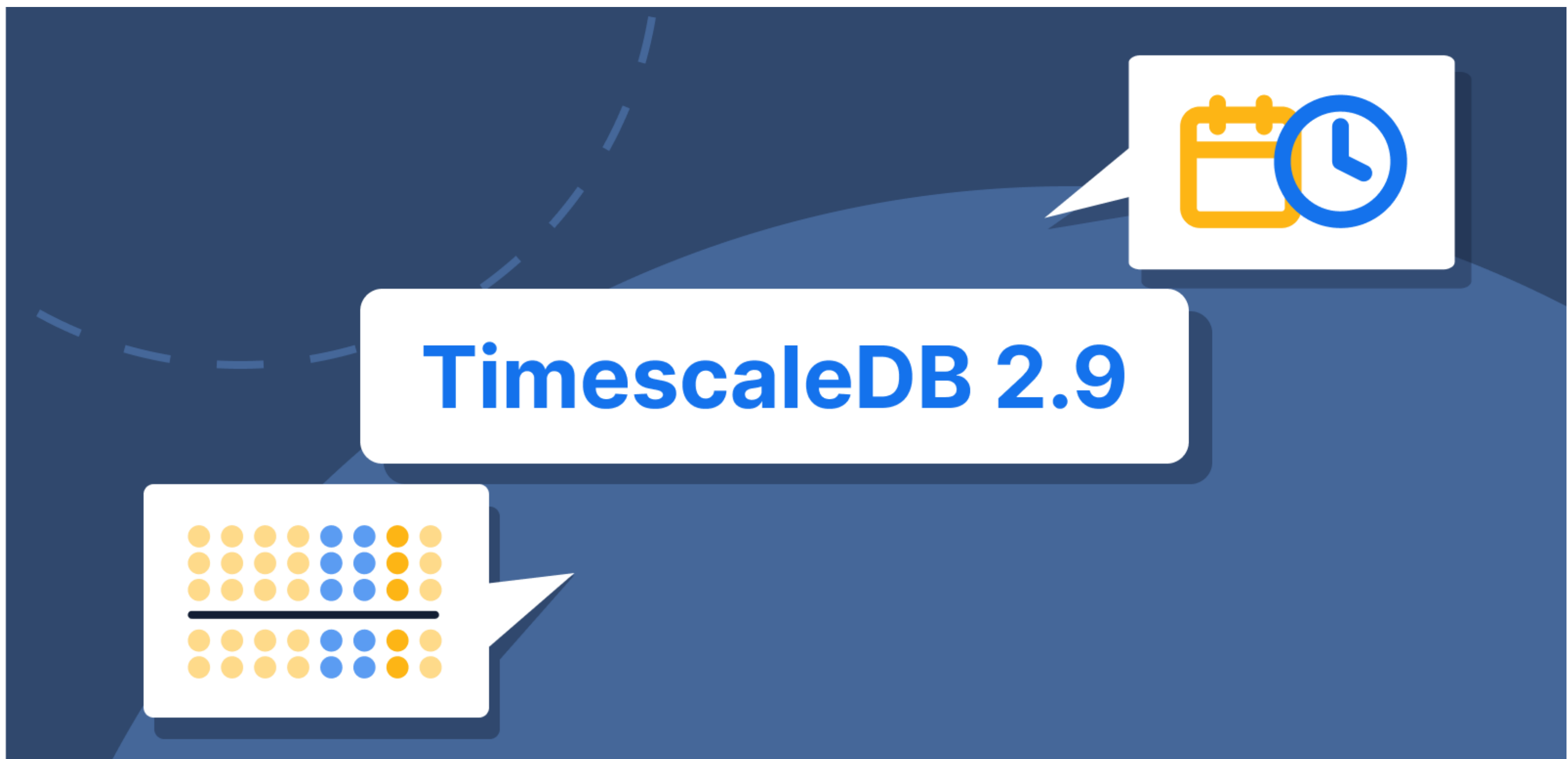


Table of contents

01 The PostgreSQL Job Scheduler Debate

02 Why We Need a Job Scheduler

This website stores data such as cookies to enable essential site functionality, as well as marketing, personalization, and analytics. By remaining on this website you indicate your consent.
Privacy Policy

See More

What is a database job scheduler? Essentially, a job scheduler is a process that kicks off in-database functions and procedures at specified times and runs them independently of user sessions. The benefits of having a scheduler built into the database are obvious: no dependencies, no inherent security leaks, fits in your existing high availability plan, and takes part in your data recovery plan, too.

As a PostgreSQL guy, it really makes you wonder why a built-in job scheduler is not a part of the core PostgreSQL project. It is one of the most requested features in the history of ever. Yet, somehow, it just isn't there.

The PostgreSQL Global Development Group has been debating for years about including a built-in job scheduler. Even after the addition of background processes that would support the feature (all the way back in 9.6), background job scheduling is unfortunately not a part of core PostgreSQL.

So being the PostgreSQL lovers we are at Timescale, we decided to build such a feature so that our users and customers can benefit from a job scheduler in PostgreSQL. And in our latest release (2.9.1), we extended it to allow you to schedule jobs with flexible intervals and provide you with better visibility of error logs.

The flexible intervals enable you to determine whether the next run of the job occurs based on the scheduled clock time or the end of the last job run. And by "better visibility" of the job logs, we mean that they are also being logged to a table where they can be queried internally. These were extended to prevent overlapping job executions, provide predictable job timing, and provide better forensics.

We extensively use the advantage of this internal scheduler for our core features, enabling us to defer compression, retention, and refreshing of continuous aggregates to a background process (among other things).

This makes the product much more responsive to the caller and results in more efficient processing of these tasks. For our own benefit, the job scheduler needs to be internal to the database. It also needs to be efficient, controllable, and scale with the installation.

And we make all this power available to you as an end user.

This website stores data such as cookies to enable essential site functionality, as well as marketing, personalization, and analytics. By remaining on this website you indicate your consent.
Privacy Policy

But before you start rejoicing, let’s review the reasons that the PostgreSQL Global Development Group chose not to include a scheduler in the database. Rather than rehashing the discussion list on the subject, let's summarize the obstacles that come up in the mailing list.

PostgreSQL is multi-process, not multi-thread. This simple fact makes having a one-to-one relationship of processes to user-defined tasks a fairly heavy implementation issue. Under normal circumstances, PostgreSQL expects to lay a process onto a CPU (affinity), load the memory through the closest non-uniform memory access (NUMA) controller, and do some fairly heavy data processing.

This works great when the expectation is that the process will be very busy the majority of the time. Schedulers do not work like that. They sit around with some cheap threads waiting to do something for the majority of the life of the thread. Just the context switching alone would make using a full-blown process very expensive.

Background workers' processes are a relatively small pool by design. This has a lot to do with the previous paragraph, but also that each process allocates the prescribed memory at startup. So, these processes compete with SQL query workers for CPU and memory. And the background processes have priority over both resources since they are allocated at system startup.

The next issue is more semantic. There are quite a few external schedulers available. Each one of them has a different implementation of the time management system. That is, there is a question about just how exactly the job should be invoked. Should it be invoked again if it is still running from the last time? Should the job be started again based on clock time or relative to the previous job run? From the beginning or the end of the last run?

There are quite a few more questions of this nature, but you get the idea. No matter how the community answers these questions, somebody will complain that the implementation is the wrong answer because `\<insert silly mathematician answer here\>`.

Why We Need a Job Scheduler

Timescale doesn't have the luxury of debating how many angels can dance on the head of a pin. As a time-series service, we face a hard requirement of background maintenance for the actions of archival, compression, and general storage.

This website stores data such as cookies to enable essential site functionality, as well as marketing, personalization, and analytics. By remaining on this website you indicate your consent.
Privacy Policy

application programming interface.

This general-purpose scheduler is available as part of TimescaleDB. You may use it to set a schedule for anything you can express as a procedure or function. In PostgreSQL, that's a huge advantage because you have the full power of the PostgreSQL extension system at your disposal. This list includes plug-in languages, which allow you to do anything the operating system can do.

Timescale assumes that the developer/administrator is a sane and reasonable person who can deal with a balance of complexity. That is longhand for "we trust you to do the right thing."

With Great Power Comes Great Responsibility

So, let's talk first about a few best design practices for using the Timescale scheduler.

> **Keep it short.** The dwell time of the background process can lead to high concurrency. You are also using a process shared by other system tasks such as sorting, sequential scans, and other system tasks.
>
> **Keep it unlocked.** Try to minimize the number of exclusive locks you create while doing your process.
>
> **Keep it down.** The processes that you are using are shared by the system, and you are competing for resources with SQL query worker processes. Keep that in mind before you kick off hundreds or thousands of scheduled jobs.

Now, assuming we are using the product fairly and judiciously, we can move on to the features and benefits of having an internal scheduler.

Built-In Job Scheduler: All the Right Stuff

Here's a list of some of the highlights when the scheduler is inside the DB:

This website stores data such as cookies to enable essential site functionality, as well as marketing, personalization, and analytics. By remaining on this website you indicate your consent.
Privacy Policy

You don't need a separate high-availability plan for your scheduler. If the system is alive, so are your scheduled jobs.

The jobs can report on their own success or failure to internal tables and the PostgreSQL log file.

The jobs can do administrative functions like dropping tables and changing table structure by monitoring the existing needs and structures.

When you install Timescale Cloud, it's already there.

How It Works

There is a quick introductory article in the Timescale documentation. Click that link if you want more detailed information.

The TL;DR version is that you make a PostgreSQL function or procedure and then call the `add_job()` function to schedule it. Of course, you can remove it from the schedule using... Wait for it... `delete_job()`.

That's it. Really. All that power is at your fingertips, and all you need to know is two function signatures.

Something to be aware of while you're using the scheduler is that the job may be scheduled to repeat from the end of the last run or from the scheduled clock time. This is a brand new feature of the current release of TimescaleDB. As of this writing, that is 2.9.1. This allows you to ensure that the previous job has completed (by picking from the end of the run) or that the job executes at a prescribed time (making job completion your responsibility).

If you feel a bit homesick and just want to look at your adorable job, there's also:

And, of course, for completeness, there's always `alter_job()` for rescheduling, renaming, etc.

Once your job has been created, it becomes the responsibility of the job scheduler to invoke it at the proper time. The job

This website stores data such as cookies to enable essential site functionality, as well as marketing, personalization, and analytics. By remaining on this website you indicate your consent.
Privacy Policy

If such a job is queued up, it will request another background process from the PostgreSQL master process. The database system will provide one (provided there are any available). The provided process becomes responsible for the execution of your job.

This basic operation has some ramifications. We have already mentioned that we need to use these background processes sparingly for resource allocation reasons. Also, there are only a few of them available. The maximum parallel count of background processes is determined by `max_worker_processes` . If you need help configuring TimescaleDB background workers, check out our documentation.

On my system (Kubuntu 22.04.1, PostgreSQL 14.6), the default is 43. That number is just an example, as the package manager for each distribution of PostgreSQL has discretion about the initial setting. Your mileage **will** vary.

Changing this parameter requires a restart, so you will need to make a judgment call about how many concurrent processes you expect to kick off. Add that to this base number and restart your system. Of course, a reasonable number has been added for you in Timescale Cloud. Remember the CPU and memory limitations while you are making this adjustment.

Some Ideas About What to Do With It

The original reasons for creating this scheduler are for time-series data management. That includes compression, continuous aggregates, retention policy implementation, downsampling, and backfilling.

You may want to use this for event notifications, sending an email, clustered index maintenance, partition creation, pruning, archiving, refreshing materialized views, or summarizing data somewhere to avoid the need for triggers. These are just a few of the obvious ideas that jump into my consciousness. You can literally do anything that the operating system allows.

Some Ideas About What Not to Do With It

This website stores data such as cookies to enable essential site functionality, as well as marketing, personalization, and analytics. By remaining on this website you indicate your consent.
Privacy Policy

`REFRESH MATERIALIZED VIEW CONCURRENTLY` is better than `REFRESH MATERIALIZED VIEW`. You get it. Use `CONCURRENTLY`, or design concurrently. Better yet, do things in a tiny atomic way that takes little time anyway.

Long-running transactions that create a lot of locks will interfere with the background writer, the planner, and the vacuum processes. If you crank up too many concurrent processes, you may also run out of memory. Please try to schedule everything to run in series. You'll thank me later.

Well Wishes to the Newly Crowned Emperor

Now you have the power to do anything your little heart desires in the background of PostgreSQL without having any external dependencies. We hope you feel empowered, awed, and a little bit special. We also hope you will use your new powers for good.

Try the Updated Job Scheduler

If you are using Timescale Cloud, upgrades are automatic, and you'll get to try our improved job scheduler in TimescaleDB 2.9.1 in your next maintenance window.

New to Timescale Cloud? Start a free 30-day trial, no credit card required, and get your new database journey started in five minutes.

If you're self-hosting TimescaleDB, follow the upgrade instructions in our documentation.

The open-source relational database for time-series and analytics.

Try Timescale for free

This website stores data such as cookies to enable essential site functionality, as well as marketing, personalization, and analytics. By remaining on this website you indicate your consent.
Privacy Policy

Share this Post

Related posts

Cloud

One-Click Multi-Node TimescaleDB: Announcing the Support for Multi-Node Deployments in Timescale Cloud

1 Nov 2021 • 7 min read

Always Be Launching

TimescaleDB 2.3: Improving Columnar Compression for Time-Series on PostgreSQL

26 May 2021 • 9 min read

Cloud

Best Practices for Time-Series Data Modeling: Narrow, Medium or Wide Table Layout

31 Jan 2023 • 11 min read

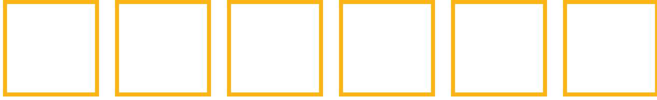
This website stores data such as cookies to enable essential site functionality, as well as marketing, personalization, and analytics. By remaining on this website you indicate your consent.
Privacy Policy

Subscribe to the Timescale Newsletter

Your email

Subscribe

By submitting you acknowledge the Timescale **Privacy Policy**.



2023 © Timescale, Inc. All Rights Reserved.

This website stores data such as cookies to enable essential site functionality, as well as marketing, personalization, and analytics. By remaining on this website you indicate your consent.

Privacy Policy