Mind the gap: Using SQL functions for time-series analysis

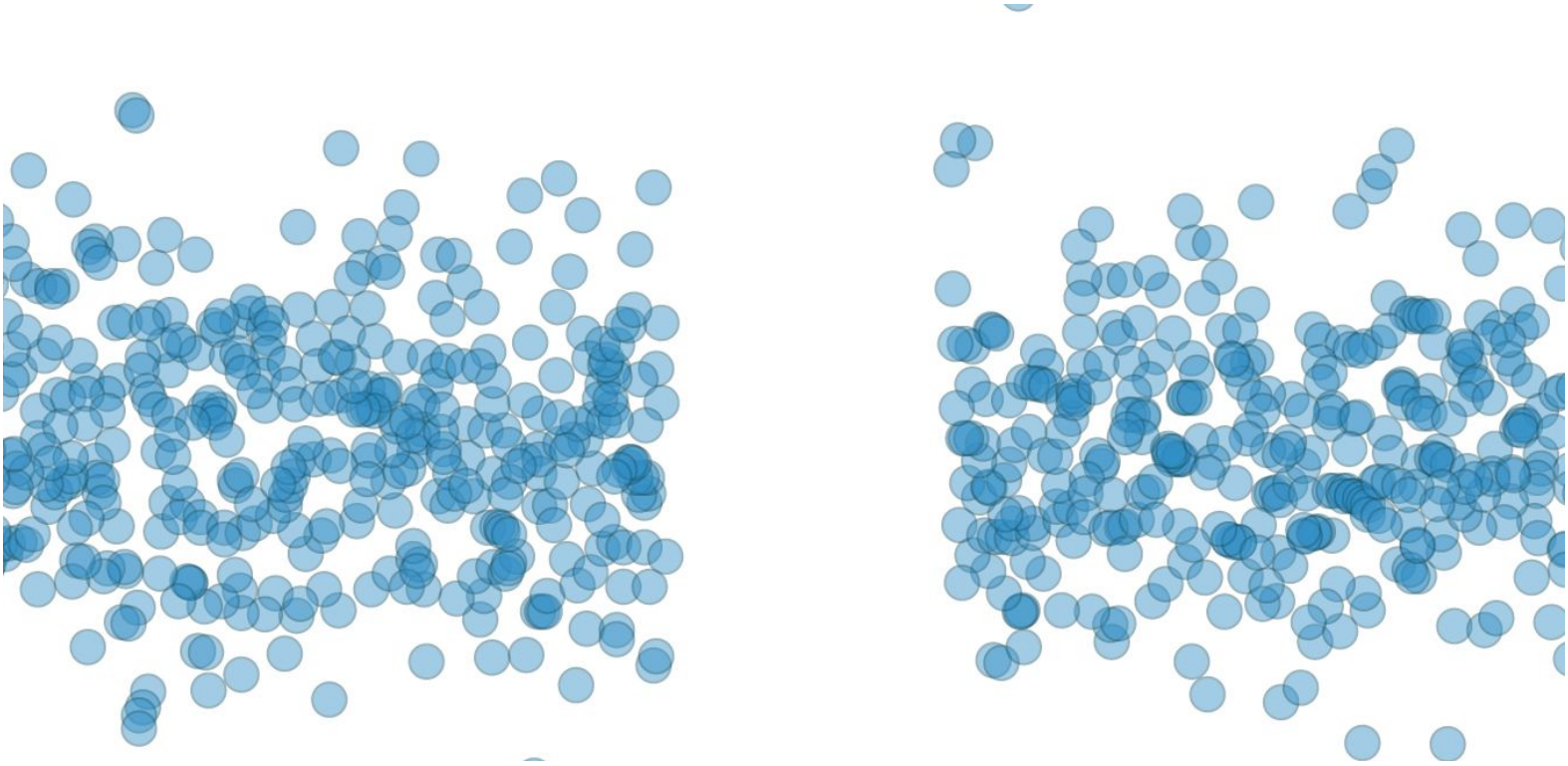


Table of contents

Write more efficient and readable queries with a new set of time-series analytic tools included in TimescaleDB 1.2

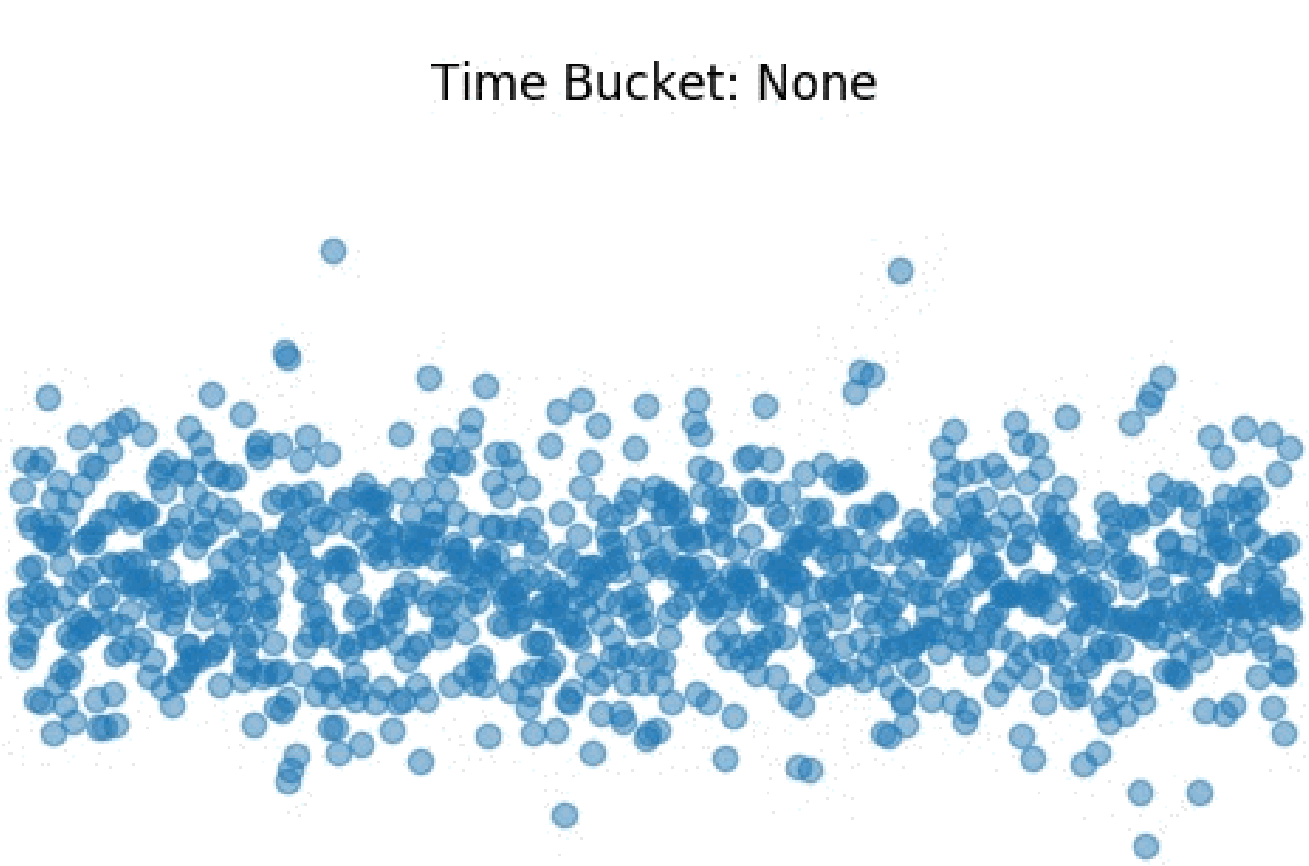
UPDATE: TimescaleDB 1.2 was released on January 29, 2019. Please refer to the TimescaleDB 1.3 release for the most recent improvements on certain features described below.

If you've been following the development of the upcoming TimescaleDB 1.2 release in GitHub, you'll notice three new SQL functions for time series analysis: `time_bucket_gapfill`, `interpolate`, and `locf`. Used together, these functions will enable you to write more efficient and readable queries for time-series analysis using SQL.

In this post, we'll talk about why you'd want to use time buckets, the related gap filling techniques, and how they're implemented under the hood. Ultimately it's the story of how we extended SQL and the PostgreSQL query planner to create a set of highly optimized functions for time-series analysis.

Introduction to time bucketing

Many common techniques for time series analysis assume that our temporal observations are aggregated to fixed time intervals. Dashboards and most visualizations of time series rely on this technique to make sense of our raw data, turning the noise into a smoother trend line that is more easily interpretable and analytically tractable.



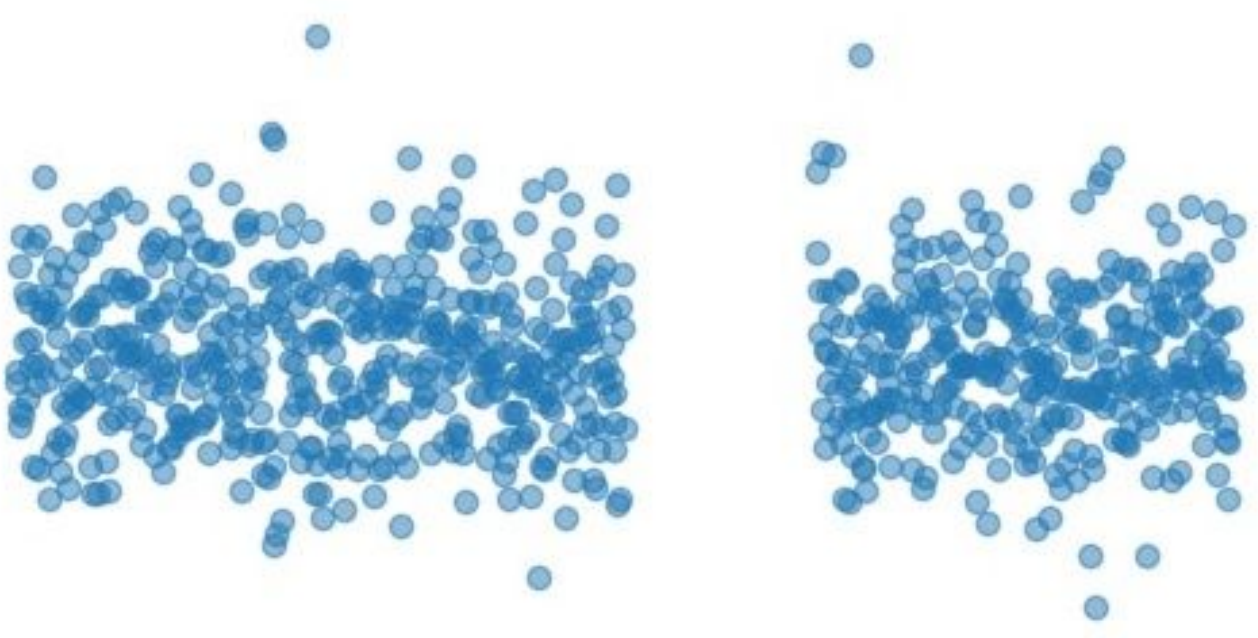
When writing queries for this type of reporting, we need an efficient way to aggregate raw observations (often noisy and irregular) to fixed time intervals. Examples of such queries might be average temperature per hour or the average CPU utilization per 5 seconds.

The solution is **time bucketing**. The `time_bucket` function has been a core feature of TimescaleDB since the first public beta release. With time bucketing, we can get a clear picture of the important data trends using a concise, declarative SQL query.

Challenges with time bucketing

The reality of time series data engineering is not always so easy.

Consider measurements recorded at **irregular sampling intervals,** either intentionally as with measurements recorded in response to external events (e.g. motion sensor). Or perhaps inadvertently due to network problems, out of sync clocks, or equipment taken offline for maintenance.



Time bucket: none

We should also consider the case of analyzing multiple measurements recorded at **mismatched sampling intervals**. For instance, you might collect some of your data every second and some every minute but still need to analyze both metrics at 15 second intervals.

The `time_bucket` function will only aggregate your data to a given time bucket if there is data in it. In both the cases of mismatched or irregular sampling, a time bucket interval might come back with missing data (i.e gaps).

Time bucket: 20 minutes

If your analysis requires data aggregated to contiguous time intervals, the time bucketing with **gap filling** solves this problem.

Time bucketing with gap filling

In upcoming TimescaleDB 1.2 release, community users will have access to:

- `time_bucket_gapfill` for creating contiguous, ordered time buckets.
- `interpolate` to perform linear interpolation between the previous and next value.
- `locf` or *last observation carried forward* to fill in gaps with the previous known value.

Gap filling

The new `time_bucket_gapfill` function is similar to `time_bucket` except that it guarantees a contiguous, ordered set of time buckets.

chronological order and contiguous.

We'll talk more below about how this is implemented below. For now, let’s look at the SQL:

Note that one of the hours is missing data entirely and the average value is represented as NULL. Gap filling gives us a contiguous set of time buckets but no data for those rows. That's where the `locf` and `interpolate` functions come into play.

LOCF or last observation carried forward

The “last observation carried forward” technique can be used to impute missing values by assuming the previous known value.

Shown here:

LOCF at 20 minutes

Linear interpolation

Linear interpolation imputes missing values by assuming a line between the previous known value and the next known value.

Shown here:

Interpolate at 20 minutes

These techniques are not exclusive; you can combine them as needed in a single time bucketed query:

Best practices

Whether you chose to use LOCF, interpolation, or gap filling with nulls depends on your assumptions about the data and your analytical approach.

Use `locf` if you assume your measurement changes only when you've received new data.

Use `interpolation` if you assume your continuous measurement would have a smooth roughly linear trend if sampled at a higher rate.

Use standard aggregate functions (without locf or interpolation) if your data is not continuous on the time axis. Where there is no data, the result is assumed NULL.

If you want to assume scalar values (typically zero) in place of NULLs, you can use PostgreSQL's coalesce function: `COALESCE(avg(val), 0)`

If you chose to explicitly `ORDER` your results, keep in mind that the gap filling will sort by time in ascending order. Any other explicit ordering may introduce additional sorting steps in the query plan.

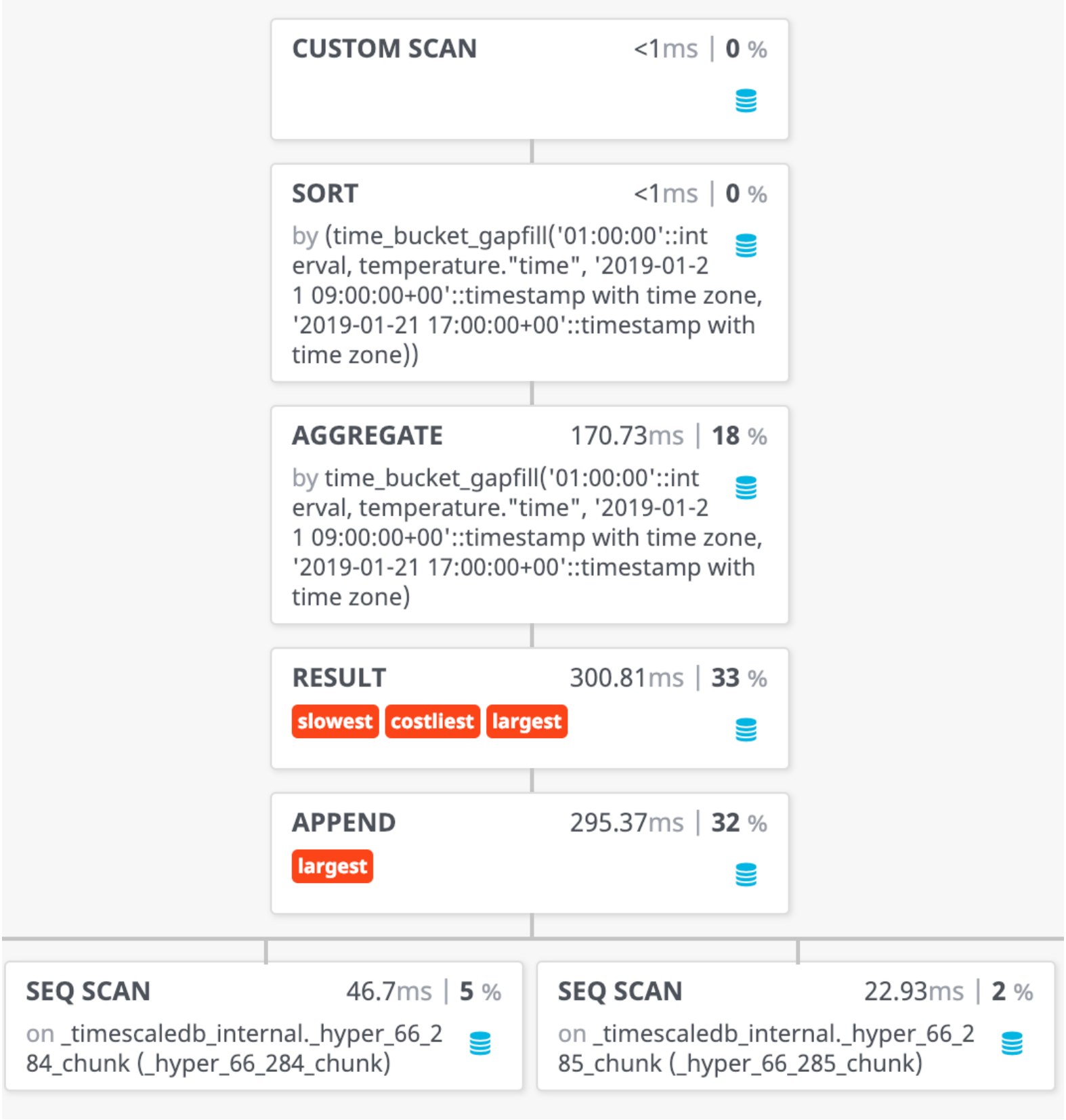
Extending SQL for time series analysis

Astute readers will note that our docs had previously contained some examples of these gap filling techniques using a different query. It used some tricks with generate_series and joins to achieve a similar effect but was verbose, limited in functionality, and challenging to write correctly.

The new `time_bucket_gapfill` query is not only significantly more readable, it's less error prone, more flexible with regards to grouping, and faster to execute.

How does TimescaleDB achieve this? Under the hood, these are not ordinary functions but specially-optimized hooks into

observations. The `locf` and `interpolate` functions are not executed directly but serve as markers so that the gap filling node can track the previous and next known values.



example of how Timescale is extending PostgresSQL for high-performance, general purpose time-series data management.

Next steps

As mentioned above, TimescaleDB 1.2 is still in development, but will be released in the *very* near future. Once released, time buckets with gap filling and the related imputation function will be available as community features under the TSL license. (For more information on the license, read this blog post.)

In the meantime, if you're interested in learning more about gapfilling, check out our docs. And if you are new to TimescaleDB and ready to get started, follow the installation instructions.

We encourage active TimescaleDB users to join our 1900+ member-strong Slack community and post any questions you may have there. Finally, if you are looking for enterprise-grade support and assistance, please let us know.

Interested in learning more? Follow us on Twitter or sign up below to receive more posts like this!

This post was written by
Sven Klemm, Matthew Perry

Cloud

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

19 Jan 2023 • 7 min read

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

Subscribe to the Timescale Newsletter

Your email

Subscribe

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