Handling Time-Series Data in a Relational DBMS: Challenges and Solutions – Abstract

Jan Kristof Nidzwetzki[1]

[1]Timescale, Inc.

Abstract

Storing and analyzing time-series data, such as stock prices or sensor readings, is crucial for many applications, and *database management systems* (DBMS) are often used for this purpose. However, handling time-series data poses challenges for DBMS, as the system needs to continuously be able to insert new data without blocking due to parallel running queries or maintenance operations. Additionally, datasets can become large, and analytical queries that utilize aggregates are common (e.g., obtaining the maximum and average temperature per sensor per hour over the last 24 hours). To meet the requirements of these time-based use cases, specialized DBMS tailored for time-series data have been developed. However, many developers already know how to write queries using the *structured query language* (SQL), and most application architectures already rely on a relational DBMS like PostgreSQL. Moreover, they prefer to use a familiar, well-established, and proven software component rather than introducing new technology. Unfortunately, PostgreSQL does not handle time-series data well.

TimescaleDB is an open-source extension for PostgreSQL that enhances its ability to efficiently handle time-series data. The extension introduces several features to PostgreSQL:

1. Operators for managing time-series data within SQL, including functions like `time_bucket()`, which facilitates grouping and aggregation queries over specific time intervals.
2. Automatic partitioning of large tables. TimescaleDB uses hypertables to partition data. A hypertable consists of multiple PostgreSQL tables called chunks.
3. Optimized query plans, which are designed to handle time-series data effectively. For instance, plan-time and execution-time partition pruning allow it to process only partitions that contain the required data.
4. Transparent and mutable columnar compression to minimize disk space usage, I/O operations, and improve data locality. This also allows aggregation queries to be performed directly on the data in its columnar form.
5. A job framework that enables the implementation of data lifecycle policies directly within PostgreSQL. For example, old data can be removed after a certain period of time. Instead of deleting individual tuples, entire chunks are removed. This is a fast and efficient operation since a complete table is dropped; it also prevents the creation of dead tuples (i.e., deleted or outdated tuples that are still present in a table and need to be physically removed by PostgreSQL during a costly VACUUM operation).
6. Continuous aggregates that pre-compute the results of aggregation queries on time-series data (like a self-updating materialized view). In addition, continuous aggregates can be used to downsample large datasets. For example, storing only the average value per hour can reduce data size while still providing sufficient accuracy for many applications.

In my presentation, I will discuss the core principles of managing time-series data using TimescaleDB, as well as the challenges I have worked on in the past few months.

35th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), May 22-24, 2024, Herdecke, Germany.

jnidzwetzki@gmx.de (J. K. Nidzwetzki)
https://jnidzwetzki.github.io/ (J. K. Nidzwetzki)
0000-0002-2650-8019 (J. K. Nidzwetzki)