TimescaleDB Research

Fadel Alshammasi

July 30, 2019

Overview

TimescaleDB is a new, open-source time-series SQL database providing fast analytics, scalability, with automated data management on a proven storage engine. It is generating a lot of "buzz" on the internet. It looks like PostgreSQL optimized for scale and speed. In fact, it is an extended version of PostgreSQL, which means it could leverage all its features for time-series data such as reliability and ease of use. The purpose of this research is to evaluate the performance of the database. This could be an "all-in-one" solution for small APS (Power Diagnostic Center's in-house software) installations. The performance will be compared separately with another candidate, InfluxDB.

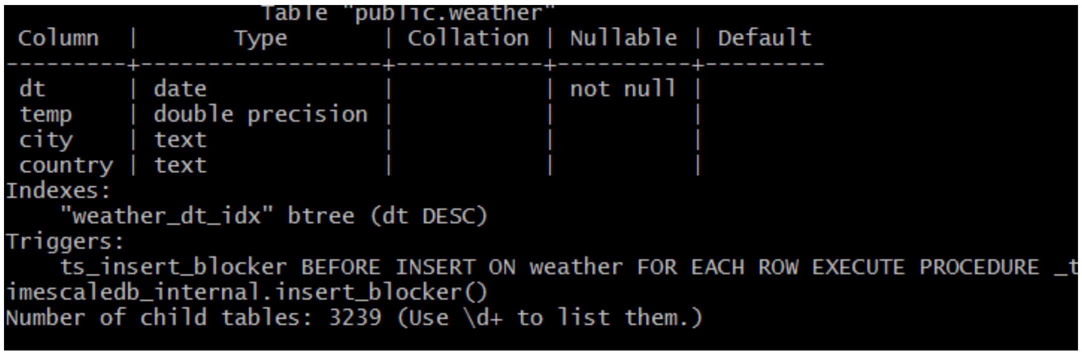
Database Schema Design

The database schema is very simple. A new empty database was created and extended with TimescaleDB. A single standard PostgreSQL table called "weather" was created. The table then was transformed into a hypertable by running the following:

```
SELECT create_hypertable('weather', 'dt');
```

The table contains four columns that identify the date, temperature, city, and country.

- dt is the date for the recorded temperature. It cannot be NULL.
- temp is a double precision floating number used to store the recorded temperature.
- city is a text that stores the name of the city that the temperature was recorded in.
- country is a text the stores the name of the country that the temperature was recorded in.



```
                  Table "public.weather"
 Column  |       Type       | Collation | Nullable | Default
---------+------------------+-----------+----------+---------
 dt      | date             |           | not null |
 temp    | double precision |           |          |
 city    | text             |           |          |
 country | text             |           |          |
Indexes:
    "weather_dt_idx" btree (dt DESC)
Triggers:
    ts_insert_blocker BEFORE INSERT ON weather FOR EACH ROW EXECUTE PROCEDURE _t
imescaledb_internal.insert_blocker()
Number of child tables: 3239 (Use \d+ to list them.)
```

Platform Used (Updated)

The evaluation was performed on a work laptop(macOS Mojave version 10.14.5) that has the following hardware configurations:

- Model: MacBookPro 11.4
- Processor: Intel® Core™ i7
  - Processor Speed: 2.2 GHz
  - # of Processors: 1
  - # of Cores: 4
  - # Threads: 8
  - L2 Cache (per Core): 256 KB
  - L3 Cache: 6 MB
- Installed memory(RAM): 16.0 GB

- System type: 64-bit Operating System

Database Version

The work was done using PostgreSQL 11.4. The version was obtained by running:

```
SELECT version();
```

The version of the TimescaldDB extension is 1.4.0 . It was obtained by simply running \dx (only works in psql). Alternatively, it could also be obtained by running the below pure SQL if psql is not used:

```
 SELECT extname,extversion FROM pg_extension WHERE extname = 'timescal
edb';
```

Method for Batch Insertion (Updated)

A weather data set was used and loaded into the database using timescaledb-parallel-copy command. I successfully ran a Go script to get that command. The weather data contains 1048575 rows. It only took 1 minute and 13 seconds to bulk load the data.

```
timescaledb-parallel-copy --db-name research --table weather --file FadelsData.csv --workers 8
```

This is not the only way to batch insert the data. There are other ways to do it, but this is the fastest way that I found.

Basic Assessment of the Storage Efficiency (Updated)

The shared buffers size is 4 GB. The temp buffers size is 8 MB. The total disk space used by the database is 698 MB for 1048575 samples . I got it by running(I wrote it this way so we get to see the size directly in MB):

```
SELECT pg_size_pretty( pg_database_size(current_database()));
```

The total size of the weather table in the current database is 16 KB. The total size of all indexes(there is only one index, b-tree) attached to the table is 8192 bytes. The toast table size is also 8192 bytes. They were obtained using similar queries as the above. Alternatively, the database sizes\table sizes could simply be obtained by running \l+ and \d+ (these two only work in psql). The following SQL script gives a comprehensive view of the size information for all tables**:**

```
SELECT *, pg_size_pretty(total_bytes) AS total
    , pg_size_pretty(index_bytes) AS INDEX
    , pg_size_pretty(toast_bytes) AS toast
    , pg_size_pretty(table_bytes) AS TABLE
  FROM (
  SELECT *, total_bytes-index_bytes-COALESCE(toast_bytes,0) AS table_bytes FR
OM (
      SELECT c.oid,nspname AS table_schema, relname AS TABLE_NAME
              , c.reltuples AS row_estimate
              , pg_total_relation_size(c.oid) AS total_bytes
              , pg_indexes_size(c.oid) AS index_bytes
              , pg_total_relation_size(reltoastrelid) AS toast_bytes
          FROM pg_class c
          LEFT JOIN pg_namespace n ON n.oid = c.relnamespace
          WHERE relkind = 'r'
  ) a
) a;
```

Example SQL Queries/Timing Metrics (Updated)

Here are some standard example queries. Each query was run 5 times and timing metrics were collected:

List all the dates, temperatures, and cities when the samples were collected in the United States. The result has an ascending order with respect to the temperature.

```
SELECT dt, temp, city FROM weather WHERE country='United States' ORDER
 BY temp;
```

Run ID	Time
1	1 s 52 ms
2	1 s 53 ms
3	1 s 40 ms
4	1 s 41 ms
5	1 s 28 ms
Average	1 s 428 ms

Return the lowest temperature value recorded in the entire dataset.

```
SELECT MIN(temp) FROM weather;
```

Run ID	Time
1	1 s 84 ms
2	1 s 50 ms
3	1 s 81 ms
4	1 s 69 ms
5	1 s 78 ms
Average	1 s 724 ms

Return the highest temperature value recorded in each city in Germany.

```
SELECT MAX(temp), city FROM weather where country='Germany' GROUP BY
city;
```

Run ID	Time
1	1 s 26 ms
2	1 s 23 ms
3	1 s 24 ms
4	1 s 26 ms
5	1 s 16 ms
Average	1 s 23 ms

Return the latest (based on the date) recorded temperature value for each city.

```
SELECT city, last(temp, dt) FROM weather

WHERE temp IS NOT NULL

GROUP BY city;
```

Run ID	Time
1	1 s 36 ms
2	1 s 25 ms
3	1 s 30 ms
4	1 s 30 ms
5	1 s 34 ms
Average	1 s 31 ms

Return all the values of a given set of weather data for a specified range of dates and a specified range of temperatures.

```
SELECT dt, temp, city, country FROM weather
WHERE dt BETWEEN '1900/6/1' AND '1900/8/31' AND
temp < 7
ORDER BY temp;
```

Run ID	Time
1	Instant
2	Instant
3	Instant
4	Instant
5	Instant
Average	Instant

Critical Performance/Operation Deficiencies (Updated)

The database generally is expected to be superior than PostgreSQL in terms of speed performance & storage efficiency. I found it to be much better in terms of memory. When it comes to speed, the batch loading method that I used only took about a minute which is super fast. The reason is because the parallel copy(unique for TimescaleDB) method performs multiple COPYs concurrently (depending on the number of workers).

However, the number of workers should not exceed the total number of CPU cores on the machine. An area of interest would be to use Timescale cloud as strongly recommend by the Timescale team that I spoke to. It's something that I haven't explored since it is not an open – source platform.