TimescaleDB

Create a hypertable

TimescaleDB's hypercore is a hybrid row-columnar store that boosts analytical query performance on your time-series and event data, while reducing data size by more than 90%. This keeps your analytics operating at lightning speed and ensures low storage costs as you scale. Data is inserted in row format in the rowstore and converted to columnar format in the columnstore based on your configuration.

```
-- Create a hypertable, with the columnstore from the hypercore engine
-- "time" as partitioning column and a segment by on location
CREATE TABLE conditions (
  time        TIMESTAMPTZ       NOT NULL,
  location    TEXT              NOT NULL,
  temperature DOUBLE PRECISION  NULL,
  humidity    DOUBLE PRECISION  NULL
)
WITH (
  timescaledb.hypertable,
  timescaledb.partition_column='time',
  timescaledb.segmentby='location'
);
```

See more:

- About hypertables
- API reference
- About columnstore
- Enable columnstore manually
- API reference

Insert and query data

Insert and query data in a hypertable via regular SQL commands. For example:

- Insert data into a hypertable named `conditions`:

  ```
  INSERT INTO conditions
    VALUES
      (NOW(), 'office',   70.0, 50.0),
      (NOW(), 'basement', 66.5, 60.0),
      (NOW(), 'garage',   77.0, 65.2);
  ```

- Return the number of entries written to the table conditions in the last 12 hours:

  ```
  SELECT
    COUNT(*)
  ```

```
FROM
  conditions
WHERE
  time > NOW() - INTERVAL '12 hours';
```

See more:

- Query data
- Write data

Create time buckets

Time buckets enable you to aggregate data in hypertables by time interval and calculate summary values.

For example, calculate the average daily temperature in a table named `conditions`. The table has a `time` and `temperature` columns:

```
SELECT
  time_bucket('1 day', time) AS bucket,
  AVG(temperature) AS avg_temp
FROM
  conditions
GROUP BY
  bucket
ORDER BY
  bucket ASC;
```

See more:

- About time buckets
- API reference
- All TimescaleDB features
- Tutorials

Create continuous aggregates

Continuous aggregates make real-time analytics run faster on very large datasets. They continuously and incrementally refresh a query in the background, so that when you run such query, only the data that has changed needs to be computed, not the entire dataset. This is what makes them different from regular PostgreSQL materialized views, which cannot be incrementally materialized and have to be rebuilt from scratch every time you want to refresh it.

For example, create a continuous aggregate view for daily weather data in two simple steps:

1. Create a materialized view:

```
CREATE MATERIALIZED VIEW conditions_summary_daily
WITH (timescaledb.continuous) AS
SELECT
  location,
  time_bucket(INTERVAL '1 day', time) AS bucket,
  AVG(temperature),
  MAX(temperature),
  MIN(temperature)
FROM
  conditions
GROUP BY
  location,
  bucket;
```

2. Create a policy to refresh the view every hour:

```
SELECT
  add_continuous_aggregate_policy(
    'conditions_summary_daily',
    start_offset => INTERVAL '1 month',
    end_offset => INTERVAL '1 day',
    schedule_interval => INTERVAL '1 hour'
);
```

See more:

- About continuous aggregates
- API reference

Documentation associated with Apache licensed "community edition" of TimescaleDB provided for convenience and built from Apache licensed source code available here.