Point-in-time query tuning and observability with pg_stat_statements

Ryan Booz (@ryanbooz)

Director, Developer Advocacy

April 2022

Agenda

01 pg_stat_statements primer

02 How to track historical data

03 Demo

04 Other alternatives

pg_stat_statements primer

What is pg_stat_statements?

- An extension included with PostgreSQL 8.4+
- It is part of the ***contrib*** module but **not enabled** by default
    - Must be loaded via ‘shared_preload_libraries’ in postgresql.conf
- Tracks aggregated statistics of all queries in the cluster
- Installing the extension in the database creates the necessary views to query the data

How does it store aggregates?

- Every dbid, userid, and queryid
- Stats are grouped based on query structure and final ID as determined by an internal hash calculation

How does it identify queries?

```
SELECT id, name FROM table1 WHERE id = 1000;
```

```
SELECT id, name FROM table1 WHERE id = $1;
```

```
SELECT id, name FROM table1 WHERE id IN
 (1000,2000,3000);
```

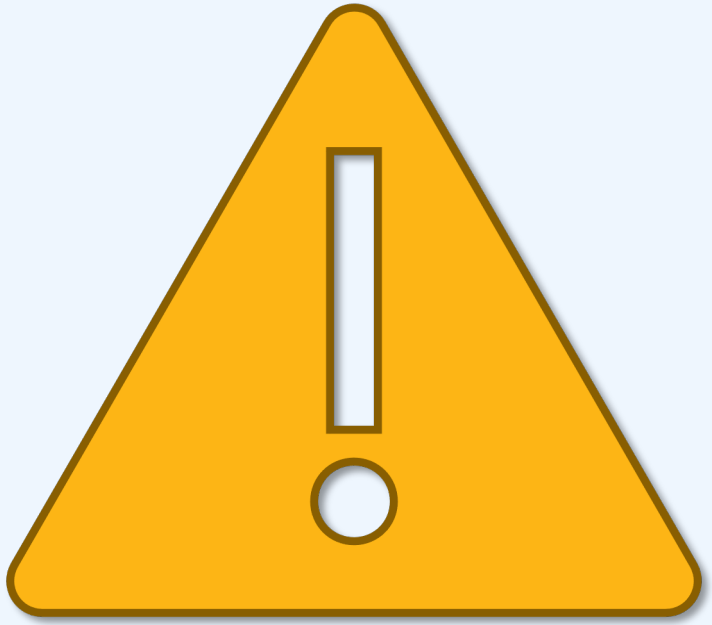
```
SELECT id, name FROM table1 WHERE id IN
 ($1,$2,$3);
```

pg_stat_statement statistics

- Execution Time (total/min/max/mean/stddev)
- Planning Time (total/min/max/mean/stddev)
- Calls (total)
- Rows (total)
- Buffers (shared/local/temp)
    - read/hit/dirtied/written
    - read/write time
- WAL

31 Columns of data
(as of PG14)

```
Name                |Value
--------------------+-----------------------
userid              |16422
dbid                |16434
queryid             |-6155333619461995114
query               |SELECT id, name FROM...
plans               |0
total_plan_time     |0.0
min_plan_time       |0.0
max_plan_time       |0.0
mean_plan_time      |0.0
stddev_plan_time    |0.0
calls               |151
total_exec_time     |8.489053
min_exec_time       |0.013751
max_exec_time       |1.356096
mean_exec_time      |0.056218894039735096
stddev_exec_time    |0.11851139585068957
rows                |151
shared_blks_hit     |450
shared_blks_read    |3
```



All statistics are cumulative from the last restart*

*or reset by a superuser

Caveats

- **PostgreSQL 13**
  - modified column names to include planning statistics
- **PostgreSQL 14**
  - Must set “compute_query_id”=true in postgresql.conf
  - Includes informational view for allocation and “last reset” information

02

Track historical data

(1) Create snapshot table(s)

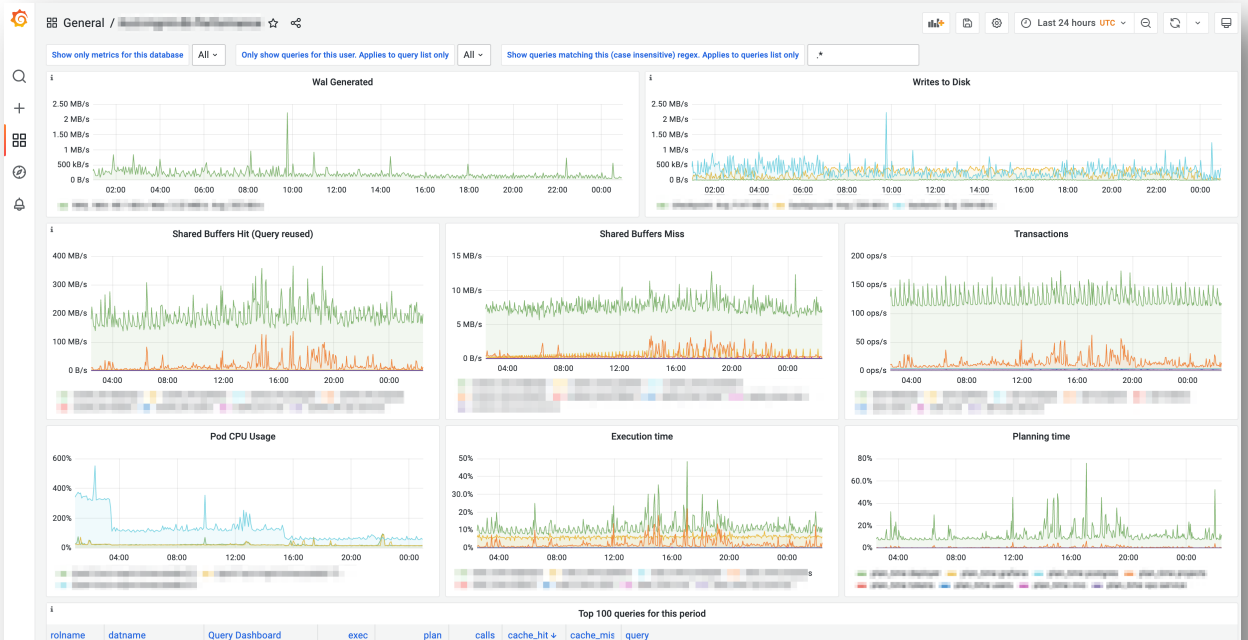
- Store all or partial statistics
- Create additional columns from joined tables (ie. `pg_roles` or `pg_database`)
- Separate database can be helpful to easily filter out monitoring queries
- Partitioning highly recommended
    - TimescaleDB Hypertable provides automatic time partitioning and 90%+ compression of `pg_stat_statements` data

(2) Automate procedure to store statistics

- Create a stored procedure to query the view and store relevant data
- Potentially customized per database
- Automate the request through multiple tools or extensions
    - TimescaleDB User Defined Actions
    - pg_cron
    - pg_timetable

(3) Create VIEW to query snapshot differentials

- Monitor recent data through dashboards
- Show query specific values for a specific interval of time



03

Demo

04

Alternative solutions

Alternative Solutions

- **pg_stat_monitor**
  - Percona open source extention
  - Captures actual parameters
  - Query plans
  - Table access statistics
- **pganalyze**
  - Paid service to collect and analyze `pg_stat_statement` data
  - Additional log analysis tools as well

Thank you!

What questions do you have?