PostgreSQL extension pg_stat_monitor

2024 p2d2.cz
Aleš Zelený

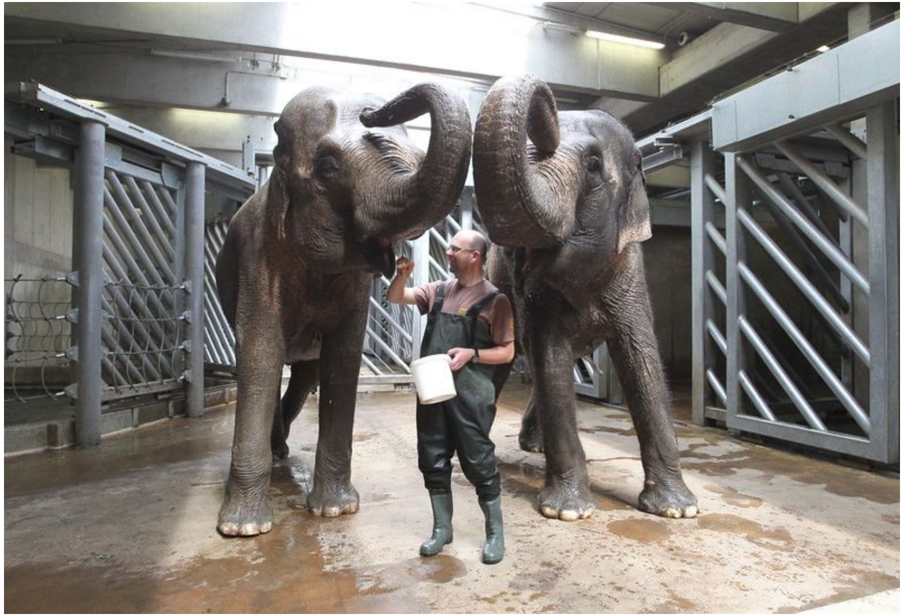
Who’s me?

InterBase / Firebird app developer, DBA (3 years)

Oracle DBA (17 years)

PostgreSQL DBA (since 2010)

Elephants enthusiast ...



Agenda

Postgres system statistics

pg_stat_statements extension

pg_stat_monitor extension

The Cumulative Statistics System

Dynamic Statistics Views

- **`pg_stat_activity`**, **`pg_stat_progress_vacuum`**... 13 views as of Pg16

Collected Statistics Views

- **`pg_stat_archiver`**, **`pg_stat_database`**, 29 views as of Pg16
    - **`stats_reset`** `timestamp with time zone`, Time at which these statistics were last reset

Additional Supplied Modules and Extensions

- **`pg_buffercache`** - real time
- **`pg_stat_statements`** - cumulative

pg_stat_statemets

- The extension track statistics of SQL planning and execution, requires adding module **`pg_stat_statements`** to **`shared_preload_libraries`**.
- Cumulative per SQL **statement**, **user** and **database**
- Never (almost) use `SELECT * FROM public.pg_stat_statements;`
- **`pg_stat_statements_info`** view
    - **`stats_reset`**
    - **`dealloc`** (`pg_stat_statements.max` was reached, least used query was removed)
- **`pg_stat_statements_reset()`** function

select pg_sleep($1) example statistics

```
SELECT pss.userid::regrole, pd.datname, calls,
min_exec_time, mean_exec_time, max_exec_time
FROM public.pg_stat_statements pss
INNER JOIN pg_catalog.pg_database pd
     ON pss.dbid = pd.oid
WHERE userid in ('u1'::regrole, 'u2'::regrole) AND queryid = 5457019535816659310
ORDER BY 1, 2;
```

```
 userid | datname | calls | min_exec_time | mean_exec_time | max_exec_time
--------+---------+-------+---------------+----------------+---------------
 u1     | db1     |     5 |   1001.066225 |   1001.0900728 |   1001.153728
 u1     | db2     |     5 |   3000.242074 |    3001.531193 |   3003.096321
 u2     | db1     |     3 |   2001.809245 | 2002.146666665 | 2002.31823199
 u2     | db2     |     3 | 4000.31013400 |    4002.307968 | 4003.88756600
(4 rows)
```

Various metrics are available: **`total_exec_time`**, **`stddev_exec_time`**, **`rows`** …

select pg_stat_statements_reset();

```
SELECT pss.userid::regrole, pd.datname, calls,
min_exec_time, mean_exec_time, max_exec_time
FROM public.pg_stat_statements pss
INNER JOIN pg_catalog.pg_database pd
     ON pss.dbid = pd.oid
WHERE userid in ('u1'::regrole, 'u2'::regrole) AND queryid = 5457019535816659310
ORDER BY 1, 2;

 userid | datname | calls | min_exec_time | mean_exec_time | max_exec_time
--------+---------+-------+---------------+----------------+---------------
(0 rows)
```

Cumulative statistics

- no calls of `pg_stat_statements_reset()`
  - easy to build long term storage by regular samples of the view
  - window functions to retrieve per interval statistics
  - easy to dilute samples by simply deleting a sample(s)
- regular calls of `pg_stat_statements_reset()`
  - easy to read statistics for period between restarts
  - more effort is needed to build/present long term statistics over couple of periods
  - more effort is needed to dilute long term statistics (eg daily resets → weekly stats)

Postgres 17 feature

```
SELECT pss.userid::regrole AS us, pd.datname AS db, calls AS c,
min_exec_time, mean_exec_time, max_exec_time, minmax_stats_since
FROM public.pg_stat_statements pss
INNER JOIN pg_catalog.pg_database pd
     ON pss.dbid = pd.oid
WHERE userid in ('u1'::regrole, 'u2'::regrole) AND queryid = 5457019535816659310
ORDER BY 1, 2;

 us | db  | c | min_exec_time | mean_exec_time | max_exec_time |      minmax_stats_since
----+-----+---+---------------+----------------+---------------+-------------------------------
 u1 | db1 | 5 |   1001.066225 |   1001.0900728 |   1001.153728 | 2024-05-05 23:26:19.057932+02
 u1 | db2 | 5 |   3000.242074 |    3001.531193 |   3003.096321 | 2024-05-05 23:28:42.938995+02
 u2 | db1 | 3 |   2001.809245 | 2002.146666665 | 2002.31823199 | 2024-05-05 23:26:38.717792+02
 u2 | db2 | 3 | 4000.31013400 |    4002.307968 | 4003.88756600 | 2024-05-05 23:29:21.18137+02
(4 rows)
```

pg_stat_statements_reset(..., minmax_only => true);

```
SELECT pss.userid::regrole AS us, pd.datname AS db, calls AS c,
min_exec_time, mean_exec_time, max_exec_time, minmax_stats_since
FROM public.pg_stat_statements pss
INNER JOIN pg_catalog.pg_database pd
     ON pss.dbid = pd.oid
WHERE userid in ('u1'::regrole, 'u2'::regrole) AND queryid = 5457019535816659310
ORDER BY 1, 2;

 us | db  | c | min_exec_time | mean_exec_time | max_exec_time |   minmax_stats_since
----+-----+---+---------------+----------------+---------------+------------------------------
 u1 | db1 | 5 |             0 |   1001.0900728 |             0 | 2024-05-05 23:30:45.380677+02
 u1 | db2 | 5 |             0 |    3001.531193 |             0 | 2024-05-05 23:30:45.380677+02
 u2 | db1 | 3 |   2001.809245 | 2002.141666665 | 2002.31823199 | 2024-05-05 23:26:38.717792+02
 u2 | db2 | 3 | 4000.31013400 |    4002.307968 | 4003.88756600 | 2024-05-05 23:29:21.18137+02
(4 rows)
```

mean exec time

```
SELECT pss.userid::regrole, pd.datname, calls,
min_exec_time, mean_exec_time, max_exec_time, stddev_exec_time, minmax_stats_since
FROM public.pg_stat_statements pss
INNER JOIN pg_catalog.pg_database pd
     ON pss.dbid = pd.oid
WHERE userid = 'u1'::regrole AND queryid = -2749826580604107531 ORDER BY 1, 2;
-[ RECORD 1 ]------+------------------------------
userid             | u1
datname            | db1
calls              | 11901
min_exec_time      | 1.016564
mean_exec_time     | 1.0910678040500839
max_exec_time      | 1.6630040000000001
stddev_exec_time   | 0.02924890373742974
minmax_stats_since | 2024-05-06 00:13:58.142156+02
```

mean is influenced by outliers

```
SELECT pss.userid::regrole, pd.datname, calls,
min_exec_time, mean_exec_time, max_exec_time, stddev_exec_time, minmax_stats_since
FROM public.pg_stat_statements pss
INNER JOIN pg_catalog.pg_database pd
     ON pss.dbid = pd.oid
WHERE userid = 'u1'::regrole AND queryid = -2749826580604107531 ORDER BY 1, 2;
-[ RECORD 1 ]------+------------------------------
userid             | u1
datname            | db1
calls              | 11904
min_exec_time      | 1.016564
mean_exec_time     | 3.6384116218918052
max_exec_time      | 10110.395931
stddev_exec_time   | 160.4422273401458
minmax_stats_since | 2024-05-06 00:13:58.142156+02
```

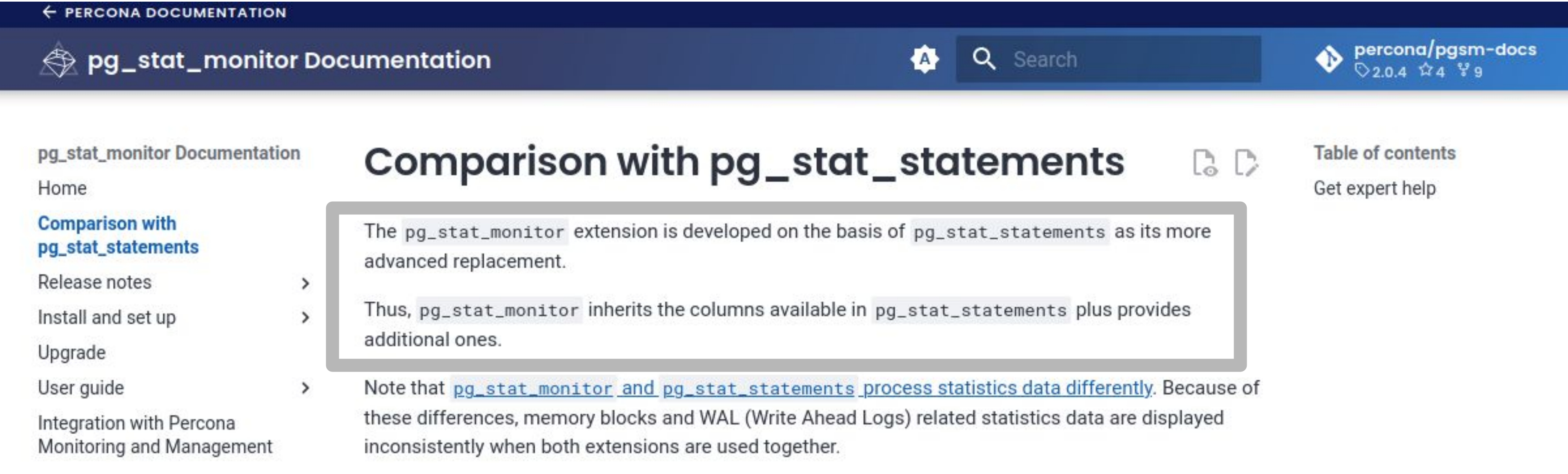
test query duration distribution

```
10 000x pg_sleep(0.001)
10 000x pg_sleep(0.003)
     1x pg_sleep(4.1)
     1x pg_sleep(5.1)
```

```
-[ RECORD 1 ]------+------------------------------
userid             | u1
datname            | db1
calls              | 20002
min_exec_time      | 1.07
mean_exec_time     | 3.59
max_exec_time      | 5103.94
stddev_exec_time   | 46.2772
minmax_stats_since | 2024-05-12 14:27:17.952729+02
```

pg_stat_monitor extension

Developed by Percona



← PERCONA DOCUMENTATION
pg_stat_monitor Documentation
Search
percona/pgsm-docs
2.0.4 ☆4 9
pg_stat_monitor Documentation
Home
Comparison with pg_stat_statements
Release notes
Install and set up
Upgrade
User guide
Integration with Percona Monitoring and Management
Table of contents
Get expert help

Comparison with pg_stat_statements

The `pg_stat_monitor` extension is developed on the basis of `pg_stat_statements` as its more advanced replacement.

Thus, `pg_stat_monitor` inherits the columns available in `pg_stat_statements` plus provides additional ones.

Note that `pg_stat_monitor` and `pg_stat_statements` process statistics data differently. Because of these differences, memory blocks and WAL (Write Ahead Logs) related statistics data are displayed inconsistently when both extensions are used together.

Comparison with pg_stat_statements

- pg_stat_statementes
  - 5 configuration parameters
  - data are stored on disk
- pg_stat_monitor
  - 17 configuration parameters
  - data are stored in shared memory - does not persist instance restart
  - `pg_stat_monitor.pgsm_max` (def. 256MB)
  - `pg_stat_monitor.pgsm_query_shared_buffer` (def. 20MB)
  - `pg_stat_monitor.pgsm_enable_overflow` (def. on, can growth to swap)

Comparison with pg_stat_statements

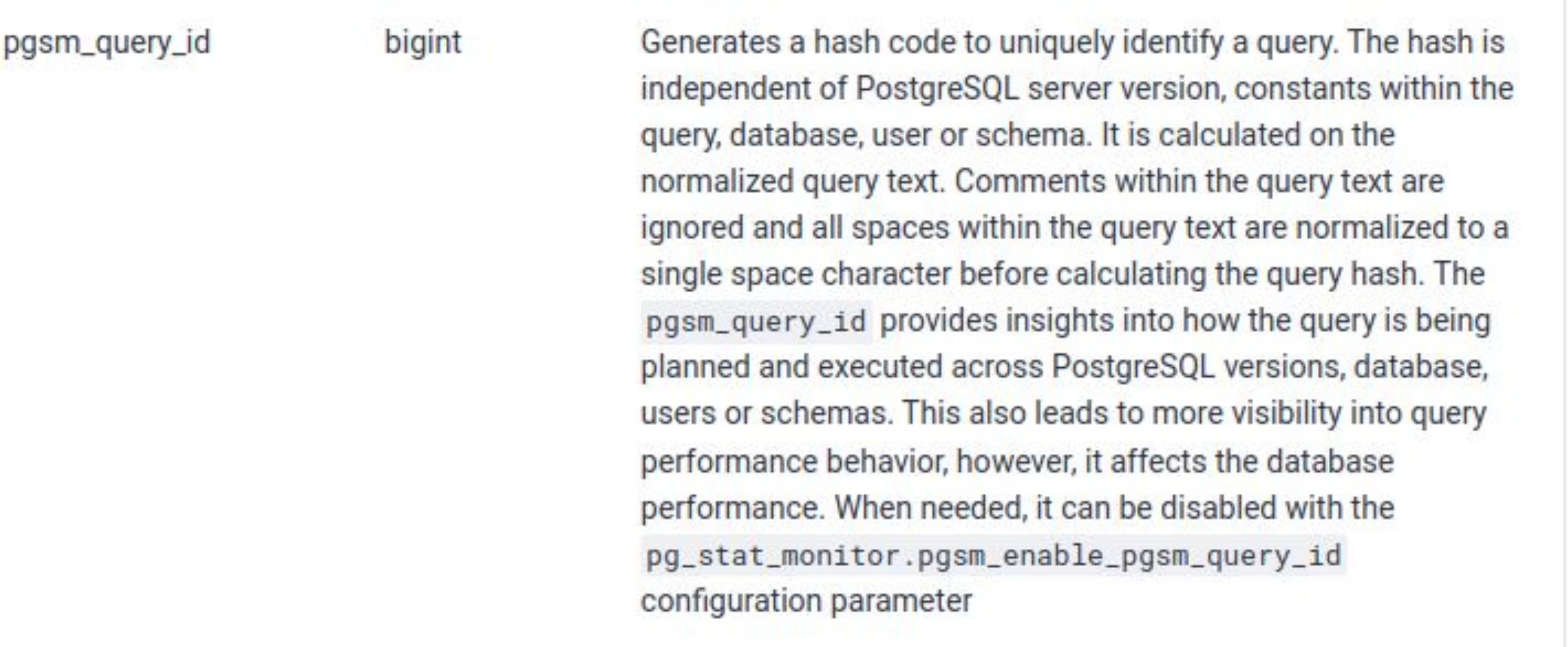
- Data are organized in a **buckets** forming cyclic buffer
  - `pg_stat_monitor.pgsm_max_buckets`
  - `pg_stat_monitor.pgsm_bucket_time`
- **Cumulative counters and aggregates are calculated within a bucket**
- Additional columns for better granularity
  - `username` no need to join/cast `userid` to `regrole`
  - `client_ip`
- `username`, `datname`, `client_ip`, `toplevel` and `queryid` forms uniqueness within a bucket (and `planid`, if enabled, potentially more columns… tests needed)

top_queryid, top_query

```
-[ RECORD 1 ]-+-------------------------
username      | postgres
datname       | db1
client_ip     | 127.0.0.1
pgsm_query_id | -3748516134953279691
queryid       | -9180915408172212016
toplevel      | f
top_queryid   | 1400612139329494583
query         | SELECT a + b
top_query     | select * from f11(2,3);
-[ RECORD 2 ]-+-------------------------
username      | postgres
datname       | db1
client_ip     | 127.0.0.1
pgsm_query_id | 4351971043219176
queryid       | 1400612139329494583
toplevel      | t
top_queryid   |
query         | select * from f11(2,3)
top_query     |
```

```
-[ RECORD 3 ]-+-------------------------
username      | u1
datname       | db1
client_ip     | 127.0.0.1
pgsm_query_id | -3748516134953279691
queryid       | -9180915408172212016
toplevel      | f
top_queryid   | 1400612139329494583
query         | SELECT a + b
top_query     | select * from f11(2,3);
-[ RECORD 4 ]-+-------------------------
username      | u1
datname       | db1
client_ip     | 127.0.0.1
pgsm_query_id | 4351971043219176
queryid       | 1400612139329494583
toplevel      | t
top_queryid   |
query         | select * from f11(2,3)
top_query     |
```

pgsm_query_id



pgsm_query_id	bigint	Generates a hash code to uniquely identify a query. The hash is independent of PostgreSQL server version, constants within the query, database, user or schema. It is calculated on the normalized query text. Comments within the query text are ignored and all spaces within the query text are normalized to a single space character before calculating the query hash. The `pgsm_query_id` provides insights into how the query is being planned and executed across PostgreSQL versions, database, users or schemas. This also leads to more visibility into query performance behavior, however, it affects the database performance. When needed, it can be disabled with the `pg_stat_monitor.pgsm_enable_pgsm_query_id` configuration parameter

pgsm_query_id

- pg_stat_statements
  - the queryid hash value is computed on the post-parse-analysis representation
- pgsm_query_id
  - The hash is independent of PostgreSQL server version, constants within the query, database, user or schema. **It is calculated on the normalized query text**.

```
postgres=# select queryid, pgsm_query_id, query
from pg_stat_monitor
where queryid in (-6701154771960778310, 877146937154600262);
       queryid        |    pgsm_query_id    |           query
----------------------+---------------------+-------------------------
   877146937154600262 | 5348698571143121017 | … pg_sleep($1) where…
 -6701154771960778310 | 5348698571143121017 | … pg_sleep($1) where…
(2 rows)
```

pg_stat_monitor.pgsm_normalized_query

```
postgres=# select queryid, pgsm_query_id, query from pg_stat_monitor
where queryid in (-6701154771960778310, 877146937154600262);

    queryid             | pgsm_query_id        |   query
----------------------+----------------------+-------------------------
   877146937154600262 | 5348698571143121017 | select pg_sleep($1) …
 -6701154771960778310 | 5348698571143121017 | select pg_sleep($1) …
   877146937154600262 | 5348698571143121017 | select pg_sleep(1) …
 -6701154771960778310 | 5348698571143121017 | select pg_sleep(1.0)...
(4 rows)
```

planid, query_plan

```
select query, planid, query_plan, application_name from pg_stat_monitor where queryid = -2311750621490427657;

-[ RECORD 1 ]----+-----------------------------------------------------------
query            | SELECT abalance FROM pgbench_accounts WHERE aid = $1;
planid           | -4461530684938051170
query_plan       | Gather                                                  +
                 |   Workers Planned: 2                                    +
                 |   ->  Parallel Seq Scan on pgbench_accounts             +
                 |       Filter: (aid = 344887)
application_name | pgbench
-[ RECORD 2 ]----+-----------------------------------------------------------
query            | SELECT abalance FROM pgbench_accounts WHERE aid = $1;
planid           | 6999392803091472102
query_plan       | Index Scan using pgbench_accounts_pkey on pgbench_accounts+
                 |   Index Cond: (aid = $1)
application_name | pgbench
-[ RECORD 3 ]----+-----------------------------------------------------------
query            | SELECT abalance FROM pgbench_accounts WHERE aid = $1;
planid           | -6715202301951857655
query_plan       | Gather                                                  +
                 |   Workers Planned: 2                                    +
                 |   ->  Parallel Seq Scan on pgbench_accounts             +
                 |       Filter: (aid = 5635518)
application_name | pgbench
```

elevel, sqlcode, message

```
postgres=# select query, elevel, sqlcode, message from
pg_stat_monitor where elevel > 0;
    query     | elevel | sqlcode |          message
--------------+--------+---------+-------------------------------
 select 1/0;  |     21 | 22012   | division by zero
 select cosi; |     21 | 42703   | column "cosi" does not exist
(2 rows)
```

Buckes & histograms

```
SSELECT * FROM histogram(3, '-7077676528008709838') AS a(range TEXT, freq INT, bar TEXT);
```

```
{{0.000 - 1.000}         |     0 |
 (1.000 - 2.778}         |     0 |
 (2.778 - 4.162}         |     0 |
 (4.162 - 6.623}         |     0 |
 (6.623 - 11.000}        |     0 |
 (11.000 - 18.783}       |     1 | ■
 (18.783 - 32.623}       |     2 | ■
 (32.623 - 57.234}       |     1 | ■
 (57.234 - 101.000}      |     6 | ■■■■
 (101.000 - 178.827}     |     7 | ■■■■■
 (178.827 - 317.226}     |    13 | ■■■■■■■■■
 (317.226 - 563.338}     |    26 | ■■■■■■■■■■■■■■■■■■
 (563.338 - 1000.994}    |    44 | ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
 (1000.994 - 1779.268}   |     0 |
 (1779.268 - 3163.256}   |     0 |
 (3163.256 - 5624.371}   |     0 |
 …
 (56234.598 - 100000.000} |    0 |
 (100000.000 - ...}}     |     0 |
```

Buckes & histograms

```
SELECT
        psm.bucket, psm.bucket_start_time, a.*, psm.pgsm_query_id, psm.queryid,
        (regexp_matches(query, '((\d+\.)?\d+) as sleep_sec'))[1] AS sleep_sec
FROM pg_stat_monitor psm, histogram(psm.bucket::int, psm.queryid) AS a(range TEXT, freq INT, bar TEXT)
WHERE psm.queryid = -229239678829605112 AND freq > 0
ORDER BY psm.bucket_start_time;
 bucket |   bucket_start_time    |       range              | freq |              bar               |      sleep_sec
--------+------------------------+--------------------------+------+--------------------------------+-------------------
       9 | 2024-05-12 19:29:00+02 |  (6.623 - 11.000}        |    1 | ■                                                   | 0.009
       9 | 2024-05-12 19:29:00+02 |  (32.623 - 57.234}       |    2 | ■                                                   | 0.009
       9 | 2024-05-12 19:29:00+02 |  (57.234 - 101.000}      |    4 | ■■■                                                | 0.009
       9 | 2024-05-12 19:29:00+02 |  (101.000 - 178.827}     |    8 | ■■■■■                                            | 0.009
       9 | 2024-05-12 19:29:00+02 |  (178.827 - 317.226}     |   15 | ■■■■■■■■■■                                   | 0.009
       9 | 2024-05-12 19:29:00+02 |  (317.226 - 563.338}     |   25 | ■■■■■■■■■■■■■■■■■                        | 0.009
       9 | 2024-05-12 19:29:00+02 |  (563.338 - 1000.994}    |   45 | ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■  | 0.009
       2 | 2024-05-12 19:32:00+02 |  (6.623 - 11.000}        |    1 | ■                                                    | 0.009
       2 | 2024-05-12 19:32:00+02 |  (11.000 - 18.783}       |    1 | ■                                                    | 0.009
       2 | 2024-05-12 19:32:00+02 |  (32.623 - 57.234}       |    3 | ■■                                                  | 0.009
       2 | 2024-05-12 19:32:00+02 |  (57.234 - 101.000}      |    3 | ■■                                                  | 0.009
       2 | 2024-05-12 19:32:00+02 |  (101.000 - 178.827}     |    6 | ■■■■                                          | 0.009
       2 | 2024-05-12 19:32:00+02 |  (178.827 - 317.226}     |    7 | ■■■■                                                 | 0.009
       2 | 2024-05-12 19:32:00+02 |  (317.226 - 563.338}     |   30 | ■■■■■■■■■■■■■■■■■■                   | 0.009
       2 | 2024-05-12 19:32:00+02 |  (563.338 - 1000.994}    |   49 | ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■  | 0.009
       3 | 2024-05-12 19:33:00+02 |  (17783.643 - 31623.492} |    2 |  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ | 22.7722
```

the extension objects

```
postgres=# \dx+ pg_stat_monitor
   Objects in extension "pg_stat_monitor"
          Object description
--------------------------------------------
 function decode_error_level(integer)
 function get_cmd_type(integer)
 function get_histogram_timings()
 function histogram(integer,bigint)
 function pgsm_create_11_view()
 function pgsm_create_13_view()
 function pgsm_create_14_view()
 function pgsm_create_15_view()
 function pgsm_create_view()
 function pg_stat_monitor_internal(boolean)
 function pg_stat_monitor_reset()
 function pg_stat_monitor_version()
 function range()
 view pg_stat_monitor
(14 rows)
```

returns time ranges based on histogram min/max parameters

highlights

- buckets & histograms
- top_query_id
- query_plan
- pgsm_query_id cross version persistency
- comments extraction - sqlcommenter by google
- relations (involved in query processing)

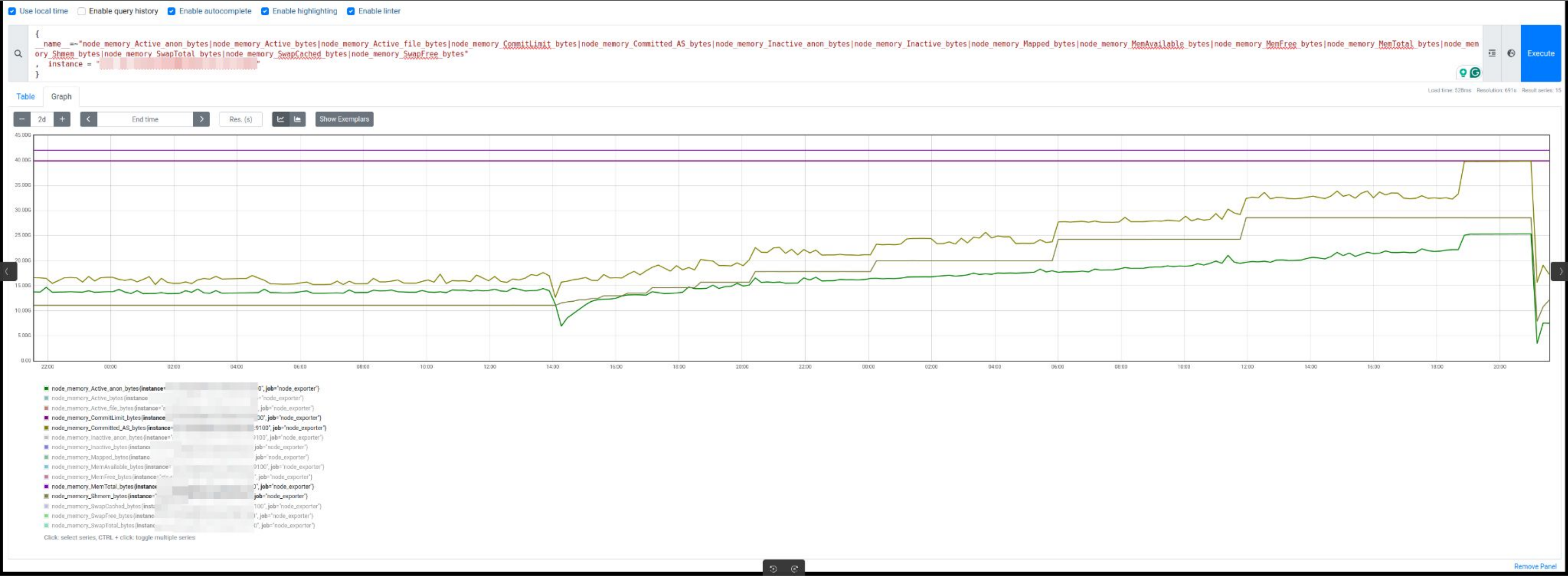
penumbra

Some of the extension function names are too generic for default public schema…

```
postgres=# create extension pg_stat_monitor schema psm;
CREATE EXTENSION
postgres=# \dx
postgres=# \dx+ pg_stat_monitor
     Objects in extension "pg_stat_monitor"
               Object description
------------------------------------------------
 function psm.decode_error_level(integer)
 function psm.get_cmd_type(integer)
 function psm.get_histogram_timings()
 function psm.histogram(integer,bigint)
 …
 function psm.range()
 view psm.pg_stat_monitor
```

shadows...

Committed memory growth up to OS commit limit...



```
{
  __name__=~"node_memory_Active_anon_bytes|node_memory_Active_bytes|node_memory_Active_file_bytes|node_memory_CommitLimit_bytes|node_memory_Committed_AS_bytes|node_memory_Inactive_anon_bytes|node_memory_Inactive_bytes|node_memory_Mapped_bytes|node_memory_MemAvailable_bytes|node_memory_MemFree_bytes|node_memory_MemTotal_bytes|node_memory_Shmem_bytes|node_memory_SwapTotal_bytes|node_memory_SwapCached_bytes|node_memory_SwapFree_bytes"
,  instance = "[redacted]"
}
```

- 🟩 node_memory_Active_anon_bytes{instance=[redacted]0", job="node_exporter"}
- ■ node_memory_Active_bytes{instance[redacted]="node_exporter"}
- ■ node_memory_Active_file_bytes{instance="s[redacted], job="node_exporter"}
- ■ node_memory_CommitLimit_bytes{instance[redacted]00", job="node_exporter"}
- ■ node_memory_Committed_AS_bytes{instance=[redacted]:9100", job="node_exporter"}
- ■ node_memory_Inactive_anon_bytes{instance="[redacted]9100", job="node_exporter"}
- ■ node_memory_Inactive_bytes{instanc[redacted]job="node_exporter"}
- ■ node_memory_Mapped_bytes{instanc[redacted]job="node_exporter"}
- ■ node_memory_MemAvailable_bytes{instance=[redacted]9100", job="node_exporter"}
- ■ node_memory_MemFree_bytes{instance="[redacted]", job="node_exporter"}
- ■ node_memory_MemTotal_bytes{instanc[redacted]0", job="node_exporter"}
- ■ node_memory_Shmem_bytes{instance="[redacted]job="node_exporter"}
- ■ node_memory_SwapCached_bytes{inst[redacted]100", job="node_exporter"}
- ■ node_memory_SwapFree_bytes{instanc[redacted]", job="node_exporter"}
- ■ node_memory_SwapTotal_bytes{instanc[redacted]0", job="node_exporter"}

Click: select series, CTRL + click: toggle multiple series

(un)expected EOF on client connection

Image by ekrem from Pixabay