Meetup PostgreSQL Lille

Que faire de pg_stat_monitor ?

DALIBO

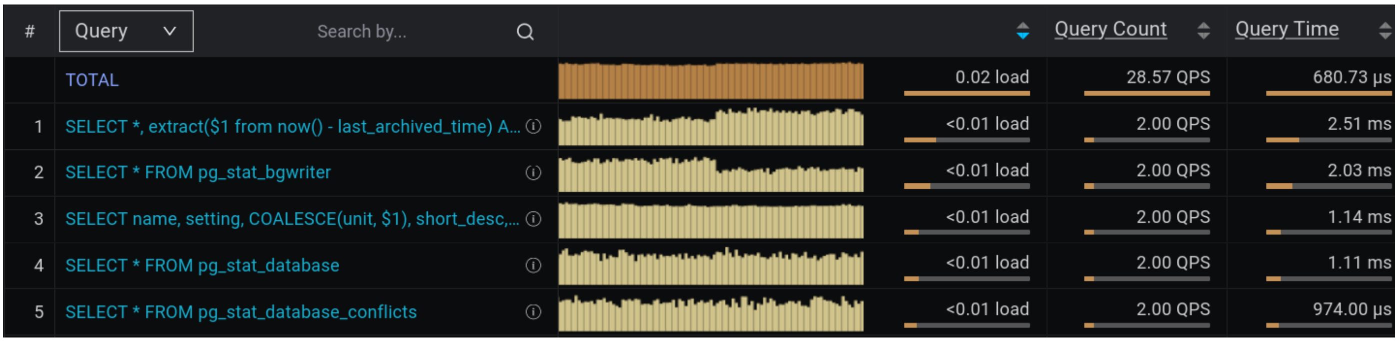
L'expertise PostgreSQL

Table des matières

Annonce de Percona

`pg_stat_monitor` est disponible en version 1.0

- Fork de `pg_stat_statements` (et `auto_explain`)
- Composant de supervision pour la solution PMM



#	Query	Search by...		Query Count	Query Time
	TOTAL		0.02 load	28.57 QPS	680.73 µs
1	SELECT *, extract($1 from now() - last_archived_time) A...		<0.01 load	2.00 QPS	2.51 ms
2	SELECT * FROM pg_stat_bgwriter		<0.01 load	2.00 QPS	2.03 ms
3	SELECT name, setting, COALESCE(unit, $1), short_desc,...		<0.01 load	2.00 QPS	1.14 ms
4	SELECT * FROM pg_stat_database		<0.01 load	2.00 QPS	1.11 ms
5	SELECT * FROM pg_stat_database_conflicts		<0.01 load	2.00 QPS	974.00 µs

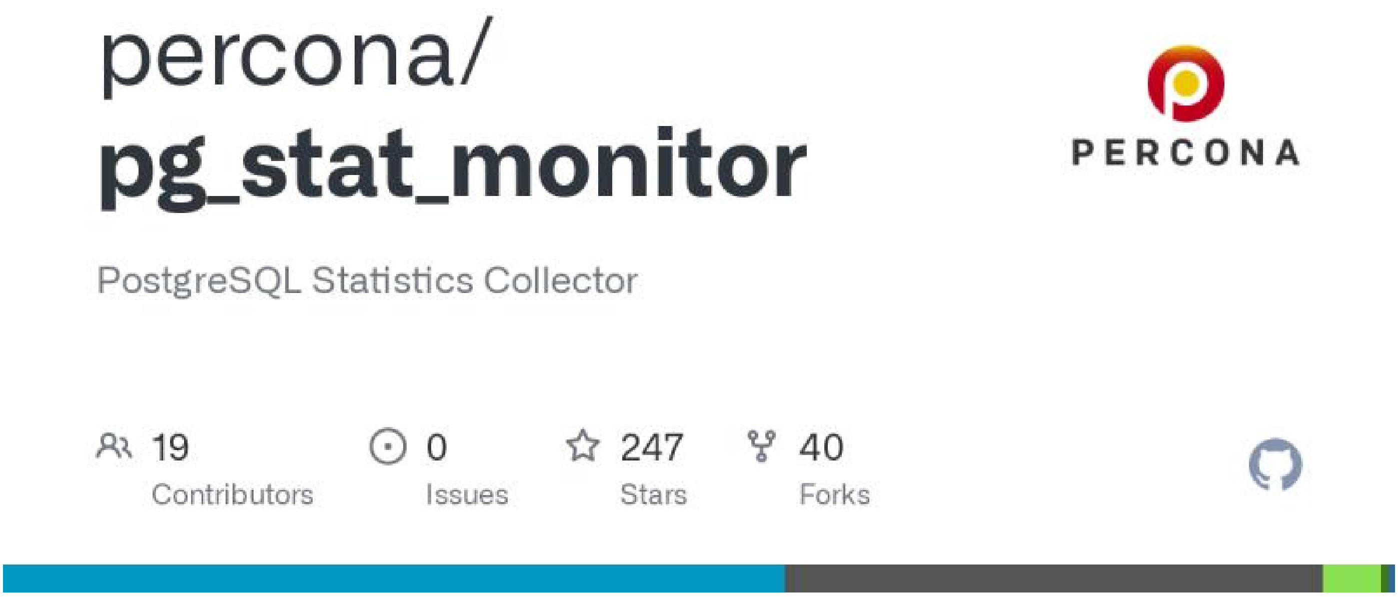
Annonce[1] du 6 mai 2022 : « Announcing general availability of pg_stat_monitor »

PMM : Percona Monitor and Management - mysql, pgsql, mongodb - composant QAN (Query Analytics) dédié écrit en typescript

---

Nouvelles fonctionnalités

(par rapport à `pg_stat_statements`...)

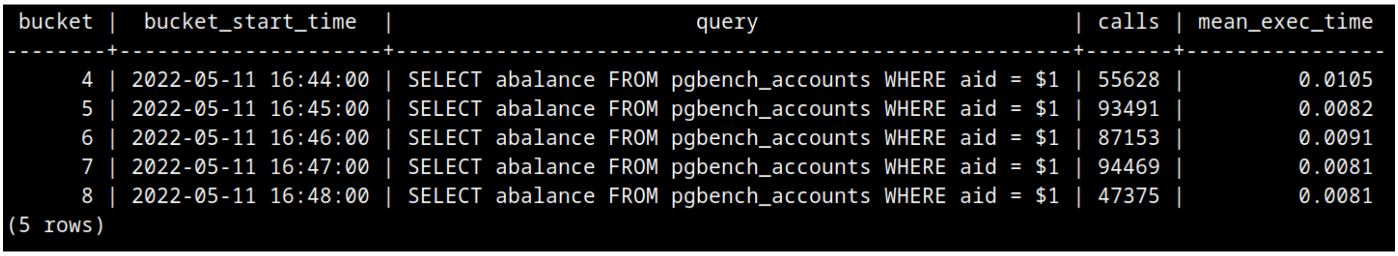


---

[1] https://www.postgresql.org/about/news/announcing-general-availability-of-pg_stat_monitor-2448/

Regroupement des requêtes en *time series buckets*

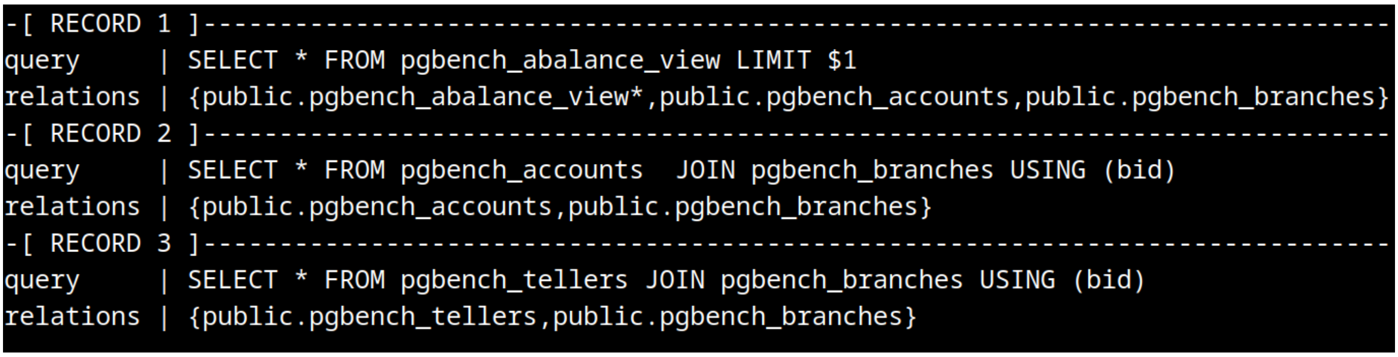
- Par défaut, un *bucket* toutes les 60 secondes
  - `pg_stat_monitor.pgsm_max_buckets` (max: 10)
  - `pg_stat_monitor.pgsm_bucket_time` (min: 1sec)



```
 bucket |  bucket_start_time   |                         query                         | calls | mean_exec_time
--------+---------------------+-------------------------------------------------------+-------+----------------
      4 | 2022-05-11 16:44:00 | SELECT abalance FROM pgbench_accounts WHERE aid = $1 | 55628 |         0.0105
      5 | 2022-05-11 16:45:00 | SELECT abalance FROM pgbench_accounts WHERE aid = $1 | 93491 |         0.0082
      6 | 2022-05-11 16:46:00 | SELECT abalance FROM pgbench_accounts WHERE aid = $1 | 87153 |         0.0091
      7 | 2022-05-11 16:47:00 | SELECT abalance FROM pgbench_accounts WHERE aid = $1 | 94469 |         0.0081
      8 | 2022-05-11 16:48:00 | SELECT abalance FROM pgbench_accounts WHERE aid = $1 | 47375 |         0.0081
(5 rows)
```

Relations de la requêtes

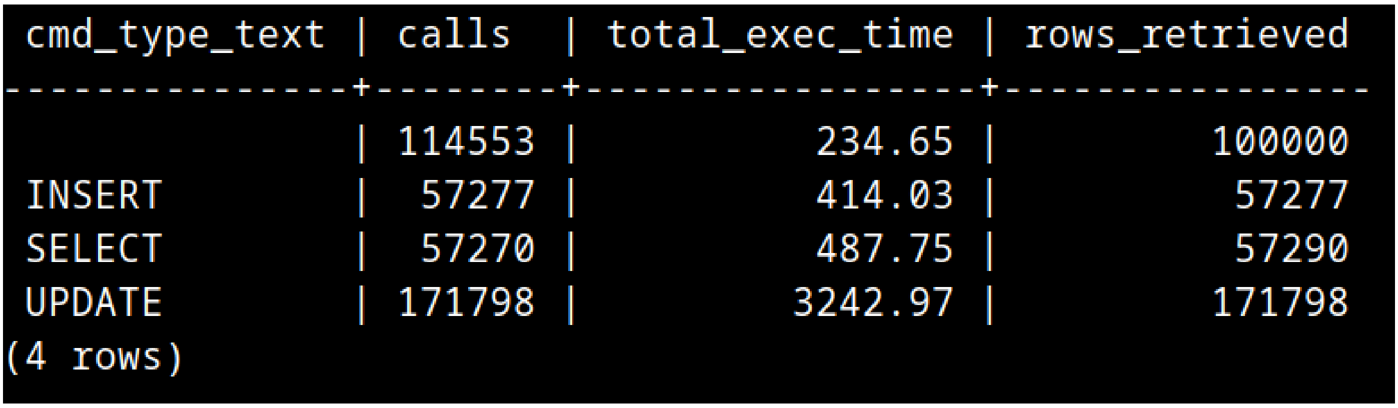
- Champ `relations`
  - Liste les tables rattachées aux requêtes
  - Parcours la définition des vues



```
-[ RECORD 1 ]----------------------------------------------------------------------
query     | SELECT * FROM pgbench_abalance_view LIMIT $1
relations | {public.pgbench_abalance_view*,public.pgbench_accounts,public.pgbench_branches}
-[ RECORD 2 ]----------------------------------------------------------------------
query     | SELECT * FROM pgbench_accounts  JOIN pgbench_branches USING (bid)
relations | {public.pgbench_accounts,public.pgbench_branches}
-[ RECORD 3 ]----------------------------------------------------------------------
query     | SELECT * FROM pgbench_tellers JOIN pgbench_branches USING (bid)
relations | {public.pgbench_tellers,public.pgbench_branches}
```

Types des requêtes

- Catégorise les requêtes selon leur type
  - `SELECT`, `INSERT`, `UPDATE`, `DELETE`
  - `(empty)`, `UTILITY`, `NOTHING`
- Champs `cmd_type` et `cmd_type_text`
  - Fonction `get_cmd_type(integer)`

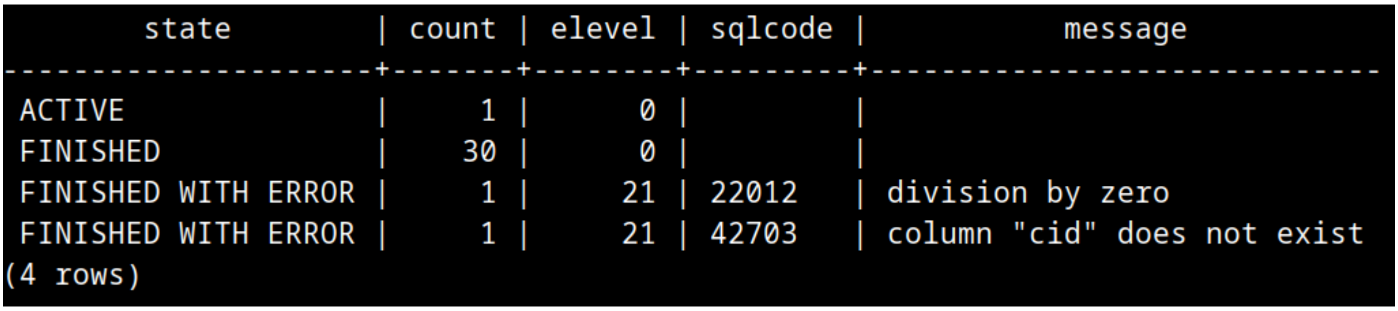


```
 cmd_type_text | calls  | total_exec_time | rows_retrieved
---------------+--------+-----------------+----------------
               | 114553 |          234.65 |         100000
 INSERT        |  57277 |          414.03 |          57277
 SELECT        |  57270 |          487.75 |          57290
 UPDATE        | 171798 |         3242.97 |         171798
(4 rows)
```

```
// src/include/nodes/nodes.h
/*
 * CmdType -
 *    enums for type of operation represented by a Query or PlannedStmt
 *
 * This is needed in both parsenodes.h and plannodes.h, so put it here...
 */
typedef enum CmdType
{
    CMD_UNKNOWN,
    CMD_SELECT,   /* select stmt */
    CMD_UPDATE,   /* update stmt */
    CMD_INSERT,   /* insert stmt */
    CMD_DELETE,   /* delete stmt */
    CMD_MERGE,    /* merge stmt */
    CMD_UTILITY,  /* cmds like create, destroy, copy, vacuum,
                               * etc. */
    CMD_NOTHING   /* dummy command for instead nothing rules
                               * with qual */
} CmdType;
```

Requêtes en erreur

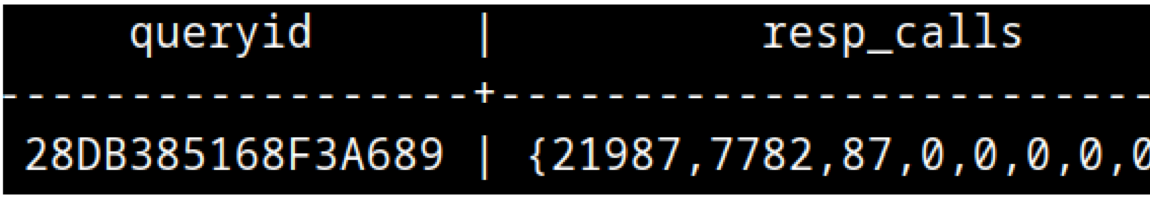
- Capte les requêtes en erreur
- Champs `state`, `elevel`, `sqlcode`, `message`



```
        state         | count | elevel | sqlcode |           message
----------------------+-------+--------+---------+------------------------------
 ACTIVE               |     1 |      0 |         |
 FINISHED             |    30 |      0 |         |
 FINISHED WITH ERROR  |     1 |     21 | 22012   | division by zero
 FINISHED WITH ERROR  |     1 |     21 | 42703   | column "cid" does not exist
(4 rows)
```

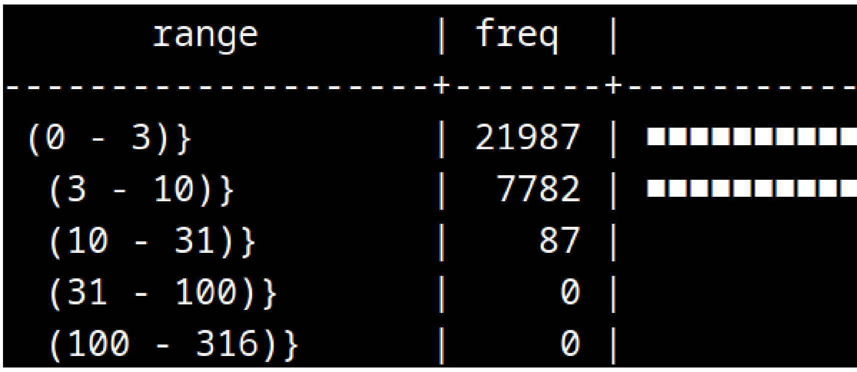
Histogramme d’exécution

- Expose les requêtes selon leur temps d’éxecution
- Champ `resp_calls`



```
      queryid       |          resp_calls
------------------+-------------------------
28DB385168F3A689 | {21987,7782,87,0,0,0,0,0
```

- Fonction `histogram(bucket, queryid)`



```
        range         | freq  |
--------------------+-------+----------
(0 - 3)}            | 21987 | ■■■■■■■■■■
 (3 - 10)}          |  7782 | ■■■■■■■■■■
 (10 - 31)}         |    87 |
 (31 - 100)}        |     0 |
 (100 - 316)}       |     0 |
```

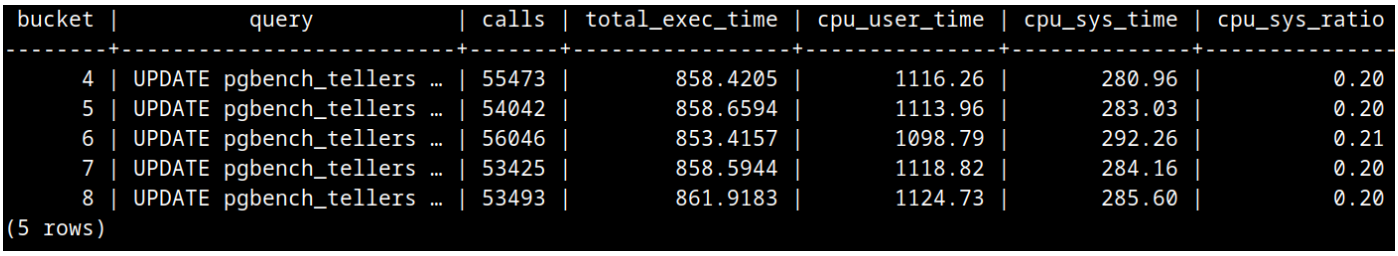
```
// pmm-agent: agents/postgres/pgstatmonitor/pgstatmonitor.go

getHistogramRangesArray() []*agentpb.HistogramItem {
    // For now we using static ranges.
    // In future we will compute range values from pg_stat_monitor_settings.
    // pgsm_histogram_min, pgsm_histogram_max, pgsm_histogram_buckets.
}
```

Plans d’éxecution

- Champs `planid` et `query_plan`
  - Affecte les performances de l’instance
  - `pg_stat_monitor.pgsm_enable_query_plan` (no)
- Équivalent de `auto_explain` mais en mémoire
  - pas d’options EXPLAIN supplémentaires

Consommation CPU

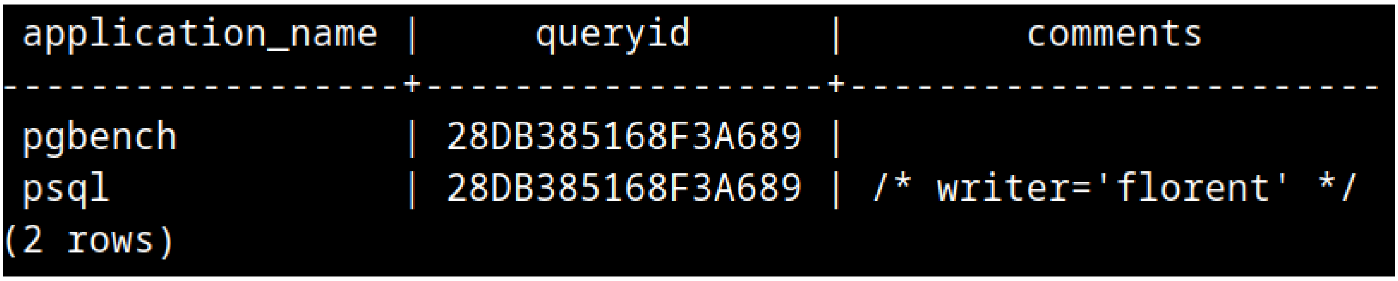
- Champs `cpu_user_time` et `cpu_sys_time`
  - Consommation CPU du tracking de requêtes
  - S'appuient sur la fonction `getrusage()`
  - Décorrélés de la valeur `total_exec_time`



```
 bucket |           query          | calls | total_exec_time | cpu_user_time | cpu_sys_time | cpu_sys_ratio
--------+--------------------------+-------+-----------------+---------------+--------------+--------------
      4 | UPDATE pgbench_tellers … | 55473 |        858.4205 |       1116.26 |       280.96 |          0.20
      5 | UPDATE pgbench_tellers … | 54042 |        858.6594 |       1113.96 |       283.03 |          0.20
      6 | UPDATE pgbench_tellers … | 56046 |        853.4157 |       1098.79 |       292.26 |          0.21
      7 | UPDATE pgbench_tellers … | 53425 |        858.5944 |       1118.82 |       284.16 |          0.20
      8 | UPDATE pgbench_tellers … | 53493 |        861.9183 |       1124.73 |       285.60 |          0.20
(5 rows)
```

Métadonnées de requête

- Spécification « Sqlcommenter » de Google
- Extrait le bloc de commentaire
  - et maintient le `queryid` intact



```
 application_name |      queryid       |         comments
------------------+--------------------+-------------------------
 pgbench          | 28DB385168F3A689 |
 psql             | 28DB385168F3A689 | /* writer='florent' */
(2 rows)
```

> an open source library that addresses the gap between the ORM libraries and understanding database performance. Sqlcommenter gives application developers visibility into which application code is generating slow queries and maps application traces to database query plans

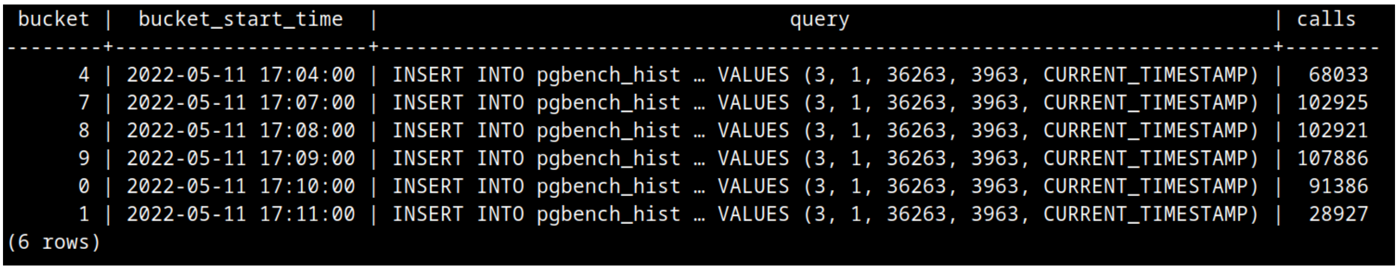
https://cloud.google.com/blog/topics/developers-practitioners/introducing-sqlcommenter-open-source-orm-auto-instrumentation-library

---

Requêtes dénormalisées

- Désactiver la normalisation des requêtes
  - Afficher les valeurs réelles
  - … Seule la première occurrence est tracée

- Facilite l'analyse des performances d'une requête
  - `pg_stat_monitor.pgsm_normalized_query`



```
 bucket |  bucket_start_time  |                                   query                                    | calls
--------+---------------------+----------------------------------------------------------------------------+--------
      4 | 2022-05-11 17:04:00 | INSERT INTO pgbench_hist … VALUES (3, 1, 36263, 3963, CURRENT_TIMESTAMP) |  68033
      7 | 2022-05-11 17:07:00 | INSERT INTO pgbench_hist … VALUES (3, 1, 36263, 3963, CURRENT_TIMESTAMP) | 102925
      8 | 2022-05-11 17:08:00 | INSERT INTO pgbench_hist … VALUES (3, 1, 36263, 3963, CURRENT_TIMESTAMP) | 102921
      9 | 2022-05-11 17:09:00 | INSERT INTO pgbench_hist … VALUES (3, 1, 36263, 3963, CURRENT_TIMESTAMP) | 107886
      0 | 2022-05-11 17:10:00 | INSERT INTO pgbench_hist … VALUES (3, 1, 36263, 3963, CURRENT_TIMESTAMP) |  91386
      1 | 2022-05-11 17:11:00 | INSERT INTO pgbench_hist … VALUES (3, 1, 36263, 3963, CURRENT_TIMESTAMP) |  28927
(6 rows)
```

Différences mineures

- `queryid` de type TEXT au lieu de BIGINT
- `userid` de type REGROLE au lieu de OID
- `datname` de type NAME au lieu de `dboid` de type OID
- `bucket_start_time` de type TEXT au lieu de TIMESTAMPTZ dans la documentation
- `rows_retrieved` au lieu de `rows`
- Colonnes inédites `application_name`, `client_ip`

https://percona.github.io/pg_stat_monitor/REL1_0_STABLE/COMPARISON.html

```c
#if PG_VERSION_NUM >= 140000
    queryId = pstmt->queryId;

    /*
     * Force utility statements to get queryId zero.  We do this even in cases
     * where the statement contains an optimizable statement for which a
     * queryId could be derived (such as EXPLAIN or DECLARE CURSOR).  For such
     * cases, runtime control will first go through ProcessUtility and then
     * the executor, and we don't want the executor hooks to do anything,
     * since we are already measuring the statement's costs at the utility
     * level.
     */
    if (PGSM_TRACK_UTILITY && pgsm_enabled(exec_nested_level))
        pstmt->queryId = UINT64CONST(0);
#endif
```

Limites actuelles

- Cohabitation difficile entre pgss et pgsm

  - pgss doit être chargé avant pgsm
  - En version 14, `compute_query_id = true`

- Les statistiques ne sont pas conservées après un redémarrage
- Un *bucket* n'est pas limité en nombre de requêtes distinctes
  - `pg_stat_monitor.pgsm_max` est exprimé en *MB* et non en quantité de requêtes

> For PostgreSQL 13 and earlier versions, `pg_stat_monitor` must follow `pg_stat_statements`. For example, `ALTER SYSTEM SET shared_preload_libraries = 'foo, pg_stat_statements, pg_stat_monitor'`.
>
> In PostgreSQL 14, you can specify `pg_stat_statements` and `pg_stat_monitor` in any order. However, due to the extensions' architecture, if both `pg_stat_statements` and `pg_stat_monitor` are loaded, only the last listed extension captures utility queries, `CREATE TABLE`, Analyze, etc. The first listed extension captures most common queries like `SELECT`, `UPDATE`, `INSERT`, but does not capture utility queries.
>
> Thus, to collect the whole statistics with `pg_stat_monitor`, we recommend to specify the extensions as follows: `ALTER SYSTEM SET shared_preload_libraries = 'pg_stat_statements, pg_stat_monitor'`.

Démonstrations

- Configuration (view settings)

```sql
SELECT name, value, default_value, options FROM pg_stat_monitor_settings;
```

- Visualiser le job

```sql
SELECT schedule, command FROM cron.job;
```

- Répartition du nombre d'appels par type

```sql
SELECT extract(epoch from bucket_start_time::timestamptz) AS time,
       cmd_type_text, avg(coalesce(calls,0)) AS calls
FROM pgsm_history
GROUP BY time, cmd_type_text
ORDER BY time;
```

- Relations les plus sollicitées

```
SELECT extract(epoch from bucket_start_time::timestamptz) AS time,
       unnest(relations) relation, COUNT(1) * sum(calls) count
 FROM pgsm_history
GROUP BY time, relation
ORDER BY time;
```

- Suivi des performances d'une requête

```
SELECT bucket, queryid, query FROM pg_stat_monitor
 WHERE 'public.pgbench_accounts' = ANY (relations)
   AND cmd_type_text = 'SELECT';

ALTER TABLE pgbench_accounts ADD CONSTRAINT pgbench_accounts_pkey
  PRIMARY KEY (aid);

-- temps d'exécution moyen
SELECT extract(epoch from bucket_start_time::timestamptz) AS time, mean_exec_time
  FROM pgsm_history WHERE queryid = '33B3468812A11DD2'
 ORDER BY time;

-- nombre d'appels total
SELECT extract(epoch from bucket_start_time::timestamptz) AS time, calls
  FROM pgsm_history WHERE queryid = '33B3468812A11DD2'
 ORDER BY time;

-- histogramme d'exécution
CREATE EXTENSION tablefunc;

SELECT * FROM crosstab($$
  SELECT extract(epoch from bucket_start_time::timestamptz) AS time,
         r range, resp_calls[ordinality]::int value
    FROM pgsm_history,
 LATERAL unnest('{r1,r2,r3,r4,r5,r6,r7,r8,r9,r10}'::text[]) WITH ORDINALITY r
   WHERE queryid = '33B3468812A11DD2'
  ORDER BY time, ordinality
$$) AS ct (time numeric, r1 int, r2 int, r3 int, r4 int,
           r5 int, r6 int, r7 int, r8 int, r9 int, r10 int);
```

CONCLUSION

- `pg_stat_monitor` est fortement couplé à PMM
- De bonnes idées pour `pg_stat_statements`
- Si vous ne connaissiez pas, essayez `pg_stat_statements`

QUESTIONS