
itm8®
Today. Tomorrow. Together

AGENDA

Where to log
How to log
What to log
pg_stat_io
Explain
Explain analyse
Auto_explain
pgBadger
pg_stat_statements
Waits
Incremental backup V17

WHERE TO LOG

The normal place to log in postgres is logfile and that it default placed in DATA/pg_log

To find out the exact location of the log file, you can check the PostgreSQL configuration file (**postgresql.conf** or **postgresql.auto.conf**).
Look for the **log_directory** or and **log_filename** parameter.

```
show data_directory;
data_directory
-------------------------------
/Users/pgr/postgres/16/dev/data

show log_directory;
log_directory
---------------
log

show log_filename;
log_filename
-------------------------------
 postgresql-%Y-%m-%d_%H%M%S.log
```

HOW TO LOG

You should use the logging collector since this is the only way to grantee messages is saved and in the correct order.

```
log_destination = 'stderr'          # stderr, csvlog, jsonlog, …

logging_collector = on              # off or on

log_rotation_age= 1d
log_rotation_size = 10MB
log_truncate_on_rotation = off      # off or on
```

WHAT TO LOG

By default, Postgres will not log anything abut DDL, DML or SELECT's.

Postgres has a lot of log options, and they are mildly saying complicated !

```
log_min_duration_statement = -1    # -1 (off), 0, N in milliseconds
log_statement = 'all'              # none, ddl, mod, all
log_duration = on                  # off or on
```

PG_STAT_IO

Postgres 16 got the view pg_stat_io with I/O timing but sadly it's turned off :-(

I recommend you turn it on since the argument that it is expensive to check the time on I/O is not an issue.

```
$ pg_test_timing -d 5
Testing timing overhead for 5 seconds.
Per loop time including overhead: 41.27 ns
Histogram of timing durations:
  < us   % of total      count
     1     95.96387  116265248
     2      4.02973    4882229
     4      0.00165       2001
     8      0.00024        289
```

PG_STAT_IO

I think you should enable ties timing and tracking parameters in Postgres

track_activities : enables monitoring of the current command being executed by any server process.

track_counts : controls whether cumulative statistics are collected about table and index accesses.

track_functions : enables tracking of usage of user-defined functions.

track_io_timing : enables monitoring of block read and write times.

track_wal_io_timing : enables monitoring of WAL write times.

EXPLAIN

Explain command will give you the execution plan of a statemnet

```
PSQL> explain
SELECT count(*) FROM pg_class, pg_index WHERE oid = indrelid AND indisunique;

------------------------------------------------------------------------------
 Aggregate  (cost=30.29..30.30 rows=1 width=8)
   ->  Hash Join  (cost=23.88..29.91 rows=150 width=0)
         Hash Cond: (pg_index.indrelid = pg_class.oid)
         ->  Seq Scan on pg_index  (cost=0.00..5.64 rows=150 width=4)
               Filter: indisunique
         ->  Hash  (cost=18.39..18.39 rows=439 width=4)
               ->  Seq Scan on pg_class  (cost=0.00..18.39 rows=439 width=4)
(7 rows)
```

https://explain.depesz.com/
https://explain.dalibo.com/

EXPLAIN ANALYSE

Explain command will give you the execution plan of a statement and ***run*** the statement to gather timing statistics !

```
BEGIN;
explain analyse
SELECT count(*) FROM pg_class, pg_index WHERE oid = indrelid AND indisunique;
Aggregate  (cost=30.29..30.30 rows=1 width=8) (actual time=0.205..0.207 rows=1 loops=1)
   ->  Hash Join  (cost=23.88..29.91 rows=150 width=0) (actual time=0.137..0.194 rows=158 loops=1)
         Hash Cond: (pg_index.indrelid = pg_class.oid)
         ->  Seq Scan on pg_index  (cost=0.00..5.64 rows=150 width=4) (actual time=0.008..0.035 rows=158
loops=1)
               Filter: indisunique
               Rows Removed by Filter: 14
         ->  Hash  (cost=18.39..18.39 rows=439 width=4) (actual time=0.124..0.124 rows=439 loops=1)
               Buckets: 1024  Batches: 1  Memory Usage: 24kB
               ->  Seq Scan on pg_class  (cost=0.00..18.39 rows=439 width=4) (actual time=0.004..0.069 rows=439
loops=1)
 Planning Time: 0.164 ms
 Execution Time: 0.380 ms
(11 rows)
ROLLBACK;
```

EXPLAIN

```
begin
explain (analyse, VERBOSE, COSTS, SETTINGS,BUFFERS, WAL,TIMING)
SELECT count(*) FROM pg_class, pg_index WHERE oid = indrelid AND indisunique;
Aggregate  (cost=29.70..29.71 rows=1 width=8) (actual time=0.262..0.263 rows=1 loops=1)
   Output: count(*)
   Buffers: shared hit=18
   ->  Hash Join  (cost=23.29..29.33 rows=150 width=0) (actual time=0.191..0.246 rows=154 loops=1)
         Inner Unique: true
         Hash Cond: (pg_index.indrelid = pg_class.oid)
         Buffers: shared hit=18
         ->  Seq Scan on pg_catalog.pg_index  (cost=0.00..5.64 rows=150 width=4) (actual time=0.017..0.046 rows=154 loops=1)
               Output: pg_index.indexrelid, pg_index.indrelid, pg_index.indnatts, pg_index.indnkeyatts, pg_index.indisunique,
pg_index.indnullsnotdistinct, pg_index.indisprimary, pg_index.indisexclusion, pg_index.indimmediate, pg_index.indisclustered, pg_index.indisvalid,
pg_index.indcheckxmin, pg_index.indisready, pg_index.indislive, pg_index.indisreplident, pg_index.indkey, pg_index.indcollation, pg_index.indclass,
pg_index.indoption, pg_index.indexprs, pg_index.indpred
               Filter: pg_index.indisunique
               Rows Removed by Filter: 14
               Buffers: shared hit=4
         ->  Hash  (cost=18.13..18.13 rows=413 width=4) (actual time=0.155..0.155 rows=425 loops=1)
               Output: pg_class.oid
               Buckets: 1024  Batches: 1  Memory Usage: 23kB
               Buffers: shared hit=14
               ->  Seq Scan on pg_catalog.pg_class  (cost=0.00..18.13 rows=413 width=4) (actual time=0.008..0.096 rows=425 loops=1)
                     Output: pg_class.oid
                     Buffers: shared hit=14
 Query Identifier: -5109968643562456341
 Planning:
   Buffers: shared hit=6
 Planning Time: 0.233 ms
 Execution Time: 0.323 ms


(24 rows)
rollback;
```

itm8

EXPLAIN V17

```
begin
explain (analyse,verbose,costs,settings,buffers,wall,timing,MEMORY)
SELECT count(*) FROM pg_class, pg_index WHERE oid = indrelid AND indisunique;
Aggregate  (cost=29.75..29.76 rows=1 width=8) (actual time=1.020..1.040 rows=1 loops=1)
   Output: count(*)
   Buffers: shared hit=16 read=2
   I/O Timings: shared read=0.383
   ->  Hash Join  (cost=23.34..29.37 rows=150 width=0) (actual time=0.782..0.847 rows=150 loops=1)
         Inner Unique: true
         Hash Cond: (pg_index.indrelid = pg_class.oid)
         Buffers: shared hit=16 read=2
         I/O Timings: shared read=0.383
         ->  Seq Scan on pg_catalog.pg_index  (cost=0.00..5.64 rows=150 width=4) (actual time=0.065..0.095 rows=150 loops=1)
               Output: pg_index.indexrelid, pg_index.indrelid, pg_index.indnatts, pg_index.indnkeyatts, pg_index.indisunique, pg_index.indnullsnotdistinct,
pg_index.indisprimary, pg_index.indisexclusion, pg_index.indimmediate, pg_index.indisclustered, pg_index.indisvalid, pg_index.indcheckxmin, pg_index.indisready,
pg_index.indislive, pg_index.indisreplident, pg_index.indkey, pg_index.indcollation, pg_index.indclass, pg_index.indoption, pg_index.indexprs, pg_index.indpred
               Filter: pg_index.indisunique
               Rows Removed by Filter: 14
               Buffers: shared hit=4
         ->  Hash  (cost=18.15..18.15 rows=415 width=4) (actual time=0.646..0.647 rows=415 loops=1)
               Output: pg_class.oid
               Buckets: 1024  Batches: 1  Memory Usage: 23kB
               Buffers: shared hit=12 read=2
               I/O Timings: shared read=0.383
               ->  Seq Scan on pg_catalog.pg_class  (cost=0.00..18.15 rows=415 width=4) (actual time=0.006..0.483 rows=415 loops=1)
                     Output: pg_class.oid
                     Buffers: shared hit=12 read=2
                     I/O Timings: shared read=0.383
 Query Identifier: 7458316119861708727
 Planning:
   Buffers: shared hit=154 read=31
   I/O Timings: shared read=125.848
   Memory: used=39kB  allocated=64kB
 Planning Time: 167.965 ms
 Execution Time: 3.148 ms
(30 rows)
rollback;
```

itm8

AUTO_EXPLAIN

The auto_explain module provides a means for logging execution plans of slow statements automatically, without having to run **EXPLAIN** by hand.

LOAD 'auto_explain' or use **session_preload_libraries** or **shared_preload_libraries**

```
auto_explain.log_min_duration
auto_explain.log_parameter_max_length
auto_explain.log_analyze
auto_explain.log_buffers
auto_explain.log_wal
auto_explain.log_timing
auto_explain.log_triggers
auto_explain.log_verbose
auto_explain.log_settings
auto_explain.log_format
auto_explain.log_level
auto_explain.log_nested_statements
auto_explain.sample_rate
```

AUTO_EXPLAIN

```
LOAD 'auto_explain';
SET auto_explain.log_min_duration = 0;
SET auto_explain.log_analyze = true;

SELECT count(*)
FROM pg_class, pg_index
WHERE oid = indrelid AND indisunique;
```

AUTO_EXPLAIN

```
2024-03-18 13:05:26 CET [21042]: user=pgr,db=postgres,app=psql,client=127.0.0.1 LOG:  duration: 0.751 ms  plan:
    Query Text: SELECT count(*)
                FROM pg_class, pg_index
                WHERE oid = indrelid AND indisunique;
    Aggregate  (cost=30.29..30.30 rows=1 width=8) (actual time=0.729..0.736 rows=1 loops=1)
      ->  Hash Join  (cost=23.88..29.91 rows=150 width=0) (actual time=0.614..0.707 rows=158 loops=1)
            Hash Cond: (pg_index.indrelid = pg_class.oid)
            ->  Seq Scan on pg_index  (cost=0.00..5.64 rows=150 width=4) (actual time=0.022..0.068 rows=158
loops=1)
                  Filter: indisunique
                  Rows Removed by Filter: 14
            ->  Hash  (cost=18.39..18.39 rows=439 width=4) (actual time=0.255..0.256 rows=439 loops=1)
                  Buckets: 1024  Batches: 1  Memory Usage: 24kB
                  ->  Seq Scan on pg_class  (cost=0.00..18.39 rows=439 width=4) (actual time=0.014..0.149
rows=439 loops=1)
2024-03-18 13:05:26 CET [21042]: user=pgr,db=postgres,app=psql,client=127.0.0.1 LOG:  duration: 56.296 ms
statement: SELECT count(*)
                FROM pg_class, pg_index
                WHERE oid = indrelid AND indisunique;
2024-03-18 13:05:31 CET [19360]: user=pgr,db=postgres,app=psql,client=127.0.0.1 LOG:  disconnection: session
time: 1:12:13.936 user=pgr databa
```

AUTO_EXPLAIN

For the cluster use :

```
alter system set auto_explain.log_analyze = 'true';
alter system set auto_explain.log_buffers = 'on';
alter system set auto_explain.log_wal = 'on';
alter system set auto_explain.log_timing = 'on';
alter system set auto_explain.log_triggers = 'on';
alter system set auto_explain.log_nested_statements = 'on';
alter system set auto_explain.log_min_duration = 0;
```

PGBADGER

Pgbadger is free tool to parse the logfiles from Postgres. It can be downloaded from **https://github.com/darold/pgbadger#** and is a Phyton program.

Recommended log setting for pgbadger is :

```
log_statement = 'none'
log_min_duration_statement = 0
log_line_prefix = '%t [%p]: user=%u,db=%d,app=%a,client=%h '
log_checkpoints = on
log_connections = on
log_disconnections = on
log_lock_waits = on
log_temp_files = 0
log_autovacuum_min_duration = 0
log_error_verbosity = default
```

```
pgbench -i -c 10 -j 2 -t 10000

pgbadger -I -q /Users/pgr/postgres/16/log/5416.log -O /Users/pgr/postgres/16/log/html
```

PG_STAT_STATEMENTS

The pg_stat_statements module provides a means for tracking planning and execution statistics of all SQL statements executed by a database.

The module must be loaded by adding pg_stat_statements to shared_preload_libraries and create the extention fir the database.

You must configure the extension :

```
alter system set shared_preload_libraries=pg_stat_statements;
alter system set pg_stat_statements.max = 5000;                 # 5000
alter system set pg_stat_statements.track = 'top';              # top, all, none
alter system set pg_stat_statements.track_planning = 'on'       # on, off
alter system set pg_stat_statements.save = 'on';                # on, off



alter system set compute_query_id = 'on';
```

PG_STAT_STATEMENTS

The text of the sampled statements are saved in the file data/pg_stat_tmp/pgss_query_texts.stat

The size of this file is number of sql statement's * length of the sql statement in my toy database with 39 statements it 10 Kb

https://www.pgmustard.com/blog/queries-for-pg-stat-statements

PG_STAT_STATEMENTS

```
 View "public.pg_stat_statements"
        Column         |       Type        | Collation | Nullable | Default
-----------------------+-------------------+-----------+----------+---------
 userid                | oid               |           |          |
 dbid                  | oid               |           |          |
 toplevel              | boolean           |           |          |
 queryid               | bigint            |           |          |
 query                 | text              |           |          |
 plans                 | bigint            |           |          |
 total_plan_time       | double precision  |           |          |
 min_plan_time         | double precision  |           |          |
 max_plan_time         | double precision  |           |          |
 mean_plan_time        | double precision  |           |          |
 stddev_plan_time      | double precision  |           |          |
 calls                 | bigint            |           |          |
 total_exec_time       | double precision  |           |          |
 min_exec_time         | double precision  |           |          |
 max_exec_time         | double precision  |           |          |
 mean_exec_time        | double precision  |           |          |
 stddev_exec_time      | double precision  |           |          |
 rows                  | bigint            |           |          |
 ...
 wal_records           | bigint            |           |          |
 wal_fpi               | bigint            |           |          |
 wal_bytes             | numeric           |           |          |
```

itm8

PG_STAT_STATEMENTS

Demo

WAITS

Postgres registers waits in the code and the can be found in the table pg_stat_activity.

One row per server process, showing information related to the current activity of that process, such as state and current query.

The waits are currently (version 16) divided into 9 main categories :

We will look at the doc now 28.2.3. pg_stat_activity

In version 17 we get the table **pg_wait_events** that contains all waits and a description of the wait !

Then a demo !

INCREMENTAL BACKUP

Postgres gets native incremental backup in V 17

1) The first idea is to replace some of the full backups you're currently doing with incremental backups, saving backup time and network transfer.

2) The second idea is to do just as many full backups as you do now, but add incremental backups between them, so that if you need to do PITR, you can use **pg_combinebackup** to reach the latest incremental backup before the point to which you want to recover, reducing the amount of WAL that you need to replay, and probably speeding up the process quite a bit.

3) The third idea is to give up on taking full backups altogether and only ever take incremental backups.

Let’s build today’s and tomorrow’s IT. Together?

Peter.Gram@itm8.dk

Tlf. +45 5374 7107