

Becoming a PG_STAT_*STAR [Returns]

Chirag Dave

Pr. PostgreSQL SA
AWS

Sami Imseih

PostgreSQL Contributors Team
AWS/RDS Open Source Databases



What is PG_STAT_?

- *pg_stat* is a prefix for collection of server activity views
- (NOT) *pg_stats* or *pg_statistics*
 - Updated by ANALYZE, used by optimizer
 - Not server activity related
- <https://www.postgresql.org/docs/current/monitoring-stats.html>
- <https://wiki.postgresql.org/wiki/Monitoring>

Dynamic views

- *PID* column to identify a process
- An entry per connection
- Entry disappears when connection closes

```
pg_stat_activity  
pg_stat_gssapi  
pg_stat_replication  
pg_stat_ssl  
pg_stat_subscription  
pg_stat_wal_receiver
```

```
pg_stat_progress_analyze  
pg_stat_progress_basebackup  
pg_stat_progress_cluster  
pg_stat_progress_copy  
pg_stat_progress_create_index  
pg_stat_progress_vacuum
```

Dynamic views

```
SELECT * FROM pg_stat_activity  
WHERE state = 'active'  
AND pid <> pg_backend_pid();
```

```
-[ RECORD 1 ]-----  
datid          | 5  
datname        | postgres  
pid            | 19182  
leader_pid     |  
usesysid       | 10  
username       | postgres  
application_name | psql  
client_addr    |  
client_hostname |  
client_port    | -1  
backend_start  | 2024-04-08 01:51:57.306027+00  
xact_start     | 2024-04-08 01:52:25.990124+00  
query_start    | 2024-04-08 01:52:25.990124+00  
state_change   | 2024-04-08 01:52:25.990128+00  
wait_event_type | IO  
wait_event     | DataFileRead  
state          | active  
backend_xid    |  
backend_xmin   | 759  
query_id       | 621416754327427450  
query          | select count(*) from demo a, demo b;  
backend_type   | client backend
```

```
SELECT * FROM pg_stat_progress_vacuum;
```

```
-[ RECORD 1 ]-----  
pid            | 19782  
datid          | 5  
datname        | postgres  
relid          | 16407  
phase          | scanning heap  
heap_blks_total | 192308  
heap_blks_scanned | 6  
heap_blks_vacuumed | 0  
index_vacuum_count | 1  
max_dead_tuple_bytes | 67108864  
dead_tuple_bytes | 0  
indexes_total   | 5  
indexes_processed | 3
```

Cumulative Statistics

```
postgres=# select * from pg_stat_database where datname = 'test';
-[ RECORD 1 ]-----+-----
datid                | 41158
datname              | test
numbackends          | 0
xact_commit          | 18914978
xact_rollback        | 12
blks_read            | 142197
blks_hit             | 19664901
tup_returned         | 4092716
tup_fetched          | 89436
tup_inserted         | 18905744
tup_updated          | 148
tup_deleted          | 0
conflicts            | 0
temp_files           | 0
temp_bytes           | 0
deadlocks            | 0
checksum_failures    |
checksum_last_failure |
blk_read_time        | 0
blk_write_time       | 0
session_time         | 14853226.948
active_time          | 12783035.536
idle_in_transaction_time | 0
sessions             | 43
sessions_abandoned   | 0
sessions_fatal       | 0
sessions_killed      | 10
stats_reset          |
```

- All backends increment the values
- Values constantly increasing

Cumulative Statistics

Cluster-wide

8.3+	pg_stat_bgwriter
12+	pg_stat_archiver
13+	pg_stat_slru
14+	pg_stat_wal
16+	pg_stat_io
16+	pg_stat_replication_slots
16+	pg_stat_recovery_prefetch
16+	pg_stat_subscription_stats
17+	pg_stat_checkpoint

Per-Database

7.2+	pg_stat_database
9.1+	pg_stat_database_conflicts

Per-Relation

7.2+	pg_stat_all_tables
7.2 +	pg_stat_all_indexes
7.2 +	pg_statio_all_tables
7.2 +	pg_statio_all_indexes
7.2 +	pg_statio_all_sequences

Per-Statement

8.4+	pg_stat_statements
------	--------------------

Per-Function

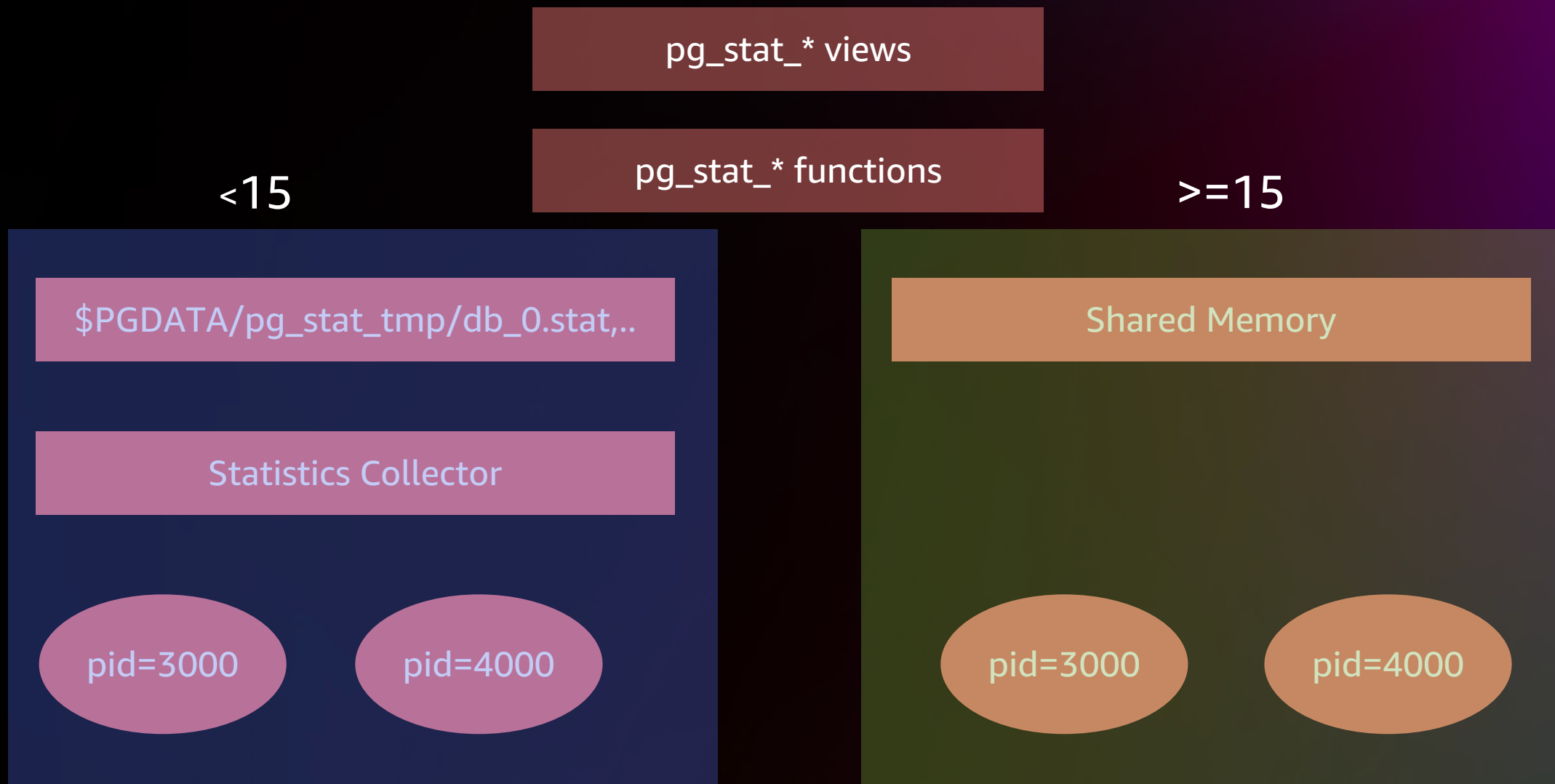
8.4+	pg_stat_user_functions
------	------------------------

Per-Backend

7.2+	pg_stat_get_backend_*
------	-----------------------

- * Not Cumulative Statistics System/Core Postgres
- * Add to *shared_preload_libraries*
- * CREATE EXTENSION pg_stat_statements

Cumulative Statistics



Custom Cumulative Statistics



<https://www.postgresql.org/docs/18/xfunc-c.html#XFUNC-ADDIN-CUSTOM-CUMULATIVE-STATISTICS>

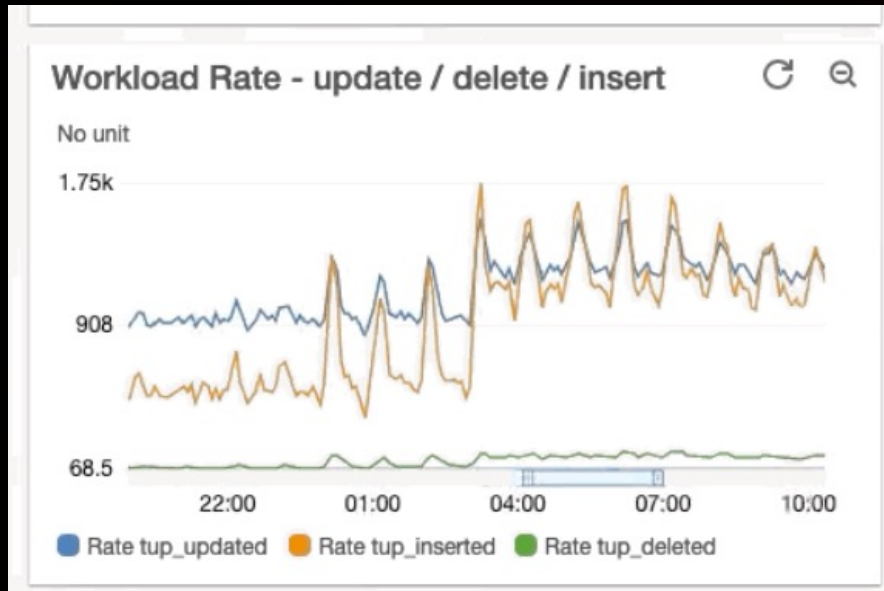
```
static const PgStat_KindInfo custom_stats = {
    .name = "custom_stats",
    .fixed_amount = false,
    .shared_size = sizeof(PgStatShared_Custom),
    .shared_data_off = offsetof(PgStatShared_Custom, stats),
    .shared_data_len = sizeof(((PgStatShared_Custom *) 0)->stats),
    .pending_size = sizeof(PgStat_StatCustomEntry),
}
```

```
extern PgStat_Kind pgstat_register_kind(PgStat_Kind kind,
                                       const PgStat_KindInfo *kind_info);
```

<https://wiki.postgresql.org/wiki/CustomCumulativeStats>

Cumulative Statistics

- Delta Metric to find changes during a time interval
- Calculates the rate of change
- $13022 - 3016 = 10006$ rows ever 10 seconds
- $10006 / 10 \approx 1000$ rows per second



```
SELECT now() timestamp, tup_inserted from  
pg_stat_database where datname = 'postgres';
```

timestamp	tup_inserted
2024-04-08 02:52:28.50213+00	3016

(1 row)

timestamp	tup_inserted
2024-04-08 02:52:38.502136+00	13022

(1 row)

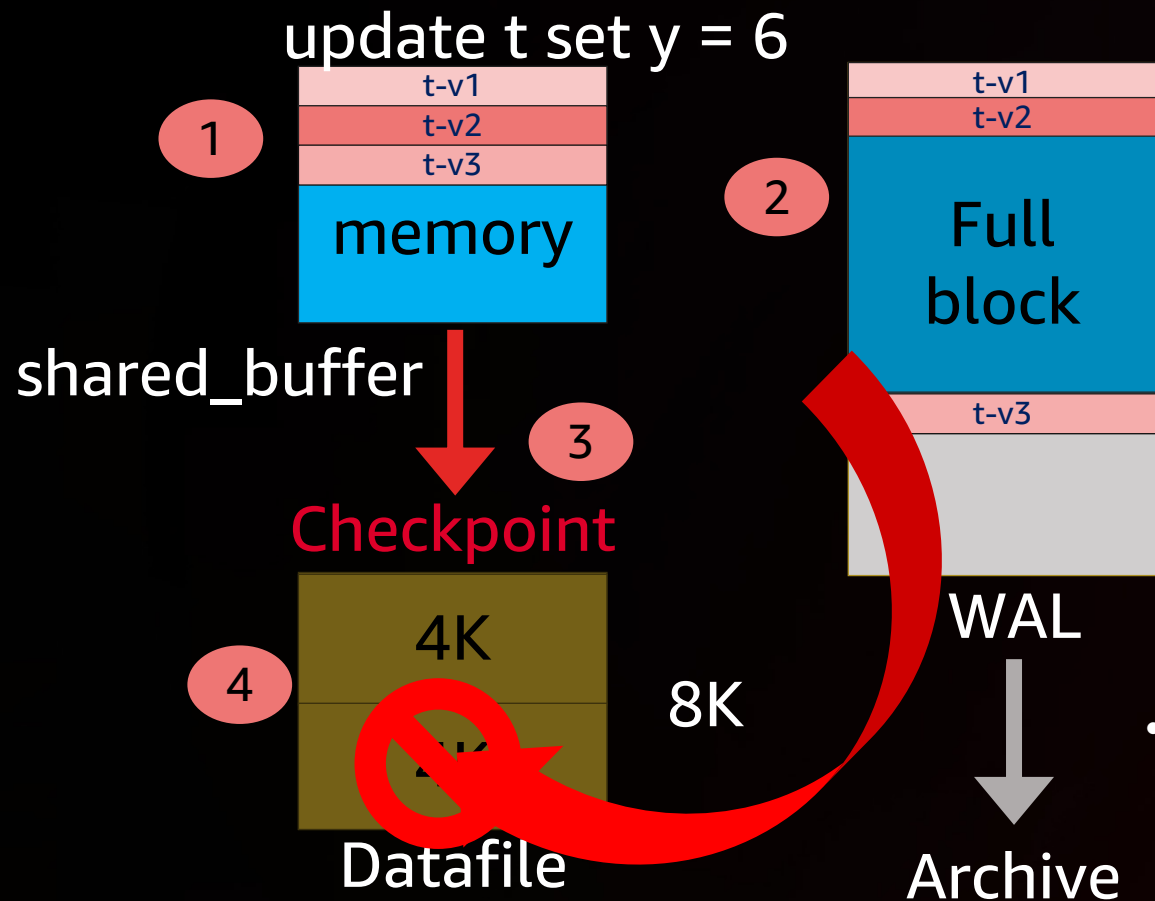


- <https://github.com/awslabs/pg-counter-metrics>

Cumulative Statistics System

- Background Writer, Checkpoint
- Vacuum /Autovacuum
- DML activity
- HOT Updates
- Index/Table Access
- I/O

PostgreSQL I/O



- 1 Write it to the WAL log
- 2 Update Shared Buffers
- 3 Checkpoint
- 4 Write to the disk

Fix this

- Dirty Buffer Flushing
 - Checkpoint
 - **increase** the checkpoint More Full Page Writes
 - **decrease** the checkpoint : longer recovery times
 - **Background Writer**

pg_stat_wal (PG14)

```
postgres=# select * from pg_stat_wal;
-[ RECORD 1 ]-----+-----
wal_records      | 414
wal_fpi          | 31
wal_bytes        | 207099
wal_buffers_full | 0
wal_write        | 27
wal_sync         | 27
wal_write_time   | 0
wal_sync_time    | 0
stats_reset      | 2024-04-14 02:17:21.429386+00
```

wal_fpi → WAL records due to a checkpoint

wal_buffers_full -→ wal_buffers setting is set too low

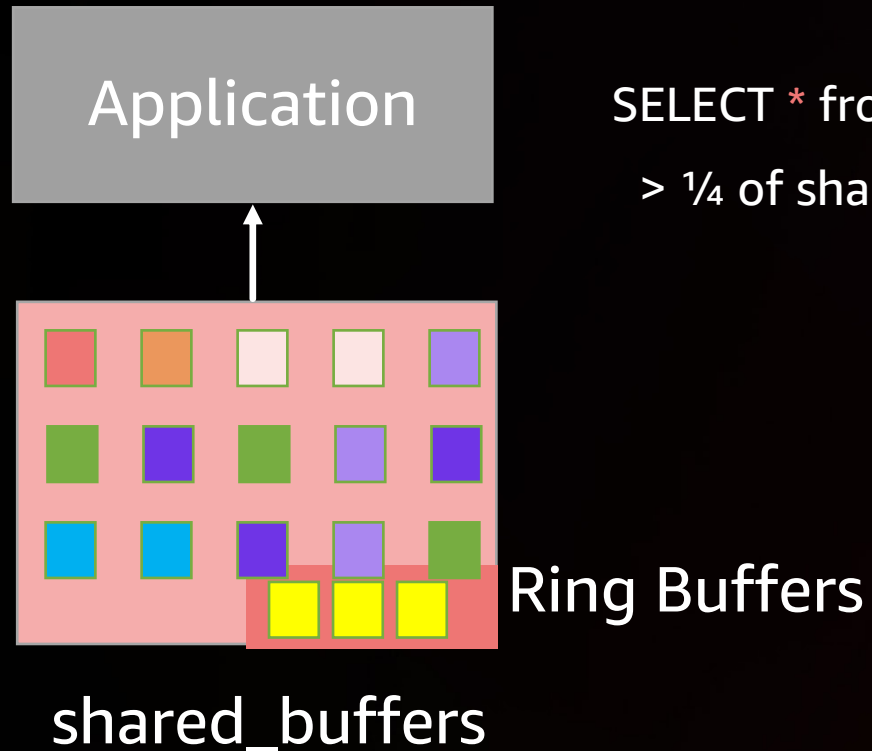
shared_buffers



Ring Buffers - Buffer Access Strategy

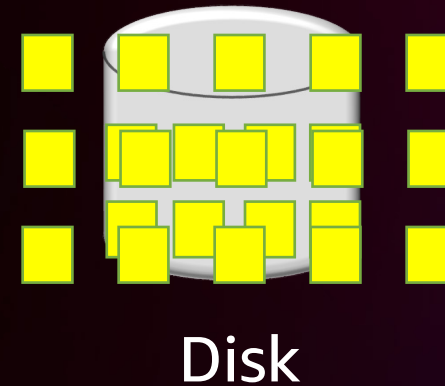
Prevent cache thrashing and maintains a higher cache hit ratio

bulk-reading	256 KB
bulk-writing	16 MB
vacuum-processing	256 KB



`SELECT * from large_table;`
> 1/4 of shared_buffers

New in PG16: `vacuum_buffer_usage_limit`



Background Writer, Checkpoint

```
postgres=# select * from pg_stat_bgwriter;
-[ RECORD 1 ]-----+-----
checkpoints_timed    | 0      > checkpoint_timeout
checkpoints_req      | 6      > max_wal_size
checkpoint_write_time | 72441
checkpoint_sync_time | 3838
buffers_checkpoint   | 6380   dirty buffers written by checkpointer
buffers_clean        | 16248  dirty buffers written by background writer
maxwritten_clean     | 161
buffers_backend      | 520106 dirty buffers written by backend
buffers_backend_fsync | 0
buffers_alloc        | 465918
stats_reset          | 2024-04-09 13:54:18.683023-05
```

- Keep `buffers_backend` close to 0 as possible
- 17+, this info will be spread between `pg_stat_bgwriter`, `pg_stat_checkpointer`, `pg_stat_io`

DML Activity

- Tracks # of inserts/updates/deletes per table

```
select * from pg_stat_all_tables
where relname = 'pgbench_accounts';
```

n_tup_ins		0
n_tup_upd		288202
n_tup_del		0

HOT Updates

- Maximize n_tup_hot_update
- Minimize n_tup_newpage_upd (16+)
- Reduce FILLFACTOR for heavily updated tables
- Drop unused Indexes

n_tup_ins		0
n_tup_upd		288202
n_tup_del		0
n_tup_hot_upd		288159
n_tup_newpage_upd		43

Vacuum

- track_counts = ON (DEFAULT)
 - pg_stat_all_indexes
 - pg_stat_all_sequences
- Autovacuum/vacuum metrics

```
select * from pg_stat_all_tables where relname =  
'pgbench_accounts';
```

```
-[ RECORD 1 ]-----+-----  
relid          | 16395  
schemaname     | public  
relname        | pgbench_accounts  
seq_scan       | 2  
last_seq_scan  | 2025-09-23 09:11:11.639694-05  
seq_tup_read   | 10000000  
idx_scan       | 2352670  
last_idx_scan  | 2025-09-23 09:16:45.263021-05  
idx_tup_fetch  | 2352670  
n_tup_ins      | 10000000  
n_tup_upd      | 1176335  
n_tup_del      | 0  
n_tup_hot_upd  | 998705  
n_tup_newpage_upd | 177630  
n_live_tup     | 9999440  
n_dead_tup     | 335089  
n_mod_since_analyze | 2554  
n_ins_since_vacuum | 0  
last_vacuum    | 2025-09-23 09:11:08.5229-05  
last_autovacuum |  
last_analyze   | 2025-09-23 09:11:08.794917-05  
last_autoanalyze | 2025-09-23 09:16:44.968341-05  
vacuum_count   | 1  
autovacuum_count | 0  
analyze_count  | 1  
autoanalyze_count | 1  
total_vacuum_time | 585  
total_autovacuum_time | 0  
total_analyze_time | 272  
total_autoanalyze_time | 9488
```



Vacuum

- Metrics used directly by autovacuum launcher
- Crash recovery/pg_stat_reset
 - Wipes out the data
 - May delay autovacuum/autoanalyze

```
/*
 * If we found stats for the table, and autovacuum is currently enabled,
 * make a threshold-based decision whether to vacuum and/or analyze. If
 * autovacuum is currently disabled, we must be here for anti-wraparound
 * vacuuming only, so don't vacuum (or analyze) anything that's not being
 * forced.
 */
if (PointerIsValid(tabentry) && AutoVacuumingActive())
{
    reltuples = classForm->reltuples;
    vactuples = tabentry->dead_tuples;
    instuples = tabentry->ins_since_vacuum;
    anltuples = tabentry->mod_since_analyze;

    /* If the table hasn't yet been vacuumed, take reltuples as zero */
    if (reltuples < 0)
        reltuples = 0;

    vacthresh = (float4) vac_base_thresh + vac_scale_factor * reltuples;
    vacinsthresh = (float4) vac_ins_base_thresh + vac_ins_scale_factor * reltuples;
    anlthresh = (float4) anl_base_thresh + anl_scale_factor * reltuples;
}
```

Vacuum

- `pg_stat_progress_vacuum.index_vacuum_count` increases every "vacuum index cleanup" cycle
- In Postgres 16 and below:
 - Scan the table for dead rows, and store the dead rows in `autovacuum_work_mem/maintenance_work_mem` (179 million dead rows max)
 - Vacuum the indexes
 - Repeat
 - Super expensive if multi-index vacuum cycles are required.
- In Postgres17:
 - Multi-index cycles will become less likely thanks to <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=30e144287a>

Index/Table Access

- last_seq_scan and last_idx_scan (16+)
- For OLTP, seq_scan should be close to 0

```
postgres=# select * from pg_stat_all_tables where relname = 'demo';
```

```
-[ RECORD 1 ]-----+-----  
relid          | 65572  
schemaname     | public  
relname        | demo  
seq_scan       | 0  
last_seq_scan  | 2024-04-07 17:08:22.454168+00  
seq_tup_read   | 0  
idx_scan       | 5  
last_idx_scan  | 2024-04-07 17:09:22.454168+00  
idx_tup_fetch  | 37199829
```

```
postgres=# select * from pg_stat_all_indexes where indexrelid = 'demo_id'::regclass::oid;
```

```
-[ RECORD 1 ]-----+-----  
relid          | 65572  
indexrelid     | 65575  
schemaname     | public  
relname        | demo  
indexrelname   | demo_id  
idx_scan       | 5  
last_idx_scan  | 2024-04-07 17:09:22.454168+00  
idx_tup_read   | 61999715  
idx_tup_fetch  | 37199829
```

Index/Table Access

- `idx_tup_fetch`
 - Index scan visits a heap
 - Projecting columns not in index

```
postgres=# select * from pg_stat_all_tables where relname =
'demo';
-[ RECORD 1 ]-----+-----
relid          | 65572
schemaname     | public
relname        | demo
seq_scan       | 0
last_seq_scan  |
seq_tup_read   | 0
idx_scan       | 5
last_idx_scan  | 2024-04-07 17:09:22.454168+00
idx_tup_fetch  | 37199829
```

```
EXPLAIN (ANALYZE) SELECT id FROM demo WHERE id = 'c40d8806-
c87e-4942-8d35-ce79819ba68c';
```

QUERY PLAN

```
-----
Index Only Scan using demo_pkey on demo (cost=0.43..4.45 rows=1
width=16) (actual time=0.045..0.047 rows
=1 loops=1)
  Index Cond: (id = 'c40d8806-c87e-4942-8d35-ce79819ba68c'::uuid)
  Heap Fetches: 0
Planning Time: 0.123 ms
Execution Time: 0.077 ms
(5 rows)
```

Index/Table Access

- Index-only scans minimize heap fetches
 - If `idx_tup_fetch` high, VACUUM more aggressive
 - Visibility Map not up-to-date

```
postgres=# select * from pg_stat_all_indexes where
indexrelid = 'demo_id'::regclass::oid;
-[ RECORD 1 ]-+-----
reloid          | 65572
indexrelid     | 65575
schemaname     | public
relname        | demo
indexrelname   | demo_id
idx_scan       | 5
last_idx_scan  | 2024-04-07 17:09:22.454168+00
idx_tup_read   | 61999715
idx_tup_fetch  | 37199829
```

```
SELECT tup_returned, tup_fetched from
pg_stat_database where datname =
'postgres';
-[ RECORD 1 ]+-----
tup_returned   | 62284143
tup_fetched    | 37204480
```

Parallel Query Statistics



```
postgres=# CREATE TABLE large (id BIGINT PRIMARY KEY, c1 TEXT);  
CREATE TABLE
```

```
postgres=# INSERT INTO large SELECT n, REPEAT(' ', 1000) FROM generate_series(1, 1000000) as n;  
INSERT 0 1000000
```

```
postgres=# SET parallel_tuple_cost = 0;  
SET
```

Force Parallelism

```
postgres=# SET parallel_setup_cost = 0;  
SET
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM large ;  
QUERY PLAN
```

Gather (cost=0.00..214287.00 rows=17142960 width=40) (actual time=1.855..1715.374 rows=1000000.00 loops=1)

Workers Planned: 2

Workers Launched: 2

Buffers: shared hit=257 read=142601 dirtied=36114 written=19325

-> Parallel Seq Scan on large (cost=0.00..214287.00 rows=7142900 width=40) (actual time=0.511..1570.444 rows=333333.33 loops=3)

Buffers: shared hit=257 read=142601 dirtied=36114 written=19325

Planning Time: 1.692 ms

Execution Time: 1739.796 ms

(8 rows)

Parallel Query Statistics



pgbench.sql

```
SET parallel_tuple_cost = 0;  
SET parallel_setup_cost = 0;  
SELECT * FROM large;
```

```
pgbench -c50 -f/tmp/pgbench.sql -T120
```

```
postgres=# EXPLAIN ANALYZE SELECT * FROM large;
```

QUERY PLAN

```
-----  
Gather (cost=0.00..147024.69 rows=1000006 width=1012) (actual time=0.223..324.756 rows=1000000.00 loops=1)  
  Workers Planned: 2  
  Workers Launched: 1  
    Buffers: shared hit=16112 read=126746  
    -> Parallel Seq Scan on large (cost=0.00..147024.69 rows=416669 width=1012) (actual time=0.459..194.591 rows=500000.00 loops=2)  
      Buffers: shared hit=16112 read=126746  
Planning Time: 0.061 ms  
Execution Time: 349.691 ms  
(8 rows)
```

Parallel Query Statistics



```
postgres=# SELECT * FROM pg_stat_database WHERE datname = current_database();
```

```
-[ RECORD 1 ]-----+-----  
datid          | 5  
datname        | postgres  
numbackends    | 1  
xact_commit    | 393  
xact_rollback  | 0  
blks_read      | 1549614  
blks_hit       | 11464195  
tup_returned   | 91019367  
tup_fetched    | 6627  
tup_inserted   | 0  
tup_updated    | 1  
tup_deleted    | 0  
conflicts      | 0  
temp_files     | 0  
temp_bytes     | 0  
deadlocks      | 0  
checksum_failures | 0  
checksum_last_failure | 0  
blk_read_time  | 0  
blk_write_time | 0  
session_time   | 10091150.988  
active_time    | 9071689.839  
idle_in_transaction_time | 0  
sessions       | 52  
sessions_abandoned | 0  
sessions_fatal | 0  
sessions_killed | 0  
parallel_workers_to_launch | 182  
parallel_workers_launched | 15  
stats_reset    |
```

MultiXacts in PostgreSQL

When multiple transactions attempt to lock the same row simultaneously, PostgreSQL turns to a specialized structure called [MultiXact](#) IDs

```
postgres=> begin ;
BEGIN
postgres=> SELECT * FROM pg_current_xact_id ();
 pg_current_xact_id
-----
          3023100
(1 row)

postgres=> SELECT * FROM pgbench_accounts WHERE aid = 1 FOR SHARE;
 aid | bid | abalance | filler
-----+-----+-----+-----
   1 |   1 |         0 |
(1 row)

postgres=> █
```

```
ec2-user@ip-10-99-0-200:~ (ssh)
postgres=> SELECT ctid, xmax, aid FROM pgbench_accounts WHERE aid < 2;
 ctid | xmax | aid
-----+-----+-----
(0,1) | 3023100 | 1
(1 row)

postgres=> SELECT * FROM pgrowlocks('pgbench_accounts');
 locked_row | locker | multi | xids | modes | pids
-----+-----+-----+-----+-----+-----
(0,1) | 3023100 | f | {3023100} | {"For Share"} | {7679}
(1 row)

postgres=> █
```

```
postgres=> SELECT ctid, xmax, aid FROM pgbench_accounts WHERE aid < 2;
 ctid | xmax | aid
-----+-----+-----
(0,1) | 24099 | 1
(1 row)

postgres=> SELECT * FROM pgrowlocks('pgbench_accounts');
 locked_row | locker | multi | xids | modes | pids
-----+-----+-----+-----+-----+-----
(0,1) | 24099 | t | {3023100,3023385} | {"For Share","For Share"} | {7679,6109}
(1 row)

postgres=> █
```

```
ec2-user@ip-10-99-0-200:~ (ssh)
pg_current_xact_id
-----
          3023385
(1 row)

postgres=> SELECT * FROM pgbench_accounts WHERE aid = 1 FOR SHARE;
 aid | bid | abalance | filler
-----+-----+-----+-----
   1 |   1 |         0 |
(1 row)

postgres=> █
```

Operations using MultiXacts

- Foreign keys
- FOR SHARE
- Savepoints
- Sub-transactions from PL/pgSQL EXCEPTION
- Drivers, ORMs

SLRU (Simple Least Recently Used)

Cache used to track transaction metadata. Backed up by a file.

Transaction status

- Commit status
- parent transaction ID (subtransaction)
- Commit timestamp
- Multixacts
- Notilfy
- Serializable isolation locks

```
postgres=> select name from pg_stat_slru;
          name
-----
CommitTs
MultiXactMember
MultiXactOffset
Notify
Serial
Subtrans
Xact
other
(8 rows)
```

SLRUs have a fixed size (< Postgres 17) measured in 8k pages



MultiXacts/Subtransactions

SLRU buffer pages are also written to the disk

MultiXacts, the directory name is `pg_multixact`

SubTransaction, the directory name is `pg_subtrans`

SLRU statistics (>=13)

```
postgres=> SELECT * FROM pg_stat_slru ;
```

name	blks_zeroed	blks_hit	blks_read	blks_written	blks_exists	flushes	truncates	stats_reset
CommitTs	0	0	0	0	0	0	0	2025-08-20 14:32:56.247996+00
MultiXactMember	505	76558	505	0	0	4	0	2025-08-20 14:32:56.247996+00
MultiXactOffset	13	75073	23	0	4	4	0	2025-08-20 14:32:56.247996+00
Notify	0	0	0	0	0	0	0	2025-08-20 14:32:56.247996+00
Serial	0	0	0	0	0	0	0	2025-08-20 14:32:56.247996+00
Subtrans	0	0	0	0	0	0	0	2025-08-20 14:32:56.247996+00
Xact	0	0	0	0	0	0	0	2025-08-20 14:32:56.247996+00
other	0	0	0	0	0	0	0	2025-08-20 14:32:56.247996+00

(8 rows)

```
postgres=# select name, setting from pg_settings where name like '%_buffers';
```

name	setting
commit_timestamp_buffers	32
multixact_member_buffers	32
multixact_offset_buffers	16
notify_buffers	16
serializable_buffers	32
shared_buffers	16384
subtransaction_buffers	32
temp_buffers	1024
transaction_buffers	32
wal_buffers	512

(10 rows)

>=17

Subtransactions Overflow Statistics

```
SELECT
  pg_stat_get_backend_pid(backendid) AS pid,
  s.*
FROM
  pg_stat_get_backend_idset() AS backendid,
  pg_stat_get_backend_subxact (backendid)
AS s
ORDER BY
  subxact_overflowed DESC,
  subxact_count DESC;
```

PG16

`pg_stat_get_backend_subxact()`

pid	subxact_count	subxact_overflowed
32211	64	t
32179	64	t
32216	64	t
32183	64	t
32208	64	t
32181	64	t
32196	64	f
32209	64	f
32201	64	f
32217	63	f
32172	60	f
32173	59	f
32199	55	f
32184	50	f
32200	44	f
32210	44	f
32207	43	f
32215	42	f
32214	40	f
32180	40	f

I/O

Pre 16, Cumulative Statistics did not make a distinction for Buffer Access Strategies

New in PG16 -> **pg_stat_io**

- CONTEXT column = Buffer Access Strategy
- Improves cache hit ratio calculation

I/O

- Context = Normal, Bulkread, Bulkwrite, Vacuum

```
postgres=# \d pg_statio_all_tables
```

```
View "pg_catalog.pg_statio_all_tables"
```

Column	Type	Collation	Nullable	Default
relid	oid			
schemaname	name			
relname	name			
heap_blks_read	bigint			
heap_blks_hit	bigint			
idx_blks_read	bigint			
idx_blks_hit	bigint			
toast_blks_read	bigint			
toast_blks_hit	bigint			
tidx_blks_read	bigint			
tidx_blks_hit	bigint			

```
postgres=# \d pg_statio_all_indexes
```

```
View "pg_catalog.pg_statio_all_indexes"
```

Column	Type	Collation	Nullable	Default
relid	oid			
indexrelid	oid			
schemaname	name			
relname	name			
indexrelname	name			
idx_blks_read	bigint			
idx_blks_hit	bigint			

```
postgres=# \d pg_stat_io
```

```
View "pg_catalog.pg_stat_io"
```

Column	Type	Collation	Nullable
Default			
backend_type	text		
object	text		
context	text		
reads	bigint		
read_time	double precision		
writes	bigint		
write_time	double precision		
writebacks	bigint		
writeback_time	double precision		
extends	bigint		
extend_time	double precision		
op_bytes	bigint		
hits	bigint		
evictions	bigint		
reuses	bigint		
fsyncs	bigint		
fsync_time	double precision		
stats_reset	timestamp with time zone		

```
postgres=# \d pg_stat_database
```

```
View "pg_catalog.pg_stat_database"
```

Column	Type	Collation	Nullable	Default
datid	oid			
datname	name			
....				
blk_read_time	double precision			
blk_write_time	double precision			

I/O – Per Backend (≥ 18)

```
SELECT *  
FROM pg_stat_get_backend_io( pg_backend_pid() )
```

I/O - Demo

```
DROP TABLE IF EXISTS demo;  
CREATE TABLE demo ( id int, c1 text );  
INSERT INTO demo SELECT n FROM generate_series(1, 8000000) as n;
```

```
SELECT  
blks_read,  
blks_hit,  
ROUND(blks_hit/(blks_hit+blks_read::numeric)*100,  
2)  
FROM pg_stat_database  
WHERE datname = 'postgres';
```

blks_read	blks_hit	round
82	1567	95.03

(1 row)

```
SELECT  
backend_type,  
context,  
reads,  
hits,  
ROUND(hits/(reads+hits)::numeric * 100, 2)  
FROM pg_stat_io  
WHERE context = 'normal'  
AND backend_type = 'client backend'  
AND object = 'relation';
```

backend_type	context	reads	hits	round
client backend	normal	86	1578	94.83

(1 row)

I/O - Demo

VACUUM demo;

```
SELECT
blks_read,
blks_hit,
ROUND(
blks_hit/(blks_hit+blks_read::numeric)*100, 2)
FROM pg_stat_database
WHERE datname = 'postgres';
```

blks_read	blks_hit	round
19657	53499	73.13

(1 row)

```
SELECT
backend_type,
context,
reads,
hits,
ROUND(hits/(reads+hits)::numeric * 100, 2)
FROM pg_stat_io
WHERE context = 'normal'
AND backend_type = 'client backend'
AND object = 'relation';
```

backend_type	context	reads	hits	round
client backend	normal	193	37438	99.49

(1 row)

I/O - Demo

```
SELECT COUNT(*) FROM demo;
```

```
select
  blks_read,
  blks_hit,
  ROUND(
    blks_hit/
    (blks_hit+blks_read)::numeric)*100, 2) chr
from pg_stat_database
where datname = 'postgres';
```

blks_read	blks_hit	chr
58593	122667	67.67

(1 row)

```
select
  backend_type,
  context,
  reads,
  hits,
  ROUND(hits/(reads+hits)::numeric * 100, 2) chr
from
  pg_stat_io
where context = 'bulkread'
and backend_type = 'client backend'
and object = 'relation';
```

backend_type	context	reads	hits	chr
client backend	bulkread	6542	5371	45.09

(1 row)

```
select
  backend_type,
  context,
  reads,
  hits,
  ROUND(hits/(reads+hits)::numeric * 100, 2) chr
from
  pg_stat_io
where context = 'normal'
and backend_type = 'client backend'
and object = 'relation';
```

backend_type	context	reads	hits	round
client backend	normal	247	73745	99.67

(1 row)

Takeaways

- `pg_stat_*` views gives you observability into PG's workload
- Postgres 15 and 16
 - Improved stability of the system, shared memory vs background worker
 - `PG_STAT_IO` improves visibility into I/O contexts = better cache hit ratio calculations
- Postgres 17
 - Configurable SLRU cache sizes
- Postgres 18
 - More per-table vacuum stats
 - Parallel counters
 - Per-backend I/O
 - <https://www.postgresql.org/docs/current/release-18.html#RELEASE-18-MONITORING>

Thank you!

