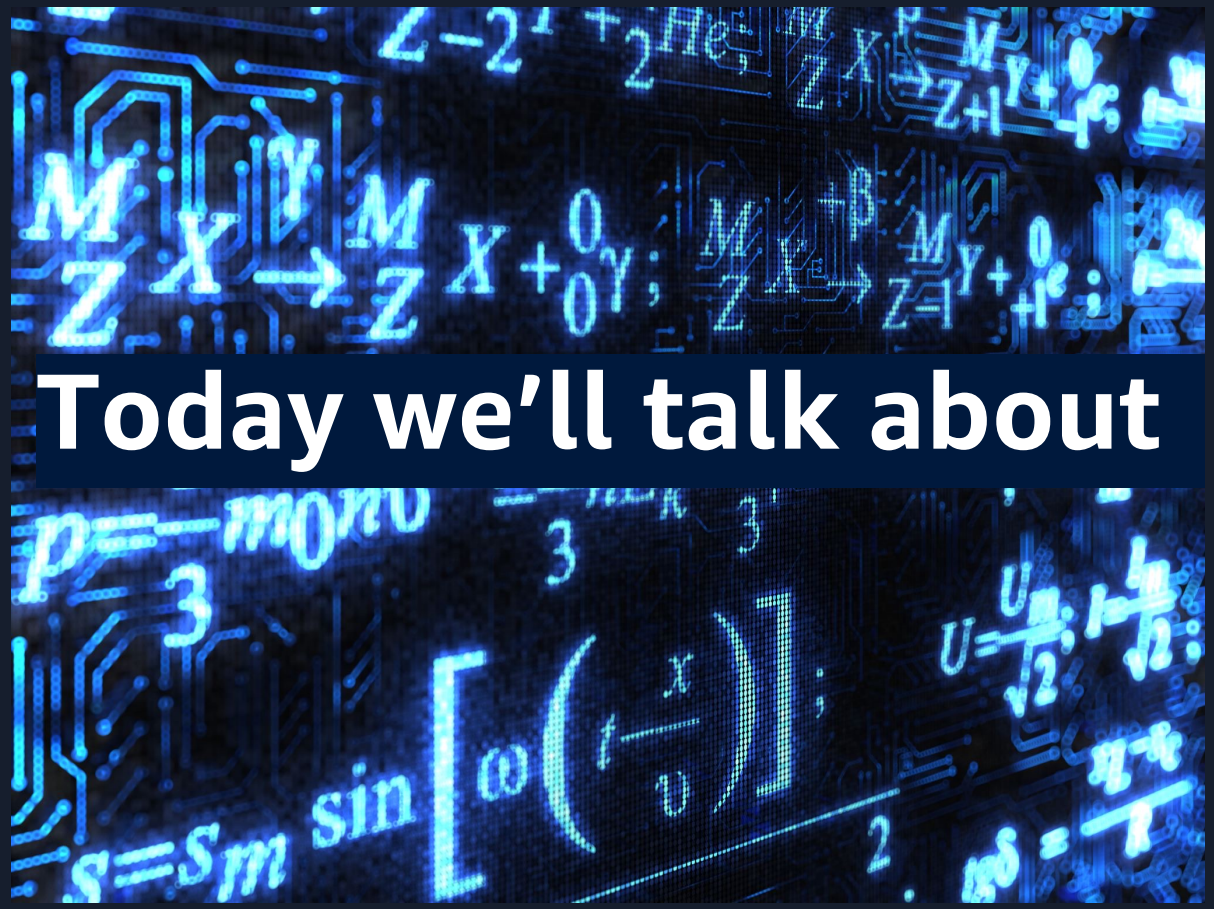
Understanding PostgreSQL statistics to optimize performance

Divya Sharma

Sr. RDS PostgreSQL SA



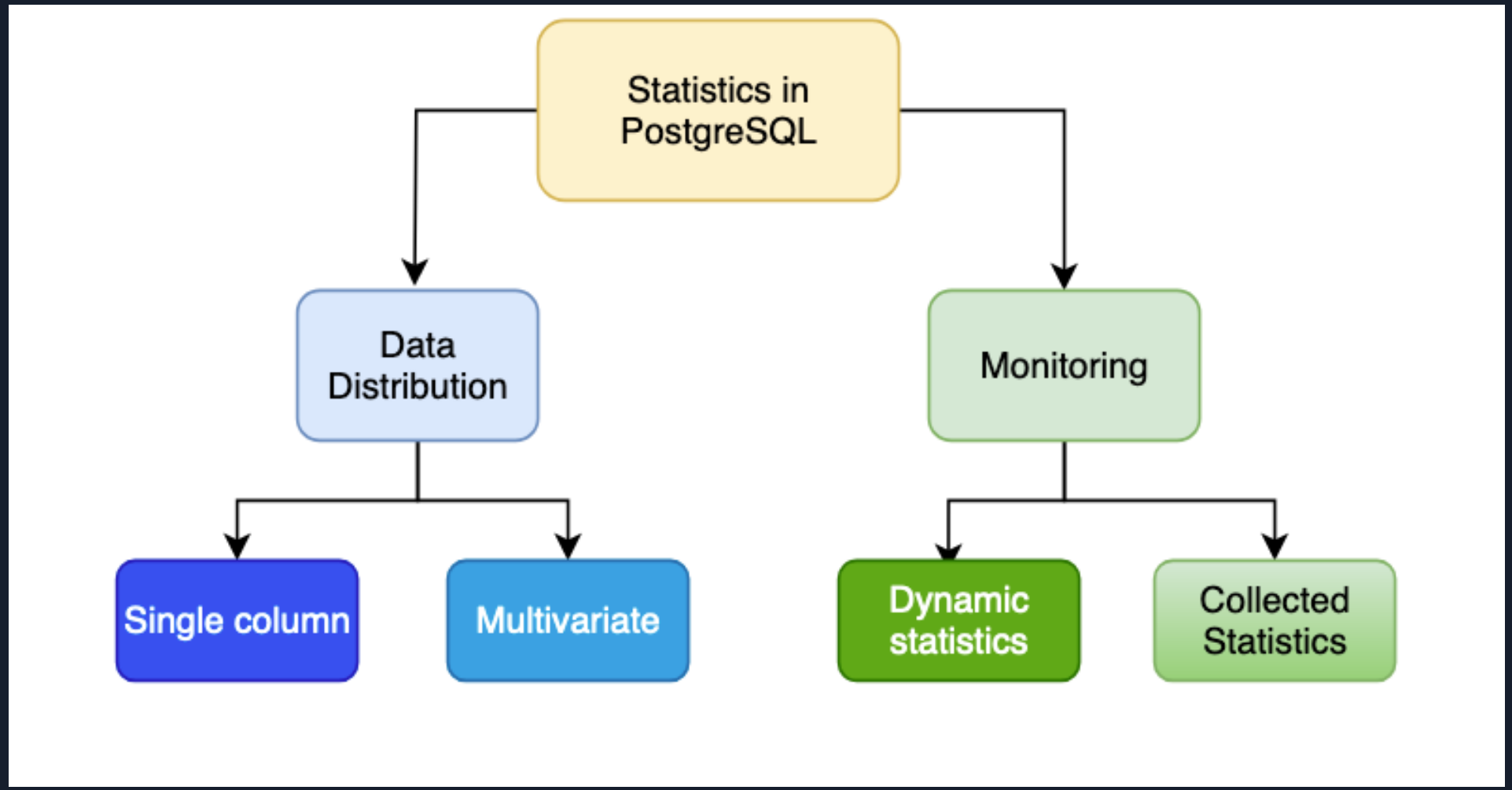
Today we'll talk about

- Types of statistics in PostgreSQL
- Data distribution statistics
- Monitoring statistics
- Troubleshooting example

But why do we need to know about them?

- To understand how queries are planned – to be able to optimize them.
- To understand when to create custom statistics by understanding correlation between data
- To know about different monitoring views available for better understanding of – when to run vaccum, when to add/remove indexes etc.

Statistics in PostgreSQL

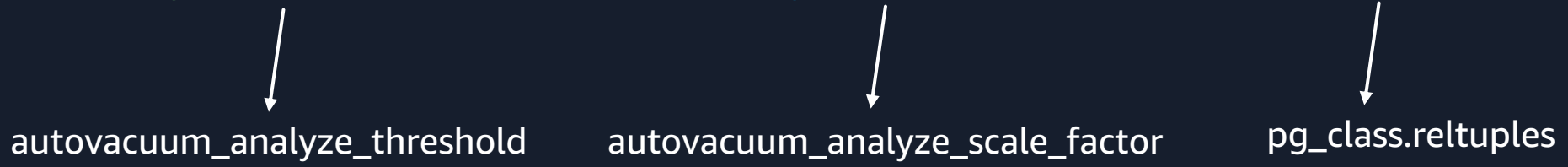


Data Distribution – single column statistics

Data Distribution – single column statistics

- Can be updated by running the “ANALYZE” command (on its own or with VACUUM)
- Can by triggered automatically by autovacuum worker if the following calculated threshold exceed total number of tuples inserted, updated, or deleted since the last ANALYZE :

analyze threshold = analyze base threshold + analyze scale factor * number of tuples



Note : The “ANALYZE” here has nothing to do with “EXPLAIN ANALYZE”

Data Distribution – single column statistics

- Information collected stores in pg_statistic system catalog (only readable by superuser)
- pg_stats is a view on top of pg_statistics which is readable by all.
- The amount of samples considered by ANALYZE depends on the default_statistics_target parameter.

Data Distribution – single column statistics

- Default value for default_statistics_target is 100, which means 100 most common values ;100 histogram bounds can be stored in those arrays.
- default_statistics_target can be set per column basis or globally for the entire database

```
postgres=> ALTER TABLE test_exp ALTER COLUMN a SET STATISTICS 100;
ALTER TABLE
postgres=> \d+ test_exp
                                   Table "public.test_exp"
 Column |  Type   | Collation | Nullable | Default | Storage | Stats target | Description
--------+---------+-----------+----------+---------+---------+--------------+-------------
 a      | integer |           | not null |         | plain   | 100          |
 b      | integer |           |          |         | plain   |              |
Indexes:
    "test_exp_pkey" PRIMARY KEY, btree (a)
Access method: heap
```

Note : Increasing the target causes a proportional increase in the time and space needed to do ANALYZE.

Data Distribution – single column statistics

```
postgres=> CREATE TABLE test_stats(id INT, name VARCHAR);
CREATE TABLE
postgres=> INSERT INTO test_stats VALUES (generate_series(1,10),'test'||generate_series(1,10));
INSERT 0 10
postgres=> INSERT INTO test_stats VALUES (generate_series(1,10),'test'||generate_series(1,10));
INSERT 0 10
postgres=> INSERT INTO test_stats VALUES (generate_series(1,10),'test'||generate_series(1,10));
INSERT 0 10
postgres=> INSERT INTO test_stats VALUES (generate_series(11,20),'test'||generate_series(11,20));
INSERT 0 10
postgres=>
postgres=> ANALYZE VERBOSE test_stats ;
INFO:  analyzing "public.test_stats"
INFO:  "test_stats": scanned 1 of 1 pages, containing 40 live rows and 0 dead rows; 40 rows in sample, 40 estimated total rows
ANALYZE
```

Data Distribution – single column statistics

```
postgres=> SELECT * FROM pg_stats WHERE tablename ='test_stats';
-[ RECORD 1 ]----------+-----------------------------------------------------------------------
schemaname             | public
tablename              | test_stats
attname                | id
inherited              | f
null_frac              | 0
avg_width              | 4
n_distinct             | -0.5
most_common_vals       | {1,2,3,4,5,6,7,8,9,10}
most_common_freqs      | {0.075,0.075,0.075,0.075,0.075,0.075,0.075,0.075,0.075,0.075}
histogram_bounds       | {11,12,13,14,15,16,17,18,19,20}
correlation            | 0.7551595
most_common_elems      |
most_common_elem_freqs |
elem_count_histogram   |
-[ RECORD 2 ]----------+-----------------------------------------------------------------------
schemaname             | public
tablename              | test_stats
attname                | name
inherited              | f
null_frac              | 0
avg_width              | 6
n_distinct             | -0.5
most_common_vals       | {test1,test10,test2,test3,test4,test5,test6,test7,test8,test9}
most_common_freqs      | {0.075,0.075,0.075,0.075,0.075,0.075,0.075,0.075,0.075,0.075}
histogram_bounds       | {test11,test12,test13,test14,test15,test16,test17,test18,test19,test20}
correlation            | -0.19043152
most_common_elems      |
most_common_elem_freqs |
elem_count_histogram   |
```

To help the planner predict the selectivity of inequality or range expressions, such as where id is between 5000–10000.

MCV helps the planner predict the selectivity of equality expressions, such as where name='test5'

When to run ANALYZE?

- After a bulk insert/delete on a relation
- Major change in data distribution
- Major version upgrade
- Recently added or dropped an index from the relation
- Estimated rows and actual returned rows do not match in the explain plan

```
Gather  (cost=53594.61..781789.24 rows=551554 width=509) (actual time=385.471..6913.891 rows=4571649 loops=1)
  Workers Planned: 4
  Workers Launched: 4
  Buffers: shared hit=494801 read=34059
  I/O Timings: shared/local read=129.933
```

Note : For some of these changes, autoanalyze will run automatically. Like for bulk insert/delete.

Data Distribution – multivariate statistics

Data Distribution – multivariate statistics

- Multiple columns used in the query clauses are sometimes correlated, and planner normally assumes that multiple conditions are independent of each other.
- Regular statistics = per column = don’t capture knowledge about cross-column correlation
- Solution = Compute multivariate statistics using CREATE STATISTICS command
- Statistics object created stored in pg_statistic_ext (pg_stats_ext is a publicly readable view)
- Data collection done by ANALYZE or autoanalyze
- Functional Dependencies, Multivariate N-distint, Multivariate MCV

Data Distribution – multivariate statistics

- Functional Dependencies :

```
CREATE STATISTICS stts (dependencies) ON city, zip FROM
zipcodes;

ANALYZE zipcodes;

SELECT stxname, stxkeys, stxddependencies FROM
pg_statistic_ext join pg_statistic_ext_data on (oid = stxoid)
WHERE stxname = 'stts';

stxname | stxkeys | stxddependencies

--------+---------+----------------------------------

stts    | 1 5     | {"1 => 5": 1.000000, "5 => 1": 0.423130}

(1 row)
```

Functional Dependencies example

```
postgres=> CREATE TABLE ext_stats(a int, b int);
CREATE TABLE
postgres=> INSERT INTO ext_stats SELECT x/1000, x/10000 FROM generate_series(1,1000000) s(x);
INSERT 0 1000000
```

n_distinct a=1000 ;
n_distinct b=100

```
postgres=> explain analyze select * from ext_stats where a=1 and b=0;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..11676.00 rows=10 width=8) (actual time=0.330..64.876 rows=1000 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on ext_stats  (cost=0.00..10675.00 rows=4 width=8) (actual time=35.696..56.307 rows=333 loops=3)
         Filter: ((a = 1) AND (b = 0))
         Rows Removed by Filter: 333000
 Planning Time: 0.049 ms
 Execution Time: 65.001 ms
(8 rows)
```

```
postgres=> create statistics s_ext_depend(dependencies) on a,b from ext_stats ;
CREATE STATISTICS
postgres=> ANALYZE ext_stats;
ANALYZE
```

Functional Dependencies example

```
postgres=> explain analyze select * from ext_stats where a=1 and b=0;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..11774.60 rows=996 width=8) (actual time=0.318..64.085 rows=1000 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on ext_stats  (cost=0.00..10675.00 rows=415 width=8) (actual time=35.004..55.364 rows=333 loops=3)
         Filter: ((a = 1) AND (b = 0))
         Rows Removed by Filter: 333000
 Planning Time: 0.053 ms
 Execution Time: 64.209 ms
(8 rows)
```

```
postgres=> explain analyze select * from ext_stats where a=1 and b=0;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..11676.00 rows=10 width=8) (actual time=0.330..64.876 rows=1000 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on ext_stats  (cost=0.00..10675.00 rows=4 width=8) (actual time=35.696..56.307 rows=333 loops=3)
         Filter: ((a = 1) AND (b = 0))
         Rows Removed by Filter: 333000
 Planning Time: 0.049 ms
 Execution Time: 65.001 ms
(8 rows)
```

Data Distribution – multivariate statistics

- Multivariate n-distinct count:

```
CREATE STATISTICS stts2 (ndistinct) ON city, state, zip FROM
zipcodes; ANALYZE zipcodes;

SELECT stxkeys AS k, stxdndistinct AS nd FROM pg_statistic_ext
join pg_statistic_ext_data on (oid = stxoid) WHERE stxname =
'stts2';

-[ RECORD 1 ]-------------------------------------

k  | 1 2 5

nd | {"1, 2": 33178, "1, 5": 33178, "2, 5": 27435, "1, 2, 5":
33178}

(1 row)
```

Data Distribution – multivariate statistics

- Multivariate MCV lists:

  - CREATE STATISTICS stts3 (mcv) ON city, state FROM zipcodes;
  - data collected only for those groups of columns appearing together in a statistics object
  - defined with the mcv option.
  - Calculates the frequencies of most common values (mcv) together for the specified columns

Only consider creating statistics objects for columns which are actually used in conditions together

Data Distribution statistics and read replica

```
postgres=> create statistics s_ext_depend(dependencies) on a,b from ext_stats ;
ERROR:  cannot execute CREATE STATISTICS in a read-only transaction
```

```
postgres=> analyze;
ERROR:  cannot execute ANALYZE during recovery
```

Because a RR can only read from the disk and not write, it will use the data distribution statistics from the primary.

Monitoring Statistics

Monitoring Statistics

- Collection and reporting of server activity
- Can by dynamic – what's happening in my server right now - controlled by **track_activities**
- Collected statistics – controlled by **Cumulative Statistics System (v15+) ;** by **Statistics Collector** (v14 and below)

Monitoring Statistics - Dynamic Statistics Views

Information about exactly what is going on in the system right now

pg_stat_activity
pg_stat_replication
pg_stat_wal_receiver
pg_stat_subscription
pg_stat_progress_create_index
pg_stat_progress_vacuum
pg_stat_progress_basebackup
pg_stat_recovery_prefetch
pg_stat_ssl
pg_stat_gssapi
pg_stat_progress_analyze
pg_stat_progress_cluster
pg_stat_progress_copy

Note : The reporting and collection of these statistics is independent of the cumulative statistics system

Dynamic Statistics Views – pg_stat_activity

```
postgres=> select * from pg_stat_activity where backend_type='client backend' and pid!=20360;
-[ RECORD 1 ]----+-------------------------------------------------
datid            | 5
datname          | postgres
pid              | 20117
leader_pid       |
usesysid         | 16397
usename          | postgres
application_name | psql
client_addr      | 172.31.36.18
client_hostname  |
client_port      | 33012
backend_start    | 2024-04-11 18:18:29.079321+00
xact_start       | 2024-04-11 18:28:28.602608+00
query_start      | 2024-04-11 18:28:31.376951+00
state_change     | 2024-04-11 18:28:31.420607+00
wait_event_type  | Client
wait_event       | ClientRead
state            | idle in transaction
backend_xid      |
backend_xmin     |
query_id         | -7652462281445876340
query            | select count(*) from foo;
backend_type     | client backend
```

log_min_duration_statement

idle_in_transaction_session_timeout

Dynamic Statistics Views – pg_stat_activity

```
postgres=> select * from pg_stat_activity where backend_type='walsender';
-[ RECORD 1 ]----+------------------------------------------------------------------------------
datid            |
datname          |
pid              | 17601
leader_pid       |
usesysid         | 16398
usename          | rdsrepladmin
application_name | walreceiver
client_addr      | 10.4.1.75
client_hostname  |
client_port      | 16058
backend_start    | 2024-04-11 17:18:41.989094+00
xact_start       |
query_start      | 2024-04-11 17:18:42.006455+00
state_change     | 2024-04-11 17:18:42.006485+00
wait_event_type  | Activity
wait_event       | WalSenderMain
state            | active
backend_xid      |
backend_xmin     |
query_id         |
query            | START_REPLICATION SLOT "rds_eu_west_1_db_kiw5yj47fmkqbcnu45ccascjum" 17E/5C000000 TIMELINE 1
backend_type     | walsender
```

Dynamic Statistics Views – pg_stat_replication

```
postgres=> select * from pg_stat_replication;
-[ RECORD 1 ]----+------------------------------
pid              | 17601
usesysid         | 16398
usename          | rdsrepladmin
application_name | walreceiver
client_addr      | 10.4.1.75
client_hostname  |
client_port      | 16058
backend_start    | 2024-04-11 17:18:41.989094+00
backend_xmin     |
sent_lsn         | 17E/CC000110
write_lsn        | 17E/CC000110
flush_lsn        | 17E/CC000110
replay_lsn       | 17E/CC000110
flush_lag        |
replay_lag       |
sync_priority    | 0
sync_state       | async
reply_time       | 2024-04-11 19:37:35.464595+00
```

Run on the primary instance

Dynamic Statistics Views

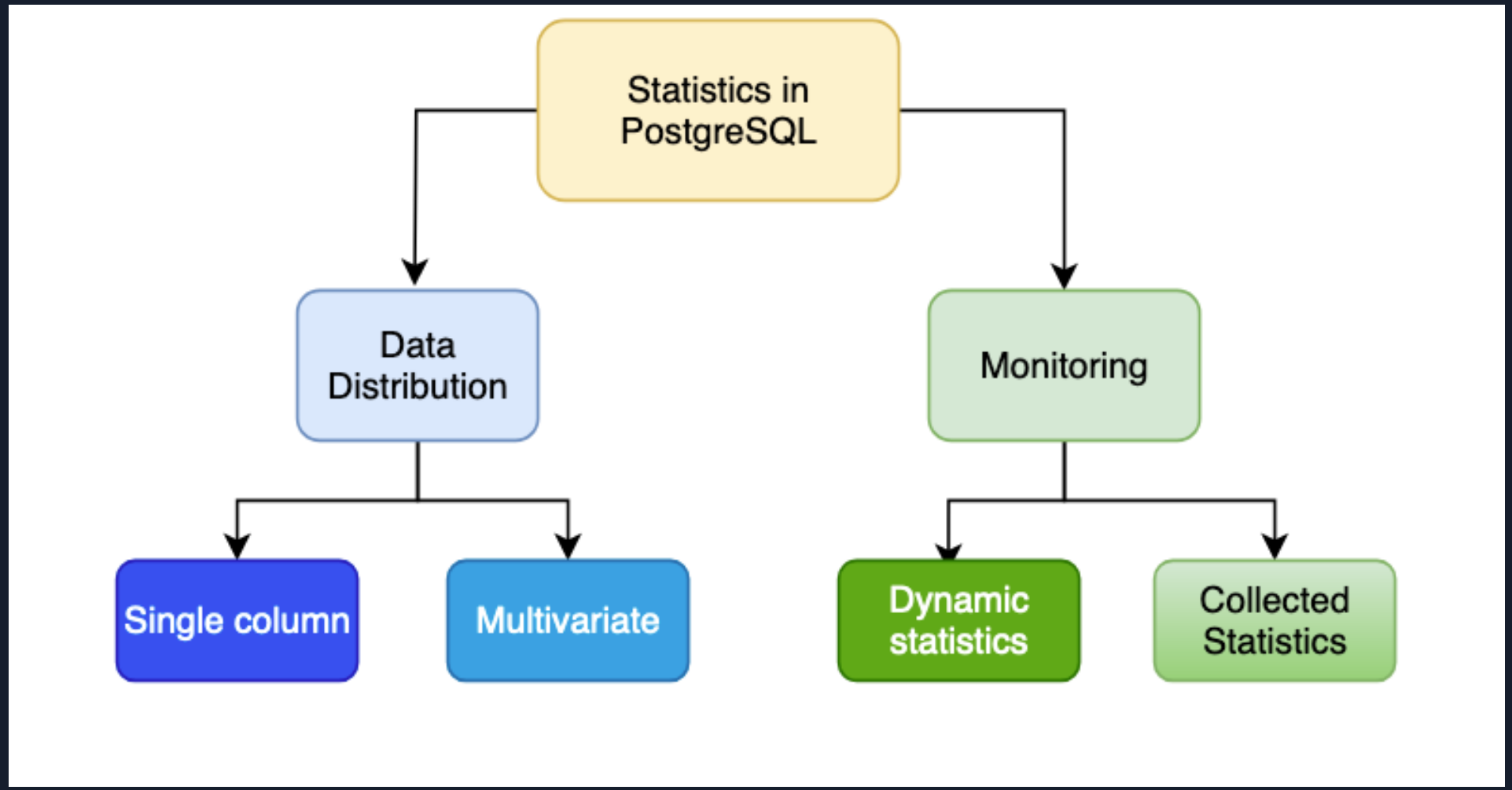
- pg_stat_progress_vacuum

```
-[ RECORD 1 ]-------+------------------
pid                 | 104701
duration            | 03:21:51.330818
waiting             | f
mode                | regular
database            | analytics
table               | events
phase               | vacuuming indexes
table_size          | 1188 GB
total_size          | 1682 GB
scanned             | 601 GB
vacuumed            | 571 GB
```

- pg_stat_progress_create_index

https://dataegret.de/2017/10/deep-dive-into-postgres-stats-pg_stat_progress_vacuum/

Statistics in PostgreSQL



Monitoring Statistics - Collected Statistics Views

Information about relations/database metadata

pg_stat_archiver
pg_stat_bgwriter
pg_stat_database
pg_stat_database_conflicts
pg_stat_user_tables
pg_stat_xact_user_tables
pg_stat_user_indexes
pg_stat_slru
pg_stat_replication_slots
pg_stat_io

Note : The reporting and collection of these statistics is independent of the cumulative statistics system

Monitoring Statistics - Collected Statistics Views

- Can be reset using - select pg_stat_reset(); - for the database you are connected to.
- Every PostgreSQL process collects statistics locally, then updates the shared memory at appropriate intervals.
- Clean shutdown – permanent copy in pg_stat directory ; unclean shutdown – all counters reset.

Collected Statistics Views - pg_stat_bgwriter

```
postgres=> select * from pg_stat_bgwriter;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 24458
checkpoints_req       | 15
checkpoint_write_time | 3595158
checkpoint_sync_time  | 80912
buffers_checkpoint    | 44392
buffers_clean         | 0
maxwritten_clean      | 0
buffers_backend       | 18194
buffers_backend_fsync | 0
buffers_alloc         | 36090
stats_reset           | 2024-01-17 19:56:46.641452+00
```

Collected Statistics Views - pg_stat_user_tables

```
postgres=> select * from pg_stat_user_tables order by (n_tup_upd+n_tup_del) desc;
-[ RECORD 1 ]-------+------------------------------
relid               | 24712
schemaname          | public
relname             | test
seq_scan            | 5
last_seq_scan       | 2024-04-07 10:55:08.074487+00
seq_tup_read        | 2522
idx_scan            | 0
last_idx_scan       |
idx_tup_fetch       | 0
n_tup_ins           | 1000
n_tup_upd           | 250
n_tup_del           | 749
n_tup_hot_upd       | 0
n_tup_newpage_upd   | 250
n_live_tup          | 251
n_dead_tup          | 0
n_mod_since_analyze | 0
n_ins_since_vacuum  | 0
last_vacuum         |
last_autovacuum     | 2024-03-09 02:04:04.405814+00
last_analyze        | 2024-04-08 22:31:36.81463+00
last_autoanalyze    | 2024-03-09 02:04:04.413322+00
vacuum_count        | 0
autovacuum_count    | 3
analyze_count       | 1
autoanalyze_count   | 4
```

Collected Statistics Views - pg_stat_user_tables

```
postgres=> select * from pg_stat_user_tables where idx_scan>0 order by idx_scan desc;
-[ RECORD 1 ]-------+------------------------------
relid               | 24672
schemaname          | public
relname             | pgbench_accounts
seq_scan            | 6
last_seq_scan       | 2024-03-27 14:32:59.35374+00
seq_tup_read        | 300007
idx_scan            | 7
last_idx_scan       | 2024-03-27 14:34:36.803696+00
idx_tup_fetch       | 5
n_tup_ins           | 100000
n_tup_upd           | 0
n_tup_del           | 0
n_tup_hot_upd       | 0
n_tup_newpage_upd   | 0
n_live_tup          | 100000
n_dead_tup          | 0
n_mod_since_analyze | 0
n_ins_since_vacuum  | 0
last_vacuum         | 2024-03-08 23:59:01.646285+00
last_autovacuum     | 2024-03-08 23:59:06.665808+00
last_analyze        | 2024-04-08 22:31:36.66896+00
last_autoanalyze    | 2024-03-08 23:59:06.726678+00
vacuum_count        | 1
autovacuum_count    | 1
analyze_count       | 2
autoanalyze_count   | 1
```

Collected Statistics Views - pg_stat_user_indexes

```
postgres=> select * from pg_stat_user_indexes where relname='pgbench_accounts';
-[ RECORD 1 ]-+------------------------------
relid         | 24672
indexrelid    | 24686
schemaname    | public
relname       | pgbench_accounts
indexrelname  | pgbench_accounts_pkey
idx_scan      | 7
last_idx_scan | 2024-03-27 14:34:36.803696+00
idx_tup_read  | 200005
idx_tup_fetch | 4
-[ RECORD 2 ]-+------------------------------
relid         | 24672
indexrelid    | 24757
schemaname    | public
relname       | pgbench_accounts
indexrelname  | idx2
idx_scan      | 0
last_idx_scan |
idx_tup_read  | 0
idx_tup_fetch | 0
```

Collected Statistics Views - pg_stat_io

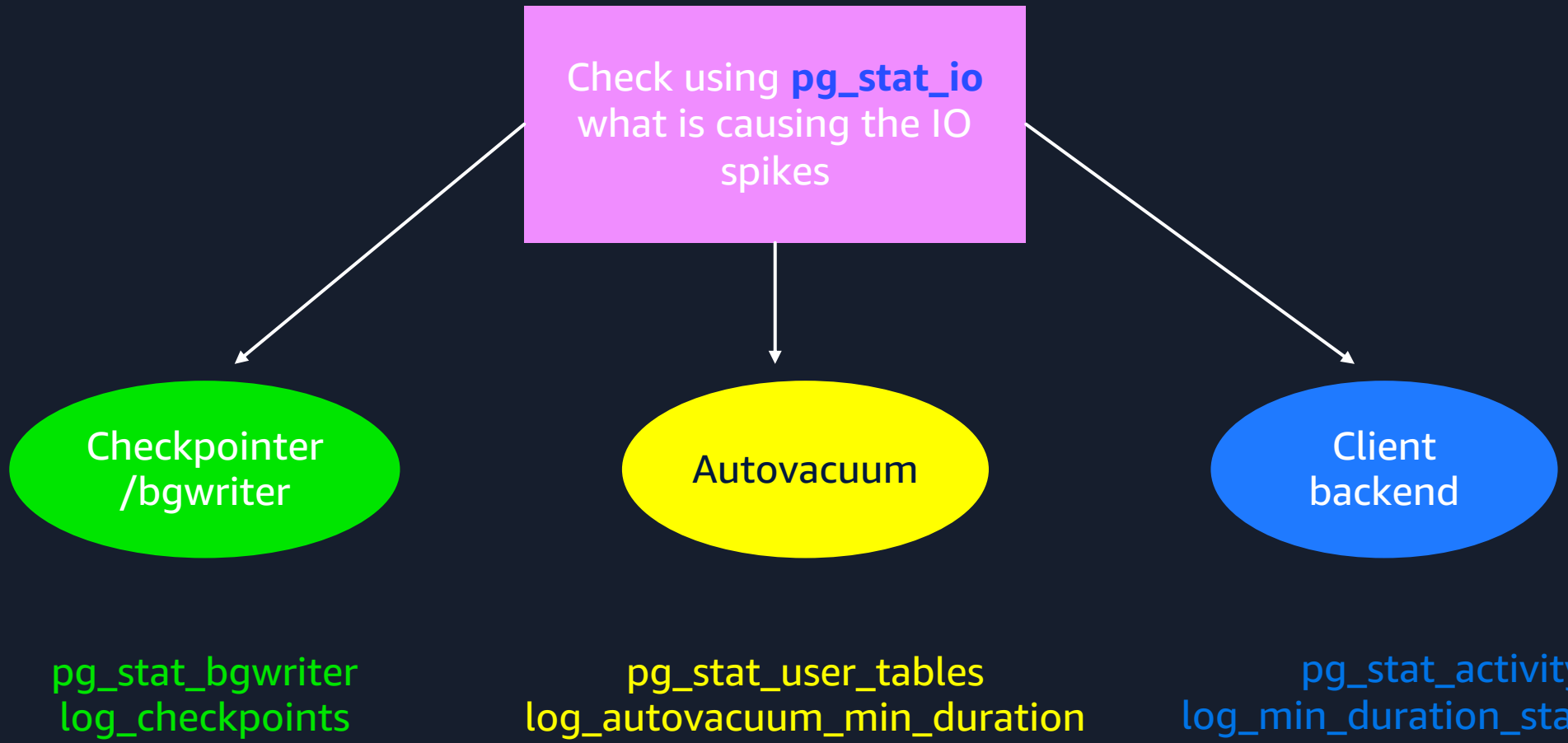
```
postgres=> select * from pg_stat_io order by reads+writes desc;
-[ RECORD 1 ]--+------------------------------
backend_type    | checkpointer
object          | relation
context         | normal
reads           |
read_time       |
writes          | 44414
write_time      | 1242.583
writebacks      | 44414
writeback_time  | 1497.037
extends         |
extend_time     |
op_bytes        | 8192
hits            |
evictions       |
reuses          |
fsyncs          | 27400
fsync_time      | 51610.241
stats_reset     | 2024-01-17 19:56:46.641452+00
```

The backend_type is same as what you see in pg_stat_activity :

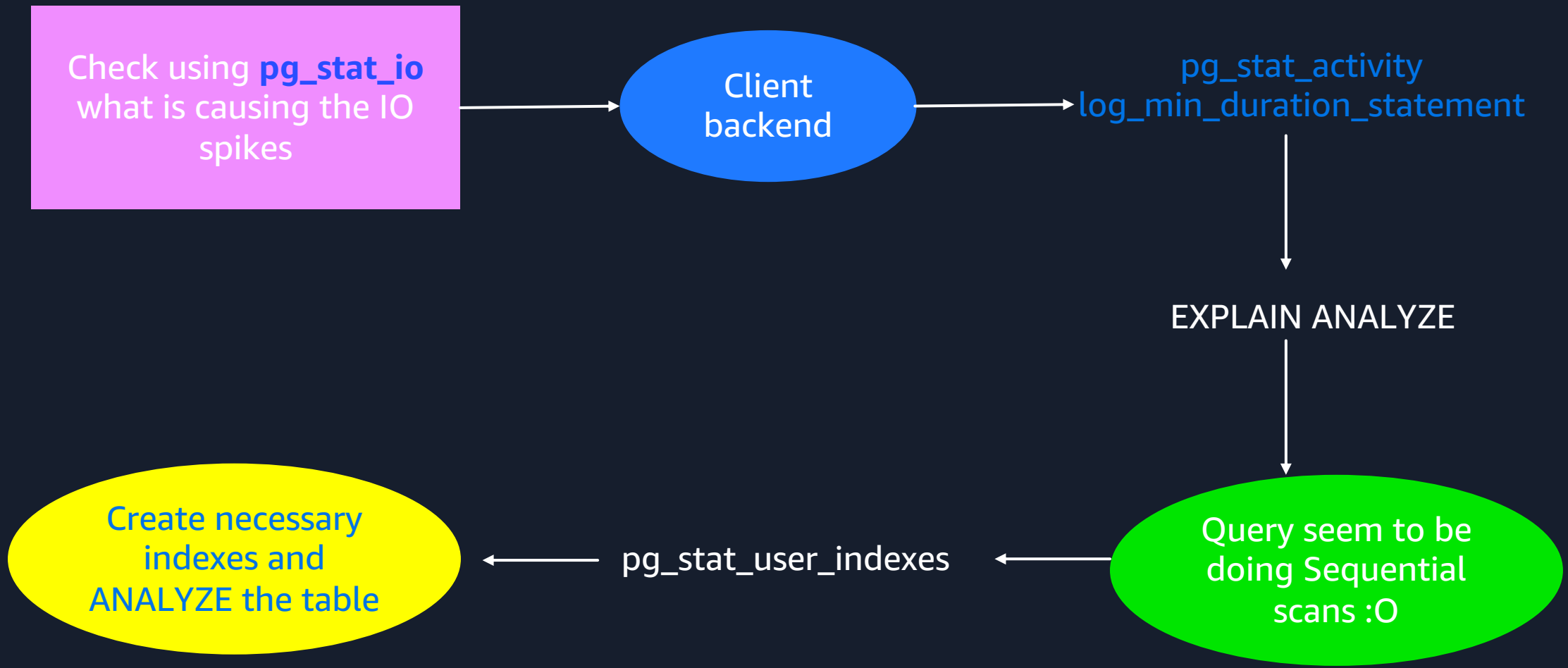
autovacuum launcher,
autovacuum worker,
logical replication launcher,
logical replication worker,
parallel worker,
background writer,
client backend etc.

Troubleshooting approach using statistics

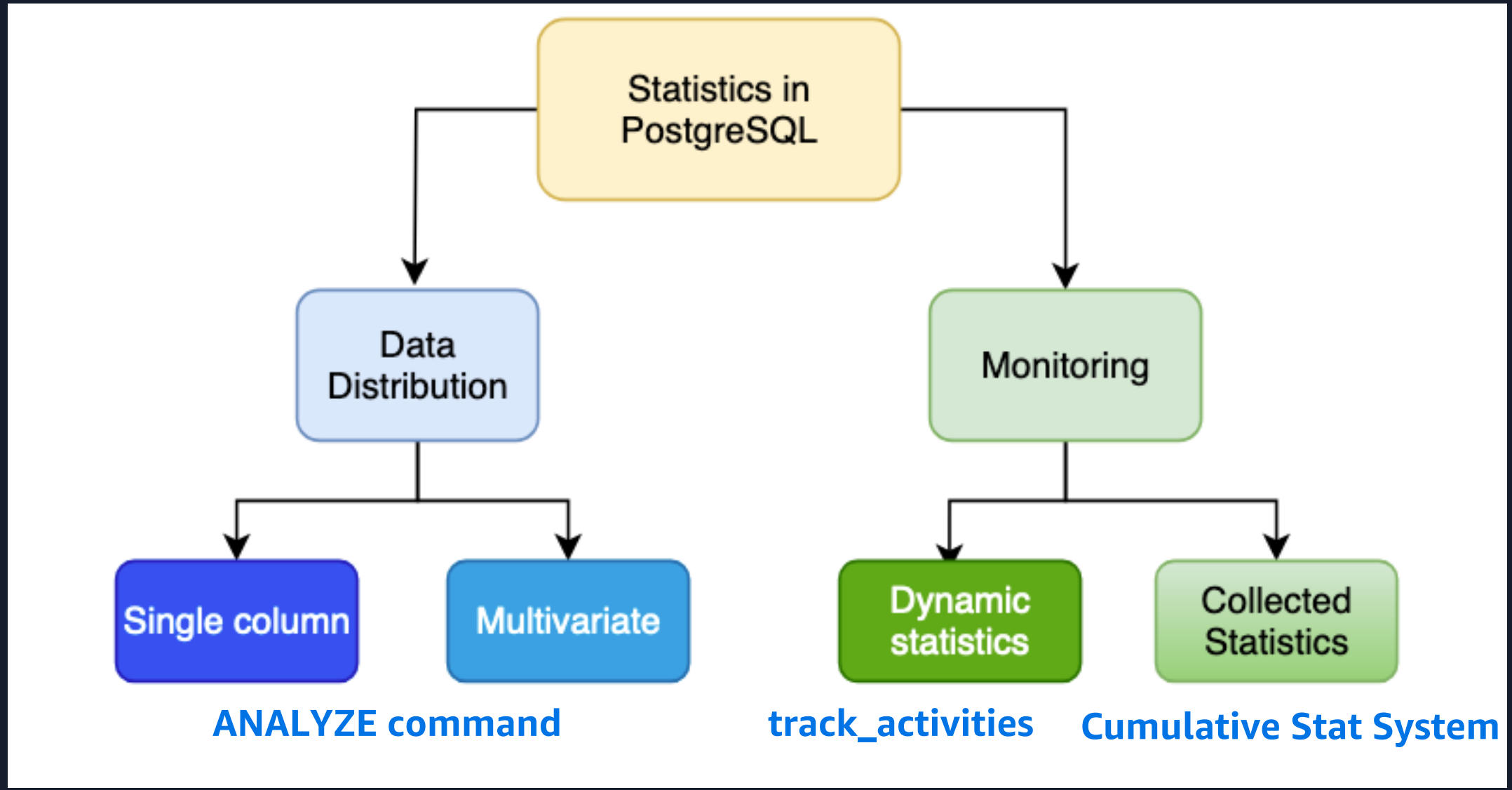
#1 - I see very high write IO spikes in my metrics. What should I do?



#1 - I see very high write IO spikes in my metrics. What should I do?



Key Takeaways



Key Takeaways

- Data distribution statistics are different from monitoring statistics
- Statistics collected by ANALYZE are used by planner to plan queries
- PostgreSQL enables you to create multivariate statistics for correlated metrics
- More samples collected to ANALYZE relation, more space and time needed
- The monitoring statistics can provide important information related to – vacuum, indexes, IO usage by processes etc.

Thank you!

Divya Sharma

https://www.linkedin.com/in/divyasharma95/