credativ

PostgreSQL Performance Tuning

Michael Banck <michael.banck@credativ.de>

PGDay UK 2025, London

credativ GmbH

- Founded 1999 in Mönchengladbach
- Strong ties to the Open-Source community
- 40 Open-Source specialists
- Consulting, Development, Trainings, Support (3rd-Level / 24x7)
- Open-Source infrastructure with Debian, Kubernetes and Proxmox
- Open-Source databases with PostgreSQL
- Open-Source DevOps with Ansible, Puppet, Terraform and others
- Independent owner-operated company again since 2025

PostgreSQL Performance Tuning

credativ

- credativ has been doing Postgres trainings for many years

```
commit cbd1c85ebfcad35d386ebacd3ebc1d94f9dd437b
Author: Peter Eisentraut <peter.eisentraut@credativ.de>
Date:   Tue Jan 10 08:29:31 2006 +0000

    Initial revision
```

- Performance-Tuning is a (very) wide subject
- Compilation of various topics from the training:
  - Buffer-cache and memory parameters tuning
  - Storage parameters and checkpoint tuning
  - Autovacuum tuning
  - Query tuning

credativ

Buffer-Cache and Memory Parameters Tuning

Buffer-Cache Tuning
Shared Buffers

credativ

- Data blocks are copied as 8 KB memory pages into buffer-cache / shared buffers
- Buffer-cache is single-source-of-truth view on the data for all backends
  - All read queries go through the buffer cache (taking up 1/8 max)
  - All data changes are first done in it
- Pages are kept in buffer-pool based on access frequency
  - Clock-sweep algorithm
- Changed (dirty) pages are written back to storage on demand
  - Background writer writes back dirty pages if there is pressure
  - Checkpointer periodically writes back dirty pages during checkpoints

Buffer-Cache Tuning

shared_buffers

credativ

```
#shared_buffers = 128MB   # (change requires restart)
shared_buffers = 0.25*RAM
shared_memory_size = <read-only>
```

- Default in most cases too small, 1/10 to 4/10 of RAM, usually 1/4
- If whole database fits in RAM, `shared_buffers` = database size
- Some operations need to scan the whole buffer pool
- Either considerably more or less than 50% of RAM, due to double-buffering
- `shared_memory_size` shows size of shared memory (buffer-cache + others)

Buffer-Cache Tuning

pg_buffercache

credativ

- Contrib extension that displays the shared buffers content

```
postgres=# SELECT c.relname, pg_size_pretty(COUNT(*)*8192) AS buffer_size,
pg_size_pretty(pg_relation_size(c.oid)) as relation_size,
ROUND(100.0 * COUNT(*) /
  (SELECT setting FROM pg_settings WHERE name = 'shared_buffers')::integer, 2) AS buffers_pct,
ROUND(COUNT(*)*8192*100/ pg_relation_size(c.oid)::numeric, 2) AS relation_pct,
ROUND(100.0 * (COUNT(*) FILTER (WHERE b.isdirty)::float/COUNT(*))::numeric, 2) AS dirty_pct
FROM pg_class c INNER JOIN pg_buffercache b ON b.relfilenode = c.relfilenode
INNER JOIN pg_database d ON ( b.reldatabase = d.oid AND d.datname = Current_database() )
GROUP BY c.relname, c.oid ORDER BY pg_total_relation_size(c.oid) DESC LIMIT 5;
        relname        | buffer_size | relation_size | buffers_pct | relation_pct | dirty_pct
-----------------------+-------------+---------------+-------------+--------------+-----------
 pgbench_accounts      | 63 MB       | 1294 MB       |       48.99 |         4.85 |     89.11
 pgbench_accounts_pkey | 64 MB       | 214 MB        |       50.29 |        30.05 |     50.39
 pgbench_history       | 832 kB      | 3072 kB       |        0.63 |        27.08 |     93.27
 pgbench_tellers       | 72 kB       | 72 kB         |        0.05 |       100.00 |    100.00
 pgbench_branches      | 8192 bytes  | 8192 bytes    |        0.01 |       100.00 |    100.00
(5 rows)
```

Memory Parameters

min_dynamic_shared_memory

credativ

```
#min_dynamic_shared_memory = 0        # (change requires restart)
min_dynamic_shared_memory = 2GB
```

- Additionally allocated shared memory area for dynamic shared memory
  - Mostly used by parallel queries
- Statically allocated at Postgres instance startup
  - Saves overhead of constantly allocating shared memory segments
  - Recommended when huge pages are used

Memory Parameters
SLRU Buffers

```
#commit_timestamp_buffers = 0   # memory for pg_commit_ts (0 = auto)
#multixact_offset_buffers = 16  # memory for pg_multixact/offsets
#multixact_member_buffers = 32  # memory for pg_multixact/members
#notify_buffers = 16            # memory for pg_notify
#serializable_buffers = 32      # memory for pg_serial
#subtransaction_buffers = 0     # memory for pg_subtrans (0 = auto)
#transaction_buffers = 0        # memory for pg_xact (0 = auto)
```

- Simple least-recently-used (SLRU) buffers
- Available since PG17
- Allows individual sizing of those buffers / shared memory segments

https://p2d2.cz/files/p2d2-2025-herrera-slru.pdf

Memory Parameters

max_connections

```
max_connections = 100                # (change requires restart)
#superuser_reserved_connections = 3 # (change requires restart)
superuser_reserved_connections = 10
```

- Additional connection attempts are rejected
- `superuser_reserved_connections` as administrative reserve
- Do not set too high (several thousand)
  - Each additional `max_connections` value takes up around 50kB shared memory
- Best tuned according to historical data (e.g. `check_postres_backends`)
- Possibly use a connection pooler

Memory Parameters

effective_cache_size

```
#effective_cache_size = 4GB
effective_cache_size = 0.5*RAM
```

- Amount of memory that is available as file system cache
- 1/2 to 3/4 of RAM
- Just a (planner) hint, not actually allocated

```
$ free -m
             total       used       free     shared    buffers     cached
Mem:        258468     223805      34663          0        283     219186
-/+ buffers/cache:       4335    *254133*
Swap:          951          2        949
```

https://www.cybertec-postgresql.com/en/effective_cache_size-a-practical-example/

Memory Parameters

work_mem

credativ

```
#work_mem = 4MB
```

- Memory used by Postgres for each plan node / parallel worker
  - Sorting (`ORDER BY`, `DISTINCT`, merge joins)
  - Hash tables (hash joins, hash-based aggregates/executions of `IN` subqueries)
  - Larger sort or hash operations spill to temporary storage afterwards
- Largely dependent on application / `max_connections` / active backends
- Setting `log_temp_files = 0` writes queries with too little `work_mem` into log

```
LOG: temporary file: path "base/pgsql_tmp/pgsql_tmp157828.1.fileset/1.0", size 8192
STATEMENT: alter table pgbench_accounts add primary key (aid)
```

Memory Parameters

hash_mem_multiplier

```
#hash_mem_multiplier = 2.0 # 1.0 till PG14
hash_mem_multiplier = 2.5 – 4.0
```

- work_mem multiplier for hash-based operations
- Hash-based nodes are more problematic than sorts when spilling to disk
- Rather set work_mem lower and increase hash_mem_multiplier

```
->  Hash Join  (cost=...)
      Hash Cond: (...)
      Buffers: shared hit=2138319, temp read=3991 written=7017
      ->  HashAggregate  (cost=...)
            Group Key: ...
            Batches: 5  Memory Usage: 262193kB  Disk Usage: 32344kB
```

Memory Parameters

{maintenance,autovacuum}_work_mem

credativ

```
#maintenance_work_mem = 64MB
maintenance_work_mem = 1GB
#autovacuum_work_mem = -1
```

- VACUUM, CREATE INDEX, ALTER TABLE ADD FOREIGN KEY
- Up to 1/10 of RAM, or 1 GB
- Index creation can use a lot of memory when doing restore in parallel
- autovacuum_work_mem accordingly (same as maintenance_work_mem by default)
- VACUUM can use only 1 GB up to PG16

Memory Parameters

vacuum_buffer_usage_limit

credativ

```
#vacuum_buffer_usage_limit = 2MB
```

- New parameter in PG16
- Size of ring-buffer for VACUUM
  - At most 1/8 of shared buffers
- Increasing it accelerates vacuum, but uses up more shared buffers
- Can also be set in VACUUM directly via BUFFER_USAGE_LIMIT option

credativ

Storage Parameters and Checkpoint Tuning

Storage Parameters

random_page_cost

```
#random_page_cost = 4.0
random_page_cost = 1.1
```

- Ratio between random and sequential storage access
- In case of SSDs/NVME storage values close to 1.0 should be used
- Pushes the planner to prefer index scans

Storage Parameters

{effective,maintenance}_io_concurrency,io_combine_limit

```
#effective_io_concurrency = 1 # PG18+: 16
effective_io_concurrency = 200
#maintenance_io_concurrency = 10
maintenance_io_concurrency = 200
#io_combine_limit = 128kB # since PG17
```

- Number of parallel I/O operations and read-ahead/prefetch distance
- `effective_io_concurrency` used for bitmap heap scans
- `maintenance_io_concurrency` used for `ANALYZE` since version 14

Storage Parameters

{effective,maintenance}_io_concurrency,io_combine_limit

credativ

- ANALYZE-timings for maintenance_io_concurrency/io_combine_limit values:

Version	0	1	5	10	50	500
13	-	-	-	61.95	-	-
14	66.34	97.60	16.91	9.53	5.05	4.75
15	66.32	90.60	16.55	9.68	5.06	4.95
16	60.37	58.31	10.48	6.31	4.68	4.86
17/8	-	-	15.79	12.96	12.89	12.87
17/32	-	-	9.19	5.45	4.51	4.60
17/128	57.13	54.23	9.03	5.41	4.50	4.51
17/256	-	-	9.05	5.46	4.45	4.50

https://www.credativ.de/en/blog/postgresql-en/quick-benchmark-analyze-vs-maintenance_io_concurrency/

Storage Parameters

{wal,default_toast}_compression

credativ

```
#default_toast_compression = pglz
default_toast_compression = lz4 # PG14+
#wal_compression = off
wal_compression = on # PG15+: pglz/lz4/zst
```

- Compression algorithm for WAL and TOAST
  - Only compresses full-page-images (FPIs) in WAL
- `lz4` much faster/uses fewer resources as `pglz` for TOAST
- `zst` compresses better than `lz4` and is roughly as fast as `pglz` for WAL

https://www.credativ.de/en/blog/postgresql-en/toasted-jsonb-data-in-postgresql-performance-tests-of-different-compression-algorithms/

Checkpoint Tuning

Checkpoints and Full-Page-Images

credativ

- Transaction log (write-ahead log, WAL) maintains WAL files on disk
  - WAL is synced on each commit before client gets commit message
- Modified table/index data is persisted into data directory periodically during checkpoints
- WAL is replayed since last checkpoint (redo-point) during crash recovery
- First change of a page after a checkpoint is written into WAL as FPI
- Further modifications until the next checkpoint are much smaller
- Automatic checkpoints are either time (`time`) or WAL based (`wal`)

```
LOG: checkpoint starting: wal
```

Checkpoint Tuning

{min,max}_wal_size, checkpoint_timeout

credativ

```
min_wal_size = 80MB
#max_wal_size = 1GB
max_wal_size = 16GB
#checkpoint_timeout = 5min
checkpoint_timeout = 15min
```

- Tries not to surpass `max_wal_size`
- Checkpoints after `checkpoint_timeout` or at ca. 1/3 `max_wal_size`
  - `max_wal_size / (2.0 + checkpoint_completion_target)`
- Advantage: less FPIs/WAL
- Disadvantage: larger `pg_wal`, longer crash recovery
- Increase `max_wal_size` until checkpoints are time-based

credativ

Autovacuum Tuning

Autovacuum Tuning

MVCC - Multi-Version Concurrency Control

credativ

- `DELETE` marks rows as deleted
- Concurrent transactions still need to read the old rows
- `UPDATE = DELETE + INSERT`
- Old/obsolete row version take up space (bloat)
- `VACUUM` cleans up when no running transactions can see old rows anymore
- `VACUUM` is blocked by
  - Long-running transactions or queries (e.g. `pg_dump`, `idle in transaction`)
  - Unused physical replication slots with `hot_standby_feedback = on`
  - Left behind prepared transactions
  - Standby servers with `hot_standby_feedback = on` and long-running transactions

Autovacuum Tuning
Autovacuum Parameters

credativ

```
#autovacuum = on
#autovacuum_max_workers = 3 # (change requires restart) (till PG17)
autovacuum_max_workers = 10
#autovacuum_worker_slots = 16 # PG18+
#autovacuum_naptime = 1min
#autovacuum_work_mem = -1 # -1 to use maintenance_work_mem
autovacuum_work_mem = 1GB
#log_autovacuum_min_duration = -1 # till PG14 / 10min since PG15
log_autovacuum_min_duration = 1min
```

- Automatic vacuum of tables and indexes
- Autovacuum launcher checks continuously if tables are in need of vacuum
- Several autovacuum workers can run in parallel
- Autovacuum is cost-based/throttled to save I/O bandwidth

Autovacuum Parameters
Cost-Based Autovacuum

credativ

```
#vacuum_cost_delay = 0ms
#autovacuum_vacuum_cost_delay = 2ms
#vacuum_cost_limit = 200
#autovacuum_vacuum_cost_limit = -1 # -1 means use vacuum_cost_limit
```

- Autovacuum waits for `autovacuum_vacuum_cost_delay` when `autovacuum_vacuum_cost_limit` work has been done
- Autovacuum throttled to 400 MB/s read / 40 MB/s write by default
- All concurrently running autovacuum processes share I/O bandwidth
- Increase `autovacuum_vacuum_cost_limit` if I/O resources are available

Autovacuum Parameters
Thresholds

credativ

```
#autovacuum_vacuum_threshold = 50
#autovacuum_vacuum_max_threshold = 100_000_000 # since PG18
#autovacuum_vacuum_scale_factor = 0.2
#autovacuum_vacuum_insert_threshold = 1000
#autovacuum_vacuum_insert_scale_factor 0.2
#autovacuum_analyze_threshold = 50
#autovacuum_analyze_scale_factor = 0.1
```

- Autovacuum is started for a table when:
  - `n_dead_tup > a_v_threshold + a_v_scale_factor * reltuples`
  - `n_ins_since_vacuum > a_v_insert_threshold + a_v_insert_scale_factor * reltuples`
- Autoanalyze is started for a table when:
  - `n_mod_since_analyze > a_a_threshold + a_a_scale_factor * retuples`

Autovacuum Parameters
Per-Table Thresholds

credativ

- Specific settings via `ALTER TABLE`
  - `autovacuum_enabled`
  - `autovacuum_vacuum_threshold`
  - `toast.autovacuum_vacuum_threshold`
  - `autovacuum_vacuum_scale_factor`
  - `toast.autovacuum_vacuum_scale_factor`
  - `autovacuum_vacuum_cost_delay`
  - `autovacuum_vacuum_cost_limit`

```
ALTER TABLE my_table1 SET (autovacuum_vacuum_scale_factor = 0.05,
                           toast.autovacuum_vacuum_scale_factor = 0.05);
ALTER TABLE my_table2 SET (autovacuum_vacuum_scale_factor = 0,
                           autovacuum_vacuum_threshold = 500000);
ALTER TABLE my_table3 SET (autovacuum_vacuum_max_threshold = 500000);
```

Autovacuum Tuning
Summary

credativ

- Avoid long-running transactions
- Use current defaults on legacy Postgres versions
- Increase `autovacuum_vacuum_cost_limit` if I/O bandwidth is abundant
- Consider increasing `autovacuum_max_workers` if there are a lot of large tables
  - Potentially adjust `autovacuum_vacuum_cost_limit`
- Set `autovacuum_work_mem = 1GB` to avoid multiple index scans
- Decrease `autovacuum_vacuum_scale_factor` for large tables
  - Alternatively set `autovacuum_vacuum_max_threshold` from PG18

credativ

Query Tuning

Query Tuning
Introduction

credativ

- Postgres query planner requires current statistics
  - Run `ANALYZE` or tune autoanalyze
  - Especially after restores/bulk-loading/in-place upgrades
- Index-only scans require uptodate visibility information
  - Run `VACUUM` or tune autovacuum
  - Especially after restores/bulk-loading/in-place upgrades
- Use real data if possible When trying to reproduce query performance issues
  - Or at least import the statistics

Query Tuning
Execution Plan Analysis

credativ

- Use `EXPLAIN (ANALYZE, BUFFERS)`
  - `BUFFERS` option (default from PG18) shows shared buffers `hit` | `read` information
  - `SERIALIZE` option (since PG17) shows deTOASTing overhead
- Overall costs/timing in first row
- Planner node timings include timings of their sub-nodes
- Compare estimated versus real rows
- Pay attention to `loops`, `actual time` is average per cycle
- Large number of "`Rows Removed by Filter`" hints at missing/unselective index
- "`Sort Method: external merge Disk: XXkB`" implies `work_mem` is too low

Query Tuning

EXPLAIN (ANALYZE, BUFFERS) Example

```
omdb=> EXPLAIN (ANALYZE, BUFFERS) SELECT movies.name FROM people, casts, movies WHERE people.name = 'Ingrid Bergman'
omdb-> AND people.id = casts.person_id AND movies.id = casts.movie_id ORDER BY movies.name DESC;
                                                            QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=29142.95..29142.96 rows=4 width=17) (actual time=109.111..109.114 rows=33 loops=1)
   Sort Key: movies.name DESC
   Sort Method: quicksort  Memory: 25kB
   Buffers: shared hit=2404 read=9128
   ->  Nested Loop  (cost=5390.52..29142.91 rows=4 width=17) (actual time=17.327..109.079 rows=33 loops=1)
         Buffers: shared hit=2404 read=9128
         ->  Hash Join  (cost=5390.10..29141.12 rows=4 width=8) (actual time=17.309..108.895 rows=33 loops=1)
               Hash Cond: (casts.person_id = people.id)
               Buffers: shared hit=2272 read=9128
               ->  Seq Scan on casts  (cost=0.00..20783.74 rows=1130374 width=16) (actual time=0.067..48.013 rows=1130374 loops=1)
                     Buffers: shared hit=352 read=9128
               ->  Hash  (cost=5390.09..5390.09 rows=1 width=8) (actual time=16.673..16.673 rows=1 loops=1)
                     Buckets: 1024  Batches: 1  Memory Usage: 9kB
                     Buffers: shared hit=1920
                     ->  Seq Scan on people  (cost=0.00..5390.09 rows=1 width=8) (actual time=0.417..16.667 rows=1 loops=1)
                           Filter: (name = 'Ingrid Bergman'::text)
                           Rows Removed by Filter: 277606
                           Buffers: shared hit=1920
         ->  Index Scan using movies_pkey on movies  (cost=0.42..0.45 rows=1 width=25) (actual time=0.005..0.005 rows=1 loops=33)
               Index Cond: (id = casts.movie_id)
               Buffers: shared hit=132
```

Query Tuning
Parallel Query Parameters

credativ

```
#max_worker_processes = 8 # (change requires restart)
max_worker_processes = num_vcores
#max_parallel_workers = 8
max_parallel_workers = num_vcores
```

- Number of processes that can be used for (among others) parallel queries
- 1-2x CPU/(v)core count

Query Tuning
Parallel Query Parameters

credativ

```
#max_parallel_workers_per_gather = 2
max_parallel_workers_per_gather = 4
```

- Amount of (additional) worker processes for parallel plan nodes
- Leader process usually takes part as well (`parallel_leader_participation`)
- Number of workers actually launched possibly lower at execution time

Query Tuning

EXPLAIN (ANALYZE, BUFFERS) – No Parallelism

```
omdb=> SET max_parallel_workers_per_gather = 0
SET
omdb=> EXPLAIN (ANALYZE, BUFFERS) SELECT COUNT(*) FROM casts;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Aggregate  (cost=23609.67..23609.68 rows=1 width=8)
            (actual time=75.842..75.843 rows=1 loops=1)
   Buffers: shared hit=576 read=8904
   ->  Seq Scan on casts  (cost=0.00..20783.74 rows=1130374 width=0)
                          (actual time=0.067..44.911 rows=1130374 loops=1)
         Buffers: shared hit=576 read=8904
 Execution Time: 75.880 ms
(6 rows)
```

Query Tuning

EXPLAIN (ANALYZE, BUFFERS) – Parallel Query

```
omdb=> SET max_parallel_workers_per_gather = 2; EXPLAIN (ANALYZE, BUFFERS) SELECT COUNT(*) FROM casts;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=16367.58..16367.59 rows=1 width=8)
                     (actual time=32.890..34.476 rows=1 loops=1)
   Buffers: shared hit=480 read=9000
   ->  Gather  (cost=16367.36..16367.57 rows=2 width=8)
               (actual time=32.763..34.471 rows=3 loops=1)
         Workers Planned: 2
         Workers Launched: 2
         Buffers: shared hit=480 read=9000
         ->  Partial Aggregate  (cost=15367.36..15367.37 rows=1 width=8)
                                (actual time=29.416..29.417 rows=1 loops=3)
               Buffers: shared hit=480 read=9000
               ->  Parallel Seq Scan on casts  (cost=0.00..14189.89 rows=470989 width=0)
                                               (actual time=0.026..17.605 rows=376791 loops=3)
                     Buffers: shared hit=480 read=9000
 Execution Time: 34.514 ms
(14 rows)
```

Query Tuning

Just-in-Time Compilation of Execution Nodes

credativ

```
#jit_above_cost = 100000
jit_above_cost = 1000000
#jit = on jit = off
```

- JIT can make very complex queries faster
- Often used by planner for regular queries and JIT startup increases execution time
- Increase `jit_above_cost` so JIT is only used for complex queries
- Alternatively, disable JIT completely
- `SHOW jit / SELECT jit_is_available()` displays JIT status

Query Tuning
Optimizer Parameters

credativ

```
#enable_seqscan = on
#enable_indexscan = on
#enable_indexonlyscan = on
#enable_bitmapscan = on
#enable_sort = on
#enable_hashjoin = on
#enable_mergejoin = on
#enable_nestloop = on
[...]
```

- Different execution node types can be disabled
- Planner sets extremely high costs for those plan nodes
- If no other option is possible those execution node types are used nevertheless
- Global setting discouraged, if at all then set per session

Query Tuning Extensions

pg_stat_statements

```
shared_preload_libraries = 'pg_stat_statements' # (change requires restart)
#pg_stat_statements.max = 5000
pg_stat_statements.max = 10000
pg_stat_statements.track = all
#track_io_timing = off
track_io_timing = on
```

- Collect cumulative statistics of executed queries
- First point of contact for query performance issues
- Used with time-series databases makes analysis of execution time changes possible
- Which queries take the longest execution time?
- Which queries cause I/O or WAL or temporary files?

Query Tuning Extensions

pg_stat_statements

credativ

```
omdb=# SELECT substr(query, 1, 75) AS query, calls, total_exec_time::bigint AS total,
min_exec_time, max_exec_time, mean_exec_time,
(shared_blks_hit+shared_blks_read)/calls AS blks_p_c, blk_read_time/calls AS readio_p_c,
temp_blks_written*8/1024/calls AS tmpmb_p_c, wal_bytes/1024/1024/calls AS walmb_p_c
FROM pg_stat_statements ORDER by total_exec_time DESC LIMIT 1 \gx
-[ RECORD 1 ]--+-----------------------------------------------------------------------------------------------------
query          | SELECT p.id,p.name, count(DISTINCT c1.movie_id) Partnerships FROM casts c1 JOIN casts
calls          | 5039
total          | 62684
min_exec_time  | 7.948906
max_exec_time  | 26.784409
mean_exec_time | 12.439838397896377
blks_p_c       | 8372
readio_p_c     | 0.01069336535026791
tmpmb_p_c      | 0
walmb_p_c      | 0
```

Query Tuning Extensions

auto_explain

credativ

```
shared_preload_libraries = 'auto_explain' # change requires restart)
#auto_explain.log_min_duration = -1
auto_explain.log_min_duration = '100ms'
#auto_explain.log_nested_statements = off
auto_explain.log_nested_statements = on
```

- Writes execution plans into the Postgres log
  - Optionally with timing information as well (overhead)
- `log_nested_statements` allows logging of plans from procedures/functions
- Comprehensively configurable

Query Tuning Extensions

hypopg

- http://dalibo.github.io/hypopg
- Hypothetical indexes
- Allows creation/deletion of indexes (in a session) and running EXPLAIN

```
omdb=# EXPLAIN SELECT * FROM people WHERE name = 'Ingrid Bergman';
                        QUERY PLAN
-----------------------------------------------------------
 Seq Scan on people  (cost=0.00..5391.61 rows=1 width=34)
   Filter: (name = 'Ingrid Bergman'::text)
(2 rows)
omdb=# SELECT * FROM hypopg_create_index('CREATE INDEX ON people (name)');
omdb=# EXPLAIN SELECT * FROM people WHERE name = 'Ingrid Bergman';
                                      QUERY PLAN
---------------------------------------------------------------------------------------
 Index Scan using "<13537>btree_people_name" on people  (cost=0.05..0.87 rows=1 width=34)
   Index Cond: (name = 'Ingrid Bergman'::text)
(2 rows)
```

Query Tuning Extensions

pg_hint_plan

- https://github.com/ossc-db/pg_hint_plan
- Optimizer Hints
- Either as SQL comment or through hint_plan.hints table

```
omdb=# EXPLAIN SELECT * FROM people WHERE name = 'Ingrid Bergman';
                                  QUERY PLAN
------------------------------------------------------------------------------
 Index Scan using people_name_idx on people  (cost=0.42..8.44 rows=1 width=34)
   Index Cond: (name = 'Ingrid Bergman'::text)
(2 rows)

omdb=# /*+ SeqScan (people) */ EXPLAIN SELECT * FROM people WHERE name = 'Ingrid Bergman';
                         QUERY PLAN
------------------------------------------------------------
 Seq Scan on people  (cost=0.00..5391.61 rows=1 width=34)
   Filter: (name = 'Ingrid Bergman'::text)
(2 rows)
```

credativ

Questions?