credativ

PostgreSQL Performance Tuning

Michael Banck <michael.banck@credativ.de>

PGConf.DE 2025

credativ GmbH

We Are Back

- Gegründet 1999 in Mönchengladbach
- Enge Verbindung zur Open-Source Community
- 40 Open-Source Spezialisten
- Beratung, Entwicklung, Schulungen, Support (3rd-Level / 24x7)
- Open-Source-Infrastruktur mit Debian, Kubernetes und Proxmox
- Open-Source-Datenbank PostgreSQL
- DevOps mit Ansible, Puppet, Terraform und andere
- Seit 2025 wieder unabhängiges, Inhaber-geführtes Unternehmen

credativ GmbH

Weitere Vorträge auf der PGConf.DE 2025

- “PostgreSQL Connections Memory Usage: How Much, Why and When?”
  - Josef Machytka, Freitag 11:10–11:55 - Raum Berlin 2+3
  - `https://www.postgresql.eu/events/pgconfde2025/schedule/session/6297-postgresql-connections-memory-usage-how-much-why-and-when/`
- “Your Data Deserves the Best: Migration to PostgreSQL”
  - Alexander Wirt & Josef Machytka, Freitag 13:35-14:20 - Raum Madrid
  - `https://www.postgresql.eu/events/pgconfde2025/schedule/session/6704-your-data-deserves-the-best-migration-to-postgresql/`

credativ

PostgreSQL Performance Tuning

PostgreSQL Performance Tuning

credativ

- credativ Postgres Schulung schon seit vielen Jahren

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

    Initial revision
```

- Performance-Tuning ein (sehr) weites Feld
- Zusammenstellung von verschiedenen Themen:
  - Speicher-Parameter Tuning
  - Storage-Parameter und Checkpoint Tuning
  - Autovacuum Tuning
  - Query Tuning

credativ

Speicher-Parameter Tuning

Speicher Parameter

shared_buffers

credativ

```
#shared_buffers = 128MB   # (change requires restart)
shared_buffers = 0.25*RAM
```

- Single-Source-of-Truth Ansicht der Daten für alle Backend-Prozesse
- Default üblicherweise zu klein
- 1/10 bis 4/10 vom RAM, üblich 1/4
- Wenn gesamte Datenbank in RAM passt, `shared_buffers` = Datenbankgröße

Speicher Parameter

min_dynamic_shared_memory

```
#min_dynamic_shared_memory = 0
min_dynamic_shared_memory = 2GB
```

- Zusätzlicher fester Shared-Memory Bereich für Parallel Query
- Insbesondere sinnvoll zu setzen bei Verwendung von Huge Pages

Speicher Parameter

max_connections

credativ

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

- Weitere Verbindungen werden abgewiesen
- superuser_reserved_connections als Admin-Reserve
- Nicht viel zu viel einstellen (keine X-Tausend)
  - Pro zusätzlicher max_connections wird ca. 50kB Shared Memory verwendet
- An Hand historischer Daten (z.B. check_postgres_backends) einstellen
- ggf. Connection-Pooler verwenden

Speicher Parameter

effective_cache_size

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

- Speicher, der Queries als Cache zur Verfügung steht
- 1/2 bis 3/4 vom RAM
- Nur Hinweis, keine wirkliche Allozierung

```
$ 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/

Speicher Parameter

work_mem

credativ

```
#work_mem = 4MB
```

- Von Postgres verwendeter Speicher pro Planknoten / Parallel Worker
  - Sortierungen `ORDER BY`, `DISTINCT`, Merge Joins)
  - Hash-Tabellen (Hash Joins, Hash-basierte Aggregierungen/Verarbeitungen von `IN` Subqueries)
  - Größere Sortierungen oder Hash-Tabellen verwenden danach temporären Storage
- Stark abhängig von der Anwendung / `max_connections` / aktive Verbindungen
- Setzen von `log_temp_files = 0` zeigt Queries mit zu kleinem `work_mem` im 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)
```

Speicher Parameter

hash_mem_multiplier

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

- Multiplikator von `work_mem` für Hash-basierte Operationen
- Hash-basierte Planknoten problematischer bei Disk-Spilling als Sortierungen
- Lieber kleineres `work_mem` und dafür `hash_mem_multiplier` hochsetzen

```
->  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
```

Speicher Parameter

{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
- Bis zu 1/10 vom RAM, bzw. 1 GB
- Bei massiv parallelen Restores kann Index-Erstellung viel Speicher verbrauchen
- autovacuum_work_mem analog (per Default gleich wie maintenance_work_mem)
- VACUUM kann bis PG16 nicht mehr als 1 GB verwenden

Speicher Parameter

`vacuum_buffer_usage_limit`

credativ

```
#vacuum_buffer_usage_limit = 2MB
```

- Neuer Parameter seit PG16
- Größe des Ring-Buffers für `VACUUM`
  - Maximal 1/8 von Shared Buffers
- Vergrößerung beschleunigt Vacuum, verbraucht aber auch mehr Shared Buffers
- Kann auch in `VACUUM` via Option `BUFFER_USAGE_LIMIT` gesetzt werden

credativ

Storage-Parameter und Checkpoint Tuning

Storage-Parameter

random_page_cost

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

- Verhältnis von zufälligem zu sequenziellem Storage-Zugriff
- Bei SSDs/NVME sollte ein Wert nahe 1.0 verwendet werden
- Lässt den Planer eher Index-Scans verwenden

Storage-Parameter

{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 # ab PG17
```

- Anzahl an parallelen I/O Vorgängen, verwendet Readahead/`posix_fadvise`
- `effective_io_concurrency` wird für Bitmap Heap Scans verwendet
- `maintenance_io_concurrency` wird ab Version 14 für `ANALYZE` verwendet

Storage-Parameter

{effective,maintenance}_io_concurrency, io_combine_limit

- ANALYZE-Zeiten für maintenance_io_concurrency/io_combine_limit-Werte:

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-Parameter

{wal,default_toast}_compression

credativ

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

- Kompressions-Methode für WAL bzw. TOAST
- `lz4` deutlich Ressourcen-schonender/schneller als `pglz` für TOAST
- `zst` komprimiert besser als `lz4` und ist ungefähr so schnell wie `pglz` für WAL

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

Checkpoint Tuning
Checkpoints und Full-Page-Images

credativ

- Transaktionslog (Write-Ahead Log, WAL) wird WAL-Dateien auf Platte geführt
  - Bei jedem Commit wird WAL gesynct bevor Client Erfolgsmeldung erhält
- Geänderte Tabellen/Index-Daten werden durch periodische Checkpoints persistent ins Datenverzeichnis geschrieben
- Im Fehlerfall wird WAL seit dem letzten Checkpoint (Redo-Punkt) nachgespielt
- Erste Änderung einer Page nach Checkpoint wird als Full-Page-Image (FPI) ins WAL geschrieben
- Weitere Änderungen bis zum nächsten Checkpoint sind deutlich kleiner
- Automatische Checkpoints entweder Zeit- (`time`) oder WAL-basiert (`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
```

- Versucht `max_wal_size` nicht zu überschreiten
- Checkpoint nach `checkpoint_timeout` oder bei ca. 1/3 `max_wal_size`
  - `max_wal_size / (2.0 + checkpoint_completion_target)`
- Vorteil: Weniger FPIs/WAL
- Nachteil: größeres `pg_wal/`, längeres Recovery
- Erhöhung von `max_wal_size` bis Checkpoints Zeit-basiert sind

Autovacuum Tuning

Autovacuum Tuning

MVCC – Multi-Version Concurrency Control

credativ

- `DELETE` markiert Zeilen als gelöscht
- Parallele Transaktionen müssen Zeilen noch lesen können
- `UPDATE = DELETE + INSERT`
- Alte Zeilen-Versionen brauchen Platz
- `VACUUM` wenn keine laufenden Transaktionen alte Zeilen mehr sehen können
- Die Arbeit von `VACUUM` wird verhindert von
  - Langlaufenden Transaktionen bzw. Queries (z.B. `pg_dump`, `idle in transaction`)
  - Nicht mehr verwendeten Replikations-Slots
  - Zurückgelassenen Prepared Transactions
  - Standby-Server mit `hot_standby_feedback = on` und langlaufenden Transaktionen

Autovacuum Tuning
Autovacuum Parameter

credativ

- Automatisches Vacuum von Tabellen und Indexen
- Autovacuum-Launcher ermittelt fortlaufend, ob Tabellen Vacuum benötigen
- Mehrere Autovacuum-Prozesse können parallel laufen
- Autovacuum ist kosten-basiert/gebremst um I/O-Bandbreite zu sparen
- Relevante Parameter:

```
#autovacuum = on
#autovacuum_max_workers = 3 # (change requires restart) (bis 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 # bis PG14
log_autovacuum_min_duration = 10min # ab PG15
```

Autovacuum Parameter

Kosten-basiertes 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 wartet jedes mal für `autovacuum_vacuum_cost_delay` wenn Arbeit von `autovacuum_vacuum_cost_limit` erledigt ist
- Autovacuum Standardmäßig gedrosselt auf 400 MB/s lesen / 40 MB/s schreiben
- Alle gleichzeitig laufenden Autovacuum-Prozesse teilen sich diese Bandbreite
- Bei genügend freien I/O-Ressourcen `vacuum_cost_limit` hochsetzen

Autovacuum Parameter

Schwellwerte

credativ

```
#autovacuum_vacuum_threshold = 50
#autovacuum_vacuum_max_threshold = 100_000_000 # ab 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 wird für eine Tabelle gestartet wenn:
  - `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 wird für eine Tabelle gestartet wenn:
  - `n_mod_since_analyze > a_a_threshold + a_a_scale_factor * retuples`

Autovacuum Parameter
Schwellwerte pro Tabelle

credativ

- Spezifische Einstellungen mit `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
Zusammenfassung

credativ

- Langlaufende Transaktionen vermeiden
- Bei veralteten Postgres-Versionen die aktuellen Defaults verwenden
- Bei genügend I/O-Bandbreite `autovacuum_vacuum_cost_limit` hochsetzen
- Bei vielen großen Tabellen evtl. `autovacuum_max_workers` hochsetzen
  - Gegebenenfalls `autovacuum_vacuum_cost_limit` anpassen
- `autovacuum_work_mem = 1GB` setzen um mehrmalige Index-Scans zu vermeiden
- Bei großen Tabellen `autovacuum_vacuum_scale_factor` heruntersetzen
  - Alternativ `autovacuum_vacuum_max_threshold` ab PG18

credativ

Query Tuning

Query Tuning
Grundlagen

credativ

- Postgres Query-Planer benötigt aktuelle Statistiken
  - `ANALYZE` verwenden bzw. Autoanalyze tunen
  - Insbesondere nach Restores/Bulk-Loading/In-Place Upgrades
- Index-Only Scans benötigen aktuelle Sichtbarkeits-Informationen
  - `VACUUM` verwenden bzw. Autovacuum tunen
  - Insbesondere nach Restores/Bulk-Loading/In-Place Upgrades
- Beim Nachstellen von Performance-Problemen möglichst echte Daten verwenden
  - Oder zumindest die Statistiken importieren

Query Tuning
Analyse von Ausführungsplänen

- `EXPLAIN (ANALYZE, BUFFERS)` verwenden
  - Option `BUFFERS` (ab PG18 Standard) zeigt Shared Buffers `hit|read` Informationen
  - Option `SERIALIZE` (ab PG17) zeigt auch deTOASTing Zeit
- Gesamt Kosten/Zeit in der ersten Zeile
- Zeit pro Planknoten ist inklusive darunter liegender Unterknoten
- Geschätzte und tatsächliche Zeilenanzahl vergleichen
- `loops` beachten, tatsächliche Zeit ist Mittelwert pro Durchlauf
- Hohe "`Rows Removed by Filter`" Anzahl deutet auf fehlenden/unselektiven Index
- "`Sort Method: external merge Disk: XXkB`" bedeutet zu kleines `work_mem`

Query Tuning

`EXPLAIN (ANALYZE, BUFFERS)` Beispiel

```
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 Parameter

credativ

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

- Anzahl der Prozesse, die für u.a. parallele Queries verwendet werden können
- 1-2x CPU/(v)Core-Anzahl

Query Tuning
Parallel Query Parameter

credativ

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

- Anzahl an (zusätzlichen) Worker-Prozessen für parallele Planknoten
- Leader-Prozess nimmt üblicherweise auch Teil (`parallel_leader_participation`)
- Zahl der geplanten Worker-Prozesse bei Ausführung evtl. geringer

Query Tuning

EXPLAIN (ANALYZE, BUFFERS) – Keine Parallelität

```
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 von Ausführungsknoten

credativ

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

- JIT kann bei komplizierten Abfragen die Ausführungszeit verkürzen
- Wird oft vom Planer bei nomalen Abfragen verwendet und verlängert Ausführungszeit
- `jit_above_cost` hochsetzen, JIT nur für komplizierte Abfragen verwenden
- Alternativ JIT komplett ausschalten
- `SHOW jit` / `SELECT jit_is_available()` zeigen JIT-Status an

Query Tuning
Optimizer Parameter

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
[...]
```

- Verschiedene Ausführungsknoten können deaktiviert werden
- Planer setzt Kosten für diese Planknoten exorbitant hoch an
- Wenn keine andere Option besteht wird Ausführungsknoten trotzdem verwendet
- Globale Einstellung nicht ratsam, wenn dann per-Session

Query Tuning Erweiterungen

pg_stat_statements

credativ

```
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
```

- Sammelt kumulative Statistiken über ausgeführte Abfragen
- Erste Anlaufstelle bei Query Performance-Problemen
- In Verbindung mit Zeitreihen-Datenbank Auswertung von Ausführungszeiten-Änderungen möglich
- Welche Queries benötigen die meiste Laufzeit?
- Welche Queries verursachen I/O oder WAL oder temporäre Dateien?

Query Tuning Erweiterungen

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 Erweiterungen

auto_explain

```
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
```

- Schreibt Ausführungspläne in das PostgreSQL-Log
  - Optional auch mit Timing-Informationen (Overhead)
- `log_nested_statements` erlaubt Protokollieren von Plänen innerhalb von Prozeduren
- Umfangreich konfigurierbar

Query Tuning Erweiterungen

hypopg

credativ

- http://dalibo.github.io/hypopg
- Hypothetische Indexe
- Erlaubt (in einer Session) Indexe anzulegen/zu löschen und EXPLAIN aufzurufen

```
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 Erweiterungen

pg_hint_plan

- https://github.com/ossc-db/pg_hint_plan
- Optimizer Hints
- Entweder als SQL-Kommentar oder via `hint_plan.hints` Tabelle

```
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

Fragen?