CYBERTEC

DATA SCIENCE & POSTGRESQL

Typische Performance Optimierungen

HANS-JÜRGEN SCHÖNIG

CEO & SENIOR DATABASE CONSULTANT

- PostgreSQL since 1999
- author of various database books

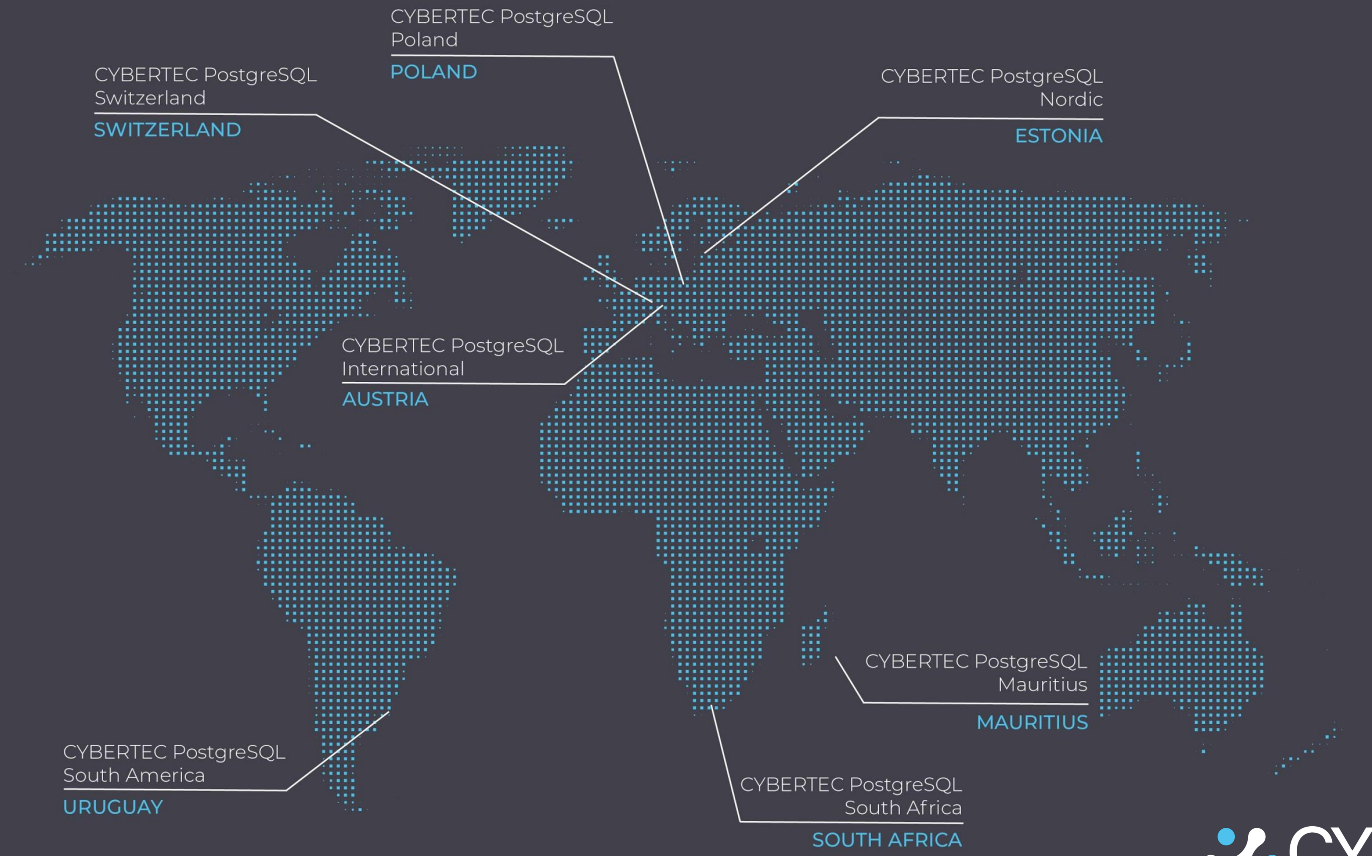
MAIL hs@cybertec.at

PHONE +43 2622 930 22-2

WEB www.cybertec-postgresql.com

CYBERTEC
DATA SCIENCE & POSTGRESQL

INTERNATIONALES POSTGRESQL CONSULTING UND SUPPORT



CYBERTEC
DATA SCIENCE & POSTGRESQL

CLIENT SECTORS

- ICT
- University
- Government
- Automotive
- Industry
- Trade
- Finance
- etc.

CYBERTEC
DATA SCIENCE & POSTGRESQL

WELCHES PROBLEM WOLLEN WIR LÖSEN?

POSTGRESQL PERFORMANCE

- Performance Themen sind durchaus vielfältig
  - Wir diskutieren einige wesentliche Themen
- In den meisten Fällen ist es der **User** und nicht die Database Engine.
- Es gibt ein “typisches” Set von Fehlern
  - Alle Branchen
  - Gilt für alle Betriebsgrößen
  - Alle Regionen der Welt

BIG FISH FIRST !

- Viele Probleme sind recht straight forward
  - Meistens sind es NICHT …
    - Die RAID Level
    - Die Kernel Versionen
    - Die I/O Scheduler
    - Obskure Parameter (“performance_on = true”)
  - Die Lösung ist in den Queries

EWIGE WAHRHEITEN

- “Queries” verursachen Load
- Viele Dinge sind die Konsequenz nicht die Ursache
  - I/O Wait
  - Hohe CPU Aktivität
  - Schlechte User Experience
- “Load = 10” hat **NULL** Informationsgehalt
- “Cache Hit Rate = 10%” hat **NULL** Informationsgehalt

MONITORING: BUNTE BILDER

- Kann hilfreich sein - muss es aber nicht
- Wer seine Workload nicht versteht, ist hilflos
- Charts sind angemalte Zahlen
  - Versteh also deine “Zahlen”

PG_STAT_STATEMENTS
DER GOLDSTANDARD

PG_STAT_STATEMENTS (1)

```
test=# \d pg_stat_statements
                    View "public.pg_stat_statements"
     Column          |     Type          ...
---------------------+------------------ ...
 userid              | oid
 dbid                | oid
 queryid             | bigint
 query               | text
 plans               | bigint
 total_plan_time     | double precision
 min_plan_time       | double precision
 max_plan_time       | double precision
 mean_plan_time      | double precision
 stddev_plan_time    | double precision
 ...
```

PG_STAT_STATEMENTS (2)

```
test=# \d pg_stat_statements
                    View "public.pg_stat_statements"
     Column          |     Type
---------------------+------------------
...
 calls               | bigint
 total_exec_time     | double precision
 min_exec_time       | double precision
 max_exec_time       | double precision
 mean_exec_time      | double precision
 stddev_exec_time    | double precision
 rows                | bigint
...
```

PG_STAT_STATEMENTS (3)

```
test=# \d pg_stat_statements
                     View "public.pg_stat_statements"
     Column          |     Type          ...
---------------------+------------------ ...
...
 shared_blks_hit     | bigint
 shared_blks_read    | bigint
 shared_blks_dirtied | bigint
 shared_blks_written | bigint
 local_blks_hit      | bigint
 local_blks_read     | bigint
 local_blks_dirtied  | bigint
 local_blks_written  | bigint
 ...
```

PG_STAT_STATEMENTS (4)

```
test=# \d pg_stat_statements
                     View "public.pg_stat_statements"
     Column          |     Type          ...
---------------------+------------------ ...
...
 temp_blks_read      | bigint
 temp_blks_written   | bigint
 blk_read_time       | double precision
 blk_write_time      | double precision
 wal_records         | bigint
 wal_fpi             | bigint
 wal_bytes           | numeric
```

EINE SINNVOLLE ABFRAGE

```
test=# SELECT   substring(query, 1, 20),
      calls,
      round(total_exec_time::numeric, 2) AS total_exec_time,
      round((100 * total_exec_time / sum(total_exec_time) OVER ())::numeric, 2) AS percent
FROM  pg_stat_statements
ORDER BY total_exec_time DESC
LIMIT 4;
      substring       | calls | total_exec_time | percent
----------------------+-------+-----------------+---------
 INSERT INTO t_owner  |     3 |       243180.64 |    34.55
 CREATE EXTENSION IF  |    35 |        58047.05 |     8.25
 COPY t_raw (data) FR |    62 |        40911.07 |     5.81
 COPY t_raw (data) FR |    24 |        16874.80 |     2.40
(4 rows)
```

CYBERTEC
DATA SCIENCE & POSTGRESQL

FEHLENDE INDEXES
DER KLASSIKER

PG_STAT_USER_TABLES

```
test=# \d pg_stat_user_tables
                    View "pg_catalog.pg_stat_user_tables"
      Column          |           Type
 ---------------------+---------------------
 relid                | oid
 schemaname           | name
 relname              | name
 seq_scan             | bigint        ← die zwei meistunterschätzten Zahlen
 seq_tup_read         | bigint
 idx_scan             | bigint
 idx_tup_fetch        | bigint
 n_tup_ins            | bigint
 n_tup_upd            | bigint
 n_tup_del            | bigint
 n_tup_hot_upd        | bigint
 n_live_tup           | bigint
 n_dead_tup           | bigint
```

EXZESSIVER
“BUFFER USAGE”

BUFFER USAGE

- Das Problem wird oft übersehen ...
- Symptome ...
  - Instabile Laufzeiten
  - Viel I/O
  - Auffällige Cache Hit Rates

BEISPIEL: BUFFER USAGE

```
Subquery Scan on some_table  (cost=463014.50..463014.91 rows=1 width=323)
     (actual time=41069.124..41071.536 rows=25 loops=1)
  Filter: (some_table.field <= 4)
  Rows Removed by Filter: 276
  Buffers: shared hit=937135 read=285779 dirtied=23 written=756
  I/O Timings: read=33215.251 write=6.482
  ->  Subquery Scan on whatever  (cost=463014.50..463014.90 rows=1 width=323)
          (actual time=41067.577..41071.512 rows=301 loops=1)
        Filter: ((some_table.a_column IS NOT NULL) AND (some_table.b_column IS NULL))
        Rows Removed by Filter: 3081
        Buffers: shared hit=937135 read=285779 dirtied=23 written=756
        I/O Timings: read=33215.251 write=6.482
        ->  Unique  (cost=463014.50..463014.62 rows=3 width=319)
              (actual time=41067.225..41070.092 rows=3382 loops=1)
            Buffers: shared hit=937103 read=285779 dirtied=23 written=756
            I/O Timings: read=33215.251 write=6.482
            ->  Sort  (cost=463014.50..463014.51 rows=3 width=319)
              (actual time=41067.223..41067.529 rows=3382 loops=1)
...
```

STORAGE
TABLE BLOAT

BLOAT ALS GEFAHR

- Große Gefahr für die Performance
- Entsteht meistens durch …
  - Zu viele UPDATEs
  - Zu lange (idle) Transactions
  - Zu wenig VACUUM

BLOAT ALS GEFAHR

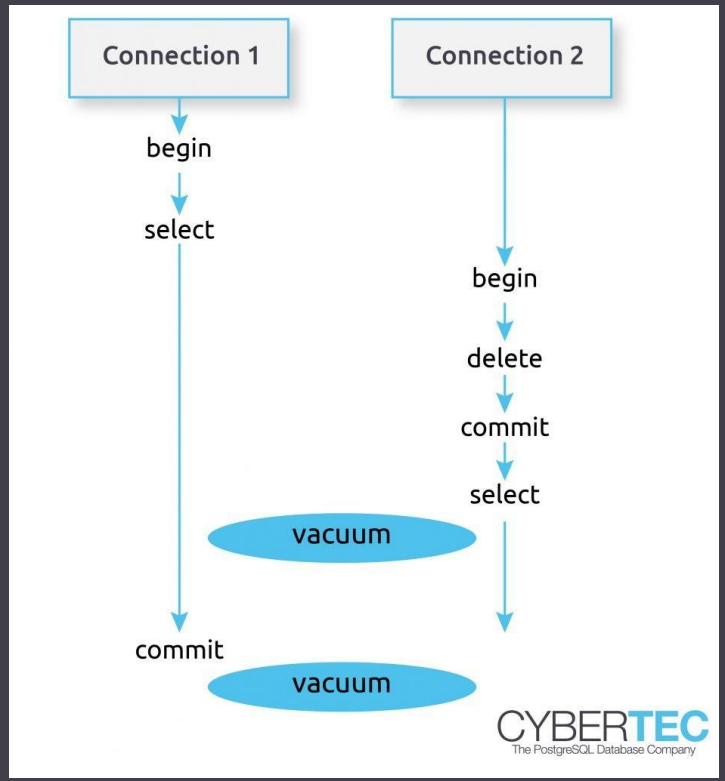


TABLE BLOAT MESSEN

```
test=# CREATE EXTENSION pgstattuple;
CREATE EXTENSION
test=# \x
Expanded display is on.
test=# SELECT * FROM pgstattuple('t_demo');
-[ RECORD 1 ]------+---------
table_len          | 36249600
tuple_count        | 1000000
tuple_len          | 28000000
tuple_percent      | 77.24
dead_tuple_count   | 0
dead_tuple_len     | 0
dead_tuple_percent | 0
free_space         | 125700
free_percent       | 0.35
```

PG_SQUEEZE ALS LÖSUNG

- VACUUM FULL benötigt einen langen Table Lock

- pg_squeeze ...
  - benötigt nur kurze Table Locks
  - kann downtime-frei Tablespaces verschieben
  - kann Tabellen reorganisieren / clustern

URL: https://www.cybertec-postgresql.com/en/products/pg_squeeze/

FINALLY

CONCLUSION

- Die Lösung ist in den Queries
  - pg_stat_statements, etc.
- Missing Indexes sind das größte Problem
  - pg_stat_user_tables
- Table Bloat kann ein Problem sein
  - VACUUM, pg_squeeze, etc.

QUESTIONS?

Feel free to contact me!

MAIL	hs@cybertec.at
WEB	www.cybertec-postgresql.com
TWITTER	@postgresql_007
YOUTUBE	CYBERTEC Data Science & PostgreSQL
	www.youtube.com/user/cybertecpostgresql

CYBERTEC
DATA SCIENCE & POSTGRESQL