CYBERTEC
POSTGRESQL SERVICES & SUPPORT

PERFORMANCE TIPS YOU HAVE NEVER SEEN BEFORE

Version 2.0


AUSTRIA (HQ)
CYBERTEC POSTGRESQL INTERNATIONAL (HQ)
SWITZERLAND
CYBERTEC POSTGRESQL SWITZERLAND
URUGUAY
CYBERTEC POSTGRESQL SOUTH AMERICA
ESTONIA
CYBERTEC POSTGRESQL NORDIC
POLAND
CYBERTEC POSTGRESQL POLAND
SOUTH AFRICA
CYBERTEC POSTGRESQL SOUTH AFRICA

CUSTOMER BUSINESS SECTORS

- ICT
- Universities
- Government
- Automotive
- Industrial
- Trade
- Financial Services
- etc.

POSTGRESQL PERFORMANCE

Sharing some less known stuff

GOAL OF THIS TALK

- Share some dirty, less well known trickery
- Hopefully help people to speed up apps
- Performance “beyond postgresql.conf”
- There is more than …
    - Adding memory
    - Adding CPUs

POSTGRESQL

CREATING CONNECTIONS

BETTER CONNECTIONS?

- Can we do something when creating connections?
- Maybe speed up stuff to come?
- Every thought about …

```
test=# SHOW session_preload_libraries;
 session_preload_libraries
---------------------------


(1 row)
```

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

WHY CARE ABOUT LIBRARIES?

- Can we do something when creating connections?
- Maybe speed up stuff to come?
- Every thought about …

```
CREATE OR REPLACE FUNCTION r_max (integer, integer)
   RETURNS integer AS
$$
 if (arg1 > arg2)
 return(arg1)
 else
 return(arg2)
$$ LANGUAGE 'plr' STRICT;
```

WHY CARE ABOUT LIBRARIES?

- Mind the first call …

```
test=# \timing
Timing is on.
test=# SELECT r_max(1, 2);
 r_max
═══════
     2
(1 row)

Time: 229.629 ms
test=# SELECT r_max(1, 2);
 r_max
═══════
     2
(1 row)

Time: 0.705 ms
```

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

PRELOADING LIBRARIES

- Load libraries when …
  - Creating the connection
  - Starting the server (shared_preload_libraries)
- More stable runtimes
- Very useful when loading large libraries
- Predictable runtimes matter
- **HINT:**
  - Initializing the library is still not “free”
  - But preloading helps

REAL LIFE: PostGIS

```
> psql -U postgres
…

test=# \timing
Timing is on.
test=# SELECT * FROM hans.points WHERE id = 1;
 id |                          p
====+==========================================================
  1 | 0101000020E610000097515B9536C33140A252824D6FDC1440
(1 row)

Time: 10.004 ms
test=# SELECT * FROM hans.points WHERE id = 1;
 id |                          p
====+==========================================================
  1 | 0101000020E610000097515B9536C33140A252824D6FDC1440
(1 row)

Time: 0.664 ms
```

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

REAL LIFE: PostGIS

```
> PGOPTIONS='-c session_preload_libraries=postgis-3' psql -U postgres
…

test=# \timing
Timing is on.
test=# SELECT * FROM hans.points WHERE id = 1;
 id |                         p
----+--------------------------------------------
  1 | 0101000020E610000097515B9536C33140A252824D6FDC1440
(1 row)

Time: 2.809 ms
test=# SELECT * FROM hans.points WHERE id = 1;
 id |                         p
----+--------------------------------------------
  1 | 0101000020E610000097515B9536C33140A252824D6FDC1440
(1 row)

Time: 0.674 ms
```

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

POSTGRESQL

STORING DATA

RUNNING A TEST

- Use pgbench to init a database
- Run a couple of transactions

```
pgbench -c 4 -t 25000 -j 4 postgres
```

- Measure the difference

```
wal_level = logical vs minimal
max_wal_size = 64 MB vs 100 GB
```

RUNNING A TEST

ON A FRESH INSTANCE

```
wal_level = logical
max_wal_size = 64MB
```

Very bad settings

```
postgres=# SELECT pg_size_pretty(pg_current_wal_lsn()
- '0/00000000'::pg_lsn) AS diff;
  diff
--------
 135 MB
(1 row)
```

How much WAL was created?

RUNNING A TEST

ON A FRESH INSTANCE

```
wal_level = minimal
max_wal_size = 100GB
```

Very good settings

```
postgres=# SELECT pg_size_pretty(pg_current_wal_lsn()
- '0/00000000'::pg_lsn) AS diff;
  diff
--------
  82 MB
(1 row)
```

How much WAL was created?

WHY IS THAT?

- “minimal” ensures that the WAL is smaller in general
- Longer checkpoint distances (max_wal_size) lead to smaller WAL
  - Not so many full page write
- Especially useful during bulk loading
  - Consider creating replicas later

Most relevant !

POSTGRESQL

INDEXING

INDEXING: SUPER IMPORTANT …

Indexes are THE most important performance features

If you don’t index properly …

Your database will be slow

Your apps won’t work

More hardware won’t fix anything

CREATING SAMPLE DATA

JUST NUMBERS …

```
test=# CREATE TABLE t_static (id int);
CREATE TABLE
```

A simple table

```
test=# INSERT INTO t_static
         SELECT  *
         FROM    generate_series(1, 25000000);
INSERT 0 25000000
```

Let us add 25 million rows

CREATING INDEXES

USING FILLFACTOR

Default value = 90%

```
test=# CREATE INDEX idx_90 ON t_static (id);
CREATE INDEX
test=# CREATE INDEX idx_100 ON t_static (id) WITH (FILLFACTOR=100);
CREATE INDEX

test=# \di+
                                      List of relations
 Schema |  Name   | Type  | Owner |  Table   | Persistence | Access method |  Size
--------+---------+-------+-------+----------+-------------+---------------+--------
 public | idx_100 | index | hs    | t_static | permanent   | btree         | 483 MB
 public | idx_90  | index | hs    | t_static | permanent   | btree         | 536 MB
(2 rows)
```

INDEXING: FILLFACTOR = 100

- ONLY do it on STATIC data
- Never if you expect changes
  - Updates will cause immediate index node splits
  - Immediate node splits are not desired

- Large, static data is quite frequent
- Nice optimization for static cases

Word of caution

POSTGRESQL

ENFORCING JOIN ORDER

CREATING TABLES

```
plan=# SELECT   'CREATE TABLE x' || id || ' (id int)'
        FROM    generate_series(1, 5) AS id;
          ?column?
--------------------------
 CREATE TABLE x1 (id int)
 CREATE TABLE x2 (id int)
 CREATE TABLE x3 (id int)
 CREATE TABLE x4 (id int)
 CREATE TABLE x5 (id int)
(5 rows)

plan=# \gexec
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
```

Generate some SQL

Use it directly

PLAN TIME DOES MATTER

“Playing with fire”

- Optimizer decides on join order
- Usually makes good decisions
- Planning is not cost free (consider prepared plans)

- **Word of caution:**
  - Know what you are doing
  - This can blow up in your face

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

INSPECTING PLAN TIME

Optimization does take time

```
plan=# explain (timing, analyze) SELECT *
        FROM    x1 JOIN x2 ON (x1.id = x2.id)
                JOIN x3 ON (x2.id = x3.id)
                JOIN x4 ON (x3.id = x4.id)
                JOIN x5 ON (x4.id = x5.id);
…
Planning Time: 0.297 ms
Execution Time: 0.049 ms
```

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

INSPECTING PLAN TIME

Optimization does take time

```
plan=# SET join_collapse_limit TO 1;
SET
plan=# explain (timing, analyze) SELECT *
        FROM    x1 JOIN x2 ON (x1.id = x2.id)
                JOIN x3 ON (x2.id = x3.id)
                JOIN x4 ON (x3.id = x4.id)
                JOIN x5 ON (x4.id = x5.id);
…
Planning Time: 0.069 ms
Execution Time: 0.046 ms
```

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

WHAT HAPPENED?

We fixed join order ...

- join_collapse_limit defines how many
  - explicit joins
  - are planned implicitly
- In short:
  - We fixed the join order

Caution !
Know what you are doing !

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

POSTGRESQL

EXECUTING MORE EFFICIENTLY

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

CHANGING EXECUTION ORDER

```
test=# CREATE TABLE t_test AS
        SELECT  *
        FROM    generate_series(1, 10000000) AS id;
SELECT 10000000
```

Creating sample data

```
CREATE FUNCTION returns_many(int)
RETURNS int AS
$$
   BEGIN
      IF $1 % 2 = 0
      THEN
         RETURN $1;
      END IF;
      RETURN 0;
   END;
$$ LANGUAGE 'plpgsql';
```

```
CREATE FUNCTION returns_few(int)
RETURNS int AS
$$
   BEGIN
      IF $1 % 1000 = 35
      THEN
         RETURN $1;
      END IF;
      RETURN 0;
   END;
$$ LANGUAGE 'plpgsql';
```

CHANGING EXECUTION ORDER

Creating sample data

```
explain analyze SELECT  *
     FROM    t_test
     WHERE   returns_many(id) = id
             AND returns_few(id) = id;

                            QUERY PLAN
--------------------------------------------------------------------
 Seq Scan on t_test  (cost=0.00..5194236.16 rows=250 width=4)
                    (actual time=2625.793..2625.794 rows=0 loops=1)
   Filter: ((returns_many(id) = id) AND (returns_few(id) = id))
   Rows Removed by Filter: 10000000
 Planning Time: 0.218 ms
 Execution Time: 2625.846 ms
(5 rows)
```

CHANGING EXECUTION ORDER

```
test=# SELECT * FROM pg_stat_xact_user_functions;

 funcid | schemaname |   funcname    |  calls    | total_time    |  self_time
--------+------------+---------------+-----------+---------------+-------------
  25581 | public     | returns_many  | 10000000  | 1596.306829   | 1596.306829
  25582 | public     | returns_few   |  5000000  |  798.209276   |  798.209276
(2 rows)
```

Mind the number of function calls

CHANGING EXECUTION ORDER

Making things more expensive

```
test=# ALTER FUNCTION returns_many(int)
        COST 10000;
```

```
                            QUERY PLAN
--------------------------------------------------------------------
 Seq Scan on t_test  (cost=0.00..252693666.91 rows=250 width=4)
                    (actual time=2154.733..2154.738 rows=0 loops=1)
   Filter: ((returns_few(id) = id) AND (returns_many(id) = id))
   Rows Removed by Filter: 10000000
 Planning Time: 0.045 ms
 Execution Time: 2154.751 ms
(5 rows)
```

What we did ...

Behind the scenes ...

- We did NOT set costs
- We set a multiplier for cpu_operator_cost
  - Internal functions are cpu_operator_cost * 1 = 0.0025 (default)
  - Procedural code is normally cpu_operator_cost * 100
  - We made our function more expensive
  - PostgreSQL therefore changed execution order

POSTGRESQL

SUMMARY

WE ARE HIRING!

CHECK OUT OUR JOB OPENINGS:

https://www.cybertec-postgresql.com/en/jobs-and-opportunities/

CYBERTEC
POSTGRESQL SERVICES & SUPPORT

CEO

Hans–Jürgen SCHÖNIG

MAIL

hs@cybertec.at

PHONE

+43 2622 930 22 - 666

WEB

www.cybertec-postgresql.com

CYBERTEC
POSTGRESQL SERVICES & SUPPORT