# Issues related to PL/pgSQL usage

Pavel Stěhule
2018-04-30



Next

# SQL execution

- Query optim. (reduced by plan cache)
- Query initialization (every time)
- Query execution (every time)

# PLpgSQL execution

- Compilation to AST (first time in session)
- AST execution (every time)
- Query optimization (first time in session)
- Query initialization (every time)
- Query execution (every time)

# PLpgSQL execution

- EVERY EXPRESSION IN PLpgSQL code is query!
- without FROM clause
  - simple expr .. faster execution
- with FROM clause
  - full query .. normal speed

# PLpgSQL execution

```
do $$
declare s int default 0;
begin
  for i in 1..1000000 loop
    s := s + 1;
  end loop;
end;
$$;
DO
Time: 319,820 ms
```

```
do $$
declare s int default 0;
begin
  for i in 1..1000000 loop
    s := (select s + 1);
  end loop;
end;
$$;
DO
Time: 5785,919 ms (00:05,786)
```

# PLpgSQL execution

```
do $$
declare s int default 0;
begin
  for i in 1..1000000 loop
    select s + 1 into s;
  end loop;
end;
$$;
DO
Time: 3724,333 ms (00:03,724)
```

# PLpgSQL execution

- Query initialization is expensive
  - when it is repeated too much
    - it is more expensive than on Oracle
  - when query is very complex (lot of input/output columns)

# PLpgSQL execution ISAM style (NO)

```
create table foo(id integer unique, v integer);
insert into foo select i, 1 from generate_series(1,1000000) g(i);
```

```
do $$
declare s integer default 0;
begin
  for i in 1..1000000 loop
    s := s +
      (select v from foo
                where id = i);
  end loop;
end;
$$;
DO
Time: 18017,332 ms (00:18,017)
```

```
do $$
declare
  s integer default 0;
  _v integer;
begin
  for _v in
    select v from foo
  loop
    s := s + _v;
  end loop;
end;
$$;
DO
Time: 577,308 ms
```

# PLpgSQL execution wrapping query

```
CREATE OR REPLACE FUNCTION public.nazev_okresu(id text)
 RETURNS text
 LANGUAGE plpgsql
AS $function$
BEGIN
  RETURN (SELECT nazev
                    FROM okresy
                   WHERE okresy.id = nazev_okresu.id);
END;
$function$
```

# PLpgSQL execution wrapping query

```
postgres=# EXPLAIN ANALYZE SELECT * FROM obce
                        WHERE nazev_okresu(okres_id) = 'Benešov';
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
 Seq Scan on obce  (cost=0.00..1699.62 rows=31 width=41)
                   (actual time=0.289..153.959 rows=114 loops=1)
   Filter: (nazev_okresu((okres_id)::text) = 'Benešov'::text)
   Rows Removed by Filter: 6136
 Planning Time: 0.165 ms
 Execution Time: 154.046 ms
(5 rows)
```

# PLpgSQL execution
# USE VIEWS!

```
CREATE VIEW obce_okresy AS
   SELECT ob.*, ok.nazev AS nazev_okresu
     FROM obce ob
          JOIN okresy ok ON ob.okres_id = ok.id;
```

```
 EXPLAIN ANALYZE SELECT * FROM obce_okresy WHERE nazev_okresu = 'Benešov';
                                  QUERY PLAN
 ------------------------------------------------------------------------------
  Nested Loop  (cost=0.28..15.19 rows=81 width=51)
                       (actual time=0.085..0.441 rows=114 loops=1)
    ->  Seq Scan on okresy ok  (cost=0.00..1.96 rows=1 width=17)
                       (actual time=0.032..0.075 rows=1 loops=1)
          Filter: (nazev = 'Benešov'::text)
          Rows Removed by Filter: 76
    ->  Index Scan using obce_okres_id_idx on obce ob
                  (cost=0.28..12.41 rows=81 width=41)
                  (actual time=0.042..0.223 rows=114 loops=1)
          Index Cond: ((okres_id)::text = ok.id)
  Execution Time: 0.555 ms
```

# PLpgSQL execution
# USE VIEWS!

```
CREATE OR REPLACE FUNCTION id_okresu(nazev text)
RETURNS text AS $$
BEGIN
RETURN (SELECT id FROM okresy WHERE okresy.nazev = id_okresu.nazev);
END;
$$ LANGUAGE plpgsql;
```

```
EXPLAIN ANALYZE SELECT * FROM obce WHERE okres_id = id_okresu('Praha');
                                                  QUERY PLAN

-------------------------------------------------------------------------------------------------
 Seq Scan on obce  (cost=0.00..1699.62 rows=81 width=41)
             (actual time=0.722..152.348 rows=1 loops=1)
   Filter: ((okres_id)::text = id_okresu('Praha'::text))
   Rows Removed by Filter: 6249
 Planning Time: 0.168 ms
 Execution Time: 152.393 ms
(5 rows)
```

# PLpgSQL execution
# USE VIEWS!

```
CREATE OR REPLACE FUNCTION id_okresu(nazev text)
RETURNS text AS $$
BEGIN
RETURN (SELECT id FROM okresy WHERE okresy.nazev = id_okresu.nazev);
END;
$$ LANGUAGE plpgsql STABLE;
```

```
EXPLAIN ANALYZE SELECT * FROM obce WHERE okres_id = id_okresu('Praha');
                                      QUERY PLAN
-------------------------------------------------------------------------------------
 Index Scan using obce_okres_id_idx on obce  (cost=0.53..8.55 rows=1 width=41)
                                             (actual time=0.305..0.308 rows=1 loops=1)
   Index Cond: ((okres_id)::text = id_okresu('Praha'::text))
 Planning Time: 0.881 ms
 Execution Time: 0.362 ms
(4 rows)
```

# PLpgSQL execution

- Use simple expressions everywhere it is possible
- Don’t use useless queries
- Don’t use ISAM style
- Don’t wrap queries (default VOLATILE function must be executed every time - sometimes STABLE helps)

# PLpgSQL execution slow queries due wrong types

```
EXPLAIN ANALYZE SELECT * FROM foo where id = 10;
                                                    QUERY PLAN

-------------------------------------------------------------------------------------------------
 Index Scan using foo_id_key on foo  (cost=0.42..8.44 rows=1 width=8)
                                (actual time=0.053..0.056 rows=1 loops=1)
   Index Cond: (id = 10)
 Planning Time: 0.100 ms
 Execution Time: 0.113 ms
(4 rows)
```

# PLpgSQL execution slow queries due wrong types

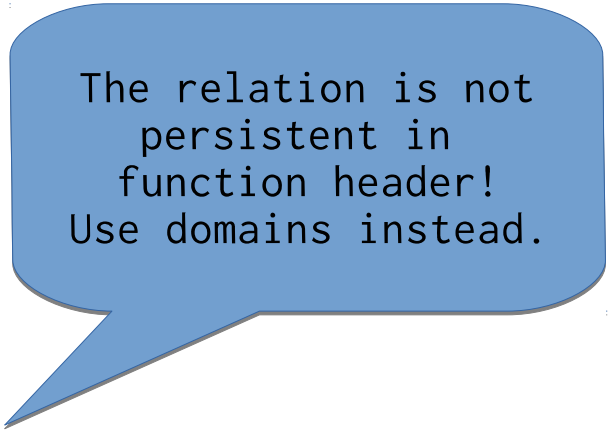
```
EXPLAIN ANALYZE SELECT * FROM foo where id = 10::numeric;
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..12175.00 rows=5000 width=8)
        (actual time=0.593..135.557 rows=1 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Seq Scan on foo  (cost=0.00..10675.00 rows=2083 width=8)
                                 (actual time=79.696..124.677 rows=0 loops=3)
         Filter: ((id)::numeric = '10'::numeric)
         Rows Removed by Filter: 333333
 Planning Time: 0.250 ms
 Execution Time: 138.087 ms
(8 rows)
```

# PLpgSQL execution slow queries due wrong types

- Often mistake
  - id in table are bigint, int
  - but variables are declared like numeric
- Use domains
- Use referenced types

# PLpgSQL execution slow queries due wrong types

```
do $$
declare _id foo.id%type;
begin
end;
$$ ;
```




```
create or replace function fx(_id foo.id%type)
returns void as $$
begin
end;
$$ language plpgsql;
```

# PLpgSQL development

- Don’t store code directly in db
  - use files
  - use git
- Don’t use reverse engineering for your code deployment
  - use rpm, deb, ...

# PLpgSQL development good tools

- plpgsql_check
- auto_explain (log_nested_statements)
- PLpgSQL profiler (plprofiler)
- pgAdmin has integrated PLpgSQL debugger

# PLpgSQL development good tools

```
SET track_functions TO 'pl';
EXPLAIN ANALYZE SELECT * FROM obce WHERE nazev_okresu(okres_id) = 'Benešov';
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------
 Seq Scan on obce  (cost=0.00..1699.62 rows=31 width=41)
                   (actual time=0.199..149.783 rows=114 loops=1)
   Filter: (nazev_okresu((okres_id)::text) = 'Benešov'::text)
   Rows Removed by Filter: 6136
 Planning Time: 0.142 ms
 Execution Time: 149.842 ms
(5 rows)

select * from pg_stat_user_functions ;
 funcid | schemaname |   funcname   | calls | total_time | self_time
--------+------------+--------------+-------+------------+-----------
  16414 | public     | nazev_okresu |  6250 |    146.022 |   146.022
(1 row)
```

# PLpgSQL hints

- Don’t use short live temp tables (if it is not really really necessary)! Use arrays.
- Don’t catch all errors.
- Catching exception is not gratis - do it only when it is necessary.
- Write tests.
- Write readable code - PLpgSQL is not good for dynamic code - don’t afraid use PLPerl, PLPython