

Optymalizacja programów korzystających z bazy PostgreSQL

<depesz@depesz.pl>

plan

- optymalizacja backendu
- optymalizacja sposobów dostępu
- stosowanie indeksów
- optymalizacja zapytań
- kiedy optymalizować
- specyficzne sytuacje/przypadki

optymalizacja sposobów dostępu

```
psql# \d first
```

```
          Table "public.first"  
Column | Type | Modifiers  
-----+-----+-----  
id     | integer | not null default nextval('public.first_id_seq'::text)  
name   | text    |
```

Indexes:

```
"first_pkey" primary key, btree (id)
```

```
psql# \d second
```

```
          Table "public.second"  
Column | Type | Modifiers  
-----+-----+-----  
first_id | integer |  
xxx     | text    |
```

Foreign-key constraints:

```
"$1" FOREIGN KEY (first_id) REFERENCES "first"(id)
```

optymalizacja sposobów dostępu

```
psql# \d test1
View "public.test1"
Column | Type | Modifiers
-----+-----+-----
id | integer |
name | text |
xxx | text |
View definition:
SELECT f.id, f.name, s.xxx
FROM "first" f
JOIN "second" s ON f.id = s.first_id;
```

```
psql# \d test2
View "public.test2"
Column | Type | Modifiers
-----+-----+-----
id | integer |
name | text |
View definition:
SELECT f.id, f.name
FROM "first" f;
```

optymalizacja sposobów dostępu

```
psql# explain select name from test1 where id = 1;
```

```
QUERY PLAN
```

```
Nested Loop (cost=22.50..27.56 rows=12 width=32)
```

```
-> Index Scan using first_pkey on "first" f (cost=0.00..4.82 rows=2 width=36)
```

```
Index Cond: (id = 1)
```

```
-> Materialize (cost=22.50..22.56 rows=6 width=4)
```

```
-> Seq Scan on "second" s (cost=0.00..22.50 rows=6 width=4)
```

```
Filter: (1 = first_id)
```

```
(6 rows)
```

```
psql# explain select name from test2 where id = 1;
```

```
QUERY PLAN
```

```
Index Scan using first_pkey on "first" f (cost=0.00..4.82 rows=2 width=32)
```

```
Index Cond: (id = 1)
```

```
(2 rows)
```

cacheowanie planów zapytań

PREPARE/EXECUTE

```
psql# \d test
```

```
Table "public.test"
```

```
Column | Type | Modifiers
```

```
-----+-----+-----
```

```
id | integer | not null default nextval('public.test_id_seq'::text)
```

```
name | text |
```

```
Indexes:
```

```
 "test_pkey" primary key, btree (id)
```

```
psql# select count(*), count(distinct name) from test;
```

```
count | count
```

```
-----+-----
```

```
100000 | 100000
```

```
(1 row)
```

cacheowanie planów zapytań

PREPARE/EXECUTE

```
psql# explain select * from test where id = 12;  
          QUERY PLAN
```

```
-----  
Index Scan using test_pkey on test (cost=0.00..3.01 rows=2 width=11)  
  Index Cond: (id = 12)  
(2 rows)
```

```
psql# prepare test_plan(integer) as select * from test where id = $1;  
PREPARE
```

```
psql# explain execute test_plan (12);  
          QUERY PLAN
```

```
-----  
Index Scan using test_pkey on test (cost=0.00..3.01 rows=2 width=11)  
  Index Cond: (id = $1)  
(2 rows)
```

cacheowanie planów zapytań

```
#!/usr/bin/perl
use Pg;
use DBI;
use Benchmark qw(:all);

my @aItems = ();
for (my $i = 0; $i < 1000; $i++) {
    push @aItems, int(rand(100000));
}

my $conn = Pg::connectdb("dbname=depez");
my $dbh = DBI->connect("dbi:Pg:dbname=depez");

cmpthese(10, {'pgstandard' => \&test1, 'pgprepare' => \&test2, 'dbistandard' => \&test3,
'dbicached' => \&test4, 'dbisingle' => \&test5});

exit;

sub test1 {
    for my $sNumber (@aItems) {
        my $result = $conn->exec("SELECT name FROM test WHERE id = $sNumber");
        my @aData = $result->fetchrow();
    }
}
```

cacheowanie planów zapytań

```
•sub test2 {
•$conn->exec('PREPARE test_plan(integer) AS SELECT name FROM test WHERE id = $1');
•for my $sNumber (@aItems) {my $result = $conn->exec("EXECUTE test_plan ($sNumber)");
•my @aData = $result->fetchrow();
•}}
•
•sub test3 {
•for my $sNumber (@aItems) {
•my $sth = $dbh->prepare("SELECT name FROM test WHERE id = ?");
•$sth->execute($sNumber);my @aData = $sth->fetchrow_array();$sth->finish();
•}}
•
•sub test4 {
•for my $sNumber (@aItems) {
•my $sth = $dbh->prepare_cached("SELECT name FROM test WHERE id = ?");
•$sth->execute($sNumber);my @aData = $sth->fetchrow_array();$sth->finish();
•}}
•
•sub test5 {
•my $sth = $dbh->prepare("SELECT name FROM test WHERE id = ?");
•for my $sNumber (@aItems) {$sth->execute($sNumber);
•my @aData = $sth->fetchrow_array();$sth->finish();
•}}
```

cacheowanie planów zapytań

	Rate dbistandard	dbicached	dbisingle	pgstandard	pgprepare	
dbistandard	1.53/s	--	-46%	-56%	-68%	-70%
dbicached	2.82/s	84%	--	-19%	-41%	-45%
dbisingle	3.48/s	128%	24%	--	-28%	-32%
pgstandard	4.81/s	214%	71%	38%	--	-7%
pgprepare	5.15/s	237%	83%	48%	7%	--

cacheowanie planów zapytań

```
SELECT n.nspname as "Schema",
       c.relname as "Name",
       CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'i' THEN 'index' WHEN 'S'
THEN 'sequence' WHEN 's' THEN 'special' END as "Type",
       u.username as "Owner"
FROM pg_catalog.pg_class c
     LEFT JOIN pg_catalog.pg_user u ON u.usesysid = c.relowner
     LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r','')
      AND n.nspname NOT IN ('pg_catalog', 'pg_toast')
      AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
```

	Rate	pgstandard	pgprepare
pgstandard	3.57/s	--	-42%
pgprepare	6.21/s	74%	--

modyfikacja skomplikowanych zapytań do postaci procedur

```
psql# \d test
```

```
                Table "public.test"  
Column | Type | Modifiers  
-----+-----+-----  
id     | integer | not null default nextval('public.test_id_seq'::text)  
name   | text    |  
Indexes:  
    "test_pkey" primary key, btree (id)
```

```
psql# create function insertItem(text) returns int4 as '  
declare  
in_name alias for $1;  
begin  
insert into test (name) values (in_name);  
return curval("test_id_seq");  
end;  
' language 'plpgsql';
```

modyfikacja skomplikowanych zapytań do postaci procedur

```
#!/usr/bin/perl
use Pg;
use Benchmark qw(:all);

my $conn = Pg::connectdb("dbname=depesz");

cmpthese(1000, {'insert' => \&test1, 'function' => \&test2});

exit;

sub test1 {
my $result = $conn->exec("insert into test (name) values ('aa')");
my @aData = $result->fetchrow();
$result = $conn->exec("select currval('test_id_seq')");
@aData = $result->fetchrow();
}

sub test2 {
my $result = $conn->exec("select insertItem('bb')");
my @aData = $result->fetchrow();
}
```

modyfikacja skomplikowanych zapytań do postaci procedur

	Rate	insert	function	
insert	2933/s	--	-33%	
function	4386/s	50%	--	

użycie indeksów wielopolowych

```
psql# select min(a), max(a), min(b), max(b), min(c), max(c) from test;
 min | max | min | max | min | max
-----+-----+-----+-----+-----+-----
  1 | 5000 | 1 | 5000 | 1 | 5000
(1 row)
```

```
psql# \d test
Table "public.test"
Column | Type | Modifiers
-----+-----+-----
 a     | integer |
 b     | integer |
 c     | integer |
Indexes:
 "ti" btree (a, b, c)
```

```
psql# select count(*), count(distinct a), count(distinct b), count(distinct c) from test;
 count | count | count | count
-----+-----+-----+-----
 5000 | 5000 | 5000 | 5000
(1 row)
```

użycie indeksów wielopolowych

```
psql> explain select * from test where a=10 and b=10 and c=10;  
          QUERY PLAN
```

```
-----  
Index Scan using ti on test (cost=0.00..5.62 rows=1 width=12)  
  Index Cond: ((a = 10) AND (b = 10) AND (c = 10))  
(2 rows)
```

```
psql> explain select * from test where a=10;  
          QUERY PLAN
```

```
-----  
Index Scan using ti on test (cost=0.00..5.62 rows=2 width=12)  
  Index Cond: (a = 10)  
(2 rows)
```

```
psql> explain select * from test where b=10;  
          QUERY PLAN
```

```
-----  
Seq Scan on test (cost=0.00..90.50 rows=2 width=12)  
  Filter: (b = 10)  
(2 rows)
```

użycie indeksów wielopoloowych

```
psql> explain select * from test where a=10 and b=10;
```

```
QUERY PLAN
```

```
-----  
Index Scan using ti on test (cost=0.00..5.62 rows=1 width=12)  
  Index Cond: ((a = 10) AND (b = 10))  
(2 rows)
```

```
psql> explain select * from test where a=10 and c=10;
```

```
QUERY PLAN
```

```
-----  
Index Scan using ti on test (cost=0.00..5.62 rows=1 width=12)  
  Index Cond: (a = 10)  
  Filter: (c = 10)  
(3 rows)
```

```
psql> explain select * from test where b=10 and c=10;
```

```
QUERY PLAN
```

```
-----  
Seq Scan on test (cost=0.00..103.00 rows=1 width=12)  
  Filter: ((b = 10) AND (c = 10))  
(2 rows)
```

indeksy funkcyjne i częściowe

```
create index nazwa on tabela (upper(pole_a));
```

```
select * from nazwa where pole_a = 'DEPESZ';
```

```
select * from nazwa where upper(pole_a) = 'DEPESZ';
```

```
create index nazwa on tabela (pole) where inne_pole = 1;
```

explain

```
psql> explain select * from search_data where au_au_id = 33;
```

```
QUERY PLAN
```

```
-----  
Nested Loop (cost=3.02..26.10 rows=4 width=2195)  
  -> Nested Loop (cost=3.02..13.94 rows=4 width=1359)  
    -> Hash Join (cost=3.02..4.33 rows=1 width=1291)  
        Hash Cond: ("outer".rp_id = "inner".rp_id)  
          -> Seq Scan on repository r (cost=0.00..1.20 rows=20 width=461)  
          -> Hash (cost=3.02..3.02 rows=1 width=830)  
              -> Index Scan using archivalunit_pkey on archivalunit au (cost=0.00..3.02 rows=1  
width=830)  
                  Index Cond: (au_id = 33)  
            -> Index Scan using hdl_i_4 on creation c (cost=0.00..9.57 rows=4 width=68)  
                Index Cond: (33 = au_id)  
          -> Index Scan using recordscreator_pkey on recordscreator rc (cost=0.00..3.01 rows=1  
width=836)  
              Index Cond: ("outer".cr_id = rc.cr_id)  
(12 rows)
```

explain analyze

```
Nested Loop (cost=3.02..26.10 rows=4 width=2195) (actual time=0.470..1.088 rows=4 loops=1)
-> Nested Loop (cost=3.02..13.94 rows=4 width=1359) (actual time=0.297..0.567 rows=4 loops=1)
    -> Hash Join (cost=3.02..4.33 rows=1 width=1291) (actual time=0.192..0.268 rows=1 loops=1)
        Hash Cond: ("outer".rp_id = "inner".rp_id)
            -> Seq Scan on repository r (cost=0.00..1.20 rows=20 width=461) (actual time=0.010..0.049
rows=20 loops=1)
                -> Hash (cost=3.02..3.02 rows=1 width=830) (actual time=0.074..0.074 rows=0 loops=1)
                    -> Index Scan using archivalunit_pkey on archivalunit au (cost=0.00..3.02 rows=1
width=830) (actual time=0.053..0.057 rows=1 loops=1)
                        Index Cond: (au_id = 33)
                            -> Index Scan using hdl_i_4 on creation c (cost=0.00..9.57 rows=4 width=68) (actual
time=0.024..0.048 rows=4 loops=1)                                Index Cond: (33 = au_id)
                                -> Index Scan using recordscreator_pkey on recordscreator rc (cost=0.00..3.01 rows=1 width=836)
(actual time=0.018..0.022 rows=1 loops=4)
                                    Index Cond: ("outer".cr_id = rc.cr_id)
Total runtime: 1.831 ms
```

wykrywanie sytuacji nieindeksowalnych

```
select * from tabela where pole like '%...%';
```

```
select min(pole) from tabela;
```

```
select pole from tabela order by pole asc limit 1;
```

wymuszanie typowania (cast)

```
psql# \d test
```

```
Table "public.test"  
Column | Type | Modifiers  
-----+-----+-----  
a      | bigint |  
Indexes:  
"xxx" unique, btree (a)
```

```
psql# select min(a), max(a), count(a) from test;
```

```
min | max | count  
----+-----+-----  
1 | 100000 | 100000  
(1 row)
```

```
psql# explain select * from test where a = 2334;
```

```
QUERY PLAN  
-----  
Seq Scan on test (cost=0.00..1741.00 rows=2 width=8)  
Filter: (a = 2334)  
(2 rows)
```

```
psql# explain select * from test where a = cast(2334 as int8);
```

```
QUERY PLAN  
-----  
Index Scan using xxx on test (cost=0.00..3.01 rows=2 width=8)  
Index Cond: (a = 2334::bigint)  
(2 rows)
```

agregacja zapytań

```
#!/usr/bin/perl
use DBI;
use Benchmark qw(:all);

my $dbh = DBI->connect("dbi:Pg:dbname=depesz");
cmpthese(1000, { 'multi' => \&test1, 'single' => \&test2, 'single-sub' => \&test3});

exit;

sub test1 {
my $rData = $dbh->selectall_arrayref('select id from test1 where pole between 100000 and 100050');
for my $rRow (@$rData) {
my $rTemp = $dbh->selectall_arrayref('select * from test2 where pole between 100000 and 205000 and t1_id = ' . $rRow->[0]);
}
}

sub test2 {
my $rData = $dbh->selectall_arrayref('select t2.* from test1 t1 join test2 t2 on t1.id = t2.t1_id where t1.pole between 100000 and 100050 and t2.pole between 100000 and 205000');
}
```

agregacja zapytań

```
sub test3 {  
my $rData = $dbh->selectall_arrayref('select * from (select * from test1 where pole between  
100000 and 100050) t1 join test2 t2 on t1.id = t2.t1_id WHERE t2.pole between 100000 and  
205000');  
}
```

	Rate	multi	single-sub	single
multi	107/s	--	-91%	-92%
single-sub	1136/s	959%	--	-14%
single	1316/s	1126%	16%	--

```
psql# explain analyze select id from test1 where pole between 100000 and 100050;
```

QUERY PLAN

```
Index Scan using h1 on test1 (cost=0.00..72.60 rows=18 width=4) (actual time=34.102..34.702  
rows=14 loops=1)  
  Index Cond: ((pole >= 100000) AND (pole <= 100050))  
Total runtime: 34.881 ms  
(3 rows)
```

agregacja zapytań

```
psql# explain analyze select * from test2 where pole between 100000 and 205000 and t1_id = 22944;
```

QUERY PLAN

```
Index Scan using h2 on test2 (cost=0.00..3.02 rows=1 width=12) (actual time=0.324..0.324 rows=0 loops=1)
```

```
  Index Cond: (t1_id = 22944)
```

```
  Filter: ((pole >= 100000) AND (pole <= 205000))
```

```
Total runtime: 0.445 ms
```

```
(4 rows)
```

agregacja zapytań

```
psql# explain analyze select t2.* from test1 t1 join test2 t2 on t1.id = t2.t1_id where t1.pole  
between 100000 and 100050 and t2.pole between 100000 and 205000;
```

QUERY PLAN

```
-----  
Nested Loop (cost=0.00..127.16 rows=4 width=12) (actual time=0.442..1.613 rows=5 loops=1)  
  -> Index Scan using h1 on test1 t1 (cost=0.00..72.60 rows=18 width=4) (actual  
time=0.296..0.615 rows=14 loops=1)  
    Index Cond: ((pole >= 100000) AND (pole <= 100050))  
  -> Index Scan using h2 on test2 t2 (cost=0.00..3.02 rows=1 width=12) (actual  
time=0.061..0.062 rows=0 loops=14)  
    Index Cond: ("outer".id = t2.t1_id)  
    Filter: ((pole >= 100000) AND (pole <= 205000))  
Total runtime: 1.828 ms  
(7 rows)
```

agregacja zapytań

```
psql# explain analyze select * from (select * from test1 where pole between 100000 and 100050)
t1 join test2 t2 on t1.id = t2.t1_id WHERE t2.pole between 100000 and 205000;
```

QUERY PLAN

```
-----
-----
Nested Loop (cost=0.00..127.16 rows=4 width=20) (actual time=1.062..3.172 rows=5 loops=1)
  -> Index Scan using h1 on test1 (cost=0.00..72.60 rows=18 width=8) (actual
time=0.673..1.314 rows=14 loops=1)
    Index Cond: ((pole >= 100000) AND (pole <= 100050))
  -> Index Scan using h2 on test2 t2 (cost=0.00..3.02 rows=1 width=12) (actual
time=0.118..0.120 rows=0 loops=14)
    Index Cond: ("outer".id = t2.t1_id)
    Filter: ((pole >= 100000) AND (pole <= 205000))
Total runtime: 3.403 ms
(7 rows)
```

użycie joina zamiast subselectów

```
select * from tabelka_a where pole_a in (select pole_a from tabelka_b where pole_b in (select pole_b from tabelka_c where ...));
```

```
select polea, (select count(*) from tabelka_b where b_polea = tabelka_a.polea) from tabelka_a where ...
```

użycie joina zamiast subselectów

```
psql# explain analyze select count(t1.*) from test1 t1 where not exists (select * from test2 where t1_id = t1.id)
```

QUERY PLAN

```
-----  
Aggregate (cost=304581.00..304581.00 rows=1 width=4) (actual time=4743.891..4743.892 rows=1 loops=1)
```

```
  -> Seq Scan on test1 t1 (cost=0.00..304331.00 rows=100000 width=4) (actual time=2.996..4736.745 rows=2006 loops=1)
```

```
    Filter: (NOT (subplan))
```

```
    SubPlan
```

```
      -> Index Scan using h2 on test2 (cost=0.00..3.01 rows=2 width=12) (actual time=0.016..0.016 rows=1 loops=200000)
```

```
        Index Cond: (t1_id = $0)
```

```
Total runtime: 4744.159 ms  
(7 rows)
```

użycie joina zamiast subselectów

```
psql# explain analyze select count(t1.*) from test1 t1 left outer join test2 t2 on t1.id = t2.t1_id
where t2.pole is null;
```

QUERY PLAN

```
-----  
-----  
Aggregate (cost=11014.60..11014.60 rows=1 width=4) (actual time=3794.841..3794.843  
rows=1 loops=1)  
  -> Merge Left Join (cost=0.00..10514.60 rows=200000 width=4) (actual  
time=2.552..3785.111 rows=2006 loops=1)  
    Merge Cond: ("outer".id = "inner".t1_id)  
    Filter: ("inner".pole IS NULL)  
      -> Index Scan using test1_pkey on test1 t1 (cost=0.00..3793.00 rows=200000 width=8)  
(actual time=0.275..1304.831 rows=200000 loops=1)  
        -> Index Scan using h2 on test2 t2 (cost=0.00..3723.00 rows=200000 width=8) (actual  
time=0.233..870.045 rows=197994 loops=1)  
Total runtime: 3798.368 ms
```

użycie subselecta zamiast joina

```
psql# explain analyze select distinct n.* from cat c join news2cat nc on c.id = nc.cat_id join newses n on nc.news_id = n.id where c.id in (1,2,3,4,5,6);
```

QUERY PLAN

```
-----  
-----  
Unique (cost=81.50..86.06 rows=200 width=12) (actual time=14.160..16.803 rows=173 loops=1)  
-> Sort (cost=81.50..83.02 rows=608 width=12) (actual time=14.155..14.745 rows=605 loops=1)  
    Sort Key: n.id, n.title  
    -> Hash Join (cost=5.89..53.39 rows=608 width=12) (actual time=1.107..12.227 rows=605  
loops=1)  
        Hash Cond: ("outer".news_id = "inner".id)  
        -> Hash Join (cost=1.39..38.24 rows=608 width=4) (actual time=0.177..8.067 rows=605  
loops=1)  
            Hash Cond: ("outer".cat_id = "inner".id)  
            -> Seq Scan on news2cat nc (cost=0.00..23.18 rows=1518 width=8) (actual  
time=0.006..3.273 rows=1518 loops=1)  
                -> Hash (cost=1.38..1.38 rows=6 width=4) (actual time=0.118..0.118 rows=0 loops=1)  
                    -> Seq Scan on cat c (cost=0.00..1.38 rows=6 width=4) (actual time=0.020..0.104  
rows=6 loops=1)  
                        Filter: ((id = 1) OR (id = 2) OR (id = 3) OR (id = 4) OR (id = 5) OR (id = 6))  
            -> Hash (cost=4.00..4.00 rows=200 width=12) (actual time=0.747..0.747 rows=0 loops=1)  
                -> Seq Scan on newses n (cost=0.00..4.00 rows=200 width=12) (actual time=0.012..0.391  
rows=200 loops=1)  
Total runtime: 17.272 ms
```

użycie subselecta zamiast joina

```
psql# explain analyze select n.* from newses n where exists (select * from news2cat n2c join cat  
c on n2c.cat_id = c.id where c.id in (1,2,3,4,5,6) and n2c.news_id = n.id);
```

QUERY PLAN

```
-----  
-----  
Seq Scan on newses n (cost=0.00..441.70 rows=100 width=12) (actual time=0.386..7.854  
rows=173 loops=1)  
  Filter: (subplan)  
  SubPlan  
    -> Hash Join (cost=1.39..4.58 rows=4 width=20) (actual time=0.033..0.033 rows=1  
loops=200)  
      Hash Cond: ("outer".cat_id = "inner".id)  
      -> Index Scan using x2 on news2cat n2c (cost=0.00..3.11 rows=9 width=8) (actual  
time=0.014..0.019 rows=2 loops=200)  
        Index Cond: (news_id = $0)  
      -> Hash (cost=1.38..1.38 rows=6 width=12) (actual time=0.107..0.107 rows=0 loops=1)  
        -> Seq Scan on cat c (cost=0.00..1.38 rows=6 width=12) (actual time=0.015..0.089  
rows=6 loops=1)  
          Filter: ((id = 1) OR (id = 2) OR (id = 3) OR (id = 4) OR (id = 5) OR (id = 6))  
Total runtime: 8.456 ms
```

sprawdzanie stanu bazy “ad-hoc”

```
psql# \dSv pg_stat*
```

List of relations

Schema	Name	Type	Owner
pg_catalog	pg_stat_activity	view	postgres
pg_catalog	pg_stat_all_indexes	view	postgres
pg_catalog	pg_stat_all_tables	view	postgres
pg_catalog	pg_stat_database	view	postgres
pg_catalog	pg_stat_sys_indexes	view	postgres
pg_catalog	pg_stat_sys_tables	view	postgres
pg_catalog	pg_stat_user_indexes	view	postgres
pg_catalog	pg_stat_user_tables	view	postgres
pg_catalog	pg_statio_all_indexes	view	postgres
pg_catalog	pg_statio_all_sequences	view	postgres
pg_catalog	pg_statio_all_tables	view	postgres
pg_catalog	pg_statio_sys_indexes	view	postgres
pg_catalog	pg_statio_sys_sequences	view	postgres
pg_catalog	pg_statio_sys_tables	view	postgres
pg_catalog	pg_statio_user_indexes	view	postgres
pg_catalog	pg_statio_user_sequences	view	postgres
pg_catalog	pg_statio_user_tables	view	postgres
pg_catalog	pg_stats	view	postgres

```
(18 rows)
```

sprawdzanie stanu bazy “ad-hoc”

`stats_start_collector = true`

`stats_command_string = true`

`stats_block_level = false`

`stats_row_level = true`

analiza statystyczna działania systemu

```
log_duration = true  
log_pid = true  
log_statement = true  
log_timestamp = true
```

```
Mar 17 21:38:59 master postgres[24880]: [4-1] LOG: statement: explain analyze      select n.*  
from newses n where exists (select * from news2cat n2c join cat c on  
Mar 17 21:38:59 master postgres[24880]: [4-2] n2c.cat_id = c.id where c.id in (1,2,3,4,5,6) and  
n2c.news_id = n.id);  
Mar 17 21:38:59 master postgres[24880]: [5-1] LOG: duration: 18.924 ms
```

```
2004-03-17 21:46:47 [12110] LOG: statement: explain analyze      select n.* from newses n  
where exists (select * from news2cat n2c join cat c on n2c.cat_id = c.id where c.id in  
(1,2,3,4,5,6) and n2c.news_id = n.id);  
2004-03-17 21:46:47 [12110] LOG: duration: 83.700 ms
```

- 12.322 select * from tabelka where pole = _D_;
- 17.216 select id from tabelka join inna on polea = poleb where mvwo(pole, _D_);

analiza zużycia pamięci

```
#!/bin/sh
PATH=/bin:/usr/bin:/usr/local/bin
date
ps -U postgres ufw
```

czw mar 18 15:40:06 CET 2004

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
postgres	1393	0.0	0.7	4456	1444	?	S	07:26	0:01	/usr/lib/postgresql/bin/pg_autovacuum -D -p 5432 -L /var/log/postgresql/autovacuum_log
postgres	30806	0.0	1.2	19292	2372	tty3	S	07:26	0:00	/usr/lib/postgresql/bin/postmaster -D /var/lib/postgres/data
postgres	12467	0.0	1.5	10092	2948	tty3	S	07:26	0:00	_ postgres: stats buffer process
postgres	32232	0.0	1.1	9232	2148	tty3	S	07:26	0:00	_ postgres: stats collector process
postgres	18647	0.0	3.1	20624	6092	tty3	S	15:05	0:00	_ postgres: marcin aukcje [local] idle

optymalizacja zapytań count(*)

```
create or replace function count_trigger() returns trigger as '  
declare  
active int4;  
temprec record;  
begin  
select count into active from counts where table_name = cast(TG_RELNAME as text);  
if not found then  
for temprec in execute "select count(*) from " || TG_RELNAME LOOP  
active := temprec.count;  
end loop;  
insert into counts (table_name, count) values (TG_RELNAME, active);  
return NEW;  
end if;  
if (TG_OP = "INSERT") then  
active := active + 1;  
ELSE  
active := active - 1;  
END IF;  
update counts set count = active where table_name = TG_RELNAME;  
return NEW;  
end;  
' language 'plpgsql';
```

optymalizacja zapytań count(*)

```
create table counts (table_name text, count int4, primary key(table_name));
```

```
create trigger ct_1 after insert or delete on test1 for each row execute procedure count_trigger();
```

```
create trigger ct_2 after insert or delete on test2 for each row execute procedure count_trigger();
```

struktury drzewiaste a ltree

Top.*{0,2}.sport*@.!football|tennis.Russ*|Spain

Europe & Russia* @ & !Transportation

Pytania