

2ndQuadrant[®] 
PostgreSQL

CREATE STATISTICS

What is it for?

Tomas Vondra <tomas.vondra@2ndquadrant.com>



Agenda

- Quick intro into planning and estimates.
- Estimates with correlated columns.
- CREATE STATISTICS to the rescue!
 - functional dependencies
 - ndistinct
- Future improvements.



ZIP_CODES

```
CREATE TABLE zip_codes (  
    postal_code      VARCHAR(20),  
    place_name       VARCHAR(180),  
    state_name       VARCHAR(100),  
    province_name    VARCHAR(100),  
    community_name   VARCHAR(100),  
    latitude         REAL,  
    longitude        REAL  
);
```

```
cat create-table.sql | psql test
```

```
cat zip-codes-portugal.csv | psql test -c "copy zip_codes from stdin"
```

```
-- http://download.geonames.org/export/zip/
```



EXPLAIN

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Lisboa';
```

QUERY PLAN

```
Seq Scan on zip_codes (cost=0.00..4860.76 rows=8588 width=56)
    (actual rows=9166 loops=1)
  Filter: ((place_name)::text = 'Lisboa'::text)
 Rows Removed by Filter: 197775
```



reltuples , relpages

```
SELECT reltuples, relpages FROM pg_class  
WHERE relname = 'zip_codes';
```

```
reltuples | relpages  
-----+-----  
    206941 |      2274
```



```
SELECT * FROM pg_stats
WHERE tablename = 'zip_codes'
AND attname = 'place_name';
```

```
-----+-----
schemaname      | public
tablename       | zip_codes
attname         | place_name
...            | ...
most_common_vals | {Lisboa, Porto, "Vila Nova de Gaia", Maia, ...}
most_common_freqs | {0.0415, 0.0206333, 0.00896667, 0.00893333, ...}
...            | ...
```



```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Lisboa';
```

QUERY PLAN

```
-----
Seq Scan on zip_codes (cost=0.00..4860.76 rows=8588 width=56)
      (actual rows=9166 loops=1)
```

```
reltuples      | 206941
most_common_vals | {Lisboa, ...}
most_common_freqs | {0.0415, ...}
```

$$206941 * 0.0415 = 8588.0515$$



```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE state_name = 'Lisboa';
```

QUERY PLAN

```
-----
Seq Scan on zip_codes (cost=0.00..4860.76 rows=34263 width=56)
      (actual rows=35230 loops=1)
```

```
reltuples          | 206941
most_common_vals   | {Lisboa, ...}
most_common_freqs  | {0.165567, ...}
```

$$206941 * 0.165567 = 34262.6$$



Underestimate

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Lisboa'
        AND state_name = 'Lisboa';
```

QUERY PLAN

```
Seq Scan on zip_codes (cost=0.00..5378.11 rows=1422 width=56)
      (actual rows=9165 loops=1)
   Filter: (((place_name)::text = 'Lisboa'::text)
            AND ((state_name)::text = 'Lisboa'::text))
  Rows Removed by Filter: 197776
```



$$P(A \& B) = P(A) * P(B)$$



```
SELECT * FROM zip_codes
  WHERE place_name = 'Lisboa'
  AND state_name = 'Lisboa';
```

```
P(place_name = 'Lisboa' & county_name = 'Lisboa')
= P(place_name = 'Lisboa') * P(state_name = 'Lisboa')
= 0.0415 * 0.165567
= 0.0068710305
```

206941 * 0.0068710305 = 1421.898



Underestimate

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Lisboa'
        AND state_name = 'Lisboa';
```

QUERY PLAN

```
Seq Scan on zip_codes (cost=0.00..5378.11 rows=1422 width=56)
      (actual rows=9165 loops=1)
   Filter: (((place_name)::text = 'Lisboa'::text)
            AND ((state_name)::text = 'Lisboa'::text))
  Rows Removed by Filter: 197776
```



Overestimate

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Lisboa'
        AND state_name != 'Lisboa';
```

QUERY PLAN

```
Seq Scan on zip_codes  (cost=0.00..5378.11 rows=7166 width=56)
    (actual rows=1 loops=1)
   Filter: (((state_name)::text <> 'Lisboa'::text)
            AND ((place_name)::text = 'Lisboa'::text))
  Rows Removed by Filter: 206940
```



Correlated columns

- Attribute Value Independence Assumption (AVIA)
 - may result in wildly inaccurate estimates
 - both underestimates and overestimates
- consequences
 - poor scan choices (Seq Scan vs. Index Scan)
 - poor join choices (Nested Loop)



Poor scan choices

Index Scan using orders_city_idx on orders
(cost=0.28..185.10 **rows=90** width=36)
(actual **rows=12248237** loops=1)

Seq Scan using on orders
(cost=0.13..129385.10 **rows=12248237** width=36)
(actual **rows=90** loops=1)



Poor join choices

- > Nested Loop (... rows=90 ...) (... **rows=12248237** ...)
- > Index Scan using orders_city_idx on orders
(cost=0.28..185.10 **rows=90** width=36)
(actual **rows=12248237** loops=1)
- ...
- > Index Scan ... (... **loops=12248237**)



Poor join choices

- > **Nested Loop** (... rows=90 ...) (... rows=12248237 ...)
- > **Nested Loop** (... rows=90 ...) (... rows=12248237 ...)
- > **Nested Loop** (... rows=90 ...) (... rows=12248237 ...)
 - > Index Scan using orders_city_idx on orders
(cost=0.28..185.10 rows=90 width=36)
(actual rows=12248237 loops=1)
 - ...
 - > Index Scan ... (... loops=12248237)
 - > Index Scan ... (... loops=12248237)
- > Index Scan ... (... loops=12248237)



functional dependencies (WHERE)



Functional Dependencies

- value in column A determines value in column B
- trivial example: primary key determines everything
 - zip code \rightarrow {place, state, province, community}
 - 4625-113 \rightarrow {Favões, Porto, Marco de Canaveses, Favões}
- other dependencies:
 - place \rightarrow community
 - community \rightarrow province
 - province \rightarrow state



CREATE STATISTICS

```
CREATE STATISTICS s (dependencies)
  ON place_name, state_name, province_name FROM zip_codes;
      2           3           4
ANALYZE zip_codes;
SELECT stxdependencies FROM pg_statistic_ext WHERE stxname = 's';
```

stxdependencies

```
{"2 => 3": 0.789467, "2 => 4": 0.774333,
 "4 => 2": 0.093300, "4 => 3": 0.993167,
 "2, 3 => 4": 0.951667, "2, 4 => 3": 0.998333,
 "3, 4 => 2": 0.093300}
```



place → state: 0.789 = d

$P(\text{place} = \text{'Lisboa'} \ \& \ \text{state} = \text{'Lisboa'}) =$

$P(\text{place} = \text{'Lisboa'}) * [d + (1-d) * P(\text{state} = \text{'Lisboa'})]$

$206941 * 0.0415 * (0.789 + (1-0.789) * 0.1656) = 7076.05$



Underestimate : fixed

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Lisboa'
        AND state_name = 'Lisboa';
```

QUERY PLAN

```
Seq Scan on zip_codes (cost=0.00..5378.11 rows=7076 width=56)
      (actual rows=9165 loops=1)
   Filter: (((place_name)::text = 'Lisboa'::text)
            AND ((state_name)::text = 'Lisboa'::text))
  Rows Removed by Filter: 197776
```



Overestimate #1: not fixed :-)

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Lisboa'
        AND state_name != 'Lisboa';
```

QUERY PLAN

```
Seq Scan on zip_codes (cost=0.00..5378.11 rows=7166 width=56)
    (actual rows=1 loops=1)
  Filter: (((state_name)::text <> 'Lisboa'::text)
    AND ((place_name)::text = 'Lisboa'::text))
Rows Removed by Filter: 206940
```

Functional dependencies only work with equalities.



Overestimate #2: not fixed :-)

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Lisboa'
                               AND state_name = 'Porto';
```

QUERY PLAN

```
Seq Scan on zip_codes (cost=0.00..5378.11 rows=1374 width=56)
    (actual rows=0 loops=1)
   Filter: (((place_name)::text = 'Lisboa'::text)
            AND ((state_name)::text = 'Porto'::text))
  Rows Removed by Filter: 206941
```

The queries need to respect the functional dependencies.



ndistinct (GROUP BY)



```
EXPLAIN (ANALYZE, TIMING off)
```

```
SELECT count(*) FROM zip_codes GROUP BY community_name;
```

QUERY PLAN

```
HashAggregate (cost=344126.92..344155.40 rows=2860 width=19)
```

```
(actual rows=3845 loops=1)
```

```
Group Key: community_name
```

```
-> Seq Scan on zip_codes (cost=0.00..277901.95 rows=13244995 width=11)
```

```
(actual rows=13244224 loops=1)
```

```
Planning Time: 0.219 ms
```

```
Execution Time: 6664.752 ms
```



```
SELECT attname, n_distinct
       FROM pg_stats WHERE tablename = 'zip_codes';
```

attname		n_distinct
place_name		6239
state_name		20
latitude		5532
longitude		5135
province_name		306
postal_code		171199
community_name		2860

(7 rows)



```
EXPLAIN (ANALYZE, TIMING off)
```

```
SELECT count(*) FROM zip_codes GROUP BY province_name, community_name;
```

QUERY PLAN

```
GroupAggregate (cost=2387970.92..2529135.75 rows=875160 width=29)
    (actual rows=3845 loops=1)
```

```
Group Key: province_name, community_name
```

```
-> Sort (cost=2387970.92..2421083.41 rows=13244995 width=21)
    (actual rows=13244224 loops=1)
```

```
Sort Key: province_name, community_name
```

```
Sort Method: external merge Disk: 415624kB
```

```
-> Seq Scan on zip_codes (cost=0.00..277901.95 rows=13244995 width=21)
    (actual rows=13244224 loops=1)
```

```
Planning Time: 1.116 ms
```

```
Execution Time: 48591.326 ms
```



```
EXPLAIN (ANALYZE, TIMING off)
```

```
SELECT count(*) FROM zip_codes GROUP BY province_name, community_name;
```

QUERY PLAN

```
GroupAggregate (cost=2387970.92..2529135.75 rows=875160 width=29)
```

```
(actual rows=3845 loops=1)
```

```
Group Key: province_name, community_name
```

```
-> Sort (cost=2387970.92..2421083.41 rows=13244995 width=21)
```

```
(actual rows=13244224 loops=1)
```

```
Sort Key: province_name, community_name
```

```
Sort Method: external merge Disk: 415624kB
```

```
-> Seq Scan on zip_codes (cost=0.00..277901.95 rows=13244995 width=21)
```

```
(actual rows=13244224 loops=1)
```

```
Planning Time: 1.116 ms
```

```
Execution Time: 48591.326 ms
```



```
ndistinct(province, community)
=
ndistinct(province) * ndistinct(community)
```

```
306 * 2860 = 875160
```



```
CREATE STATISTICS s (ndistinct)
  ON state_name, province_name, community_name
  FROM zip_codes;
```

```
ANALYZE zip_codes;
```

```
SELECT stxndistinct FROM pg_statistic_ext;
```

stxndistinct

```
{"3, 4": 308, "3, 5": 2858, "4, 5": 2858, "3, 4, 5": 2858}
```



```
EXPLAIN (ANALYZE, TIMING off)
```

```
SELECT count(*) FROM zip_codes GROUP BY province_name, community_name;
```

QUERY PLAN

```
HashAggregate (cost=102569.26..102597.84 rows=2858 width=36)
  (actual rows=3845 loops=1)
```

```
Group Key: state_name, province_name, community_name
```

```
-> Seq Scan on zip_codes (cost=0.00..69467.13 rows=3310213 width=28)
    (actual rows=3311056 loops=1)
```

```
Planning Time: 1.367 ms
```

```
Execution Time: 2343.846 ms
```




ndistinct

- the “old behavior” was defensive
 - unreliable estimates with multiple columns
 - HashAggregate can’t spill to disk (OOM)
 - rather than crash do Sort+GroupAggregate (slow)
- ndistinct coefficients
 - make multi-column ndistinct estimates more reliable
 - reduced danger of OOM
 - large tables + GROUP BY multiple columns



Future Improvements

- additional types of statistics
 - MCV lists, histograms, ...
- statistics on expressions
 - currently only simple column references
 - alternative to functional indexes
- improving join estimates
 - using MCV lists
 - special multi-table statistics (syntax already supports it)



Questions?

Tomas Vondra

tomas.vondra@2ndquadrant.com

tomas@pgaddict.com



@fuzzycz