

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
 - MCV lists
- Future improvements.



ZIP_CODES

```
CREATE TABLE zip_codes (
```

```
    postal_code      INT,  
    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-belgium.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
3386		38



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

```
-----+-----  
schemaname      | public  
tablename       | zip_codes  
attname         | place_name  
...            | ...  
most_common_vals | {Tournai, Bruxelles, Antwerpen, Charleroi, ...}  
most_common_freqs | {0.0059067, 0.005611, 0.004134, 0.0041347, ...}  
...            | ...
```



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

QUERY PLAN

```
-----
Seq Scan on zip_codes  (cost=0.00..80.33 rows=20 width=56)
    (actual rows=20 loops=1)
```

```
reltuples      | 3386
most_common_vals | {Tournai, ...}
most_common_freqs | {0.0059067, ...}
```

$$3386 * 0.0059067 = 20.0$$



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

QUERY PLAN

```
-----
Seq Scan on zip_codes  (cost=0.00..80.33 rows=131 width=56)
    (actual rows=131 loops=1)
```

```
reltuples      | 3386
most_common_vals | {..., Tournai, ...}
most_common_freqs | {..., 0.03868872, ...}
```

$$3386 * 0.03868872 = 131.0$$



Underestimate

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

QUERY PLAN

```
Seq Scan on zip_codes  (cost=0.00..88.79 rows=1 width=56)
    (actual rows=20 loops=1)
  Filter: (((place_name)::text = 'Tournai'::text)
    AND ((community_name)::text = 'Tournai'::text))
Rows Removed by Filter: 3366
```



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



```
SELECT * FROM zip_codes
    WHERE place_name = 'Tournai'
        AND community_name = 'Tournai';
```

```
P(place_name = 'Tournai' & county_name = 'Tournai')
= P(place_name = 'Tournai') * P(state_name = 'Tournai')
= 0.0059067 * 0.03868872
= 0.0002285
```

$3386 * 0.0002285 = 0.774$



Underestimate

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

QUERY PLAN

```
Seq Scan on zip_codes  (cost=0.00..88.79 rows=1 width=56)
    (actual rows=20 loops=1)
   Filter: (((place_name)::text = 'Tournai'::text)
            AND ((community_name)::text = 'Tournai'::text))
  Rows Removed by Filter: 3366
```



Overestimate

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

QUERY PLAN

```
Seq Scan on zip_codes  (cost=0.00..88.79 rows=19 width=56)
    (actual rows=0 loops=1)
  Filter: (((state_name)::text <> 'Tournai'::text)
    AND ((community_name)::text = 'Tournai'::text))
Rows Removed by Filter: 3386
```



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)
-> 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}
 - 7506 \rightarrow {Tournai, Wallonie, Hainaut, Tournai}
- other dependencies:
 - place \rightarrow community
 - community \rightarrow province
 - province \rightarrow state



CREATE STATISTICS

```
CREATE STATISTICS s (dependencies)
  ON place_name, state_name, community_name FROM zip_codes;
      2           3           5
ANALYZE zip_codes;
SELECT stxdependencies FROM pg_statistic_ext WHERE stxname = 's';
```

stxdependencies

```
-----
{"2 => 3": 1.000000, "2 => 5": 1.000000,
 "3 => 5": 0.018015, "5 => 2": 0.000886,
 "5 => 3": 0.981985, "2, 3 => 5": 1.000000,
 "2, 5 => 3": 1.000000, "3, 5 => 2": 0.000886}
```



place → community: 1.0 = d

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

$$3386 * 0.0059 * (1.0 + (1.0 - 1.0) * 0.0386) = 19.97$$



Underestimate : fixed

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

QUERY PLAN

```
Seq Scan on zip_codes  (cost=0.00..88.79 rows=20 width=56)
    (actual rows=20 loops=1)
   Filter: (((place_name)::text = 'Tournai'::text) AND
            ((community_name)::text = 'Tournai'::text))
  Rows Removed by Filter: 3366
```



Overestimate #1: not fixed :-)

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

QUERY PLAN

```
Seq Scan on zip_codes  (cost=0.00..88.79 rows=19 width=56)
    (actual rows=0 loops=1)
   Filter: (((community_name)::text <> 'Tournai'::text) AND
            ((place_name)::text = 'Tournai'::text))
  Rows Removed by Filter: 3386
```

Functional dependencies only work with equalities.



Overestimate #2: not fixed :-)

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Tournai'
        AND community_name = 'Bruxelles';
```

QUERY PLAN

```
Seq Scan on zip_codes  (cost=0.00..88.79 rows=20 width=56)
    (actual rows=0 loops=1)
   Filter: (((place_name)::text = 'Tournai'::text) AND
            ((community_name)::text = 'Bruxelles'::text))
  Rows Removed by Filter: 3386
```

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=45130.48..45130.95 rows=47 width=16)
    (actual rows=47 loops=1)
    Group Key: community_name
    -> Seq Scan on zip_codes  (cost=0.00..36462.32 rows=1733632 width=8)
        (actual rows=1733632 loops=1)
```

Planning Time: 0.111 ms

Execution Time: 382.878 ms

(5 rows)



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

attname		n_distinct
postal_code		1175
place_name		2847
state_name		4
province_name		12
community_name		47
latitude		496
longitude		596

(7 rows)



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

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

QUERY PLAN

```
-----  
GroupAggregate  (cost=287219.11..305893.52 rows=133809 width=32)
```

```
    (actual rows=2847 loops=1)
```

```
    Group Key: place_name, community_name
```

```
    -> Sort  (cost=287219.11..291553.19 rows=1733632 width=24)
```

```
        (actual rows=1733632 loops=1)
```

```
        Sort Key: place_name, community_name
```

```
        Sort Method: external merge  Disk: 59008kB
```

```
        -> Seq Scan on zip_codes  (cost=0.00..36462.32 rows=1733632 width=24)
```

```
            (actual rows=1733632 loops=1)
```

```
Planning Time: 0.334 ms
```

```
Execution Time: 2656.216 ms
```



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

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

QUERY PLAN

```
GroupAggregate (cost=287219.11..305893.52 rows=133809 width=32)
```

```
(actual rows=2847 loops=1)
```

```
Group Key: place_name, community_name
```

```
-> Sort (cost=287219.11..291553.19 rows=1733632 width=24)
```

```
(actual rows=1733632 loops=1)
```

```
Sort Key: place_name, community_name
```

```
Sort Method: external merge Disk: 59008kB
```

```
-> Seq Scan on zip_codes (cost=0.00..36462.32 rows=1733632 width=24)
```

```
(actual rows=1733632 loops=1)
```

```
Planning Time: 0.334 ms
```

```
Execution Time: 2656.216 ms
```



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

2847 * 47 = 133809
```



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

```
ANALYZE zip_codes;
```

```
SELECT stxndistinct FROM pg_statistic_ext;
```

stxndistinct

```
{"2, 4": 2847, "2, 5": 2847, "4, 5": 48, "2, 4, 5": 2847}
```



```
EXPLAIN (ANALYZE, TIMING off)
SELECT count(*) FROM zip_codes GROUP BY place_name, community_name;
```

QUERY PLAN

```
-----
HashAggregate  (cost=49464.56..49493.03 rows=2847 width=32)
    (actual rows=2847 loops=1)
    Group Key: place_name, community_name
    -> Seq Scan on zip_codes  (cost=0.00..36462.32 rows=1733632 width=24)
        (actual rows=1733632 loops=1)
```

Planning Time: 0.197 ms

Execution Time: 518.319 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