2ndQuadrant®
PostgreSQL

CREATE STATISTICS

What is it for?

Tomas Vondra, 2ndQuadrant
<tomas.vondra@2ndquadrant.com> / <tomas@pgaddict.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),
  latitude          REAL,
  longitude         REAL
);

cat create-table.sql | psql test
cat zip-codes-ukraine.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 = 'Львів';

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..45864.80 rows=3154 width=61)
                        (actual rows=2944 loops=1)
   Filter: ((place_name)::text = 'Львів'::text)
   Rows Removed by Filter: 1889600
 Planning Time: 0.125 ms
 Execution Time: 166.881 ms
(5 rows)
```

reltuples , relpages

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

  reltuples   | relpages
--------------+----------
 1.892544e+06 |    22208
```

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

------------------+---------------------------------
schemaname        | public
tablename         | zip_codes
attname           | place_name
...               | ...
most_common_vals  | {Київ, Харків, Дніпро, ..., Львів, ...}
most_common_freqs | {0.006367, 0.003067, 0.0029, ..., 0.001667, ...}
...               | ...
```

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

                              QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..45864.80 rows=3154 width=61)
                        (actual rows=2944 loops=1)
   Filter: ((place_name)::text = 'Львів'::text)
   Rows Removed by Filter: 1889600

reltuples          | 1.892544e+06
most_common_vals   | {..., Львів, ...}
most_common_freqs  | {..., 0.001667, ...}

                        0.001667 * 1.892544e+06 = 3154
```

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

                              QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..45864.80 rows=2870 width=61)
                        (actual rows=4288 loops=1)
   Filter: ((province_name)::text = 'Lvivska'::text)
   Rows Removed by Filter: 1888256

reltuples          | 1.892544e+06
most_common_vals   | {..., Lvivska, ...}
most_common_freqs  | {..., , 0.0015165, …}

                        1.892544e+06 * 0.0015165 = 2870
```

Underestimate

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Львів'
                          AND province_name = 'Lvivska';

                             QUERY PLAN
--------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..50596.16 rows=5 width=61)
                        (actual rows=2944 loops=1)
   Filter: (((place_name)::text = 'Львів'::text)
        AND ((province_name)::text = 'Lvivska'::text))
   Rows Removed by Filter: 1889600
```

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

```
SELECT * FROM zip_codes
        WHERE place_name = 'Львів'
          AND province_name = 'Lvivska';
```

```
P(place_name = 'Львів' & province_name = 'Lvivska')
  = P(place_name = 'Львів') * P(province_name = 'Lvivska')
  = 0.001667 * 0.0015165
  = 0.0000025280055
```

```
0.0000025280055 * 1.892544e+06 = 4.78
```

Underestimate

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Львів'
                          AND province_name = 'Lvivska';

                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..50596.16 rows=5 width=61)
                        (actual rows=2944 loops=1)
   Filter: (((place_name)::text = 'Львів'::text)
        AND ((province_name)::text = 'Lvivska'::text))
   Rows Removed by Filter: 1889600
```

Overestimate

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Київ'
                          AND province_name != 'Kyiv';

                              QUERY PLAN
-------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..50596.16 rows=11972 width=61)
                        (actual rows=0 loops=1)
   Filter: (((province_name)::text <> 'Kyiv'::text)
        AND ((place_name)::text = 'Київ'::text))
   Rows Removed by Filter: 1892544
```

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 → {place, state, province}
  - 4176 → {Київ, Kyiv, Kyiv}
- other dependencies:
  - place → state
  - state → province
  - place → province

CREATE STATISTICS

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

                    dependencies
-------------------------------------------------
{"2 => 3": 0.683733, "2 => 4": 0.658233,
 "3 => 2": 0.005067, "3 => 4": 0.005067,
 "4 => 2": 0.029000, "4 => 3": 0.949033,
 "2, 3 => 4": 0.862367, "2, 4 => 3": 0.999400, "3, 4 => 2": 0.029000}
```

```
place → province: 0.658233 = d
```

```
P(place = 'Львів' & province = 'Lvivska') =
P(place = 'Львів') * [d + (1-d) * P(province = 'Lvivska')]
```

```
1.892544e+06 * 0.001667 * (0.658233 + (1.0 - 0.658233) * 0.0015165)
= 2078.2
```

Underestimate : fixed

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Львів'
                          AND province_name = 'Lvivska';

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..50596.16 rows=2078 width=61)
                        (actual rows=2944 loops=1)
   Filter: (((place_name)::text = 'Львів'::text)
        AND ((province_name)::text = 'Lvivska'::text))
   Rows Removed by Filter: 1889600
```

Overestimate #1: not fixed :-(

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..50131.16 rows=9290 width=60)
                        (actual rows=0 loops=1)
   Filter: (((state_name)::text <> 'Kyiv'::text) AND
            ((province_name)::text = 'Kyiv'::text))
   Rows Removed by Filter: 1892544
```

Functional dependencies only work with equalities.

Overestimate #2: not fixed :-(

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE province_name = 'Kyiv'
                          AND state_name = 'Lvivska';

                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..50131.16 rows=10351 width=60)
                        (actual rows=0 loops=1)
   Filter: (((province_name)::text = 'Kyiv'::text)
        AND ((state_name)::text = 'Lvivska'::text))
   Rows Removed by Filter: 1892544
```

The queries need to respect the functional dependencies.

ndistinct (GROUP BY)

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

                                   QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=50596.16..50601.38 rows=522 width=20)
                (actual rows=524 loops=1)
   Group Key: province_name
   ->  Seq Scan on zip_codes  (cost=0.00..41133.44 rows=1892544 width=12)
                              (actual rows=1892544 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
----------------+------------
 state_name     |         25
 postal_code    |      25274
 place_name     |      16788
 latitude       |      12154
 longitude      |      13694
 province_name  |        522
(7 rows)
```

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

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=341951.09..362769.07 rows=189254 width=46)
                 (actual rows=27631 loops=1)
   Group Key: place_name, province_name
   ->  Sort  (cost=341951.09..346682.45 rows=1892544 width=38)
             (actual rows=1892544 loops=1)
         Sort Key: place_name, province_name
         Sort Method: external merge  Disk: 91360kB
         ->  Seq Scan on zip_codes  (cost=0.00..41133.44 rows=1892544 width=38)
                                    (actual rows=1892544 loops=1)
```

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

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=341951.09..362769.07 rows=189254 width=46)
                 (actual rows=27631 loops=1)
   Group Key: place_name, province_name
   ->  Sort  (cost=341951.09..346682.45 rows=1892544 width=38)
             (actual rows=1892544 loops=1)
         Sort Key: place_name, province_name
         Sort Method: external merge  Disk: 91360kB
         ->  Seq Scan on zip_codes  (cost=0.00..41133.44 rows=1892544 width=38)
                                    (actual rows=1892544 loops=1)
```

```
ndistinct(place, province)

=

ndistinct(place) * ndistinct(province)
```

```
16788 * 522 = 8763336
```

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

```
16788 * 522 = 8763336 => 189254
```

(capped to 10% of the table)

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

ANALYZE zip_codes;

SELECT n_distinct FROM pg_stats_ext WHERE statistics_name = 's'
```

```
                            n_distinct
-------------------------------------------------------------------
 {"2, 3": 22666, "2, 4": 26573, "3, 4": 530, "2, 3, 4": 26585}
```

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

```
                                    QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=54862.52..55128.25 rows=26573 width=46)
                (actual rows=27631 loops=1)
   Group Key: place_name, province_name
   ->  Seq Scan on zip_codes  (cost=0.00..40668.44 rows=1892544 width=38)
                              (actual rows=1892544 loops=1)
```

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

MCV lists (PG12)

```
--
CREATE STATISTICS s (mcv) ON state_name, province_name FROM zip_codes;

-- more detailed stats
SET default_statistics_target = 10000;

-- show the MCV list contents
SELECT * FROM pg_mcv_list_items((
  SELECT stxdmcv FROM pg_statistic_ext_data d
                 JOIN pg_statistic_ext s on (s.oid = d.stxoid)
   WHERE stxname = 's'));
```

Overestimate #1: fixed

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

                          QUERY PLAN
-----------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..50131.16 rows=1 width=60)
                        (actual rows=0 loops=1)
   Filter: (((state_name)::text <> 'Kyiv'::text) AND
            ((province_name)::text = 'Kyiv'::text))
   Rows Removed by Filter: 1892544
```

Overestimate #2

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE province_name = 'Kyiv'
                          AND state_name = 'Lvivska';

                          QUERY PLAN
-----------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..50131.16 rows=1 width=60)
                        (actual rows=0 loops=1)
   Filter: (((province_name)::text = 'Kyiv'::text) AND
            ((state_name)::text = 'Lvivska'::text))
   Rows Removed by Filter: 1892544
```

summary

Future Improvements

- additional types of statistics
  - MCV lists (PG12), 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)

2ndQuadrant PostgreSQL

Questions?

Tomas Vondra

tomas.vondra@2ndquadrant.com
tomas@pgaddict.com

@fuzzycz