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
  - MCV lists
- Future improvements.

ZIP_CODES

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

cat create-table.sql | psql test
cat zip-codes-poland.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 = 'Warszawa';

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..54030.80 rows=118770 width=66)
                        (actual rows=119712 loops=1)
   Filter: ((place_name)::text = 'Warszawa'::text)
   Rows Removed by Filter: 2070272
 Planning Time: 0.184 ms
 Execution Time: 175.427 ms
(5 rows)
```

reltuples , relpages

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

  reltuples   | relpages
--------------+----------
 2.189984e+06 |    26656
```

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

------------------+---------------------------------
schemaname       | public
tablename        | zip_codes
attname          | place_name
...              | ...
most_common_vals | {Warszawa, Łódź, Wrocław, ...}
most_common_freqs | {0.054233335, 0.0231, 0.0188, ...}
...              | ...
```

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..54030.80 rows=118770 width=66)
                        (actual rows=119712 loops=1)
   Filter: ((place_name)::text = 'Warszawa'::text)
   Rows Removed by Filter: 2070272

reltuples          | 2.189984e+06
most_common_vals   | {Warszawa, Łódź, Wrocław, ...}
most_common_freqs  | {0.054233335, 0.0231, 0.0188, ...}

                2.189984e+06 * 0.054233335 = 118770.135916640
```

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..54030.80 rows=118770 width=66)
                        (actual rows=119648 loops=1)
   Filter: ((community_name)::text = 'Warsaw'::text)
   Rows Removed by Filter: 2070336

reltuples          | 2.189984e+06
most_common_vals   | {Warsaw, Łódź, Wrocław, ...}
most_common_freqs  | {0.054233335, 0.023066666, 0.0188, ...}

                2.189984e+06 * 0.054233335 =  118770.135916640
```

Underestimate

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..59505.76 rows=6441 width=66)
                        (actual rows=119648 loops=1)
   Filter: (((place_name)::text = 'Warszawa'::text)
        AND ((community_name)::text = 'Warsaw'::text))
   Rows Removed by Filter: 2070336
```

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

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

```
P(place_name = 'Warszawa' & community_name = 'Warsaw')
  = P(place_name = 'Warszawa') * P(community_name = 'Warsaw')
  = 0.054233335 * 0.054233335
  = 0.00294125462522225
```

```
0.00294125462522225 * 2.189984e+06 = 6441.301
```

Underestimate

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..59505.76 rows=6441 width=66)
                        (actual rows=119648 loops=1)
   Filter: (((place_name)::text = 'Warszawa'::text)
        AND ((community_name)::text = 'Warsaw'::text))
   Rows Removed by Filter: 2070336
```

Overestimate

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..59505.76 rows=112329 width=66)
                        (actual rows=0 loops=1)
   Filter: (((place_name)::text <> 'Warszawa'::text)
        AND ((community_name)::text = 'Warsaw'::text))
   Rows Removed by Filter: 2189984
```

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, county, community}
  - 04-163 → {Warszawa, Mazovia, Warszawa, Warsaw}
- other dependencies:
  - zip code → place → community → county → state

CREATE STATISTICS

```
CREATE STATISTICS s (dependencies)
    ON place_name, community_name FROM zip_codes;
            2           5
ANALYZE zip_codes;

SELECT stxdependencies FROM pg_statistic_ext WHERE stxname = 's';

                dependencies
-----------------------------------------
 {"2 => 5": 0.643100, "5 => 2": 0.225900}
```

```
place → community: 0.643100 = d
```

```
P(place = 'Warszawa' & community = 'Warsaw') =
P(place = 'Warszawa') * [d + (1-d) * P(community = 'Warsaw')]
```

```
2.189984e+06 * 0.054233335 * (0.6431 + (1.0 - 0.6431) * 0.054233335)
                           = 78679.97458
```

Underestimate : fixed

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..59505.76 rows=78680 width=66)
                        (actual rows=119648 loops=1)
   Filter: (((place_name)::text = 'Warszawa'::text)
        AND ((community_name)::text = 'Warsaw'::text))
   Rows Removed by Filter: 2070336
```

Overestimate #1: not fixed :-(

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..59505.76 rows=112329 width=66)
                        (actual rows=0 loops=1)
   Filter: (((place_name)::text <> 'Warszawa'::text)
        AND ((community_name)::text = 'Warsaw'::text))
   Rows Removed by Filter: 2189984
```

Functional dependencies only work with equalities.

Overestimate #2: not fixed :-(

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

                                QUERY PLAN
-------------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..59505.76 rows=34661 width=66)
                        (actual rows=0 loops=1)
   Filter: (((place_name)::text = 'Łódź'::text)
        AND ((community_name)::text = 'Warsaw'::text))
   Rows Removed by Filter: 2189984
```

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=59505.76..59527.44 rows=2168 width=20)
                (actual rows=2262 loops=1)
   Group Key: community_name
   ->  Seq Scan on zip_codes  (cost=0.00..48555.84 rows=2189984 width=12)
                              (actual rows=2189984 loops=1)
 Planning Time: 0.094 ms
 Execution Time: 424.562 ms
(5 rows)
```

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

     attname      | n_distinct
------------------+------------
 community_name |       2168
 county_name    |        357
 latitude       |      29180
 longitude      |      38742
 place_name     |      21870
 postal_code    |      10519
 state_name     |         16
(7 rows)
```

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

                                   QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=59505.76..59509.33 rows=357 width=24)
                (actual rows=370 loops=1)
   Group Key: county_name
   ->  Seq Scan on zip_codes  (cost=0.00..48555.84 rows=2189984 width=16)
                              (actual rows=2189984 loops=1)
 Planning Time: 0.091 ms
 Execution Time: 424.645 ms
(5 rows)
```

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

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=383985.41..408075.23 rows=218998 width=36)
                 (actual rows=2333 loops=1)
   Group Key: community_name, county_name
   ->  Sort  (cost=383985.41..389460.37 rows=2189984 width=28)
             (actual rows=2189984 loops=1)
         Sort Key: community_name, county_name
         Sort Method: external merge  Disk: 82304kB
         ->  Seq Scan on zip_codes  (cost=0.00..48555.84 rows=2189984 width=28)
                                    (actual rows=2189984 loops=1)
 Planning Time: 2.040 ms
 Execution Time: 2590.548 ms
(8 rows)
```

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

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=383985.41..408075.23 rows=218998 width=36)
                 (actual rows=2333 loops=1)
   Group Key: community_name, county_name
   ->  Sort  (cost=383985.41..389460.37 rows=2189984 width=28)
             (actual rows=2189984 loops=1)
         Sort Key: community_name, county_name
         Sort Method: external merge  Disk: 82304kB
         ->  Seq Scan on zip_codes  (cost=0.00..48555.84 rows=2189984 width=28)
                                    (actual rows=2189984 loops=1)
 Planning Time: 2.040 ms
 Execution Time: 2590.548 ms
(8 rows)
```

```
ndistinct(community, place)

=

ndistinct(community) * ndistinct(place)
```

```
357 * 2168 = 934626
```

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

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=383985.41..408075.23 rows=218998 width=36)
                 (actual rows=2333 loops=1)
   Group Key: community_name, county_name
   ->  Sort  (cost=383985.41..389460.37 rows=2189984 width=28)
             (actual rows=2189984 loops=1)
         Sort Key: community_name, county_name
         Sort Method: external merge  Disk: 82304kB
         ->  Seq Scan on zip_codes  (cost=0.00..48555.84 rows=2189984 width=28)
                                    (actual rows=2189984 loops=1)
 Planning Time: 2.040 ms
 Execution Time: 2590.548 ms
(8 rows)
```

```
ndistinct(community, place)

=

ndistinct(community) * ndistinct(place)
```

```
357 * 2168 = 934626 (= 218998)
```

(capped to 10% of the table)

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

ANALYZE zip_codes;

SELECT stxndistinct FROM pg_statistic_ext WHERE stxname = 's';

                          n_distinct
---------------------------------------------------------------
 {"2, 4": 34528, "2, 5": 35094, "4, 5": 2217, "2, 4, 5": 35150}
```

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

                                  QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=64980.72..65002.89 rows=2217 width=36)
                (actual rows=2333 loops=1)
   Group Key: community_name, county_name
   ->  Seq Scan on zip_codes  (cost=0.00..48555.84 rows=2189984 width=28)
                              (actual rows=2189984 loops=1)
 Planning Time: 0.307 ms
 Execution Time: 539.230 ms
(5 rows)
```

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 (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