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       INT,
  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-gb.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 = 'Manchester';

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..42175.91 rows=14028 width=67)
                        (actual rows=13889 loops=1)
   Filter: ((place_name)::text = 'Manchester'::text)
   Rows Removed by Filter: 1683064
 Planning Time: 0.113 ms
 Execution Time: 151.340 ms
(5 rows)
```

# reltuples , relpages

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

  reltuples   | relpages
--------------+----------
 1.696953e+06 |    20964
```

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

------------------+---------------------------------
schemaname       | public
tablename        | zip_codes
attname          | place_name
...              | ...
most_common_vals | {... , Manchester, ...}
most_common_freqs | {..., 0.0082665813, ...}
...              | ...
```

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..42175.91 rows=14028 width=67)
                        (actual rows=13889 loops=1)
   Filter: ((place_name)::text = 'Manchester'::text)
   Rows Removed by Filter: 1683064

reltuples          | 1.696953e+06
most_common_vals   | {..., Manchester, ...}
most_common_freqs  | {..., 0.0082665813, ...}

                1.696953e+06 * 0.0082665813 = 14027.9999367789
```

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..42175.91 rows=13858 width=67)
                        (actual rows=13912 loops=1)
   Filter: ((community_name)::text = 'Manchester'::text)
   Rows Removed by Filter: 1683041

reltuples          | 1.696953e+06
most_common_vals   | {..., Manchester, ...}
most_common_freqs  | {..., 0.0081664017, ...}

                1.696953e+06 * 0.0081664017 =  13857.9998640201
```

# Underestimate

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..46418.29 rows=115 width=67)
                        (actual rows=11744 loops=1)
   Filter: (((place_name)::text = 'Manchester'::text) AND
((community_name)::text = 'Manchester'::text))
   Rows Removed by Filter: 1685209
```

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

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

```
P(place_name = 'Manchester' & community_name = 'Manchester')
  = P(place_name = 'Manchester') * P(community_name = 'Manchester')
  = 0.0082665813 * 0.0081664017
  = 0.00006750822358150821
```

```
0.00006750822358150821 * 1.696953e+06 = 114.558282531
```

# Underestimate

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..46418.29 rows=115 width=67)
                        (actual rows=11744 loops=1)
   Filter: (((place_name)::text = 'Manchester'::text) AND
((community_name)::text = 'Manchester'::text))
   Rows Removed by Filter: 1685209
```

# Overestimate

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

                                QUERY PLAN
-------------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..46418.29 rows=10896 width=67)
                        (actual rows=4 loops=1)
   Filter: (((place_name)::text <> 'London'::text) AND
            ((community_name)::text = 'Westminster'::text))
   Rows Removed by Filter: 1696949
```

# 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}
  - M11 0AT → {Manchester, England, Greater Manchester, Manchester District (B)}
- other dependencies:
  - zip code → place → state → county → community

# 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": 0.697633, "5 => 2": 0.095800}
```

```
place → community: 0.697633 = d
```

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

```
1.696953e+06 * 0.008266581 * (0.697633 + (1.0 - 0.697633) * 0.0081664017)
                                                    = 9281.03
```

# Underestimate : fixed

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

                                QUERY PLAN
-----------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..46418.29 rows=9307 width=67)
                        (actual rows=11744 loops=1)
   Filter: (((place_name)::text = 'Manchester'::text) AND
            ((community_name)::text = 'Manchester'::text))
   Rows Removed by Filter: 1685209
```

# Overestimate #1: not fixed :-(

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

                                QUERY PLAN
-------------------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..46418.29 rows=10896 width=67)
                        (actual rows=4 loops=1)
   Filter: (((place_name)::text <> 'London'::text) AND
            ((community_name)::text = 'Westminster'::text))
   Rows Removed by Filter: 1696949
```

**Functional dependencies only work with equalities.**

# Overestimate #2: not fixed :-(

```
EXPLAIN (ANALYZE, TIMING off)
SELECT * FROM zip_codes WHERE place_name = 'Manchester'
                          AND county_name = 'Westminster';

                           QUERY PLAN
-----------------------------------------------------------------
 Seq Scan on zip_codes  (cost=0.00..46418.29 rows=9305 width=67)
                        (actual rows=0 loops=1)
   Filter: (((place_name)::text = 'Manchester'::text) AND
            ((community_name)::text = 'Westminster'::text))
   Rows Removed by Filter: 1696953
```

**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=46418.29..46421.86 rows=358 width=29)
                (actual rows=359 loops=1)
   Group Key: community_name
   ->  Seq Scan on zip_codes  (cost=0.00..37933.53 rows=1696953 width=21)
                              (actual rows=1696953 loops=1)
 Planning Time: 0.087 ms
 Execution Time: 337.718 ms
(5 rows)
```

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

    attname     | n_distinct
----------------+------------
 community_name |        358
 county_name    |         91
 latitude       |      59925
 longitude      |      64559
 place_name     |      12281
 postal_code    |         -1
 state_name     |          3
(7 rows)
```

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

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=294728.63..313395.11 rows=169695 width=40)
                 (actual rows=15194 loops=1)
   Group Key: community_name, place_name
   ->  Sort  (cost=294728.63..298971.01 rows=1696953 width=32)
             (actual rows=1696953 loops=1)
         Sort Key: community_name, place_name
         Sort Method: external merge  Disk: 69648kB
         ->  Seq Scan on zip_codes  (cost=0.00..37933.53 rows=1696953 width=32)
                                    (actual rows=1696953 loops=1)
 Planning Time: 0.374 ms
 Execution Time: 1554.933 ms
```

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

                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=294728.63..313395.11 rows=169695 width=40)
                 (actual rows=15194 loops=1)
   Group Key: community_name, place_name
   ->  Sort  (cost=294728.63..298971.01 rows=1696953 width=32)
             (actual rows=1696953 loops=1)
         Sort Key: community_name, place_name
         Sort Method: external merge  Disk: 69648kB
         ->  Seq Scan on zip_codes  (cost=0.00..37933.53 rows=1696953 width=32)
                                    (actual rows=1696953 loops=1)
 Planning Time: 0.374 ms
 Execution Time: 1554.933 ms
```

```
ndistinct(community, place)

=

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

```
358 * 12281 = 4396598
```

```
ndistinct(community, place)
```

=

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

```
358 * 12281 = 4396598 (169695)
```

(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": 12996, "2, 5": 13221, "4, 5": 399, "2, 4, 5": 13252}
```

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

                                  QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=50660.68..50792.89 rows=13221 width=40)
                (actual rows=15194 loops=1)
   Group Key: community_name, place_name
   ->  Seq Scan on zip_codes  (cost=0.00..37933.53 rows=1696953 width=32)
                              (actual rows=1696953 loops=1)
 Planning Time: 0.056 ms
 Execution Time: 436.828 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**