Lies, damned lies, and statistics

A journey into the PostgreSQL statistics subsystem

Jan Urbański
j.urbanski@wulczer.org

New Relic

PGCon 2017, Ottawa, May 25

For those following at home

Getting the slides

https://wulczer.org/lies-damned-lies-and-statistics.pdf

Getting the source

https://github.com/wulczer/lies-damned-lies-and-statistics

1. Overview of the statistics subsystem
   - Why gather statistics?
   - What gets calculated
   - Accessing statistics
2. The internals
   - Populating statistics tables
   - Into the math
   - Configuration
3. Advanced features
   - Typanalyze functions
   - Multivariate statistics

Outline

1 Overview of the statistics subsystem
- Why gather statistics?
- What gets calculated
- Accessing statistics

2 The internals

3 Advanced features

Lifecycle of a SQL query

- parsing
    - transforming SQL text into an internal structure
    - determining types of all expressions
- planning
    - deciding how to execute the query
- execution
    - reading actual data from disk
    - formatting and returning the result

Lifecycle of a SQL query

- parsing
  - transforming SQL text into an internal structure
  - determining types of all expressions
- planning
  - deciding how to execute the query
- execution
  - reading actual data from disk
  - formatting and returning the result

The importance of cardinality estimation

- one of the basic questions the planner has to answer is how much rows will an expression return
- estimating the cardinality important for several reasons
  - choosing join type and join ordering (nested loop, hash join)
  - choosing table access method (sequential scan, index scan)
  - deciding whether to materialise or not

Selectivity estimation

- estimating cardinality boils down to two things
  - figuring out the total number of rows in a table
  - figuring out how many rows will be filtered out by the WHERE clause
- tracking the size of a table is relatively straightforward
- determining the selectivity of a WHERE clause is much more difficult

Selectivity estimation puzzlers

Estimation problem examples

```
SELECT * FROM places;

SELECT * FROM stores WHERE province = 'alberta';

SELECT * FROM places WHERE population > 200000;

SELECT * FROM stores WHERE
    province = 'alberta' AND
    zip = 'T0A';
```

Selectivity estimation puzzlers cont.

Estimation problem examples cont.

```sql
SELECT * FROM places JOIN stores USING (province);

SELECT * FROM places JOIN stores USING (province) WHERE
    profit > 10000;

SELECT province, count(*) FROM stores
    GROUP BY province;
```

Outline

1. Overview of the statistics subsystem
   - Why gather statistics?
   - What gets calculated
   - Accessing statistics
2. The internals
3. Advanced features

Overall table statistics

- number of rows in the table
  - useful when deciding which join method to use
  - processing each row has a cost, so a precise count is necessary
- number of disk pages used by the table
  - the real measure of how much will it cost to read data off disk
- the same stats are kept for every index

Per-column stats

- fraction of values that are `NULL`
- average value width in bytes
  - includes TOASTed value width if applicable
- number of distinct values in the column
  - if positive it's the actual number of distinct values
  - if negative it's the number of distinct values as a fraction of all non-null values

Per-column stats cont.

- an array of most common values
- an array of frequencies of the most common values
  - same length as the values array
- histogram bounds for the spectrum of values in the column
  - only present if the column type can be ordered
  - excludes most common values
- correlation between physical row ordering and logical ordering
  - used to determine how much random IO will an index scan require
  - only present if the column type can be ordered

Special cases

- some special data types maintain their own statistic data
  - statistic calculation is pluggable
  - the storage format is flexible enough to work for different data types
  - more on that later
- foreign tables can provide their own statistic calculation implementations
- the same statistics are gathered for expression indexes
  - regular indexes don't need their own per-column statistics

Outline

1. Overview of the statistics subsystem
   - Why gather statistics?
   - What gets calculated
   - Accessing statistics
2. The internals
3. Advanced features

pg_class

- keeps table-wide statistics
  - `relpages` is the number of disk pages used by the table
  - `reltuples` is the approximate number of rows in the table
- row count information is approximate, but can still useful for monitoring
- also has an entry for every index

pg_statistics

- keeps per-column statistics
- information that's not datatype-dependent is stored directly
  - `stanullfrac` is the `NULL` fraction
  - `stawidth` is the average width
  - `stadistinct` is the number of distinct values
- the remainder are five loosely typed "slots"
- each slot is formed out of four fields
  - code identifying what kind data the slot contains
  - optional operator OID
  - `anyarray` for column values
  - `real` array for statistical data

pg_stats

- a more user-friendly view built on top of `pg_statistics`
- table names instead of OIDs
- slightly better column names (`null_frac` vs `stanullfrac`)
- "slots" codes decoded to their meanings
- readable for everyone and limited to columns accessible to the user

Estimation problem examples

```sql
SELECT * FROM places;

SELECT * FROM stores WHERE province = 'alberta';

SELECT * FROM places WHERE population > 200000;

SELECT * FROM stores WHERE
    province = 'alberta' AND
    zip = 'T0A';
```

Estimation problem examples cont.

```
SELECT * FROM places JOIN stores USING (province);

SELECT * FROM places JOIN stores USING (province) WHERE
    profit > 10000;

SELECT province, count(*) FROM stores
    GROUP BY province;
```

Outline

1 Overview of the statistics subsystem

2 The internals
- Populating statistics tables
- Into the math
- Configuration

3 Advanced features

ANALYZE and VACUUM

- ANALYZE is the primary command that updates statistics
  - a plain ANALYZE updates statistics for all tables in the cluster
  - it just loops over all tables in pg_class so we'll focus on a single-table ANALYZE
  - it's also possible to analyze a specific set of columns only
- both VACUUM and ANALYZE update the `pg_class` per-table statistics
- things like CLUSTER and CREATE INDEX also update `pg_class`

ANALYZE internals

- switch to the table owner's user
  - index functions will get evaluated
  - at least three CVEs out of that one...
- find out the ordering operators and custom type analysis functions for the column
- determine the number of rows that need to be fetched
- fetch sample rows from the table
  - the number of rows is the maximum over all columns
- calculate statistics
- update `pg_statistics`

Automated statistics collection

- the autovacuum worker will run ANALYZE if needed
  - decision taken based on the number of rows changed since last ANALYZE
  - trigger expressed as a fraction of the total number of rows, with an additional minimum floor
  - the `pg_class` data comes in handy here...
- after bulk loading a new table, it will initially have no statistic information
  - remember to run ANALYZE as part of the bulk load
- no autovacuum = no stats

Statistics collector vs planner statistics

- confusingly, there is a Postgres process called the stats collector
- it handles runtime stats which are **different** from the planner stats
- examples of runtime stats are:
  - number of times a table has been accessed
  - number of rows read from an index
  - number of buffer hits for data in a table
- runtime stats live in `pg_stat_*` tables

Outline

1. Overview of the statistics subsystem
2. The internals
   - Populating statistics tables
   - Into the math
   - Configuration
3. Advanced features

Determining row sample size

- the number of rows to acquire from the table is based on target histogram size
- histograms are stored as sets of values that divide the column data into equal-sized bins
- relative bin size error is the relative size difference between the histogram bin and a perfectly equal-sized bin
- error probability is the probability that the relative bin size error will be greater than our target error

Histogram bin size error

Histogram bin size error example

$$
\begin{aligned}
\text{assuming values } V &= \{1, 2, 3, \ldots, 1000\} \\
\text{and histogram size } k &= 4 \\
\text{equal-sized bins } B_{equal} &= \{1, 250, 500, 750, 1000\} \\
\text{actual bins } B_{actual} &= \{1, 300, 550, 780, 1000\} \\
\text{max relative bin size error } f &= \frac{|300 - 250|}{250} = 0.2
\end{aligned}
$$

Histogram minimum sample size

Bounding max relative bin size error

Given a table with n rows, the required sample size for a histogram size k, maximum relative bin size f and error probability γ is

$$r = \frac{4k \ln \frac{2n}{\gamma}}{f^2}$$

Postgres assumes $f = 0.5$, $n = 10^6$ and $\gamma = 0.01$, which yields

$$r = 305.82k$$

Histogram minimum sample size

Bounding max relative bin size error

Given a table with n rows, the required sample size for a histogram size k, maximum relative bin size f and error probability γ is

$$r = \frac{4k \ln \frac{2n}{\gamma}}{f^2}$$

Postgres assumes $f = 0.5$, $n = 10^6$ and $\gamma = 0.01$, which yields

$$r = 305.82k$$

Histogram minimum sample size cont.

- for histogram size k you need to fetch 300 times as much rows
- that factor depends on the size of the table, but logarithmically
  - even if the table is much larger, a factor of 300 should still be enough
- the default histogram size is 100

Sampling table blocks

- once we know how many rows we need, that many table blocks are fetched
- table blocks are sampled using Knuth's algorithm S

Sampling a table with N blocks to obtain n blocks

let K be the number of remaining blocks
let k be n minus current sample size
skip current block it with a probability of $1 - \frac{k}{K}$
otherwise put it in the sample

Sampling rows

- as table blocks are fetched, rows contained in them are being sampled
- can't use algorithm S because the total number of rows is not known
- Vitter's algorithm Z is used, which is a variation of reservoir sampling that requires less random numbers to be generated

Naive reservoir sampling to obtain n rows

put the first n rows in the sample
for each next row number i choose it with probability $\frac{n}{i}$
if block is chosen, have it replace a random one from the sample

Calculating statistics

- most statistics are straightforward to compute
  - fraction of non-null values
  - histogram
  - most common values and their frequencies
  - correlation
- the number of distinct values is a bit more involved

Estimating number of distinct values

- if all values in the sample are different, assume the column is unique
- if all values appear more than once, assume the column contains only these values
- otherwise, use the Haas and Stokes estimator

Number of distinct values

$$\frac{nd}{n - f_1 + \frac{f_1 n}{N}}$$

Where n is the sample size, N is the total number of values, f_1 is the number of distinct values that appeared exactly once and d is the total number of distinct values.

Outline

1 Overview of the statistics subsystem

2 The internals
- Populating statistics tables
- Into the math
- Configuration

3 Advanced features

Configuring the statistics subsystem

- `default_statistics_target` is the histogram width
  - configurable on a per-column basis
- `autovacuum_analyze_threshold` is the minimum number of row updates before autovacuum runs ANALYZE
- `autovacuum_analyze_scale_factor` is the fraction of table size to change before autovacuum runs ANALYZE
  - configurable on a per-table basis

Outline

1 Overview of the statistics subsystem

2 The internals

3 Advanced features
- Typanalyze functions
- Multivariate statistics

Array type statistics

- certain data types have their own statistics routines
- a typical example are arrays
  - histograms and most common elements don't make sense for arrays
  - array columns are more often constrained with set operators and ANY/ALL than with equality
- the type analysis function calculates most common elements (as opposed to most common values)
- additionally, a distinct element counts histogram is calculated

Tsvector statistics

- similar to array statistics
- tsvectors only store unique occurrences of lexems, so no distinct elements count histogram
- also accounts for the fact that the frequency of words in a natural language text is not linear

Range statistics

- stores three additional histograms
- the first one is a length histogram of all the non-empty ranges
  - the format is similar to a values histogram for scalar types
  - the "measurements" slot contains the fraction of empty ranges
- the second one is a bounds histogram, which are actually two histograms
  - uses ranges instead of integers for histogram values
  - the lower bounds form a histogram of lower bounds for all the ranges and the upper bounds form a histogram of upper bounds

Outline

1 Overview of the statistics subsystem

2 The internals

3 Advanced features
- Typanalyze functions
- Multivariate statistics

Coming in 10.0

Correlated columns example

Overestimating selectivity

```
SELECT * FROM
    places JOIN stores USING (province)
WHERE
    zip = 'YOA' AND
    province = 'yukon';
```

Correlated columns

- a functional dependency between columns is when a value in one determines the value in the other
  - for example, a store in zip code YOA will always be in Yukon
- if the planner is not aware of them, it will overestimate the selectivity of a two-clause WHERE constraint

Correlated columns cont.

- functional dependencies are rarely "hard" because of bad data, special cases and so on
- a functional dependency statistic will store how "strong" the dependency is
- functional dependencies know nothing about individual values
  - only works if `WHERE` are consistent with the dependency
  - `zip = 'L5M' AND province = 'yukon'` will give wrong estimates

n-distinct statistics

- a similar problem to `WHERE` selectivity for correlated columns is estimating distinct values
- if the planner is not aware, if will overestimate the number of rows a `GROUP BY` will produce
- an ndistinct statistic will store distinct counts for the user-defined column groupings

Multivariate n-distinct example

Overestimating row counts

```
SELECT provice, zip, count(*) FROM stores
GROUP BY province, zip;
```

Creating extended statistics

Creating extended statistics

```
CREATE STATISTICS zip_province_correlation(dependencies)
ON zip, province FROM stores;

CREATE STATISTICS zip_province_distinct(ndistinct)
ON zip, province FROM stores;
```

Questions?