Lecture 5: Statistics for Cost Estimation

Statistical Information for Cost Estimation

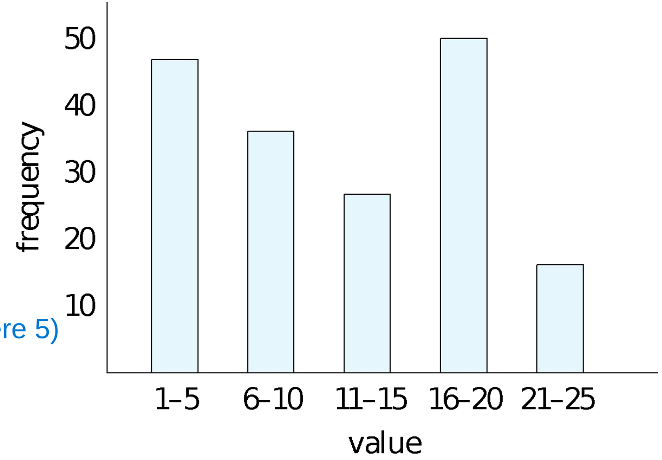
- n_r: number of tuples in a relation *r*.
- b_r: number of blocks containing tuples of *r*.
- l_r: size of a tuple of *r*.
- f_r: blocking factor of *r* — i.e., the number of tuples of *r* that fit into one block.
- *V(A, r):* number of distinct values that appear in *r* for attribute *A;* same as the size of $\prod_A(r)$.
- If tuples of *r* are stored together physically in a file, then:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

Histograms

- Histogram on attribute *age* of relation *person*

width -> assemble by the same intervals (here 5)



- **Equi-width** histograms
- **Equi-depth** histograms break up range such that each range has (approximately) the same number of tuples
  - E.g. (4, 8, 14, 19)
- Many databases also store *n* **most-frequent values** and their counts
  - Histogram is built on remaining values only

Histograms (cont.)

- Histograms and other statistics usually computed based on a **random sample**
- Statistics may be out of date
  - Some database require a **analyze (vacuum)** command to be executed to update statistics
  - Others automatically recompute statistics
    - e.g., when number of tuples in a relation changes by some percentage

Postgres Statistics

Table 51.89. pg_stats Columns

Column Type Description
schemaname name (references pg_namespace.nspname) Name of schema containing table
tablename name (references pg_class.relname) Name of table
attname name (references pg_attribute.attname) Name of the column described by this row
inherited bool If true, this row includes inheritance child columns, not just the values in the specified table
null_frac float4 Fraction of column entries that are null
avg_width int4 Average width in bytes of column's entries
n_distinct float4 If greater than zero, the estimated number of distinct values in the column. If less than zero, the negative of the number of distinct values divided by the number of rows. (The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows; the positive form is used when the column seems to have a fixed number of possible values.) For example, -1 indicates a unique column in which the number of distinct values is the same as the number of rows.

`most_common_vals anyarray`
: A list of the most common values in the column. (Null if no values seem to be more common than any others.)

`most_common_freqs float4[]`
: A list of the frequencies of the most common values, i.e., number of occurrences of each divided by total number of rows. (Null when `most_common_vals` is.)

`histogram_bounds anyarray`
: A list of values that divide the column's values into groups of approximately equal population. The values in `most_common_vals`, if present, are omitted from this histogram calculation. (This column is null if the column data type does not have a < operator or if the `most_common_vals` list accounts for the entire population.)

`correlation float4`
: Statistical correlation between physical row ordering and logical ordering of the column values. This ranges from -1 to +1. When the value is near -1 or +1, an index scan on the column will be estimated to be cheaper than when it is near zero, due to reduction of random access to the disk. (This column is null if the column data type does not have a < operator.)

`most_common_elems anyarray`
: A list of non-null element values most often appearing within values of the column. (Null for scalar types.)

`most_common_elem_freqs float4[]`
: A list of the frequencies of the most common element values, i.e., the fraction of rows containing at least one instance of the given value. Two or three additional values follow the per-element frequencies; these are the minimum and maximum of the preceding per-element frequencies, and optionally the frequency of null elements. (Null when `most_common_elems` is.)

`elem_count_histogram float4[]`
: A histogram of the counts of distinct non-null element values within the values of the column, followed by the average number of distinct non-null elements. (Null for scalar types.)

Selection Size Estimation

$\sigma_{A=v}(r)$

- n_r / $V(A,r)$: number of records that will satisfy the selection
- Equality condition on a key attribute: *size estimate* = 1

$\sigma_{A \le V}(r)$ (case of $\sigma_{A \ge V}(r)$ is symmetric)

- Let c denote the estimated number of tuples satisfying the condition.
- If min(A,r) and max(A,r) are available in catalog
  - c = 0 if v < min(A,r)
  - c = $n_r \cdot \dfrac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$
  - If histograms available, can refine above estimate
  - In absence of statistical information *c* is assumed to be n_r / 2.

Size Estimation of Complex Selections

- The **selectivity** of a condition θ_i is the probability that a tuple in the relation *r* satisfies θ_i .
  - If s_i is the number of satisfying tuples in *r*, the selectivity of θ_i is given by s_i / n_r.
- **Conjunction:** $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n}(r)$. *Assuming independence,* estimate of tuples in the result is: $n_r * \frac{s_1 * s_2 * \ldots * s_n}{n_r^n}$
- **Disjunction:** $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n}(r)$. Estimated number of tuples:

$$n_r * \left(1 - (1 - \frac{s_1}{n_r}) * (1 - \frac{s_2}{n_r}) * \ldots * (1 - \frac{s_n}{n_r})\right)$$

- **Negation:** $\sigma_{\neg\theta}(r)$. Estimated number of tuples: $n_r - size(\sigma_\theta(r))$

Join Operation: Running Example

Running example: *student* ⋈ *takes*

Catalog information for join examples:

- $n_{student}$ = 5,000. $f_{student}$ = *50,* which implies that $b_{student}$ =5000/50 = 100.
- n_{takes} = 10000. f_{takes} = 25, which implies that b_{takes} = 10000/25 = 400.
- *V(ID, takes)* = 2500, which implies that on average, each student who has taken a course has taken 4 courses.
  - Attribute *ID* in *takes* is a foreign key referencing *student.*
  - *V*(*ID, student*) = 5000 (*primary key!*)

```
create table student
   (ID              varchar(5),
    name            varchar(20) not null,
    dept_name       varchar(20),
    tot_cred        numeric(3,0) check (tot_cred >= 0),
    primary key (ID),
    foreign key (dept_name) references department (dept_name)
          on delete set null
   );
```

```
create table takes
   (ID          varchar(5),
    course_id   varchar(8),
    sec_id      varchar(8),
    semester    varchar(6),
    year        numeric(4,0),
    grade       varchar(2),
    primary key (ID, course_id, sec_id, semester, year),
    foreign key (course_id, sec_id, semester, year) references section
(course_id, sec_id, semester, year)
          on delete cascade,
    foreign key (ID) references student (ID)
          on delete cascade
   );
```

Estimation of the Size of Joins

- The Cartesian product $r \times s$ contains $n_r . n_s$ tuples; each tuple occupies $s_r + s_s$ bytes.
- If $R \cap S = \varnothing$, then $r \bowtie s$ is the same as $n_r \times n_s$.
- If $R \cap S$ is a key for R, then a tuple of s will join with at most one tuple from r
  - therefore, the number of tuples in $r \bowtie s$ is no greater than the number of tuples in s.
- If $R \cap S$ *in* S is a foreign key in S referencing R, then the number of tuples in $r \bowtie s$ is exactly the same as the number of tuples in s.
    - The case for $R \cap S$ being a foreign key referencing S is symmetric.
- In the example query *student* $\bowtie$ *takes, ID* in *takes* is a foreign key referencing *student*
  - hence, the result has exactly n_{takes} tuples, which is 10000

Estimation of the Size of Joins (Cont.)

- If $R \cap S = \{A\}$ is not a key for R or S.
  If we assume that every tuple t in R produces tuples in $R \bowtie S$, the number of tuples in $R \bowtie S$ is estimated to be:

  $$\frac{n_r * n_s}{V(A,s)}$$

  If the reverse is true, the estimate obtained will be:

  $$\frac{n_r * n_s}{V(A,r)}$$

  The lower of these two estimates is probably the more accurate one.
- Can improve on above if histograms are available
  - Use formula similar to above, for each cell of histograms on the two relations

Estimation of the Size of Joins (Cont.)

- Compute the size estimates for *student* ⋈ *takes* without using information about foreign keys:
  - *V(ID, takes)* = 2500, and
    V(ID, student) = 5000
  - The two estimates are 5000 * 10000/2500 = 20,000 and 5000 * 10000/5000 = 10000
  - We choose the lower estimate, which in this case, is the same as our earlier computation using foreign keys.

The Internals of PostgreSQL

Chapter 3

Query Processing

https://www.interdb.jp/pg/pgsql03.html

Postgres optimizer code snippets

Postgres genetic query optimizer

https://www.postgresql.org/docs/13/geqo-intro.html

https://doxygen.postgresql.org/geqo_8h_source.html

var=const selectivity

https://doxygen.postgresql.org/selfuncs_8h.html#a31ee9824c23028c56ca3d6ca92c39a7e

Range typanalyze

https://doxygen.postgresql.org/rangetypes__typanalyze_8c_source.html

Range overlap

https://github.com/postgres/postgres/blob/cd3f429d9565b2e5caf0980ea7c707e37bc3b317/src/include/catalog/pg_operator.dat#L3110

rangesel

https://doxygen.postgresql.org/rangetypes__selfuncs_8c.html#a632d39f45c72d18cf792fb33014155ee

Selectivity Estimation of Inequality Joins In Databases

Diogo Repas, Zhicheng Luo, Maxime Schoemans, Mahmoud Sakr

https://arxiv.org/abs/2206.07396

Credits

Many slides in this lecture are taken from:

- Avi Silberschatz, Henry F. Korth, S. Sudarshan. Database System Concepts

Recommended reading

- The Internals of PostgreSQL (https://www.interdb.jp/pg/)