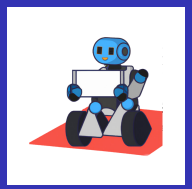


pg_index_stats

manage extended statistics automatically

Lepikhov A., Rybakina A.

Postgres Professional

2024

PostgresPro

Self Introduction

- Core Developer in Postgres Professional since 2017.
- Contributing to the PostgreSQL project since 2017: *Self-Join Elimination, GROUP-BY optimisation, OR <-> ANY transformation*.
- Ph.D. in Computer Sciences (Distributed Databases), MSU, 2008.
- Designed extensions: AQO, sr_plan, pg_index_stats ...

PostgresPro

Extended Statistics

CREATE STATISTICS ON x1,x2,x3,x4 FROM tablename;

- **MCV** - **M**ost **C**ommon **V**alues on composite value of (x1,x2,x3).
- **ndistinct** - number of distinct values on all combinations of columns: (x1,x2),(x1,x3),(x1,x2,x3)...
- **dependencies** - functional dependencies between combinations of columns: x1->x2, (x1,x2)->x3,...

Vondra. T. CREATE STATISTICS improvements, PGConf.DE 2022

PostgresPro

Laboriousness of Extended Statistics

- **MCV** - two arrays: values[] and frequences[].
- **ndistinct** - 1 integer for each of $2^n - (n + 1)$ combinations
- **dependencies** - 1 float value for each of combinations

columns:	2	3	4	...	8
distinct combinations:	1	4	11	...	247
dependency combinations:	2	9	28	...	1016

PostgresPro

Quick start

```
CREATE EXTENSION 'pg_index_stats';
CREATE TABLE test (x int,y int,z text);
CREATE INDEX ON test (x,y);
CREATE INDEX ON test (z,y);
\dX
```

```
                          List of extended statistics
 Schema |     Name      |   Definition    | Ndistinct | Dependencies |   MCV
--------+---------------+-----------------+-----------+--------------+---------
 public | test_x_y_stat | x, y FROM test | defined   | defined      | defined
 public | test_z_y_stat | y, z FROM test | defined   | defined      | defined
```

PostgresPro

Quick start - II

```
DROP INDEX test_x_y_idx;
```

```
                          List of extended statistics
 Schema |      Name       |    Definition    | Ndistinct | Dependencies |   MCV
--------+-----------------+------------------+-----------+--------------+---------
 public | test_z_y_stat | y, z FROM test | defined   | defined      | defined
```

```
SELECT stxname,obj_description(oid, 'pg_statistic_ext')
FROM (SELECT oid,stxname FROM pg_statistic_ext);
```

```
    stxname     |              obj_description
----------------+-------------------------------------------
 test_z_y_stat | pg_index_stats - multivariate statistics
```

GUCs & Funcs

- *mode* - disabled | all | univariate | multivariate
- *columns_limit*
- *pg_index_stats_build(relname, mode)*
- *pg_index_stats_rebuild()*
- *pg_index_stats_remove()*

PostgresPro

Short Tech Dive

Two-step process:

- Object Access Hook - gather candidate OIDs
- Utility Hook - create extended statistics after a successful utility statement

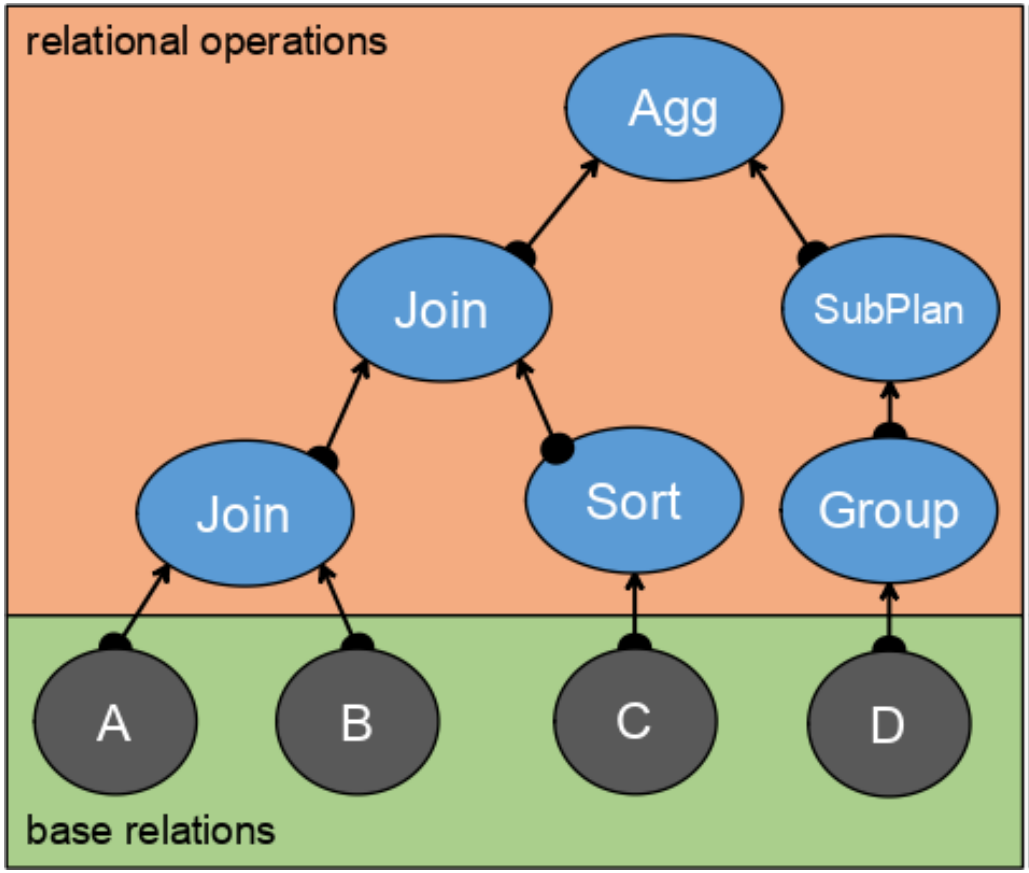
Set dependency of auto-generated statistics on the extension and index relation.
Add specific description for auto-generated statistics

PostgresPro

Query auto-generation frameworks

PostgresPro

What is the reason?



Cardinality estimation:

- Adaptive Query Optimizer (*aqo*)
- Query replanning (*replan*)
- Plan freezing (*sr_plan*)
- Selectivity by index (*Joinsel*)

Cardinality estimation:

- statistics ???

PostgresPro

Redundant expressions

Query	Planned rows	Actual rows
SELECT * **FROM** power_plants **WHERE** country = 'RUS';	544	544
SELECT * **FROM** power_plants **WHERE** country = 'RUS' **AND** country_long = 'Russia';	8	544

*Global Power Plant Database

PostgresPro

The question

Why can database systems, which manage all the data, not analyse it and find at least in-table dependencies and interconnections?

PostgresPro

Why indexes?

Table ”Parcels”:
Indexes:
”parcel_pkey” **PRIMARY KEY**, btree (id)
”parcel_parcel_id” btree (parcel_id)
”parcel_id_prik” btree (parcel_id, prik)
”parcel_id_sn_pol” btree (parcel_id, sn_pol)
”parcel_par_begin” btree (parcel_id, prik_begin_period)
”parcel_par_patient” btree (parcel_id, patient_id)
”parcel_par_recid” btree (parcel_id, recid)
”parcel_per_prik” btree (period, prik)

PostgresPro

Extended v/s Plain Statistics

SELECT * FROM power_plants WHERE ...

Query	Plain stat	Extended stat	Actual rows
country = 'RUS';	544	545	544
AND primary_fuel **IN** ('Solar')	166	57	57
primary_fuel **IN** ('Solar', 'Biomass')	189	60	60
primary_fuel **IN** ('Solar', 'Biomass', 'Coal')	225	156	156
AND source = 'Wiki–Solar'	24	46	40
AND longitude > 40. **AND** longitude < 70.	1	1	33

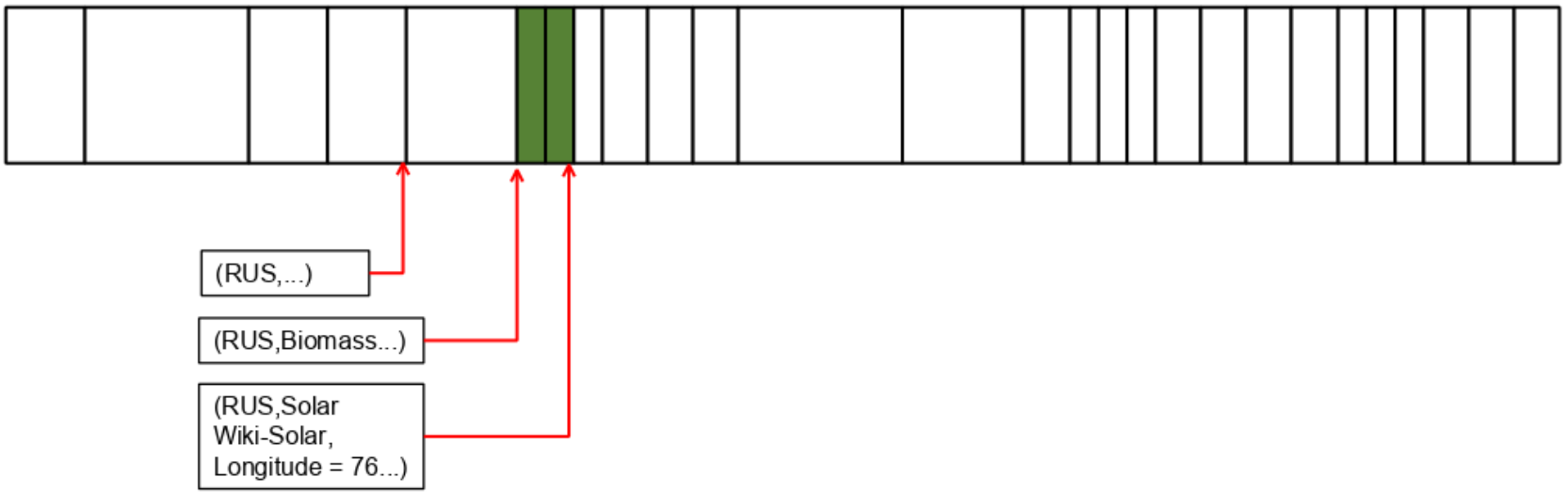
*Global Power Plant Database

PostgresPro

Does multicolumn histogram make sense?

In general - no. We need a multidimensional histogram. But it can make sense if we follow the definition of indexes ...

```
SELECT * FROM power_plants
WHERE
  country = 'RUS' AND
  primary_fuel IN ('Solar', 'Biomass', 'Coal') AND
  AND source = 'Wiki–Solar' AND
  longitude > 40. AND longitude < 70.;
```



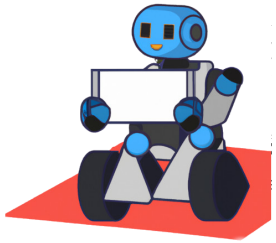
Multicolumn histogram advantage

SELECT * FROM power_plants WHERE ...

Query	Plain stat	Extended stat	Histogram	Actual rows
country = 'RUS';	544	545	544	544
AND primary_fuel **IN** ('Solar')	166	57	57	57
primary_fuel **IN** ('Solar', 'Biomass')	189	60	60	60
primary_fuel **IN** ('Solar', 'Biomass', 'Coal')	225	156	156	156
AND source = 'Wiki—Solar'	24	46	42	40
AND longitude > 40. **AND** longitude < 70.	1	1	35	33

*Global Power Plant Database

PostgresPro



Questions ?

The pg_index_stats extension
github link

Postgres Professional LLC
The Russian Postgres Company

PostgresPro