Запредельная селективность

Проблемы оценки селективности

Старков Максим, разработчик “ТанторЛабс”

Обо мне

- Большой опыт эксплуатации различных СУБД
- SRE в части эксплуатации крупных ИТ систем
- Эксперт по технологическим вопросам платформы 1С
- Разработчик ТанторЛабс

https://github.com/maxstarkov

В этом докладе:

1

Вы узнаете:

- Не все проблемы производительности запросов можно понять через EXPLAIN
- Как выполняется стоимостная оценка некоторых операторов плана запроса
- О проблемах оценки селективности

2

Вы научитесь:

- Пониманию некоторых тонкостей работы планировщика запросов
- Собирать компоненты оценки стоимости оператора плана запроса с помощью отладки

3

И еще:

- Сделаем доработку планировщика запросов
- Поговорим о расширенной статистике

Проблемный запрос

```
SELECT
  wide_table.field1 + T2.field1 AS field1,
  wide_table.field2 + T2.field2 AS field2,
  wide_table.field3 + T2.field3 AS field3,
  wide_table.field4 + T2.field4 AS field4
FROM wide_table AS wide_table
INNER JOIN pg_temp.temp_table AS T2
ON
  T2.field5 = wide_table.field5 AND T2.field6 = wide_table.field6
  AND T2.field7 = wide_table.field7 AND T2.field8 = wide_table.field8
  AND T2.field9 = wide_table.field9 AND T2.field10 = wide_table.field10
  AND T2.field11 = wide_table.field11 AND T2.field12 = wide_table.field12
  AND T2.field13 = wide_table.field13
WHERE
  wide_table.field13 = CAST(0 AS NUMERIC);
```

Готовим таблицу для воспроизведения проблемы

```
drop table if exists wide_table;

-- Create a wide table.
create table wide_table (
    c1 int not null, c2 int not null,
    c3 int not null, c4 int not null,
    c5 int not null, c6 int not null,
    c7 int not null, c8 int not null,
    c9 int not null, c10 int not null,
    c11 char(1000) not null,
    c12 char(1000) not null
);
```

Заполняем таблицу для воспроизведения проблемы

```
-- Fill the table with synthetic data.
insert into wide_table
select
    i % 3 as c1,
    (i + 1) % 101010 as c2,
    (i + 2) % 10101 as c3,
    (i + 3) % 5 as c4,
    (i + 4) % 13 as c5,
    (i + 5) % 17 as c6,
    (i + 6) % 19 as c7,
    (i) as c8,
    (i + 10) as c9,
    (i + 20) as c10,
    'FOO' as c11,
    'BAR' as c12
from generate_series(1,500000) as i;
```

Добавим индексы на таблице

```
-- Create wide unique index on table.
create unique index idx_wide on wide_table (
    c1, c2, c3, c4, c5, c6, c7, c8, c11, c12
);

-- Create narrow non-unique index on table.
create index idx_narrow on wide_table (
    c1, c2, c3
);

-- Analyze table.
analyze wide_table;
```

Создаем временную таблицу

```
/*
 Create a temporary table
 to store some of the data
 from the wide_table.
*/
create temp table tt1 (
    c1 int not null,
    c2 int not null,
    c3 int not null,
    c4 int not null,
    c5 int not null,
    c6 int not null,
    c7 int not null,
    c8 int not null,
    c9 int not null,
    c10 int not null,
    c11 char(1000) not null,
    c12 char(1000) not null
);
```

Заполняем временную таблицу случайной выборкой

```
/*
 Fill temporary table
 with a random sample
 from the wide_table.
*/
insert into tt1
select
    c1, c2, c3, c4, c5, c6,
    c7, c8, c9, c10, c11, c12
from wide_table
order by random()
limit 1000;

-- Analyze temporary table.
analyze tt1;
```

Запрос для воспроизведения проблемы

```
explain (analyze, buffers)
select
    t1.c1, t1.c2, t1.c3, t1.c4, t1.c5, t1.c6, t1.c7,
    t1.c8, t1.c9 + t2.c9, t1.c10 + t2.c10, t1.c11, t1.c12
from wide_table as t1
inner join tt1 as t2
on
    t1.c1 = t2.c1 and t1.c2 = t2.c2
    and t1.c3 = t2.c3 and t1.c4 = t2.c4
    and t1.c5 = t2.c5 and t1.c6 = t2.c6
    and t1.c7 = t2.c7 and t1.c8 = t2.c8
    and t1.c11 = t2.c11 and t1.c12 = t2.c12
where
    t1.c11 = 'FOO';
```

План проблемного запроса

Узкий индекс

```
 Nested Loop  (cost=0.42..7100.01 rows=1 width=1069)
              (actual time=0.056..22.690 rows=1000 loops=1)
   Buffers: shared hit=5971, local hit=143
   ->  Seq Scan on tt1 t2  (cost=0.00..155.50 rows=1000 width=1069)
                           (actual time=0.009..1.348 rows=1000 loops=1)
         Filter: (c11 = 'FOO'::bpchar)
        Buffers: local hit=143
   ->  Index Scan using idx_narrow on wide_table t1  (cost=0.42..6.93 rows=1 width=1069)
                                                     (actual time=0.020..0.020 rows=1 loops=1000)
          Index Cond: ((c1 = t2.c1) AND (c2 = t2.c2) AND (c3 = t2.c3))
          Filter: ((c11 = 'FOO'::bpchar) AND (t2.c4 = c4) AND (t2.c5 = c5) AND (t2.c6 = c6)
                 AND (t2.c7 = c7) AND (t2.c8 = c8) AND (t2.c12 = c12))
         Rows Removed by Filter: 2
        Buffers: shared hit=5937
 Planning:
   Buffers: shared hit=125
 Planning Time: 0.626 ms
 Execution Time: 22.806 ms
(14 rows)
```

Отключаем узкий индекс idx_narrow

```
-- Disable narrow index idx_narrow.
update pg_index set indisvalid = false
where indexrelid::regclass = 'idx_narrow'::regclass;

explain (analyze, buffers)
. . .

-- Enable narrow index idx_narrow.
update pg_index set indisvalid = true
where indexrelid::regclass = 'idx_narrow'::regclass;
```

План проблемного запроса без индекса idx_narrow

```
Nested Loop  (cost=0.42..8326.01 rows=1 width=1069)
             (actual time=0.039..16.076 rows=1000 loops=1)
   Buffers: shared hit=4000, local hit=143
   ->  Seq Scan on tt1 t2  (cost=0.00..155.50 rows=1000 width=1069)
                           (actual time=0.007..1.349 rows=1000 loops=1)
         Filter: (c11 = 'FOO'::bpchar)
         Buffers: local hit=143
   ->  Index Scan using idx_wide on wide_table t1  (cost=0.42..8.17 rows=1 width=1069)
                                                   (actual time=0.013..0.013 rows=1 loops=1000)
          Index Cond: ((c1 = t2.c1) AND (c2 = t2.c2) AND (c3 = t2.c3) AND (c4 = t2.c4) AND (c5 = t2.c5)
                      AND (c6 = t2.c6) AND (c7 = t2.c7) AND (c8 = t2.c8) AND (c11 = 'FOO'::bpchar) AND (c12 = t2.c12))
         Buffers: shared hit=4000
 Planning:
   Buffers: shared hit=8
 Planning Time: 0.368 ms
 Execution Time: 16.182 ms
(12 rows)
```

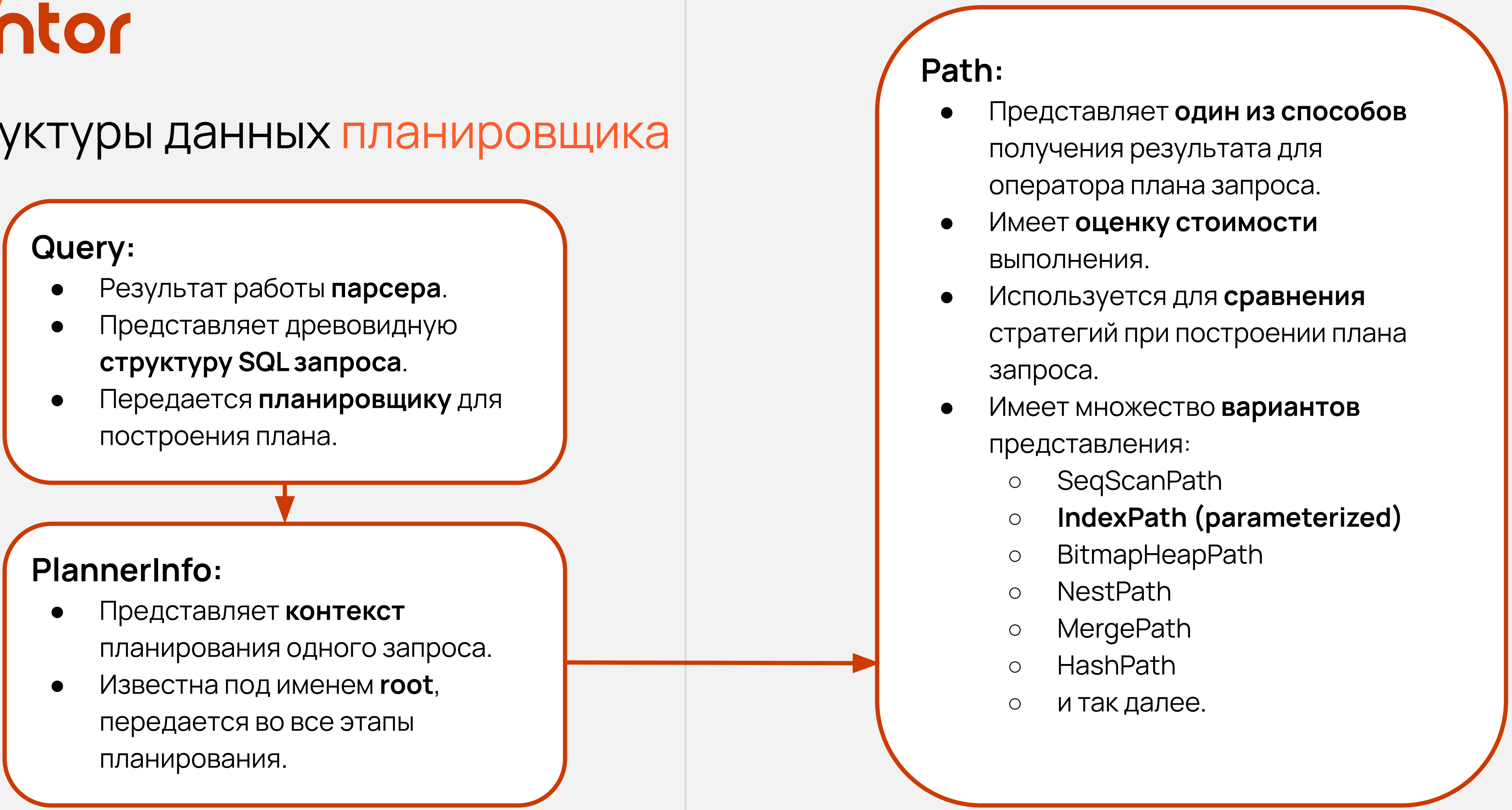
Широкий индекс

Декорреляция оценки стоимости и времени выполнения

```
-- Cost and actual time with narrow index idx_narrow.
Index Scan using idx_narrow on wide_table t1
  (cost=0.42..6.93 rows=1 width=1069)
  (actual time=0.020..0.020 rows=1 loops=1000)
    Index Cond: ((c1 = t2.c1) AND (c2 = t2.c2) AND (c3 = t2.c3))
    Filter: ((c11 = 'FOO'::bpchar) AND (t2.c4 = c4) AND (t2.c5 = c5)
      AND (t2.c6 = c6) AND (t2.c7 = c7) AND (t2.c8 = c8) AND (t2.c12 = c12))

-- Cost and actual time with wide index idx_wide.
Index Scan using idx_wide on wide_table t1
  (cost=0.42..8.17 rows=1 width=1069)
  (actual time=0.013..0.013 rows=1 loops=1000)
     Index Cond: ((c1 = t2.c1) AND (c2 = t2.c2) AND (c3 = t2.c3)
       AND (c4 = t2.c4) AND (c5 = t2.c5) AND (c6 = t2.c6)
       AND (c7 = t2.c7) AND (c8 = t2.c8) AND (c11 = 'FOO'::bpchar) AND (c12 = t2.c12))
```

Структуры данных планировщика



Query:

- Результат работы **парсера**.
- Представляет древовидную **структуру SQL запроса**.
- Передается **планировщику** для построения плана.

PlannerInfo:

- Представляет **контекст** планирования одного запроса.
- Известна под именем **root**, передается во все этапы планирования.

Path:

- Представляет **один из способов** получения результата для оператора плана запроса.
- Имеет **оценку стоимости** выполнения.
- Используется для **сравнения** стратегий при построении плана запроса.
- Имеет множество **вариантов** представления:
  - SeqScanPath
  - **IndexPath (parameterized)**
  - BitmapHeapPath
  - NestPath
  - MergePath
  - HashPath
  - и так далее.

Планировщик работает по принципу bottom up

3. Планируются другие операции:

- Группировка.
- Сортировка.
- Агрегации.
- Ограничение выборки.

2. Планируются соединения (JoinPath):

- Используются **Пути** для таблиц с **предыдущего этапа**.
- Выбираются **способы выполнения** соединений (NestedLoop, HashJoin, MergeJoin).

1. Планируются базовые отношения (RelOptInfo):

- Строятся Пути для **получения строк** из таблиц (SeqScanPath, IndexPath и так далее).
- Каждая таблица получает набор Путей с **оценкой стоимости**.

Точки останова для отладки оценки стоимости IndexPaths

```
/* Given an index and a set of index clauses for it, construct IndexPaths. */
get_index_paths() /* src/backend/optimizer/path/indxpath.c */
{
        add_path();
}

/* Determines and returns the cost of scanning a relation using an index */
cost_index() /* src/backend/optimizer/path/costsize.c */
{
        amcostestimate();
}

/* Index cost estimation functions for standard btree index */
btcostestimate() /* src/backend/utils/adt/selfuncs.c */
{
        genericcostestimate();
}
```

Компоненты оценки стоимости Index Scan

Index leaf pages cost:

оценка стоимости извлечения листовых страниц индекса и применения предикатов (IO и CPU costs).

Index traversal (descent) cost:

оценка стоимости прохода по btree в глубину (планировщик считает, что это CPU cost).

Heap pages cost:

оценка стоимости извлечения страниц таблицы индекса и применения фильтра, если часть предикатов не содержит атрибуты ключа индекса (IO и CPU costs).

Компоненты оценки стоимости Index Scan

```
Index leaf pages cost:
  IO cost:

  index_pages_fetched(
    CEIL(
      MIN(
        MAX(1, indexSelectivity * index->rel->tuples),
        index->tuples
      )
      * index->pages / index->tuples)
    * num_scans)
  * random_page_cost
```

effective_cache_size

numIndexTuples

numIndexPages

Компоненты оценки стоимости Index Scan

```
Index leaf pages cost:
  CPU cost:

  numIndexTuples *
  (
    (
      list_length(indexQuals) + list_length(indexOrderBys)
    ) * cpu_operator_cost
    + cpu_index_tuple_cost
  )
```

Компоненты оценки стоимости Index Scan

```
Index traversal (descent) cost:
  CPU cost:

  (CEIL
      (ln(index->tuples) / log(2.0))
      ) * cpu_operator_cost
  )
  +
  (
      (index->tree_height + 1)
      * DEFAULT_PAGE_CPU_MULTIPLIER
      * cpu_operator_cost
  )
```

50.0

Компоненты оценки стоимости Index Scan

Heap pages cost:

Max IO cost:

effective_cache_size

```
index_pages_fetched(
  MIN(
      MAX(1, indexSelectivity * baserel->tuples),
      MAXIMUM_ROWCOUNT
  )
  * loop_count)
* random_page_cost / loop_count
```

1e100

tuples_fetched

pages_fetched

Компоненты оценки стоимости Index Scan

```
Heap pages cost:
  Min IO cost:
```

effective_cache_size

```
index_pages_fetched(
  CEIL(indexSelectivity * baserel->pages)
   * loop_count)
* random_page_cost / loop_count
```

pages_fetched

Компоненты оценки стоимости Index Scan

```
Heap pages cost:
  IO cost:
```

indexCorrelation определена на этапе
Index leaf page cost

```
  (indexCorrelation * indexCorrelation) * min_IO_cost
  +
  (1 - indexCorrelation * indexCorrelation) * max_IO_cost
```

Компоненты оценки стоимости Index Scan

Heap pages cost:

CPU cost:

cpu_operator_cost

```
qpqual_cost = cost_qual_eval(qpquals)

(cpu_tuple_cost + qpqual_cost.per_tuple) * tuples_fetched
    + path->path.pathtarget->cost.per_tuple * path->path.rows
```

Значения параметров конфигурации

```
random_page_cost = 4
cpu_operator_cost = 0.0025
cpu_index_tuple_cost = 0.005
cpu_tuple_cost = 0.01
```

Компоненты оценки стоимости Index Scan для idx_narrow

```
Index leaf pages cost:
     IO cost = 2.4960
     CPU cost = 0.0125
     Index total cost = 2.5085

              Index traversal (descent) cost:
                    CPU cost = 0.4225
                    Index total cost = 2.9310

                          Heap pages cost:
                                IO cost = 3.9760
                                CPU cost = 0.0275
                                Index total cost = 6.9345
```

Компоненты оценки стоимости Index Scan для idx_wide

```
Index leaf pages cost:
    IO cost = 3.7320
    CPU cost = 0.0300
    Index total cost = 3.7620

              Index traversal (descent) cost:
                    CPU cost = 0.4225
                    Index total cost = 4.1845

                          Heap pages cost:
                                IO cost = 3.9760
                                CPU cost = 0.0100
                                Index total cost = 8.1705
```

Сравнение оценок стоимостей Index Scan

idx_narrow

```
Index leaf pages cost:
    IO cost = 2.4960
    CPU cost = 0.0125
    Index total cost = 2.5085

Index traversal (descent) cost:
    CPU cost = 0.4225
    Index total cost = 2.9310

Heap pages cost:
    IO cost = 3.9760
    CPU cost = 0.0275
    Index total cost = 6.9345
```

idx_wide

```
Index leaf pages cost:
    IO cost = 3.7320
    CPU cost = 0.0300
    Index total cost = 3.7620

Index traversal (descent) cost:
    CPU cost = 0.4225
    Index total cost = 4.1845

Heap pages cost:
    IO cost = 3.9760
    CPU cost = 0.0100
    Index total cost = 8.1705
```

delta

```
Index leaf pages cost:
    Delta IO cost = 1.2360
    Delta CPU cost = 0.0175
    Delta Index total cost = 1.2535

Index traversal (descent) cost:
    Delta CPU cost = 0.0000
    Delta Index total cost = 1.2535

Heap pages cost:
    Delta IO cost = 0.0000
    Delta CPU cost = -0.0175
    Index total cost = 1.2360
```

Сравнение оценок стоимостей Index Scan

```
Index leaf pages cost:
     Delta IO cost = +1.2360
     Delta CPU cost = +0.0175
     Delta Index total cost = +1.2535

              Index traversal (descent) cost:
                    Delta CPU cost = 0.0000
                    Delta Index total cost = +1.2535

                          Heap pages cost:
                                Delta IO cost = 0.0000
                                Delta CPU cost = -0.0175
                                Index total cost = +1.2360
```

Оценка селективности

clauselist_selectivity()

idx_narrow:

- Index key: **3 attributes**
- Join clauses: **3**
- Filter clauses: **7**
- estimate selectivity:
  3.2856322580743958e-10

idx_wide:

- Index key: **10 attributes**
- Join clauses: **10**
- Filter clauses: **0**
- estimate selectivity:
  3.1298687199803531e-20

wide_table:

- tuples: **500 000**

Оценка селективности clauselist_selectivity()

$$Sel = \frac{1}{Attr1_{n\_distinct}} \times \frac{1}{Attr2_{n\_distinct}} \times \frac{1}{Attr3_{n\_distinct}} \times \cdots \times \frac{1}{AttrN_{n\_distinct}}$$

```
SELECT
    *
FROM R JOIN S
ON
R.Attr1 = S.Attr1
AND R.Attr2 = S.Attr2
AND R.Attr3 = S.Attr3
AND … R.AttrN = S.AttrN
```

Оценка количества различных запросом

```
/* Real n_distinct for the index
idx_narrow key attributes */
select
  count(distinct c1) c1_n_distinct,
  count(distinct c2) c2_n_distinct,
  count(distinct c3) c3_n_distinct
from wide_table;
```

```
/* Real n_distinct for the index
idx_narrow key */
select
  count(1) idx_narrow_key_n_distinct
from (
  select distinct c1, c2, c3
from wide_table);
```

Functional dependency

```
 c1_n_distinct | c2_n_distinct | c3_n_distinct
---------------+---------------+---------------
             3 |        101010 |         10101
(1 row)
```

=

```
 idx_narrow_key_n_distinct
------------------------------
                    101010
(1 row)
```

Компоненты оценки стоимости Index Scan для idx_narrow **FIX**

```
Index leaf pages cost:
    IO cost = 2.4960
    CPU cost = 0.0625 (+0.05)
    Index total cost = 2.5585 (+0.05)

              Index traversal (descent) cost:
                   CPU cost = 0.4225
                   Index total cost = 2.9810 (+0.05)

                        Heap pages cost:
                             IO cost = 18.4408 (+14.4648) ←
                             CPU cost = 0.1375 (+0.1100)
                             Index total cost = 21.5093 (+14.5748)
```

Расширенная статистика Dependencies и Ndistinct

```
/* Functional dependency ext stats */
create statistics stx_dep(dependencies)
on c1,c2,c3
from wide_table;

analyze wide_table;

select dependencies from pg_stats_ext;
```

```
/* Multivariate ndistinct ext stats */
create statistics stx_ndist(ndistinct)
on c1,c2,c3
from wide_table;

analyze wide_table;

select n_distinct from pg_stats_ext;
```

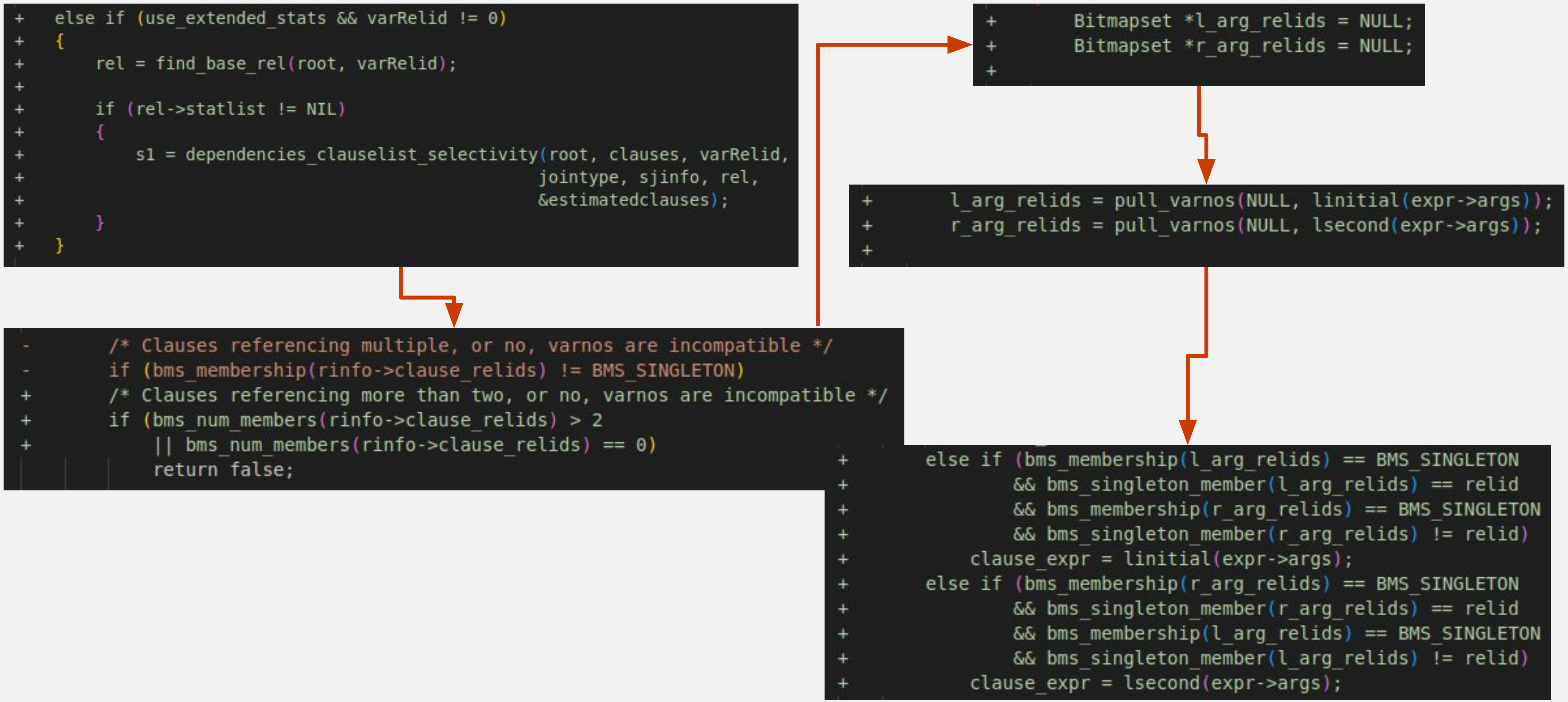
```
                dependencies
------------------------------------------
 {"2 => 1": 1.000000, "2 => 3": 1.000000,
  "3 => 1": 1.000000, "3 => 2": 0.061267,
  "1, 2 => 3": 1.000000, "1, 3 => 2": 0.061267,
"2, 3 => 1": 1.000000}
```

```
                n_distinct
------------------------------------------
 {"1, 2": 100517, "1, 3": 10093,
  "2, 3": 100517, "1, 2, 3": 100517}
```

Расширенная статистика Dependencies и Ndistinct

Не работает при оценке селективности параметризованных Index Scan

Расширенная статистика Dependencies и Ndistinct **PATCH**



```
+   else if (use_extended_stats && varRelid != 0)
+   {
+       rel = find_base_rel(root, varRelid);
+
+       if (rel->statlist != NIL)
+       {
+           s1 = dependencies_clauselist_selectivity(root, clauses, varRelid,
+                                                    jointype, sjinfo, rel,
+                                                    &estimatedclauses);
+       }
+   }
```

```
-       /* Clauses referencing multiple, or no, varnos are incompatible */
-       if (bms_membership(rinfo->clause_relids) != BMS_SINGLETON)
+       /* Clauses referencing more than two, or no, varnos are incompatible */
+       if (bms_num_members(rinfo->clause_relids) > 2
+           || bms_num_members(rinfo->clause_relids) == 0)
            return false;
```

```
+           Bitmapset *l_arg_relids = NULL;
+           Bitmapset *r_arg_relids = NULL;
+
```

```
+           l_arg_relids = pull_varnos(NULL, linitial(expr->args));
+           r_arg_relids = pull_varnos(NULL, lsecond(expr->args));
+
```

```
+       else if (bms_membership(l_arg_relids) == BMS_SINGLETON
+                && bms_singleton_member(l_arg_relids) == relid
+                && bms_membership(r_arg_relids) == BMS_SINGLETON
+                && bms_singleton_member(r_arg_relids) != relid)
+           clause_expr = linitial(expr->args);
+       else if (bms_membership(r_arg_relids) == BMS_SINGLETON
+                && bms_singleton_member(r_arg_relids) == relid
+                && bms_membership(l_arg_relids) == BMS_SINGLETON
+                && bms_singleton_member(l_arg_relids) != relid)
+           clause_expr = lsecond(expr->args);
```

План проблемного запроса FIX

Широкий индекс

```
Nested Loop  (cost=0.42..8326.01 rows=1 width=1069)
             (actual time=0.036..13.520 rows=1000 loops=1)
   Buffers: shared hit=4000, local hit=143
   ->  Seq Scan on tt1 t2  (cost=0.00..155.50 rows=1000 width=1069)
                           (actual time=0.010..1.380 rows=1000 loops=1)
         Filter: (c11 = 'FOO'::bpchar)
         Buffers: local hit=143
   ->  Index Scan using idx_wide on wide_table t1  (cost=0.42..8.17 rows=1 width=1069)
                                                   (actual time=0.011..0.011 rows=1 loops=1000)
          Index Cond: ((c1 = t2.c1) AND (c2 = t2.c2) AND (c3 = t2.c3) AND (c4 = t2.c4) AND (c5 = t2.c5)
                     AND (c6 = t2.c6) AND (c7 = t2.c7) AND (c8 = t2.c8) AND (c11 = 'FOO'::bpchar) AND (c12 = t2.c12))
         Buffers: shared hit=4000
 Planning Time: 0.422 ms
 Execution Time: 13.631 ms
(10 rows)
```

План проблемного запроса без индекса idx_wide

```
Nested Loop  (cost=0.42..21674.77 rows=1 width=1069)
             (actual time=0.040..19.064 rows=1000 loops=1)
   Buffers: shared hit=8015, local hit=143
   ->  Seq Scan on tt1 t2  (cost=0.00..155.50 rows=1000 width=1069)
                           (actual time=0.011..1.356 rows=1000 loops=1)
        Filter: (c11 = 'FOO'::bpchar)
        Buffers: local hit=143
   ->  Index Scan using idx_narrow on wide_table t1  (cost=0.42..21.51 rows=1 width=1069)
                                                     (actual time=0.012..0.017 rows=1 loops=1000)
         Index Cond: ((c1 = t2.c1) AND (c2 = t2.c2) AND (c3 = t2.c3))
        Filter: ((c11 = 'FOO'::bpchar) AND (t2.c4 = c4) AND (t2.c5 = c5) AND (t2.c6 = c6)
                AND (t2.c7 = c7) AND (t2.c8 = c8) AND (t2.c12 = c12))
        Rows Removed by Filter: 4
        Buffers: shared hit=8015
 Planning Time: 0.388 ms
 Execution Time: 19.173 ms
(12 rows)
```

Узкий индекс

~~Декорреляция~~ оценки стоимости и времени выполнения

```
-- Cost and actual time with narrow index idx_narrow.
Index Scan using idx_narrow on wide_table t1
  (cost=0.42..21.51 rows=1 width=1069)
  (actual time=0.012..0.017 rows=1 loops=1000)
   Index Cond: ((c1 = t2.c1) AND (c2 = t2.c2) AND (c3 = t2.c3))
   Filter: ((c11 = 'FOO'::bpchar) AND (t2.c4 = c4) AND (t2.c5 = c5)
     AND (t2.c6 = c6) AND (t2.c7 = c7) AND (t2.c8 = c8) AND (t2.c12 = c12))

-- Cost and actual time with wide index idx_wide.
Index Scan using idx_wide on wide_table t1
  (cost=0.42..8.17 rows=1 width=1069)
  (actual time=0.013..0.013 rows=1 loops=1000)
    Index Cond: ((c1 = t2.c1) AND (c2 = t2.c2) AND (c3 = t2.c3)
      AND (c4 = t2.c4) AND (c5 = t2.c5) AND (c6 = t2.c6)
      AND (c7 = t2.c7) AND (c8 = t2.c8) AND (c11 = 'FOO'::bpchar) AND (c12 = t2.c12))
```

Итоги

EXPLAIN

Есть запросы, проблемы которых невозможно понять только из вывода EXPLAIN. Требуется воспроизведение проблемы и отладка на уровне исходного кода Postgres.

Selectivity

Оценка селективности сложный процесс, который может дать запредельный результат далекий от реальной статистики отношения.

Extension statistics

Расширенная статистика - мощный инструмент повышения точности выполнения стоимостной оценки.

Develop

Требуются доработки планировщика для развития применения расширенной статистики в более сложных случаях.

tantor

Спасибо!

www.tantorlabs.ru