



Хакинг оптимизатора запросов, СНОВА... И СНОВА

Часть 1. Препроцессинг исходного дерева запросов

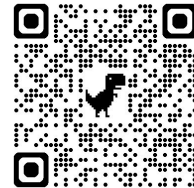
Максим Милютин
Руководитель R&D
Тантор Лабс

Соловьев Сергей
Разработчик
Тантор Лабс

Докладчики

Максим

*



- 2+ года в R&D (ex-Huawei)
- 5+ лет системной разработки ядер СУБД (PostgreSQL, Greenplum, GaussDB)
- 8+ лет работы с базами и PostgreSQL
- веду открытые курсы по реализации СУБД для студентов профиля “Системное программирование” мехмата НГУ *

Сергей

- базами данных интересуется с вуза
- в Танторе занимается системной разработкой PostgreSQL и интеграцией с ОС
- в свободное время пишет статьи на Хабре
- есть статья на *dev.to* по отладке и разработке оптимизатора **

**



Стадии исполнения запроса



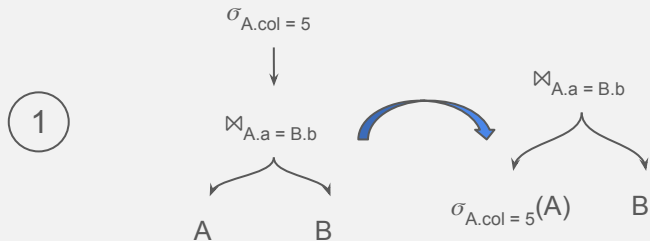
Стадии исполнения запроса

- один из наиболее **сложных** компонентов СУБД
- конкурентное преимущество одних движков перед другими
- разработчики соревнуются между собой по части продвинутости в реализации

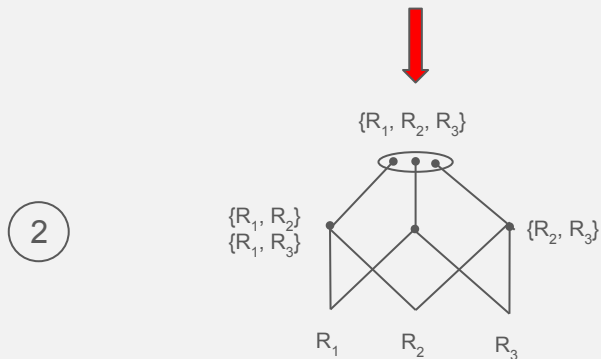


Query structure

Фазы планирования запросов



Initial query rewrites
Query normalization



Cost-based
join enumeration

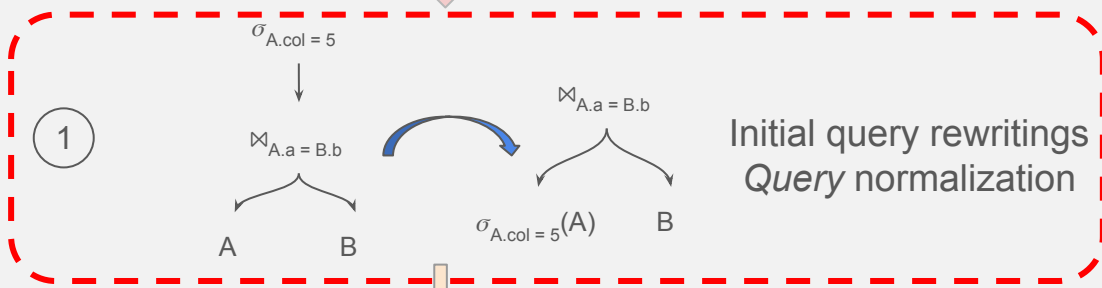


Query **Plan**



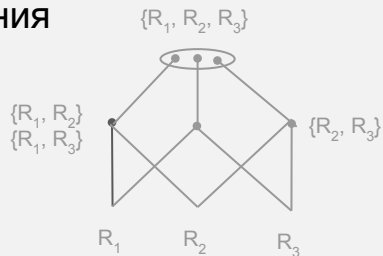
Query structure

Фазы планирования запросов



Тема сегодняшнего выступления

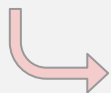
2



Cost-based
join enumeration

4

Post-join (group by, distinct, aggregate, etc.) planning



Query *Plan*



5



Plan Management
(plan caching, hints, execution
feedback, etc.)

не фаза, но отдельная
компонента, влияющая
на выбор итогового
плана запроса

Query structure

Фазы планирования запросов

- 1
- [PGCon 2011, Hacking the Query Planner](#) - Tom Lane
 - [PGCon 2020, Hacking the Query Planner. Again](#) 📄 - Richard Guo - [slides](#)
 - [PGCon 2019, Learning to Hack on Postgres Planner](#) 📄 - Melanie Plageman
 - [PostgreSQL planner development and debugging](#) - Sergey Solovev, 2025

3

[Статистика в ретроспективе](#) 📄 - Егор Поров

[PGConf.EU 2024, A Deep Dive into Postgres Statistics](#) 📄 - Louise Grandjonc
доклады на текущем PGBootcamp

- 2
- [CMU Database Group, PostgreSQL Optimizer Methodology](#) 📄 - Robert Haas
 - [Познакомимся с GEQO за 20 минут](#) 📄 - Павел Толмачев

- 4
- [PGCon 2018, What's in a Plan](#) 📄 - Robert Haas - [slides](#)

5

Plan Management
(plan caching, hints, execution
feedback, etc.)

Query *Plan*

Общие доклады по внутренке оптимизатора

Hacking the Query Planner

Tom Lane
Red Hat
PGCon 2011

Hacking the Query Planner, Again

Richard Guo / VMware
PGCon 2020

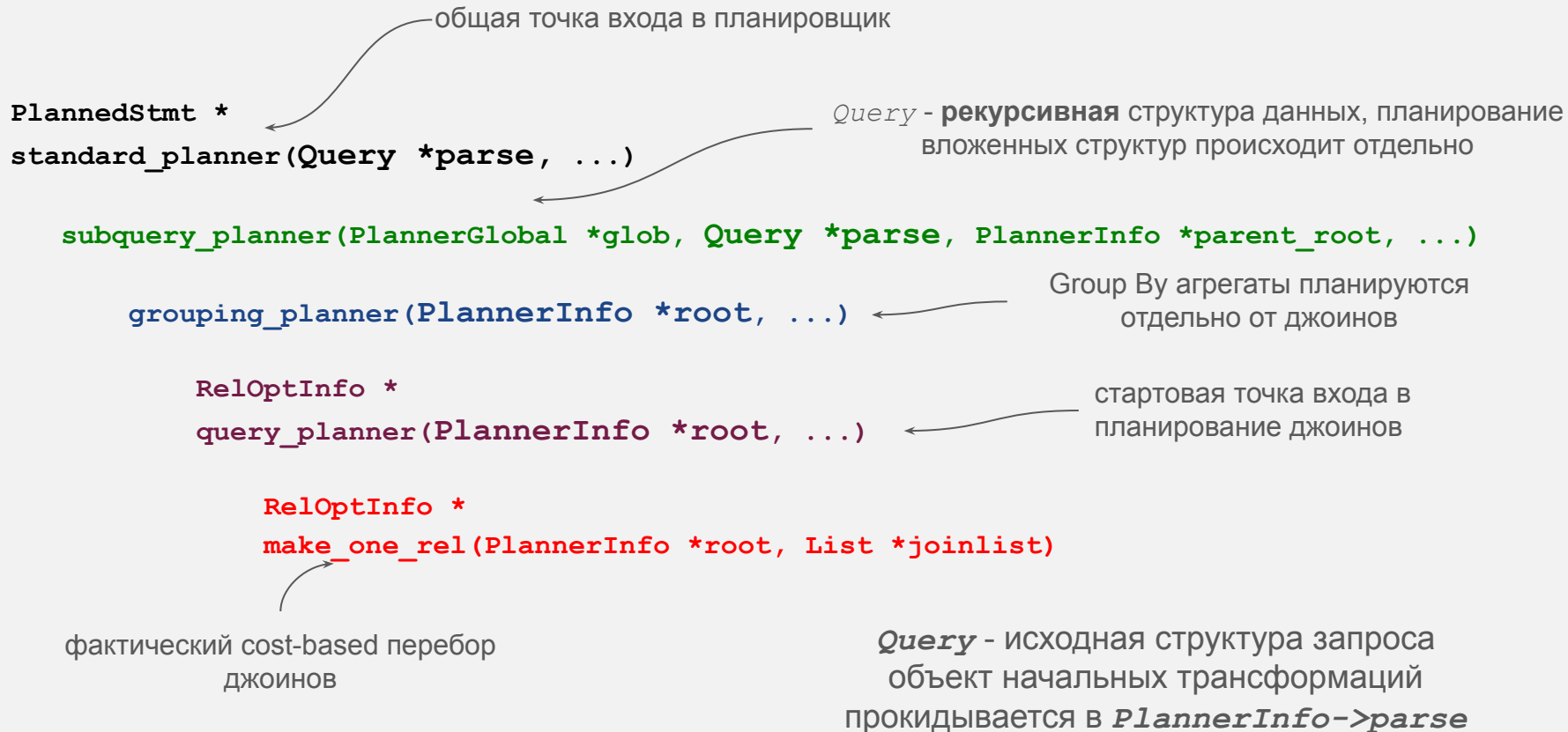
мы решили продолжить ряд :)

старые доклады по теме:

[Recent PostgreSQL Optimizer Improvements](#) by Tom Lane 2003

[Inside the PostgreSQL Query Optimizer](#) by Neil Conway 2005

Workflow оптимизатора

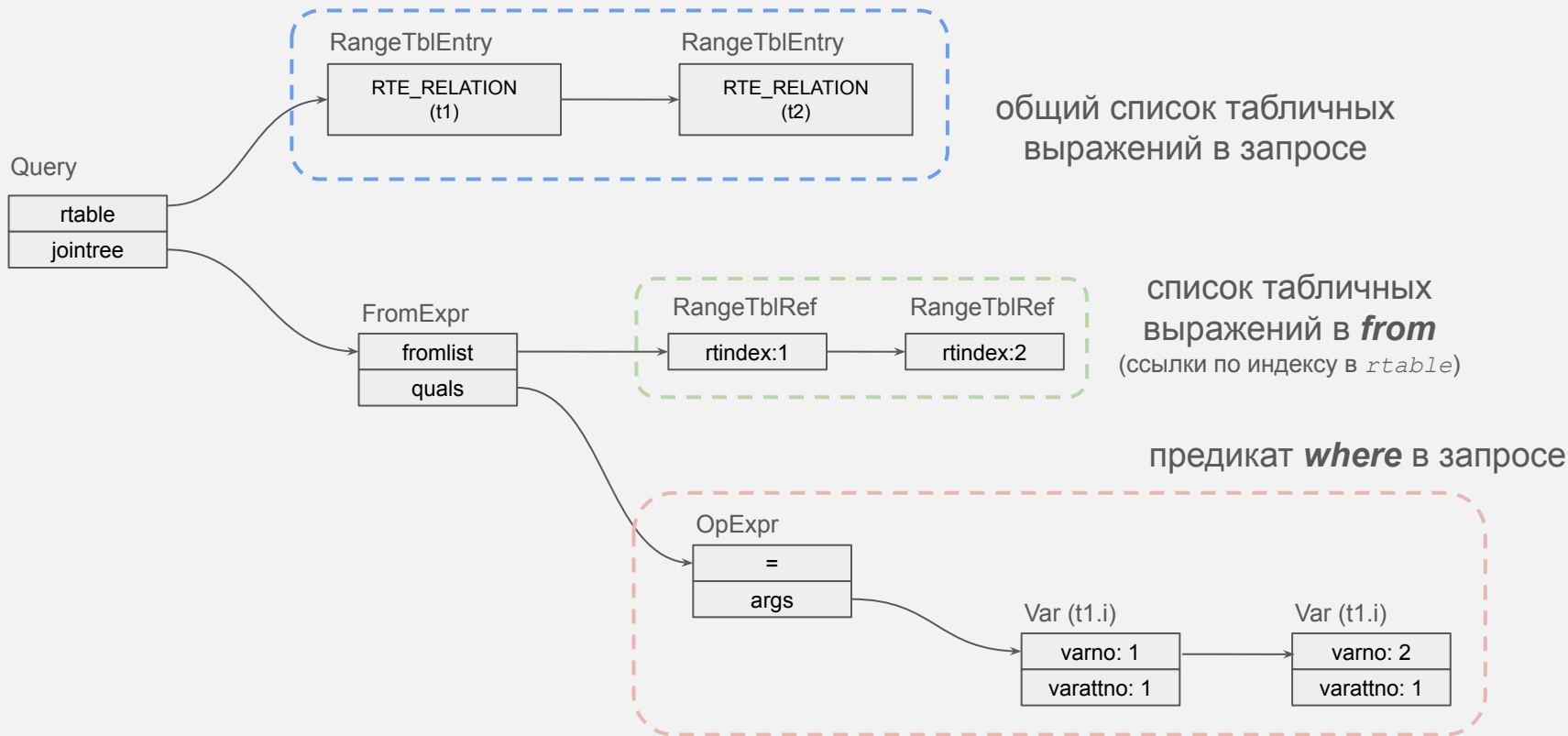


Query структура

- Стартовая структура для планирования
 - формируется после стадии **Analyze-Rewrite** на базе AST исходного запроса
- **Каноническое** представление исходного запроса
 - любой запрос описывается **Query** структурой
 - ... в т.ч. рекурсивно - подзапросы как **Query** объекты внутри верхнеуровневых запросов
- Начальное представление **алгебраического** дерева плана
 - преобразуется **as-is** в первичное алгебраическое дерево плана для применения преобразований в оптимизаторе ORCA

Query структура

```
select * from t1, t2 where t1.i = t2.i
```

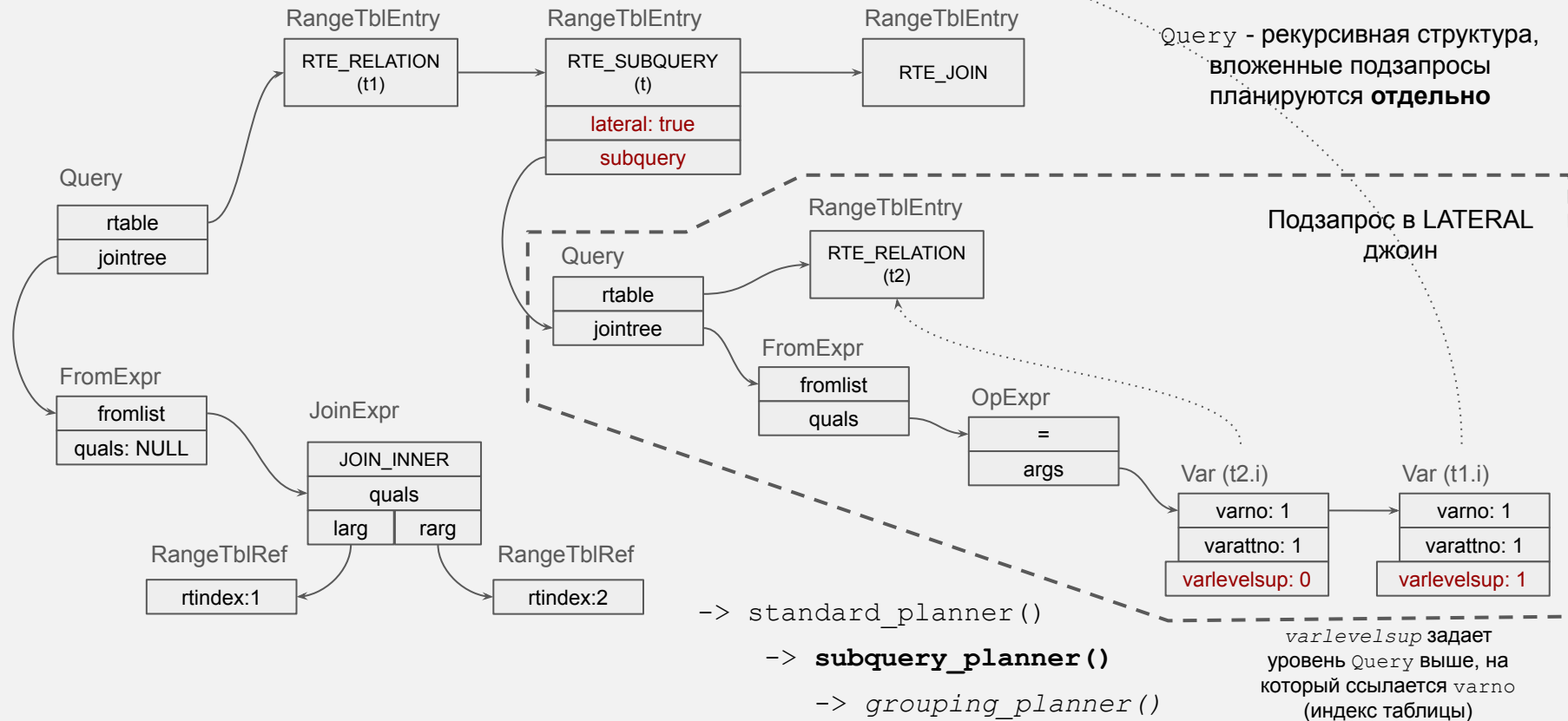


Жаргонные термины

- **Var** = variable = ссылка на колонку в таблице (выражении)
 - **varno** = ссылка (индекс в *rtable*) на таблицу, **varattno** = номер атрибута в *pg_attribute*
- **Rel** = relation = реальная или виртуальная таблица
- **Base rel** = примитив из списка FROM
 - реальная таблица, подзапрос (вложенный Query) или функция, возвращающая набор строк
- **Join rel** = табличное выражение от результата джоина
 - описывается набором base rels, участвующих в джоине
- **Qual** = qualifier = предикат (clause) в условии WHERE
- **Join qual** = qual, включающий Vars от нескольких base rels
- **Target list** = результирующий список выражений в SELECT
- **RTE** = Range Table entry = описание relation в запросе
 - на неё по индексу ссылаются все другие элементы Query

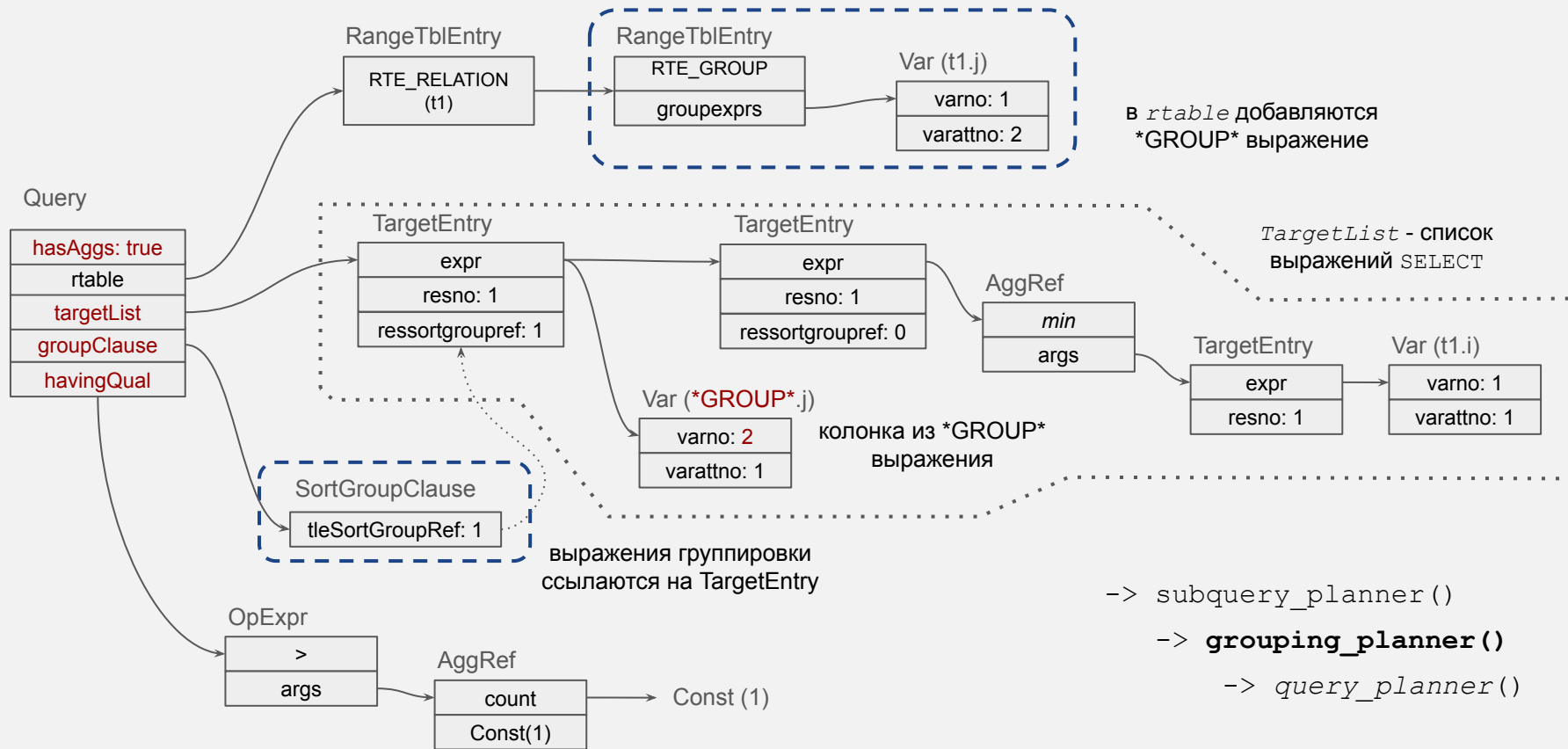
Случай подзапроса

```
select * from t1 cross join lateral (  
    select * from t2 where t2.i = t1.i) t
```



Случай GROUP BY

```
select j, min(i) from t1 group by j having count(1) > 1
```



Query как начальное алгебраическое дерево плана

```
select j, min(i)
from t1 join t2 using(i)
group by j
having count(1) > 1
```

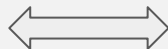
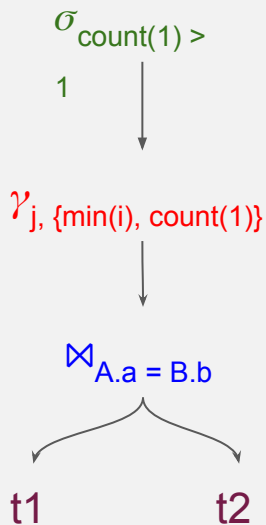
трансляция *Query*
структуры в исходное
алгебраическое дерево
плана в оптимизаторе
ORCA *

```
+--CLogicalSelect
|  |--CLogicalGbAgg( Global ) Grp Cols: [""j"" (1)] ...
|  |  |--CLogicalNaryJoin
|  |    |--CLogicalGet ""t1"" (""t1"") ...
|  |    |--CLogicalGet ""t2"" (""t2"") ...
|  |      +--CScalarCmp (=)
|  |        |--CScalarIdent ""i"" (0)
|  |        +--CScalarIdent ""i"" (9)
|  +--CScalarProjectList
|    |--CScalarProjectElement ""min"" (18)
|    |  +--CScalarAggFunc (min , ...)
|    |    +--CScalarValuesList
|    |      +--CScalarIdent ""i"" (9)
|    +--CScalarProjectElement ""count"" (19)
|      +--CScalarAggFunc (count , ...)
|        +--CScalarValuesList
|          +--CScalarConst (1)
+--CScalarCmp (>)
  |--CScalarIdent ""count"" (19)
  +--CScalarConst (1)
```

вывод при включенной опции
optimizer_print_query

* исходный код трансляции в <https://github.com/greenplum-db/gpdb-archive/blob/main/src/backend/gpopt/translate/CTranslatorQueryToDXL.cpp>

Query как начальное алгебраическое дерево плана



```
+--CLogicalSelect
|  |--CLogicalGbAgg( Global ) Grp Cols: ["j" (1)] ...
|  |  |--CLogicalNaryJoin
|  |    |--CLogicalGet ""t1"" ("t1") ...
|  |    |--CLogicalGet ""t2"" ("t2") ...
|  |    +--CScalarCmp (=)
|  |      |--CScalarIdent ""i"" (0)
|  |      +--CScalarIdent ""i"" (9)
|  +--CScalarProjectList
|    |--CScalarProjectElement ""min"" (18)
|    |  +--CScalarAggFunc (min , ...)
|    |    +--CScalarValuesList
|    |      +--CScalarIdent ""i"" (9)
|    +--CScalarProjectElement ""count"" (19)
|    |  +--CScalarAggFunc (count , ...)
|    |    +--CScalarValuesList
|    |      +--CScalarConst (1)
+--CScalarCmp (>)
  |--CScalarIdent ""count"" (19)
  +--CScalarConst (1)
```

вывод при включенной опции
optimizer_print_query

Стадии перезаписи Query

PlannedStmt *

standard_planner(Query *parse, ...)

Ранний препроцессинг

применение логических
трансформаций к структуре Query

subquery_planner(PlannerGlobal *glob, Query *parse, PlannerInfo *parent_root)

grouping_planner(PlannerInfo *root, ...)

RelOptInfo *

query_planner(PlannerInfo *root, ...)

Поздний препроцессинг

подготовка к перебору джоинов
устранение лишних джоинов

RelOptInfo *

make_one_rel(PlannerInfo *root, List *joinlist)

Ранний препроцессинг Query

- Выравнивание CTE
- Преобразование `MERGE` в `JOIN`
- Выравнивание (**flattening, unnesting**) подзапросов в главный запрос
- Встраивание (**inline**) SQL функций в основное тело запроса
- Упрощение скалярных выражений
- Выравнивание подзапросов (**pull up subqueries**)
- Сведение `OUTER` джоинов к `INNER/ANTI`

Ранний препроцессинг Query

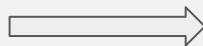
Практически все преобразования работают по алгоритму:

- найти подходящий паттерн выражения внутри `Query`
 - структура `Query` и внутренние выражения (`Expr`) обходятся в глубину многократно
- ... и применить эквивалентное преобразование
 - желательно чтобы всегда выигрышное

Выравнивание Common Table Expressions (CTEs)

PG12+

```
with t as (  
    select * from t1 where i > 0  
)  
select * from t2 join t using(i)
```



```
Hash Join  
Hash Cond: (t1.i = t2.i)  
-> Seq Scan on t1  
    Filter: (i > 0)  
-> Hash  
    -> Seq Scan on t2
```

CTE подзапрос переписывается в **классический** подзапрос
“магия” встраивания происходит позже

```
with t as materialized (  
    select * from t1 where i > 0  
)  
select * from t2 join t using(i)
```



```
Hash Join  
Hash Cond: (t.i = t2.i)  
CTE t  
-> Seq Scan on t1  
    Filter: (i > 0)  
-> CTE Scan on t  
-> Hash  
    -> Seq Scan on t2
```

Критерии встраивания подзапроса CTE

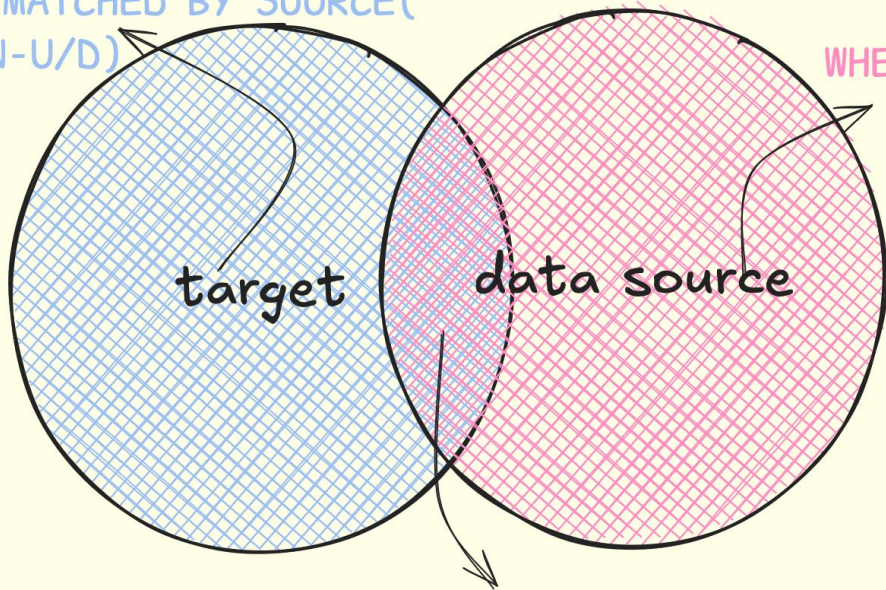
- **Не** содержит опцию-”хинт” `MATERIALIZED`
- **Не** рекурсивный
- Подзапрос **не** содержит побочные эффекты
 - модифицирующий DML
 - `volatile` функции
- Множественные ссылки на CTE с опцией-”хинтом” `NOT MATERIALIZED`
 - но даже с `NOT MATERIALIZED`, если внутри зовется **рекурсивный CTE**

Преобразование MERGE в JOIN

```
MERGE INTO target  
    USING data_source ON ...  
    WHEN MATCHED THEN ...  
    WHEN NOT MATCHED BY TARGET THEN  
    WHEN NOT MATCHED BY SOURCE THEN
```

Merge - PostgreSQL 17

WHEN NOT MATCHED BY SOURCE(
OPERATION-U/D)



WHEN NOT MATCHED
(OPERATION - I)

WHEN MATCHED(Operation - U/D)

I-INSERT
U-UPDATE
D-DELETE

Преобразование MERGE в JOIN

Варианты слияния MERGE

ВСЕ ВОЗМОЖНЫЕ:

MATCHED

NOT MATCHED BY TARGET

NOT MATCHED BY SOURCE

ТОЛЬКО MATCHED

+ NOT MATCHED BY TARGET

+ NOT MATCHED BY SOURCE

Вариант джоина

`target FULL OUTER JOIN source`

`target INNER JOIN source`

`target RIGHT OUTER JOIN source`

`target LEFT OUTER JOIN source`

Преобразование MERGE в JOIN

преобразование видно в плане
под Merge оператором

```
MERGE INTO t2 USING (  
  select i from t1 where i < 10  
) as t ON t2.i = t.i  
WHEN MATCHED THEN  
  delete  
WHEN NOT MATCHED THEN  
  insert values(i)  
WHEN NOT MATCHED BY SOURCE THEN  
  delete
```

```
Merge on t2  
-> Hash Full Join  
    Hash Cond: (t2.i = t1.i)  
    -> Seq Scan on t2  
    -> Hash  
        -> Seq Scan on t1  
            Filter: (i < 10)
```

по факту для MATCHED набора имеем semi-join ($\text{target} \ltimes \text{source}$) с проверкой на дубликаты от source

- более чем одна меняющая строка (change row) от source для каждой сопоставленной строки target образует **неоднозначность** слияния, и выбрасывается ошибка

Выравнивание EXISTS/ANY подзапросов

```
select * from t1 where exists (  
    select 1 from t2 where t1.i = t2.i  
)
```

```
select * from t1 where i ~=any() in (  
    select i from t2  
)
```



```
select * from t1 *SEMI JOIN* t2  
using(i)
```

Hash Semi Join

Hash Cond: (t1.i = t2.i)

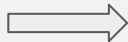
-> Seq Scan on t1

-> Hash

-> Seq Scan on t2

Выравнивание NOT EXISTS/ANY подзапросов

```
select * from t1 where not exists (  
    select 1 from t2  
    where t1.i = t2.i  
)
```



```
select * from t1 *ANTI JOIN* t2  
using(i)
```

Hash Anti Join

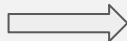
Hash Cond: (t1.i = t2.i)

-> Seq Scan on t1

-> Hash

-> Seq Scan on t2

```
select * from t1 where i not in (  
    select i from t2  
)
```



```
select * from t1 where  
    not i = any  $\Pi_i(t_2)$ 
```

Seq Scan on t1

Filter: (NOT (ANY (i =
 (hashed SubPlan 1).col1)))

SubPlan 1

-> Seq Scan on t2

NOT IN / NOT =ANY просто так в ANTI JOIN не преобразовать из-за семантики NULL значений

NOT IN и NULL семантика

`col IN (1, null)` \Rightarrow $\begin{cases} \text{TRUE если } col = 1 \\ \text{NULL (unknown) если } col \neq 1 \end{cases} \Rightarrow \text{никогда не FALSE}$

`col NOT IN (1, null)` \Rightarrow $\begin{cases} \text{FALSE если } col = 1 \\ \text{NULL (unknown) если } col \neq 1 \end{cases} \Rightarrow \text{никогда не TRUE}$

`SELECT ... WHERE col NOT IN (select)` \Rightarrow $\begin{cases} \text{*ANTI JOIN* если select не содержит NULLS} \\ \emptyset \text{ иначе, т.к NULL (unknown) фильтрующий} \end{cases}$

Стратегии вычисления NOT IN / NOT =ANY

Seq Scan on t1

Filter: (NOT (ANY (i = (hashed SubPlan 1).col1)))

SubPlan 1

-> Seq Scan on t2

ключи подплана кэшируются в **хэш-таблице**
работает только для **небольшого** датасета t2

Seq Scan on t1

Filter: (NOT (ANY (i = (SubPlan 1).col1)))

SubPlan 1

-> **Materialize**

-> Seq Scan on t2

во всех остальных случаях:
на каждый ключ t1 **NestLoop** поиск в
материализованном датасете t2

Hash Left Anti Semi (**Not-In**) Join

Hash Cond: (t1.i = t2.i)

-> Seq Scan on t1

-> Hash

-> Seq Scan on t2

Greenplum **адаптирует** стандартный Anti Join под NOT IN семантику
до джоина правое (inner) поддерево полностью **сканируется** для выявления NULLS
для MergeJoin NULL-ы должны идти **в начале** упорядоченной выборки

Встраивание функций в тело основного запроса

```
CREATE FUNCTION incr4(integer)
  RETURNS integer
  LANGUAGE sql
AS $function$
  select $1 + (2 + 2)
$function$
```

встраивание
скалярной функции



```
explain (costs off, verbose)
select incr4(i) from t1
```

Seq Scan on public.t1

Output: (i + 4)

константные выражения
предвычисляются заранее
(constant-folding)

```
CREATE FUNCTION t2_filter(integer)
  RETURNS SETOF t2
  LANGUAGE sql STABLE
AS $function$
  select * from t2 where t2.i = $1
$function$
```

set-returning функции
встраиваются как подзапросы



```
explain (costs off)
select * from t1 cross join t2_filter(t1.i)
```

Hash Join

Hash Cond: (t1.i = t2.i)

-> Seq Scan on t1

-> Hash

-> Seq Scan on t2

“магия” встраивания случается позже

Критерии встраивания функций

- Язык реализации **SQL**
- `Not STRICT`
- `Not VOLATILE` (stable или immutable)
- Без заданных конфигов (`SET` опция) и внешних хуков на вызов
- Аргументы не содержат вызовы volatile функций, либо подзапросы

Упрощение (предвычисление) скалярных выражений

- Скалярные функции

```
int4eq(1, NULL) => NULL
```

```
2 + 2 (int4pl(2, 2)) => 4
```

- Булевы выражения

```
x OR True => True
```

```
x AND False => False
```

- CASE выражения

```
CASE WHEN 2+2 = 4 THEN x+1  
ELSE 1/0 END => x+1
```

ошибка "ERROR: division by zero" не случается

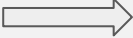
- ...

функция `eval_const_expressions_mutator()`

Упрощение (предвычисление) скалярных выражений

- Пронизывает всю стадию переписывания `Query`
- Каждая новая нетривиальная трансформация (например, встраивание `view`, либо тела функции) создает новые возможности для применения правил упрощения
- Приводит предикаты к виду “`Var op Constant`”, пригодному для расчета селективности
- В PostgreSQL типы и операции над ними **абстрагированы** => оптимизатор не всегда осведомлен о семантике операций
 - интуитивные преобразования могут не работать: $x + 0 \neq x$

Выравнивание “простых” подзапросов

```
select * from t1 join (  
    select i from t2 join t3 using(i)  
) using(i)  select * from t1 join (t2 join t3 using(i))  
using(i)
```

lateral джоин с
коррелированным подзапросом

```
select * from t1 cross join lateral (  
    select * from t2 where t2.i = t1.i  
)  select * from t1 join t2 on t2.i = t1.i
```

та самая “магия” встраивания подзапросов

Критерии “простоты”

- `SELECT` оператор
- без множественных операторов `UNION / INTERSECT / EXCEPT`
 - кроме `UNION ALL`
- без агрегатов, оконных функций, `GROUP BY`, `ORDER BY`, `DISTINCT`, `LIMIT`
- не `SELECT FOR SHARE / FOR UPDATE`
- не `CTE`
- без `volatile` функций в списке `SELECT (TargetList)`

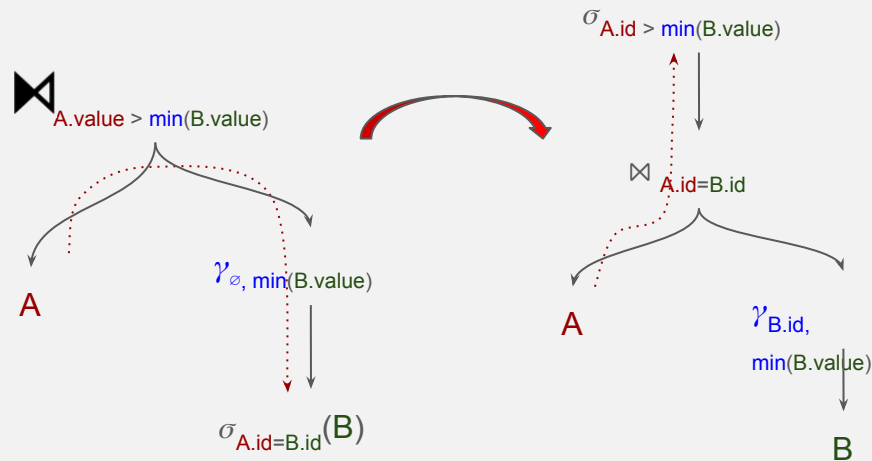
За бортом

- Сведение OUTER джоинов к INNER или ANTI
- Трансформации с PlaceholderVar
 - структурой, обратной ф-ции COALESCE (константа → NULL) *
 - контейнер для выражения из inner подплана поверх OUTER JOIN
- Удаление лишних ключей GROUP BY
- Проброс предикатов WHERE и JOIN/ON к таблицам (**predicate pushdown**)
- Построение классов эквивалентных выражений
- Формирование ограничений о порядке джоинов
- Удаление бесполезных джоинов
- Удаление self-джоинов
- ...

* <https://www.postgresql.org/message-id/AANLkTinP01jOWUI2nTifW5OPggO1Fsd27SqkjRLQ0HT0%40mail.gmail.com>

Практическая часть

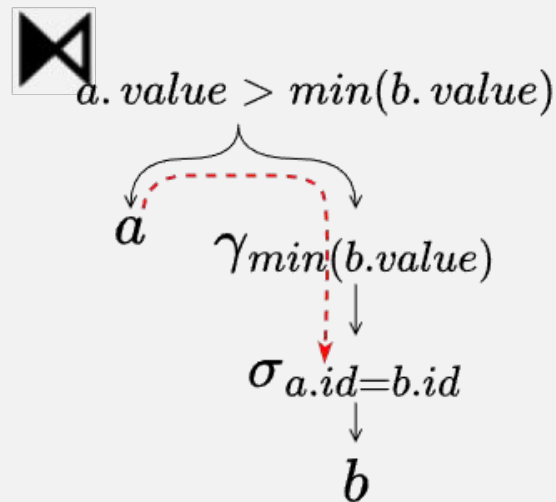
Декорреляция агрегатного подзапроса в предикате



Декорреляция подзапросов

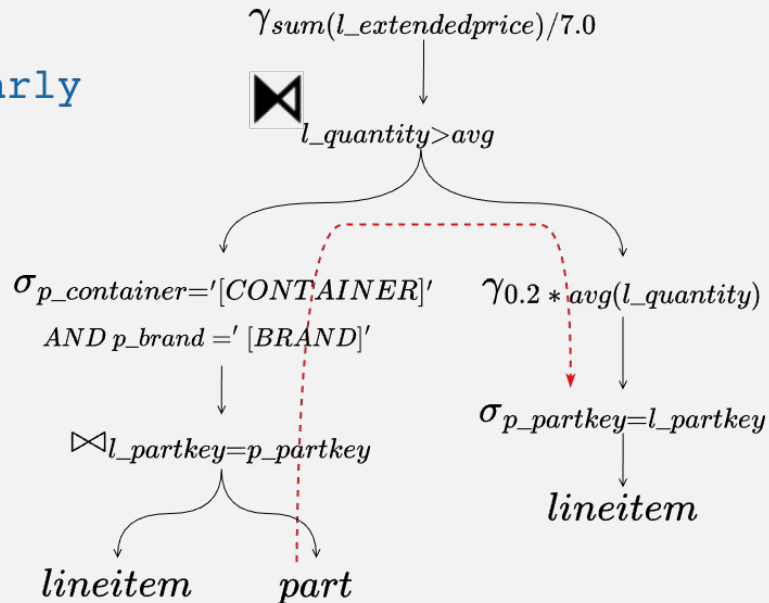
- > **Коррелированный подзапрос** - подзапрос, который использует атрибуты из внешнего запроса и зависит от него
- > **Декорреляция** - процесс удаления коррелированного подзапроса, например, превращением его в соединение с подзапросом

```
SELECT *
FROM a
WHERE a.value > (SELECT min(b.value)
                  FROM b
                  WHERE b.id = a.id)
```



TPC-H Q17

```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part
WHERE p_partkey = l_partkey
  AND p_brand = '[BRAND]'
  AND p_container = '[CONTAINER]'
  AND l_quantity < (
    SELECT 0.2 * avg(l_quantity)
    FROM lineitem
    WHERE l_partkey = p_partkey)
```



План запроса Q17 TPC-H

Aggregate

-> Nested Loop

-> Seq Scan on part

-> Index Scan using idx_lineitem_part_supp on lineitem (actual rows=3 loops=2'008)

Index Cond: (l_partkey = part.p_partkey)

Filter: (l_quantity < (SubPlan 1))

Rows Removed by Filter: 27

SubPlan 1

-> Aggregate

-> Index Scan using idx_lineitem_part_supp on lineitem lineitem_1 (actual rows=31 loops=60'156)

Index Cond: (l_partkey = part.p_partkey)

Planning Time

Execution Time: 2'308.801 ms

Декорреляция подзапроса

```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part
WHERE p_partkey = l_partkey
      AND p_brand = '[BRAND]'
      AND p_container = '[CONTAINER]'
      AND l_quantity < (
          SELECT 0.2 * avg(l_quantity)
          FROM lineitem
          WHERE l_partkey = p_partkey)
```


Декорреляция подзапроса

```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part,
(
    SELECT 0.2 * avg(l_quantity) AS result,
           l_partkey
    FROM lineitem
    GROUP BY l_partkey
) sub
WHERE p_partkey = sub.l_partkey
      AND p_brand = '[BRAND]'
      AND p_container = '[CONTAINER]'
      AND l_quantity < sub.result
      AND p_partkey = l_partkey
```

Декорреляция подзапроса

```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part
WHERE p_partkey = l_partkey
AND p_brand = '[BRAND]'
AND p_container = '[CONTAINER]'
AND l_quantity < (
```

```
SELECT 0.2 * avg(l_quantity)
FROM lineitem
WHERE l_partkey = p_partkey)
```

```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part,
```

```
(
  SELECT 0.2 * avg(l_quantity) AS result,
        l_partkey
  FROM lineitem
  GROUP BY l_partkey
) sub
```

```
WHERE p_partkey = sub.l_partkey
AND p_brand = '[BRAND]'
AND p_container = '[CONTAINER]'
AND l_quantity < sub.result
AND p_partkey = l_partkey
```

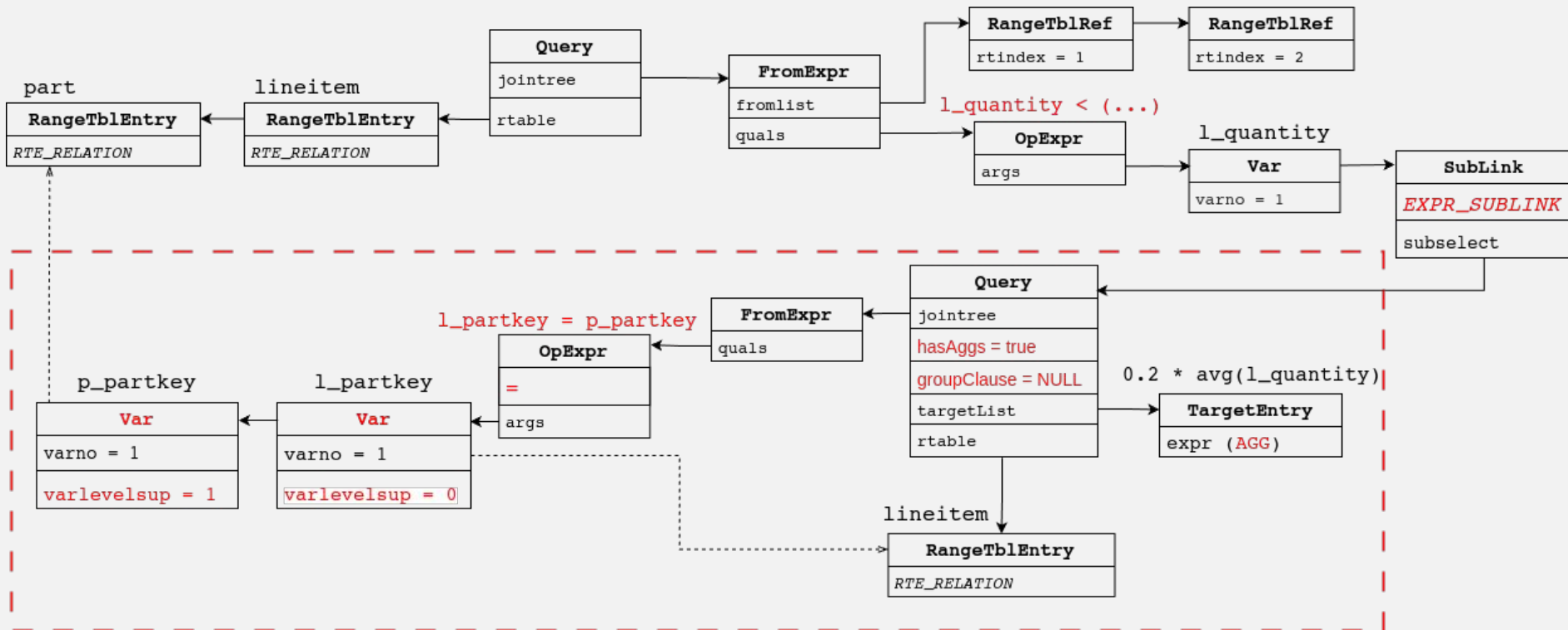
Декорреляция подзапроса

```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part LEFT JOIN
(
    SELECT 0.2 * count(l_quantity) AS result,
           l_partkey
    FROM lineitem
    GROUP BY l_partkey
) sub ON p_partkey = sub.l_partkey
WHERE p_partkey = l_partkey
    AND p_brand = '[BRAND]'
    AND p_container = '[CONTAINER]'
    AND l_quantity < COALESCE(sub.result, 0)
```

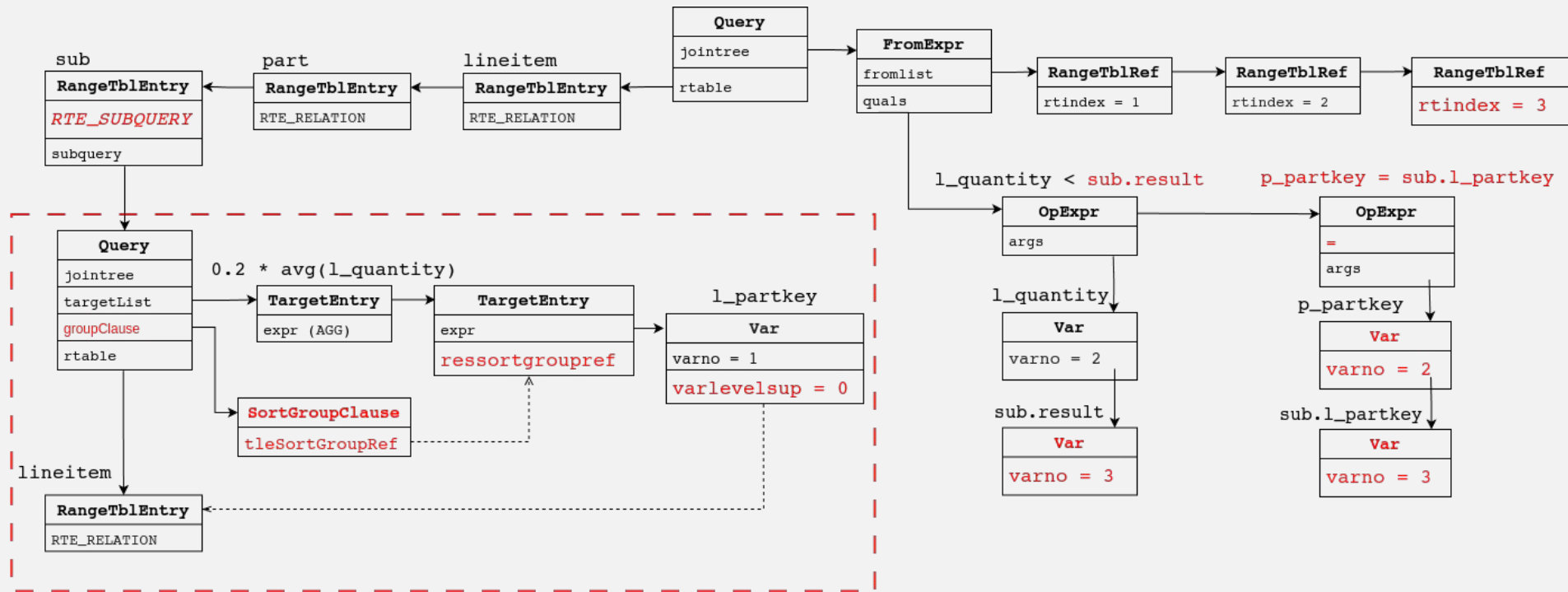
Декорреляция подзапроса

```
SELECT *  
FROM outer LEFT JOIN (...) subquery  
    ON p_partkey.attr = subquery.attr  
WHERE outer.attr < COALESCE(subquery.result, [default])
```

Декорреляция подзапроса



Декорреляция подзапроса



Один из операндов - подзапрос, возвращающий скалярное значение

```
OpExpr *op = (OpExpr *)node;
SubLink *sublink;

if (list_length(op->args) != 2)
    return node;

/* Один из операндов - подзапрос */
if (IsA(linitial(op->args), SubLink))
    sublink = (SubLink *)linitial(op->args);
else if (IsA(llast(op->args), SubLink))
    sublink = (SubLink *) llast(op->args);
else
    return node;

/* Подзапрос возвращает скалярное выражение */
if (sublink->subLinkType != EXPR_SUBLINK)
{
    return node;
}
```

Подзапрос возвращает единственный агрегат

```
static bool has_aggs_walker(Node *node, void *arg)
{
    /* Находим Aggref - агрегирующую функцию */
    if (IsA(node, Aggref))
        return true;

    return expression_tree_walker(node, &has_aggs_walker, arg);
}

static bool returns_single_agg(Query *parse)
{
    return parse->groupClause == NIL &&
        list_length(parse->targetList) == 1 &&
        has_aggs_walker(linitial(parse->targetList), NULL);
}
```


Агрегат возвращает NULL на пустом входе

```
Aggref *agg = (Aggref *)node;

/* Получаем aggfinalfn, agginitval, aggdeserialfn из pg_aggregate */
if (!get_finalfunc_and_initval(agg->aggfnoid, &aggfinalfunc,
                               &agginitval, &deserialfunc))
    return true;

/* Десериализуем initval во внутреннее состояние */
if (agginitval != NULL)
{
    fmgr_info(deserialfunc, &flinfo);
    InitFunctionCallInfoData(*fcinfo, &flinfo, 2, InvalidOid, NULL, NULL);
    fcinfo->args[0].value = PointerGetDatum(agginitval);

    fcinfo->args[1].value = (Datum) 0;
    fcinfo->args[1].isnull = true;

    initval = FunctionCallInvoke(fcinfo);
}
else
{
    initval = (Datum) 0;
}
```

Агрегат возвращает NULL на пустом входе

```
/* Вызываем финализатор */  
fmgr_info(aggfinalfunc, &flinfo);  
InitFunctionCallInfoData(*fcinfo, &flinfo, 1, InvalidOid, NULL, NULL);  
fcinfo->args[0].value = initval;  
fcinfo->args[0].isnull = agginitval == NULL;  
  
/* Проверяем, что агрегат возвращает NULL */  
FunctionCallInvoke(fcinfo);  
return !fcinfo->isnull;
```

Предикат внутри подзапроса - равенство 2 атрибутов

```
case T_OpExpr:
{
    OpExpr *op = (OpExpr *)expr;
    if (list_length(op->args) == 2 &&
        IsA(get_lefttop(op), Var) &&
        IsA(get_righttop(op), Var))
    {
        /*
         * OUTER.VAR = INNER.VAR или INNER.VAR = OUTER.VAR
         */
        Var *left = (Var *) get_lefttop(op);
        Var *right = (Var *) get_righttop(op);
        if ((left->varlevelsup == 1 || right->varlevelsup == 1) &&
            get_oprrest(op->opno) == F_EQSEL)
        {
            List **ops = (List **)context;
            *ops = lappend(*ops, op);
            return makeBoolConst(true, false);
        }
    }
}
break;
```

Добавляем подзапрос в Range Table родителя

```
/* Создаем RTE для подзапроса */  
parse = (Query *) sublink->subselect;  
nsitem = addRangeTableEntryForSubquery(make_parsestate(NULL),  
                                        parse,  
                                        makeAlias("CORRELATED_subquery", colnames),  
                                        false, false);  
/* Добавляем RTE в родительский запрос */  
parent->rtable = lappend(parent->rtable, nsitem->p_rte);
```

Создаем узел JOIN для подзапроса

```
rtr = makeNode(RangeTblRef);  
rtr->rtindex = new_subquery_rtindex;  
  
join = makeNode(JoinExpr);  
join->jointype = JOIN_INNER;  
join->larg = NULL;  
join->rarg = (Node *)rtr;
```

Добавляем атрибуты подзапроса в SELECT и GROUP BY

```
/* Добавляем атрибут в список SELECT */
targetEntry = makeTargetEntry((Expr *)inner,
                               list_length(parse->targetList) + 1,
                               pstrdup(""),
                               false);
parse->targetList = lappend(parse->targetList, targetEntry);

/* Создаем GROUP BY для этого атрибута */
sgClause = makeNode(SortGroupClause);
sgClause->eqop = opExpr->opno;
sgClause->hashable = true;
sgClause->tleSortGroupRef = assignSortGroupRef(targetEntry, parse->targetList);
parse->groupClause = lappend(parse->groupClause, sgClause);
```


Заменяем весь подзапрос на атрибут из подзапроса

```

if ((j = convert_correlated_sublink_to_join(root->parse, sublink, available_rels1)) != NULL)
{
    Query *subquery = (Query *)sublink->subselect;
    Var *var;
    int rtindex;

    /* Заменяем коррелированный подзапрос на атрибут */
    rtindex = list_length(root->parse->rtable);
    result = (TargetEntry *) linitial(subquery->targetList);
    var = makeVarFromTargetEntry(rtindex, result);
    llast(op->args) = var;

    /* Выносим этот оператор в JOIN предикат */
    j->quals = (Node *)makeBoolExpr(AND_EXPR, list_make2(op, j->quals), -1);
    j->larg = *jtlink1;
    *jtlink1 = (Node *)j;

    j->rarg = pull_up_sublinks_jointree_recurse(root, j->rarg, &child_rels);
    j->quals = pull_up_sublinks_qual_recurse(root, j->quals, &j->rarg,
                                             child_rels, NULL, NULL);
    return NULL;
}

```

План после оптимизации

Aggregate

-> Nested Loop

-> Nested Loop

-> HashAggregate

-> Seq Scan on lineitem lineitem_1

-> Index Scan using part_pkey on part

-> Index Scan using idx_lineitem_part_supp on lineitem)

Index Cond: (l_partkey = lineitem_1.l_partkey)

Filter: (l_quantity < ((0.2 * avg(lineitem_1.l_quantity))))

Rows Removed by Filter: 27

План после оптимизации

Aggregate

-> Nested Loop

-> Nested Loop

-> HashAggregate

-> Seq Scan on lineitem lineitem_1

-> Index Scan using part_pkey on part

-> Index Scan using idx_lineitem_part_supp on lineitem)

Index Cond: (l_partkey = lineitem_1.l_partkey)

Filter: (l_quantity < ((0.2 * avg(lineitem_1.l_quantity))))

Rows Removed by Filter: 27

Planning Time

Execution Time: 77'633.718 ms

План после оптимизации

Aggregate

-> Nested Loop

-> Nested Loop

-> HashAggregate (actual time=37'615.114..60'351.691 rows=2'000'000 loops=1)

Group Key: lineitem_1.l_partkey

Batches: 129 Memory Usage: 8209kB Disk Usage: 2219184kB

-> Seq Scan on lineitem lineitem_1 (actual time=0.106..8'451.107 rows=59'986'052 loops=1)

-> Index Scan using part_pkey on part

-> Index Scan using idx_lineitem_part_supp on lineitem)

Index Cond: (l_partkey = lineitem_1.l_partkey)

Filter: (l_quantity < ((0.2 * avg(lineitem_1.l_quantity))))

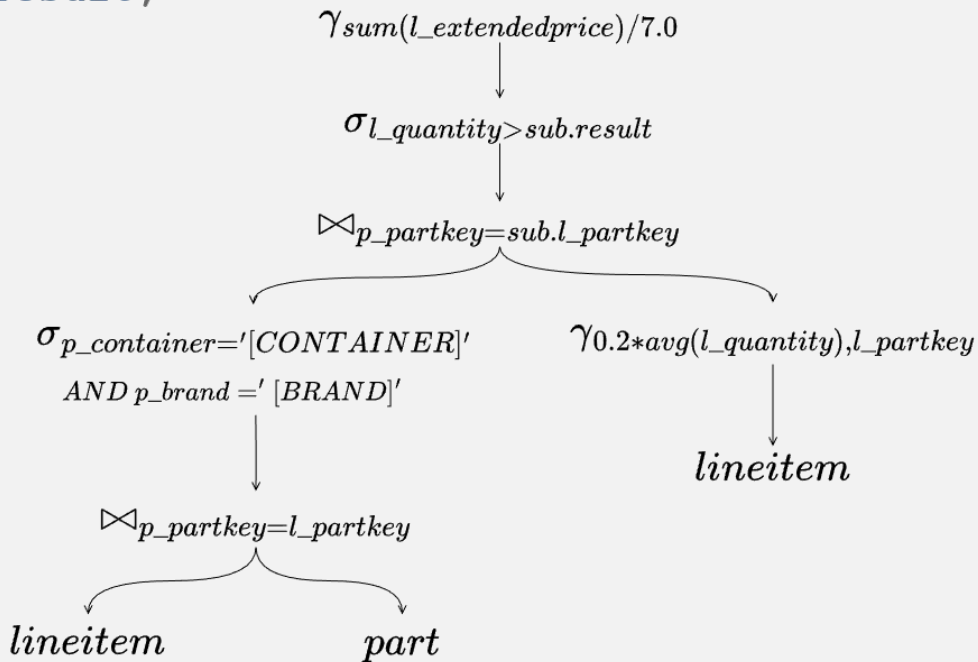
Rows Removed by Filter: 27

Planning Time

Execution Time: 77'633.718 ms

Join pushdown

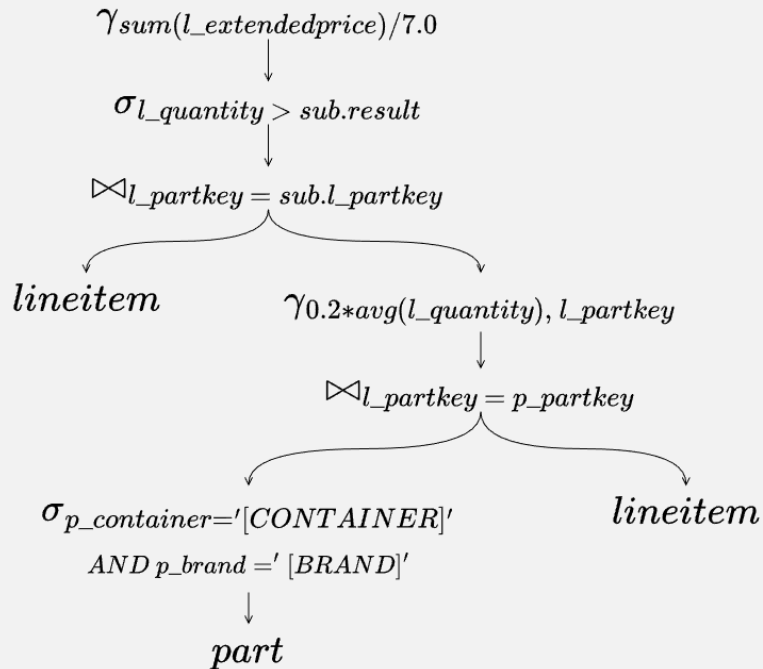
```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part,
(
  SELECT 0.2 * avg(l_quantity) AS result,
         l_partkey
  FROM lineitem
  GROUP BY l_partkey
) sub
WHERE p_partkey = sub.l_partkey
   AND p_brand = '[BRAND]'
   AND p_container = '[CONTAINER]'
   AND l_quantity < sub.result
   AND p_partkey = l_partkey
```



Join pushdown

```

SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem,
(
  SELECT 0.2 * avg(l_quantity) AS result,
         l_partkey
  FROM lineitem, part
  WHERE p_partkey = l_partkey
        AND p_brand = '[BRAND]'
        AND p_container = '[CONTAINER]'
  GROUP BY l_partkey
) sub
WHERE sub.l_partkey = l_partkey
      AND l_quantity < sub.result
  
```



Join pushdown

```

SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part,
(
  SELECT 0.2 * avg(l_quantity) AS result,
         l_partkey
  FROM lineitem
  GROUP BY l_partkey
) sub
WHERE p_partkey = sub.l_partkey
AND p_brand = '[BRAND]'
AND p_container = '[CONTAINER]'
AND l_quantity < sub.result
AND p_partkey = l_partkey

```

```

SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem,
(
  SELECT 0.2 * avg(l_quantity) AS result,
         l_partkey
  FROM lineitem, part
  WHERE p_partkey = l_partkey
        AND p_brand = '[BRAND]'
        AND p_container = '[CONTAINER]'
  GROUP BY l_partkey
) sub
WHERE sub.l_partkey = l_partkey
AND l_quantity < sub.result

```

Join pushdown

- > У отношения должен быть **уникальный ключ**
- > Предикат JOIN не использует результатов **агрегирующих функций**

```
part.p_partkey = sub.l_partkey
```

Join pushdown

- У отношения должен быть **уникальный ключ**
- Предикат JOIN не использует результатов **агрегирующих функций**

```
part.p_partkey = sub.l_partkey
```

$$S \bowtie_p (\gamma_{A, F} R) = \gamma_{A \cup \text{columns}(S), F} (S \bowtie_p R)$$

Join pushdown

- > У отношения должен быть **уникальный ключ**
- > Предикат JOIN не использует результатов **агрегирующих функций**

```
part.p_partkey = sub.l_partkey
```

$$S \bowtie_p (\gamma_{A, F} R) = \gamma_{A \cup \boxed{columns(S)}, F} (S \bowtie_p R)$$

Join pushdown

```
SELECT sum(l_extendedprice) / 7.0 AS avg_yearly
FROM lineitem, part,
(
    SELECT 0.2 * avg(l_quantity) AS result,
           l_partkey
    FROM lineitem
    GROUP BY l_partkey
) sub
WHERE p_partkey = sub.l_partkey
      AND p_brand = '[BRAND]'
      AND p_container = '[CONTAINER]'
      AND l_quantity < sub.result
      AND p_partkey = l_partkey
```

Новый код после *deconstruct_jointree*

```
joinlist = deconstruct_jointree(root);  
  
/* ... */  
  
joinlist = pushdown_joins(root, joinlist);
```

Находим класс эквивалентности, содержащий атрибут подзапроса

```
EquivalenceClass *ec = lfirst_node(EquivalenceClass, lc);
ListCell *lc2;

subquery_var = NULL;
foreach(lc2, ec->ec_members)
{
    EquivalenceMember *member = (EquivalenceMember *)lfirst(lc2);
    if (IsA(member->em_expr, Var))
    {
        Var *var = (Var *)member->em_expr;

        /* Проверяем, что атрибут принадлежит подзапросу */
        if (var->varno == subquery_varno && var->varlevelsup == 0)
        {
            subquery_var = var;
            break;
        }
    }
}
```

Находим атрибут другого отношения, НЕ подзапроса

```
joined_var = NULL;
eclass = NULL;
foreach(lc2, ec->ec_members)
{
    EquivalenceMember *member = (EquivalenceMember *)lfirst(lc2);
    if (IsA(member->em_expr, Var))
    {
        Var *var = (Var *)member->em_expr;

        /* Находим атрибут другого отношения, не подзапроса */
        if (var->varno != subquery_varno && var->varlevelsup == 0)
        {
            joined_var = var;
            eclass = ec;
            break;
        }
    }
}
```

Атрибут подзапроса не ссылается на агрегат

```
static bool has_aggs_walker(Node *node, void *arg)
{
    /* Находим Aggref - агрегирующую функцию */
    if (IsA(node, Aggref))
        return true;

    return expression_tree_walker(node, &has_aggs_walker, arg);
}

static bool is_agg(Query *subquery, Var *subquery_join_var)
{
    Bitmapset *aggs_attnums;
    ListCell *lc;

    aggs_attnums = NULL;
    /* Обходим все выражения в SELECT подзапроса */
    foreach(lc, subquery->targetList)
    {
        TargetEntry *entry = (TargetEntry *)lfirst(lc);

        if (entry->resno == subquery_join_var->varattno)
        {
            return has_aggs_walker((Node *)entry->expr, NULL);
        }
    }

    return false;
}
```

Атрибут отношения составляет ключ

```

/* Создаем условие SUB.VAR = REL.VAR */
join_clause = generate_joinclause(eclass, subquery_var, joined_var);
if (join_clause == NULL)
    return joinlist;

/* Проверяем ключ на уникальность */
joined_rel = root->simple_rel_array[joined_var->varno];
if (!relation_has_unique_index_for(root, joined_rel, list_make1(join_clause), NIL, NIL))
    return joinlist;

```

Добавляем отношение в Range Table подзапроса

```
/* Берем RangeTblEntry из родителя */  
rte = root->simple_rte_array[rtindex];  
remove_rte_fromlist(root->parse, rtindex);  
  
/* Переносим его в Range Table подзапроса */  
subquery->rtable = lappend(subquery->rtable, rte);  
  
/* И добавляем в список FROM подзапроса */  
rtr = makeNode(RangeTblRef);  
rtr->rtindex = list_length(subquery->rtable);  
subquery->jointree->fromlist = lappend(subquery->jointree->fromlist, rtr);
```


Обновляем атрибуты подзапроса и отношения

```
if (IsA(node, Var))
{
    Var *var = (Var *)node;
    if (var->varlevelsup != 0)
        return node;

    if (var->varno == context->old_joined_rti)
    {
        /* Обновляем varno соединяемого отношения */
        var->varno = context->new_joined_rti;
    }
    else if (var->varno == context->subquery_rti)
    {
        /* Заменяем Var на выражение в списке SELECT подзапроса */
        TargetEntry *te = get_target_entry(context->subquery->targetList,
                                           var->varattno);
        return (Node *) copyObject(te->expr);
    }
}
```


Join pushdown

Aggregate

-> Nested Loop

-> HashAggregate (actual rows=2'008 loops=1)

Group Key: lineitem_1.l_partkey

Batches: 1 Memory Usage: 1809kB

-> Nested Loop (actual rows=60'156 loops=1)

-> Seq Scan on part

-> Index Scan using idx_lineitem_part_supp on lineitem lineitem_1

-> Index Scan using idx_lineitem_part_supp on lineitem (actual rows=3 loops=2'008)

Index Cond: (l_partkey = lineitem_1.l_partkey)

Filter: (l_quantity < ((0.2 * avg(lineitem_1.l_quantity))))

Rows Removed by Filter: 27

Planning Time

Execution Time: 781.798 ms

Join pushdown

Aggregate

-> Nested Loop

-> HashAggregate (actual rows=2'008 loops=1)

Group Key: lineitem_1.l_partkey

Batches: 1 Memory Usage: 1809kB

-> Nested Loop (actual rows=60'156 loops=1)

-> Seq Scan on part

-> Index Scan using idx_lineitem_part_supp on lineitem lineitem_1

-> Index Scan using idx_lineitem_part_supp on lineitem (actual rows=3 loops=2'008)

Index Cond: (l_partkey = lineitem_1.l_partkey)

Filter: (l_quantity < ((0.2 * avg(lineitem_1.l_quantity))))

Rows Removed by Filter: 27

Planning Time

Execution Time: 781.798 ms

Проделанный путь

1



Исходный запрос

- Есть коррелированный подзапрос

0:2.3

2



Декорреляция

- Вынесли подзапрос в FROM
- Сгруппировали по атрибуту равенства
- Подняли JOIN предикат

1:10

3



Join pushdown

- Перенесли таблицу part в подзапрос
- Обновили атрибуты предикатов

0:0.7

Финалочка

- > Декорреляция агрегатных подзапросов и проброс JOIN'ов под GROUP BY - **эвристические** преобразования
 - выигрыша от трансформаций может и не быть
- > Лучше замкнуть их в **стоимостной** перебор JOIN'ов
 - для алгоритма перебора “снизу-вверх” (bottom-up) задача челленджевая
- > ... либо закрыть их активацию GUC'ами

- > Первая стадия оптимизации (применение начальных трансформаций) - наиболее сложная и запутанная часть планировщика
 - теоретическая основа - реляционная алгебра
 - по мере совершенствования сложность кода и кол-во рассматриваемых трансформаций будут **расти... и расти**



Спасибо за внимание!
Вопросы? Критика! Пожелания



www.tantorlabs.ru