GREENPLUM DATABASE®

阿里云 aliyun.com

Greenplum内核揭秘之查询优化

钉钉直播 | 6月19日 16：00 - 17：00

Agenda

- Greenplum查询优化器介绍
- Greenplum查询优化的具体处理过程
  - 查询树的预处理
  - 扫描/连接优化
  - 扫描/连接之外的优化
  - 计划树的后处理

查询优化器介绍

- 对于给定的查询语句，找到“代价”最小的查询计划
    - 对于同一个查询语句，一般可以由多种方式 执行
    - 查询优化器尽可能去遍 历每一种可能的执行方式
    - 找到“代价”最小的执行方式，并把它 转换成可执行的计划树

```
# explain select * from a join b on a.i = b.i;
                              QUERY PLAN
------------------------------------------------------------------------
 Nested Loop  (cost=0.29..9.32 rows=1 width=24)
   ->  Seq Scan on a  (cost=0.00..1.01 rows=1 width=12)
   ->  Index Scan using b_i_idx on b  (cost=0.29..8.30 rows=1 width=12)
         Index Cond: (i = a.i)
(4 rows)
```

```
# explain select * from a join b on a.i = b.i;
                           QUERY PLAN
-------------------------------------------------------------
 Hash Join  (cost=1.02..193.53 rows=1 width=24)
   Hash Cond: (b.i = a.i)
   ->  Seq Scan on b  (cost=0.00..155.00 rows=10000 width=12)
   ->  Hash  (cost=1.01..1.01 rows=1 width=12)
         ->  Seq Scan on a  (cost=0.00..1.01 rows=1 width=12)
(5 rows)
```

```
# explain select * from a join b on a.i = b.i;
                                 QUERY PLAN
------------------------------------------------------------------------------
 Merge Join  (cost=1.31..354.32 rows=1 width=24)
   Merge Cond: (b.i = a.i)
   ->  Index Scan using b_i_idx on b  (cost=0.29..328.29 rows=10000 width=12)
   ->  Sort  (cost=1.02..1.02 rows=1 width=12)
         Sort Key: a.i
         ->  Seq Scan on a  (cost=0.00..1.01 rows=1 width=12)
(6 rows)
```

查询计划介绍

- 一个查询计划就是一个由计划节点组成的树
- 每个计划节点代表了一个特定类型的处理操作，计划节点中包含了执行器执行所需的全部信息
- 在执行时，计划节点产生输出元组
- 一般来说，扫描节点从数据表中获取输入元组
- 大部分其他节点从它们的子计划节点中获取输入元组，并处理产生输出元组

计划节点的类型

- 扫描节点
  - 顺序扫描，索引扫描，位图扫描
- 连接节点
  - Nestloop, hash, merge
- 非SPJ节点
  - Sort, aggregate, set operations (UNION etc)

```
# explain (costs off)
# select a.i, avg(a.j) from a join b on a.i = b.i group by 1 order by 2;
              QUERY PLAN
---------------------------------------
 Sort
   Sort Key: (avg(a.j))
   ->  HashAggregate
         Group Key: a.i
         ->  Hash Join
               Hash Cond: (b.i = a.i)
               ->  Seq Scan on b
               ->  Hash
                     ->  Seq Scan on a
(9 rows)
```

Sort ← 排序节点

HashAggregate ← 聚集节点

Hash Join ← 连接节点

Seq Scan on b ← 扫描节点

Greenplum查询优化的具体处理过程

GREENPLUM DATABASE®

- 查询树的预处理
  - 尽可能的简化查询树;收集有用信息
- 扫描/连接优化
  - 为查询语句中扫描和连接部分做计划
- 扫描/连接之外的优化
  - 为查询语句中扫描和连接之外的部分做计划
- 计划树的后处理
  - 把优化结果转换成执行器可以执行的形式

查询树的预处理（早期）

GREENPLUM DATABASE®

- 简化常量表达式
- 内联简单的SQL函数
- 提升IN, EXISTS类型的子连接
- 提升子查询
- 消除外连接
- etc.

简化常量表达式

GREENPLUM DATABASE®

- 简化函数表达式
  - 函数本身是“严格”的，并且输入参数中包含NULL值

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

  - 函数本身是“IMMUTABLE”的，并且输入参数全都是常量

    ```
    2 + 2 => 4
    ```

简化常量表达式

- 简化布尔表达式

```
"x OR true" => "true"

"x AND false" => "false"
```

简化常量表达式

GREENPLUM DATABASE®

- 简化CASE表达式

```
CASE WHEN 2+2 = 4 THEN x+1

ELSE 1/0 END

    ⇒ x+1

    ... not “ERROR: division by zero”
```

为什么简化常量表达式

- 仅需做一次计算，而不是为每行元组都做一次计算
- 视图展开和函数内联都可能会带来新的常量表达式 简化的机会
- 简化常量表达式也 为统计信息类的函数减少了 计算量

内联简单的SQL函数

```
CREATE FUNCTION incr4(int) RETURNS int

AS 'SELECT $1 + (2 + 2)' LANGUAGE SQL;



SELECT incr4(a) FROM foo;

=>

SELECT a + 4 FROM foo;
```

为什么内联简单的SQL函数

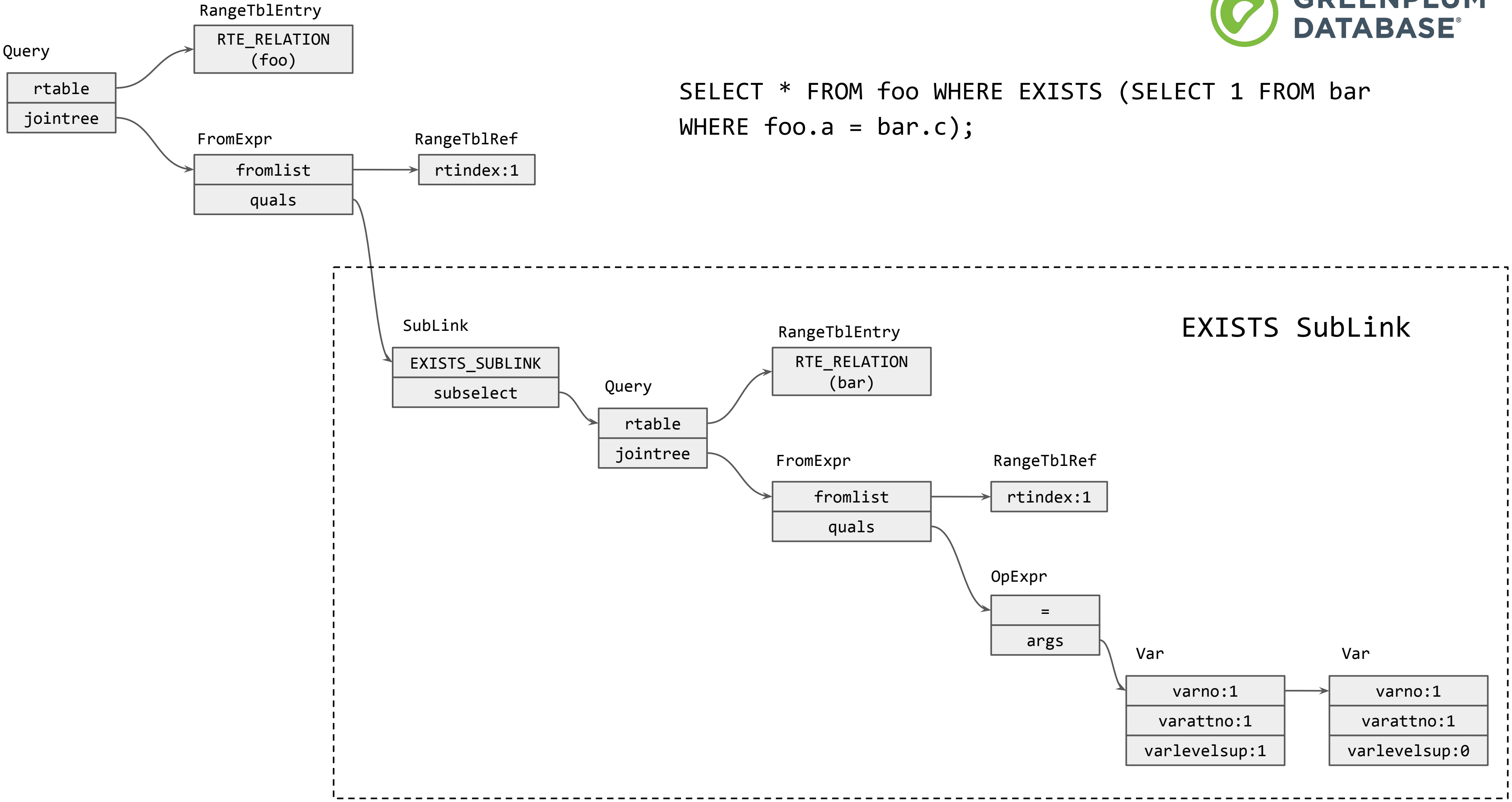
- 避免SQL函数调用的代价
- 为简化常量表达式提供新的机会

提升子连接

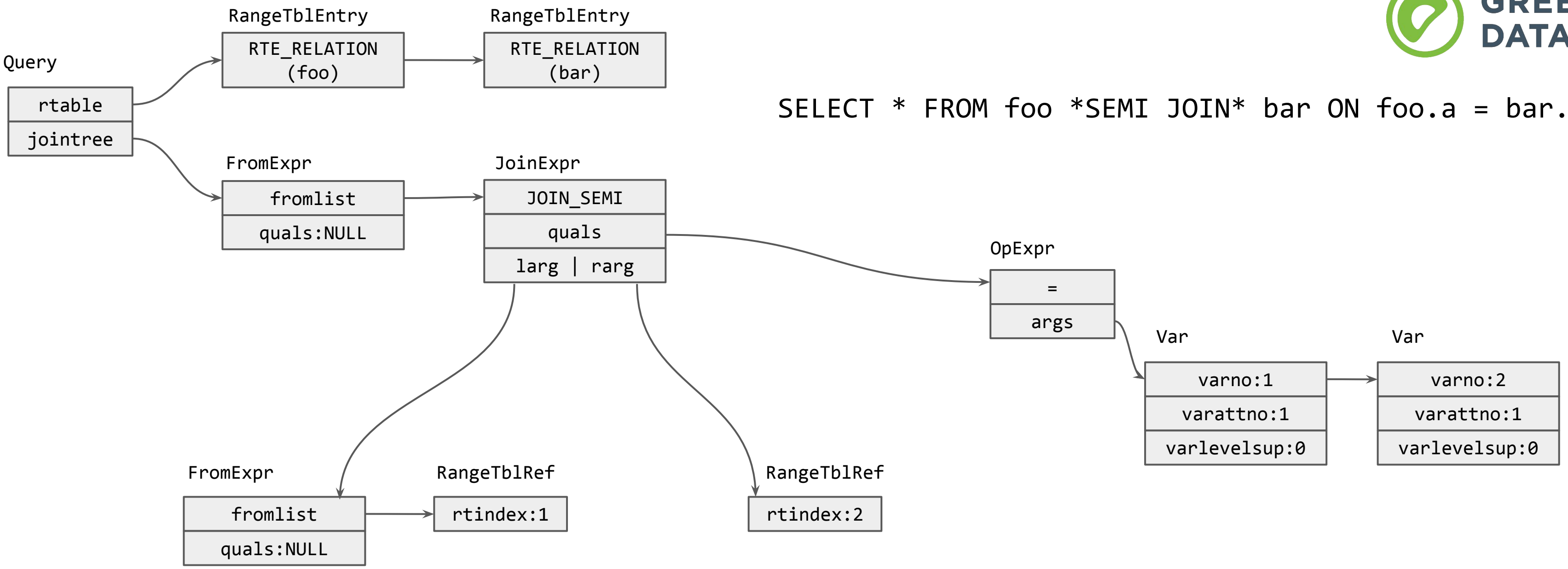
子连接(SubLink)是指出现在表达式中的子 查询，通常出现在WHERE或JOIN/ON子句中

```
SELECT * FROM foo WHERE EXISTS (SELECT 1 FROM bar WHERE foo.a = bar.c);

=>

SELECT * FROM foo *SEMI JOIN* bar ON foo.a = bar.c;
```

```
SELECT * FROM foo WHERE EXISTS (SELECT 1 FROM bar
WHERE foo.a = bar.c);
```



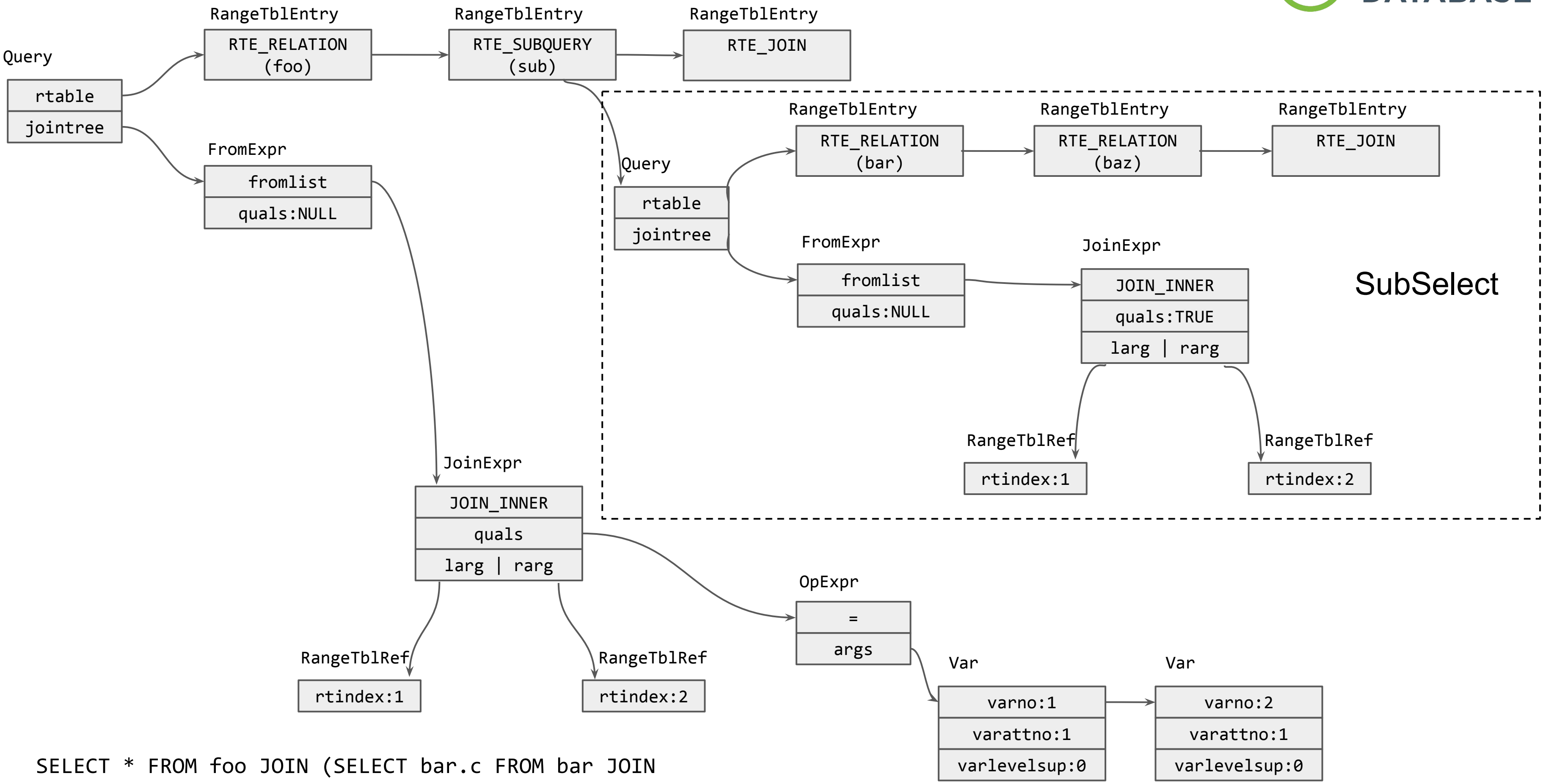
SELECT * FROM foo *SEMI JOIN* bar ON foo.a = bar.c;



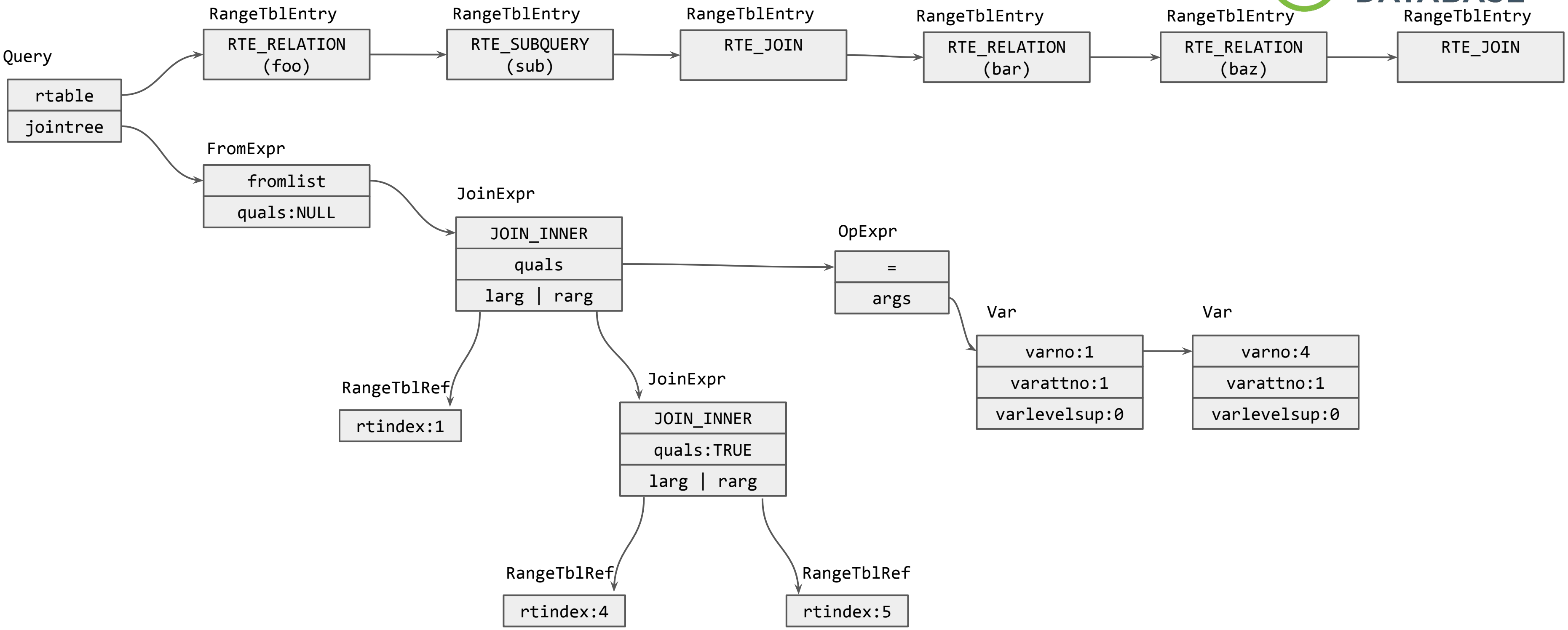
提升子查询

子查询一般以范围表的方式存在，通常出现在FROM子句中

```
SELECT * FROM foo JOIN (SELECT bar.c FROM bar JOIN baz ON TRUE) AS sub ON
foo.a = sub.c;

=>

SELECT * FROM foo JOIN (bar JOIN baz ON TRUE) ON foo.a = bar.c;
```




```
SELECT * FROM foo JOIN (SELECT bar.c FROM bar JOIN
baz ON TRUE) AS sub ON foo.a = sub.c;
```




```
SELECT * FROM foo JOIN (bar JOIN baz ON TRUE) ON
foo.a = bar.c;
```

为什么提升子查询

- 通过把子查询提升到父查询之中，就可以使子查询参与整个计划搜索空间，从而找到更好的执行计划
- 否则，我们不得不为子查询单独做计划，然后在为父查询做计划时把子查询当做是一个“黑盒子”

消除外连接

```
# select * from student;
 sid | sname  | sage
-----+--------+------
   1 | Tom    |   18
   2 | Robert |   16
(2 rows)
```

```
# select * from enrolled;
 sid | cid | score
-----+-----+-------
   1 | 100 |    60
(1 row)
```

```
# select * from student inner join enrolled on student.sid = enrolled.sid;
 sid | sname | sage | sid | cid | score
-----+-------+------+-----+-----+-------
   1 | Tom   |   18 |   1 | 100 |    60
(1 row)
```

```
# select * from student left join enrolled on student.sid = enrolled.sid;
 sid | sname  | sage | sid | cid | score
-----+--------+------+-----+-----+-------
   1 | Tom    |   18 |   1 | 100 |    60
   2 | Robert |   16 |     |     |
(2 rows)
```

```
# select * from student left join enrolled on student.sid = enrolled.sid
where enrolled.sid is not null;
 sid | sname | sage | sid | cid | score
-----+-------+------+-----+-----+-------
   1 | Tom   |   18 |   1 | 100 |    60
(1 row)
```

消除外连接

- 外连接的上层有“严格”的约束条件，且该约束条件限定了来自 nullable side的某一变量为非NULL值，则外连接可以转换成内连接

```
SELECT ... FROM foo LEFT JOIN bar ON (...) WHERE bar.d = 42;

=>

SELECT ... FROM foo INNER JOIN bar ON (...) WHERE bar.d = 42;
```

消除外连接

GREENPLUM DATABASE®

```
# select * from student;
 sid | sname  | sage
-----+--------+------
   1 | Tom    |   18
   2 | Robert |   16
(2 rows)
```

```
# select * from enrolled;
 sid | cid | score
-----+-----+-------
   1 | 100 |    60
(1 row)
```

```
# select * from student left join enrolled on student.sid = enrolled.sid;
 sid | sname  | sage | sid | cid | score
-----+--------+------+-----+-----+-------
   1 | Tom    |   18 |   1 | 100 |    60
   2 | Robert |   16 |     |     |
(2 rows)
```

```
# select * from student left join enrolled on student.sid = enrolled.sid
where enrolled.sid is null;
 sid | sname  | sage | sid | cid | score
-----+--------+------+-----+-----+-------
   2 | Robert |   16 |     |     |
(1 row)
```

```
# explain (costs off) select * from student left join enrolled on
student.sid = enrolled.sid where enrolled.sid is null;
                 QUERY PLAN
---------------------------------------------
 Nested Loop Anti Join
   Join Filter: (student.sid = enrolled.sid)
   ->  Seq Scan on student
   ->  Materialize
         ->  Seq Scan on enrolled
(5 rows)
```

消除外连接

- 外连接本身有“严格”的连接条件，且该连接条件引用了来自 nullable side的某一变量，且该变量被上层的约束条件限定为NULL值，则外连接可以转换成反半连接(Anti Join)

```
SELECT * FROM foo LEFT JOIN bar ON foo.a = bar.c WHERE bar.c IS NULL;

=>

SELECT * FROM foo *ANTI JOIN* bar on foo.a = bar.c;
```

查询树的预处理(后期)

- 分发WHERE和JOIN/ON约束条件
- 收集关于连接顺序限制的信息
- 消除无用连接
- etc.

分发约束条件

GREENPLUM DATABASE®

- 一般来说，我们期望可以尽可能的下推 约束条件
- 如果只有内连接，我们可以把一个约束条件下推到它的“自然语义”位置
- 如果存在外连接，那么约束条件的下推可能会受到阻碍，从而无法下推到它的 “自然语义”位置
- 对于被外连接阻碍的约束条件，我们通过让它的“required_relids”包含进外连接所需要的所有基表，从而避免 该约束条件被下推到外 连接之下

被外连接阻碍的约束条件(1/2)

```
# explain (costs off)
# select * from student left join enrolled on student.sid =
enrolled.sid and student.sage = 18;
                              QUERY PLAN
-----------------------------------------------------------------
-------
 Nested Loop Left Join
   Join Filter: ((student.sage = 18) AND (student.sid =
enrolled.sid))
   ->  Seq Scan on student
   ->  Materialize
         ->  Seq Scan on enrolled
(5 rows)


# select * from student left join enrolled on student.sid =
enrolled.sid and student.sage = 18;
 sid | sname  | sage | sid | cid | score
-----+--------+------+-----+-----+-------
   1 | Tom    |   18 |   1 | 100 |    60
   2 | Robert |   16 |     |     |
(2 rows)
```

```
# explain (costs off)
# select * from (select * from student where sage = 18) sub left
join enrolled on sub.sid = enrolled.sid;
                 QUERY PLAN
---------------------------------------------
 Nested Loop Left Join
   Join Filter: (student.sid = enrolled.sid)
   ->  Seq Scan on student
         Filter: (sage = 18)
   ->  Materialize
         ->  Seq Scan on enrolled
(6 rows)


# select * from (select * from student where sage = 18) sub left
join enrolled on sub.sid = enrolled.sid;
 sid | sname | sage | sid | cid | score
-----+-------+------+-----+-----+-------
   1 | Tom   |   18 |   1 | 100 |    60
(1 row)
```

被外连接阻碍的约束条件(1/2)

```
# explain (costs off)
# select * from student left join enrolled on student.sid =
enrolled.sid and student.sage = 18;
                              QUERY PLAN
----------------------------------------------------------------
-------
 Nested Loop Left Join
   Join Filter: ((student.sage = 18) AND (student.sid =
enrolled.sid))
   ->  Seq Scan on student
   ->  Materialize
         ->  Seq Scan on enrolled
(5 rows)
```

```
# select * from student left join enrolled on student.sid =
enrolled.sid and student.sage = 18;
 sid | sname  | sage | sid | cid | score
-----+--------+------+-----+-----+-------
   1 | Tom    |   18 |   1 | 100 |    60
   2 | Robert |   16 |     |     |
(2 rows)
```

```
# explain (costs off)
# select * from (select * from student where sage = 18) sub left
join enrolled on sub.sid = enrolled.sid;
                      QUERY PLAN
---------------------------------------------
 Nested Loop Left Join
   Join Filter: (student.sid = enrolled.sid)
   ->  Seq Scan on student
         Filter: (sage = 18)
   ->  Materialize
         ->  Seq Scan on enrolled
(6 rows)
```

```
# select * from (select * from student where sage = 18) sub left
join enrolled on sub.sid = enrolled.sid;
 sid | sname | sage | sid | cid | score
-----+-------+------+-----+-----+-------
   1 | Tom   |   18 |   1 | 100 |    60
(1 row)
```

如果外连接本身的连接条件引用了non-nullable side的表，那么该连接条件不能下推到外连接之下，否则我们可能会丢失一些null-extended元组

被外连接阻碍的约束条件(2/2)

GREENPLUM DATABASE

```
# explain (costs off)
# select * from student left join enrolled on student.sid =
enrolled.sid where coalesce(enrolled.cid, 1) = 100;
                  QUERY PLAN
-------------------------------------------------
 Nested Loop Left Join
   Join Filter: (student.sid = enrolled.sid)
   Filter: (COALESCE(enrolled.cid, 1) = 100)
   ->  Seq Scan on student
   ->  Materialize
         ->  Seq Scan on enrolled
(6 rows)
```

```
# select * from student left join enrolled on student.sid =
enrolled.sid where coalesce(enrolled.cid, 1) = 100;
 sid | sname | sage | sid | cid | score
-----+-------+------+-----+-----+-------
   1 | Tom   |   18 |   1 | 100 |    60
(1 row)
```

```
# explain (costs off)
# select * from student left join (select * from enrolled where
coalesce(cid, 1) = 100) sub on student.sid = sub.sid;
                    QUERY PLAN
----------------------------------------------------
 Nested Loop Left Join
   Join Filter: (student.sid = enrolled.sid)
   ->  Seq Scan on student
   ->  Materialize
         ->  Seq Scan on enrolled
               Filter: (COALESCE(cid, 1) = 100)
(6 rows)
```

```
# select * from student left join (select * from enrolled where
coalesce(cid, 1) = 100) sub on student.sid = sub.sid;
 sid | sname  | sage | sid | cid | score
-----+--------+------+-----+-----+-------
   1 | Tom    |   18 |   1 | 100 |    60
   2 | Robert |   16 |     |     |
(2 rows)
```

被外连接阻碍的约束条件(2/2)

GREENPLUM DATABASE®

```
# explain (costs off)
# select * from student left join enrolled on student.sid =
enrolled.sid where coalesce(enrolled.cid, 1) = 100;
                  QUERY PLAN
---------------------------------------------
 Nested Loop Left Join
   Join Filter: (student.sid = enrolled.sid)
   Filter: (COALESCE(enrolled.cid, 1) = 100)
   ->  Seq Scan on student
   ->  Materialize
         ->  Seq Scan on enrolled
(6 rows)
```

```
# select * from student left join enrolled on student.sid =
enrolled.sid where coalesce(enrolled.cid, 1) = 100;
 sid | sname | sage | sid | cid | score
-----+-------+------+-----+-----+-------
   1 | Tom   |   18 |   1 | 100 |    60
(1 row)
```

```
# explain (costs off)
# select * from student left join (select * from enrolled where
coalesce(cid, 1) = 100) sub on student.sid = sub.sid;
                   QUERY PLAN
------------------------------------------------
 Nested Loop Left Join
   Join Filter: (student.sid = enrolled.sid)
   ->  Seq Scan on student
   ->  Materialize
         ->  Seq Scan on enrolled
               Filter: (COALESCE(cid, 1) = 100)
(6 rows)
```

```
# select * from student left join (select * from enrolled where
coalesce(cid, 1) = 100) sub on student.sid = sub.sid;
 sid | sname  | sage | sid | cid | score
-----+--------+------+-----+-----+-------
   1 | Tom    |   18 |   1 | 100 |    60
   2 | Robert |   16 |     |     |
(2 rows)
```

如果外连接上层的约束条件引用了nullable side的变量，
那么该约束条件不能下推到外连接之下，否则可能会多出
一下null-extended元组

连接顺序限制

GREENPLUM DATABASE®

- 外连接会在一定程度上限制 连接顺序的交换
- 非FULL-JOIN可以和一个外连接的左端(LHS)自由结合
- 通常非FULL-JOIN不可以和外连接的右端(RHS)结合

```
(A leftjoin B on (Pab)) innerjoin C on (Pac)
= (A innerjoin C on (Pac)) leftjoin B on (Pab)
```

```
# explain (costs off)
# select * from (a left join b on true) inner join c on
a.i = c.i;
          QUERY PLAN
---------------------------------
 Nested Loop Left Join
   ->  Hash Join
         Hash Cond: (a.i = c.i)
         ->  Seq Scan on a
         ->  Hash
               ->  Seq Scan on c
   ->  Materialize
         ->  Seq Scan on b
(8 rows)
```

```
(A leftjoin B on (Pab)) innerjoin C on (Pbc)
!= A leftjoin (B innerjoin C on (Pbc)) on (Pab)
```

```
# explain (costs off)
# select * from (a left join b on true) inner join c on
coalesce(b.i, 1) = c.i;
              QUERY PLAN
---------------------------------------
 Hash Join
   Hash Cond: (COALESCE(b.i, 1) = c.i)
   ->  Nested Loop Left Join
         ->  Seq Scan on a
         ->  Materialize
               ->  Seq Scan on b
   ->  Hash
         ->  Seq Scan on c
(8 rows)
```

消除无用连接

```
# explain (costs off) select student.sid from student left join
(select distinct sid as sid from enrolled) sub on student.sid =
sub.sid;
     QUERY PLAN
---------------------
 Seq Scan on student
(1 row)
```

- 必须是左连接，且内表是基表
- 内表的列没有在 该连接之上使用
- 连接条件最多只可能匹配内表中的一个元 组

扫描/连接优化

- 主要处理查询语句中FROM和WHERE部分
- 同时也会考虑到ORDER BY信息

```
# EXPLAIN (COSTS OFF) SELECT * FROM foo JOIN bar ON foo.a = bar.c AND foo.b = bar.d ORDER BY b, a;
                     QUERY PLAN
-----------------------------------------------------
 Merge Join
   Merge Cond: ((foo.b = bar.d) AND (foo.a = bar.c))
   ->  Sort
         Sort Key: foo.b, foo.a
         ->  Seq Scan on foo
   ->  Sort
         Sort Key: bar.d, bar.c
         ->  Seq Scan on bar
(8 rows)
```

- 由代价来驱动

扫描/连接优化

- 首先为基表确定扫描路径，估计扫描路径的代价和大小
- 利用动态规划算法，搜索整个连接顺序空间，生成连接路径
- 在搜索连接顺序空间时，需要考虑到由外连接带来的连接顺序的限制

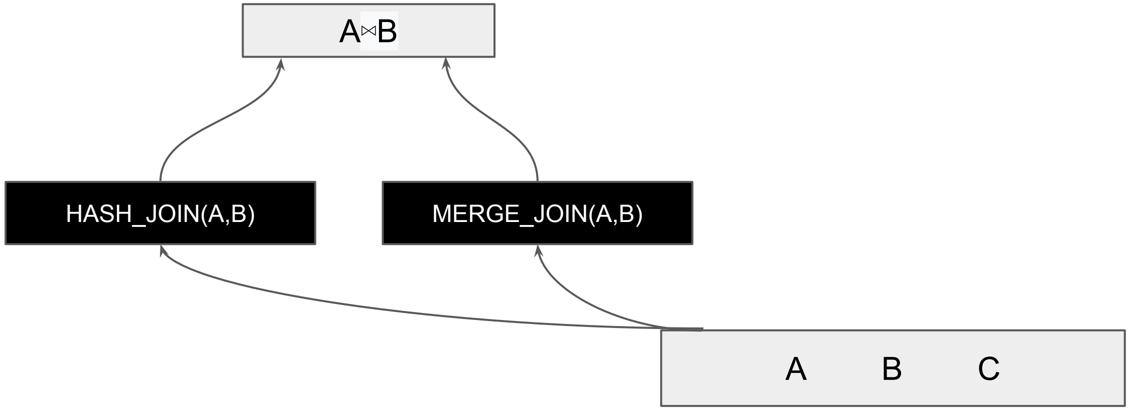
动态规划

- 为每一个基表生成 扫描路径
- 为所有可能的两个表的 连接生成连接路径
- 为所有可能的三个表的 连接生成连接路径
- 为所有可能的四个表的 连接生成连接路径
- …
- 直到所有基表都 连接在了一起

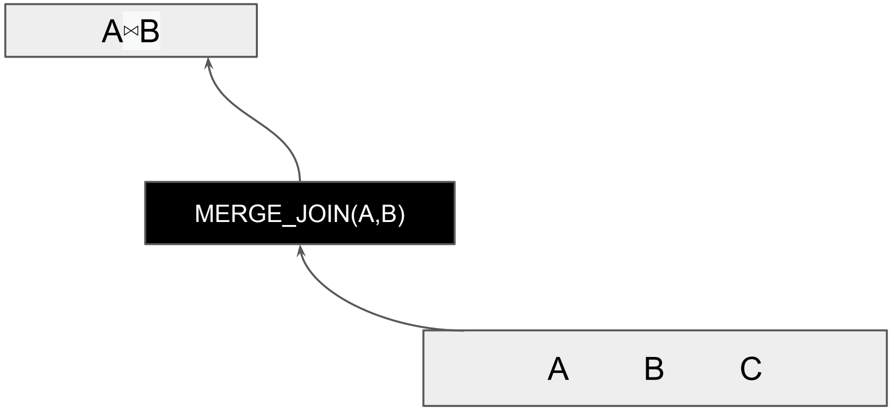
```
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;
```

A B C

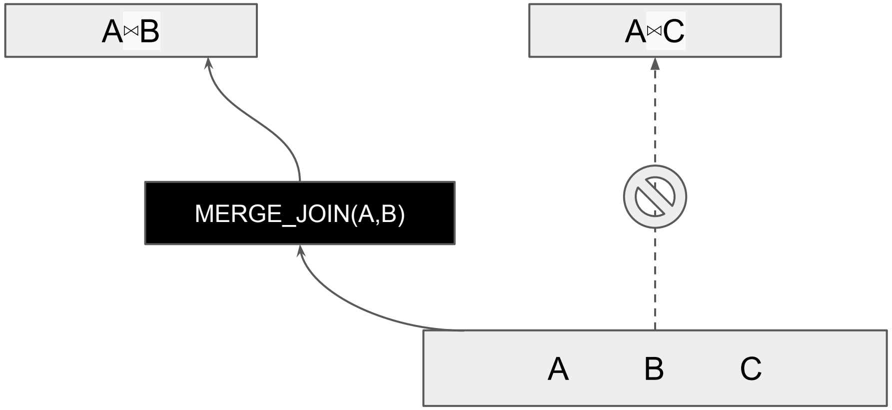
```
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;
```



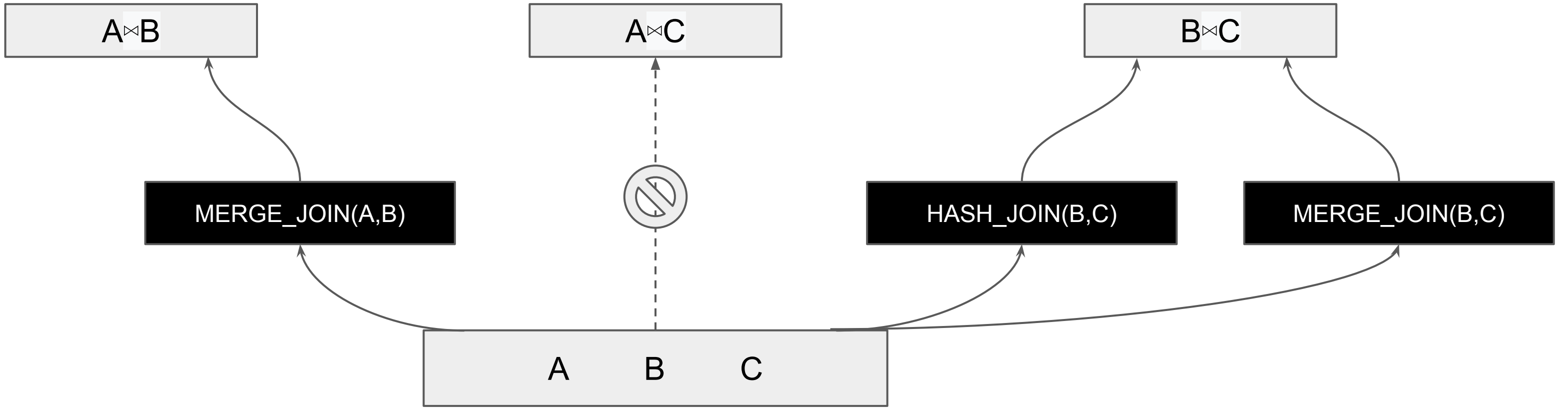
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;



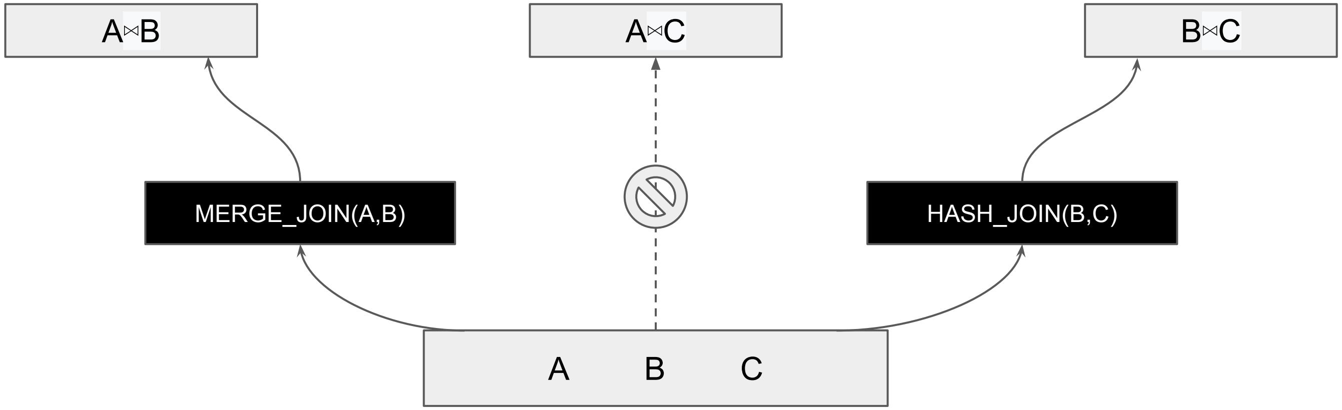
```
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;
```



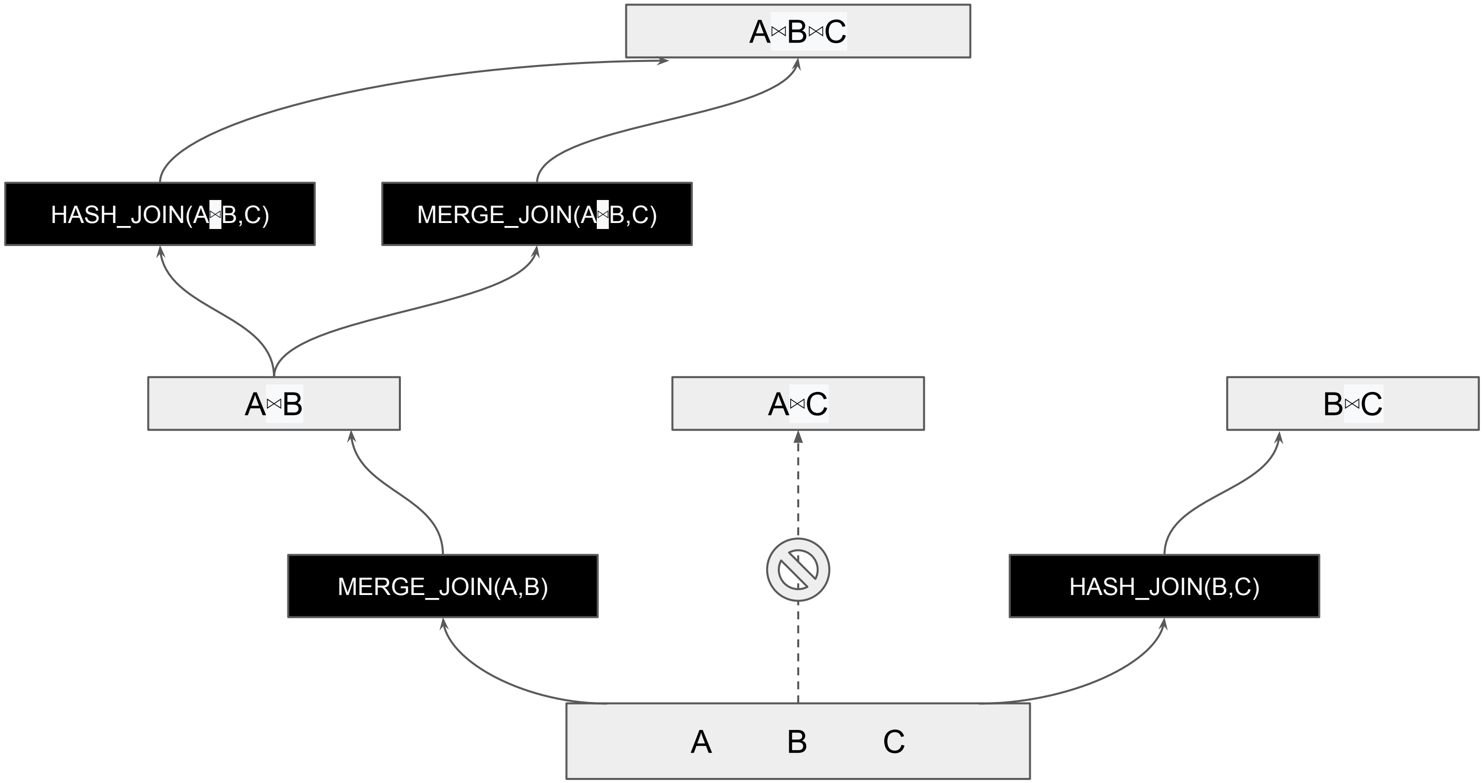
```
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;
```



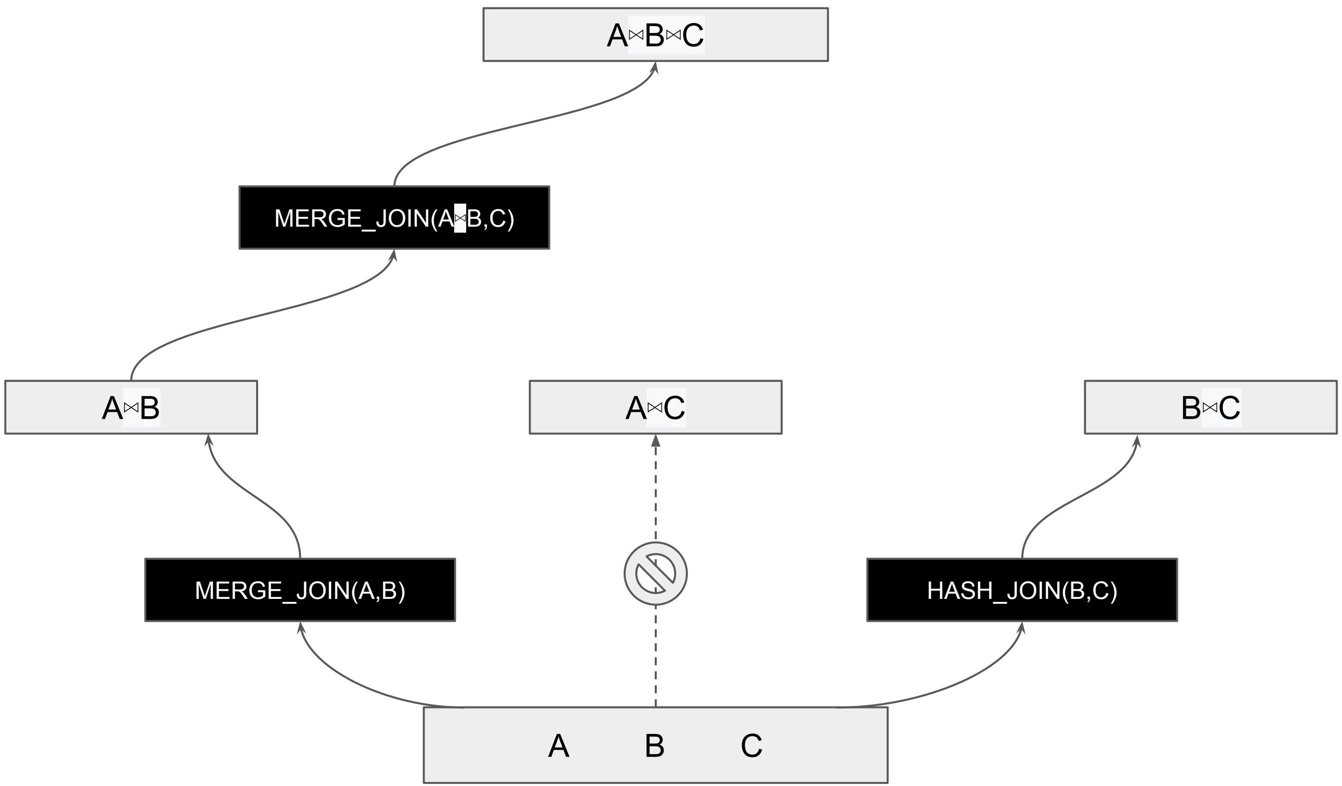
```
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;
```



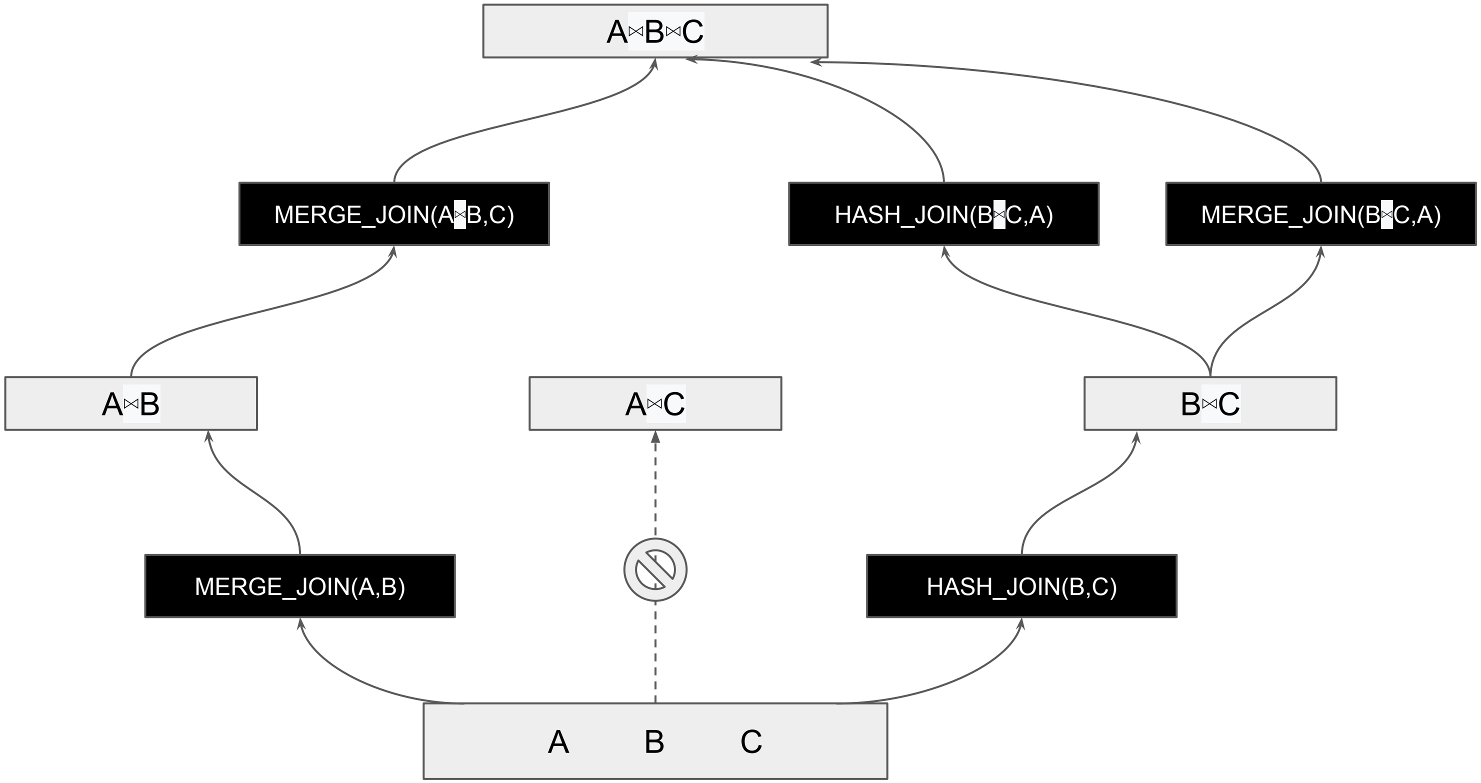
```
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;
```



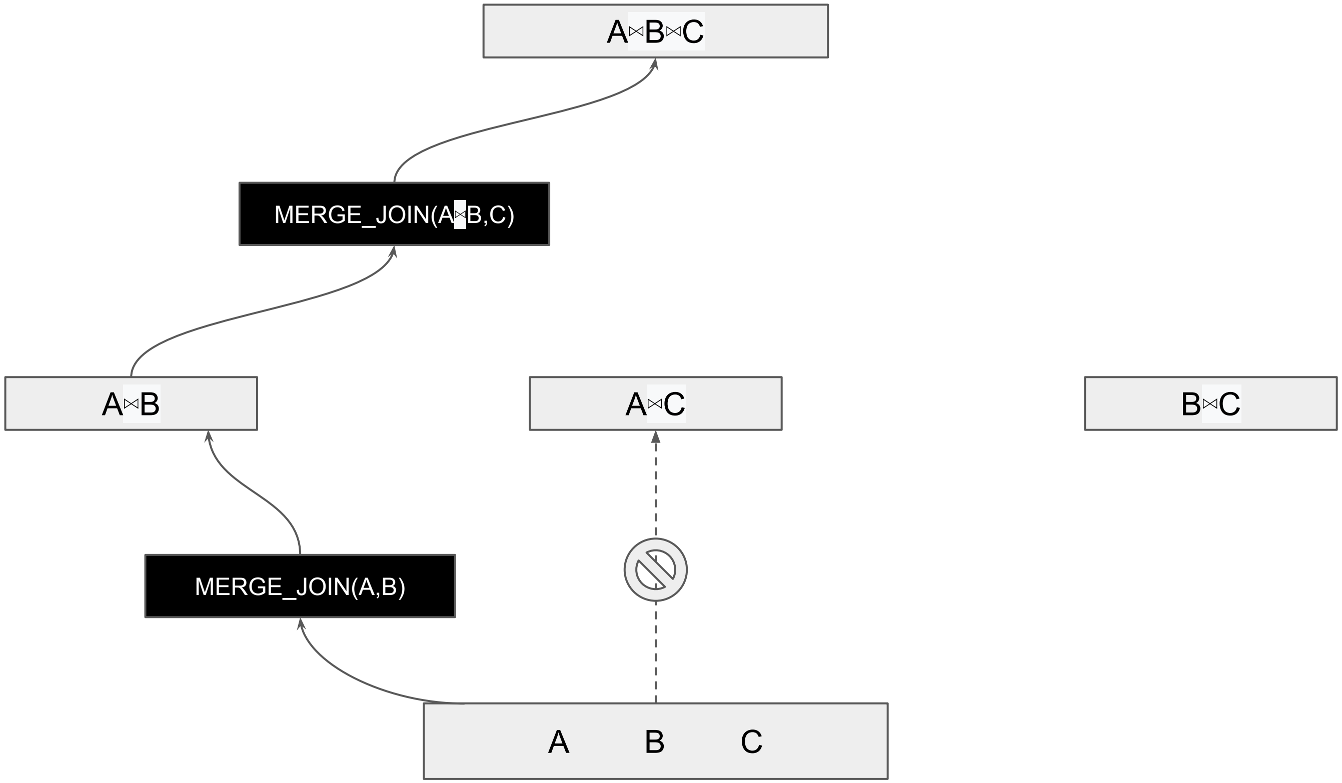
```
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;
```



```
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;
```

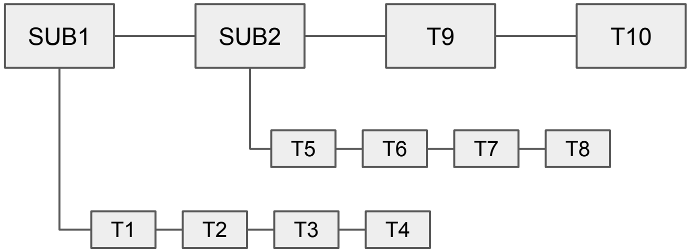


```
SELECT * FROM A JOIN (B JOIN C ON B.j = C.j) ON A.i = B.i;
```



- n个表的连接，理论上有 n! 个不同的连接顺序
- 遍历所有可能的连接顺序是不可行的
- 我们使用一些启发式办法，减少搜索空间
    - 对于不存在连接条件的两个表，尽量不做 连接
    - 把一个大的问题，分解成多个子 问题

```
SELECT * FROM
    (SELECT * FROM T1, T2, T3, T4) SUB1 JOIN
    (SELECT * FROM T5, T6, T7, T8) SUB2 ON TRUE JOIN
    (SELECT * FROM T9, T10) SUB3 ON TRUE;

SET join_collapse_limit TO 4;
```



扫描/连接之外的优化

- 处理GROUP BY, aggregation, window functions, DISTINCT
- 处理集合操作，UNION/INTERSECT/EXCEPT
- 如果ORDER BY需要，添加最后的SORT节点
- 添加LockRows, Limit, ModifyTable节点

计划树的后处理

- 把代价最小的路径 转换成计划树
- 调整计划树中的一些细节
    - 展平子查询的范围表
    - 把上层计划节点中的变量变成OUTER_VAR或是INNER_VAR的形式，来指向子计划的输出
    - 删除不必要的SubqueryScan, Append等节点

Thank You

GREENPLUM DATABASE®

```
Output: Thank You
Gather Motion 3:1  (slice1; segments: 3)
   ->  Index Scan using Common_phrases_idx on Common_phrases
         Index Cond: (value = ‘Thank You’)
         Filter: (Language = ‘English’)
```

Q&A