

数据库查询优化技术

The Query Optimize Engine Of PostgreSQL(9.2.3)

LiHaiXiang

1

What is the technology Of Query Optimization ?

2

What is the Query Optimization Technology Of PostgreSQL ?

3

How to implement the Query Optimization Of PostgreSQL ?

The Technology Of Query Optimization

1. Query reuse

- ① The query result reuse
- ② The query plan reuse

2. The Rule Of Query Rewrite

- ① Based on Relational Algebra and Heuristic Rule
- ② View Rewrite、Sub-query Optimization、Equivalent Predicate Rewrite、Condition Simplification、Outer Join Elimination、Join Elimination、Nest Join Elimination
- ③ Sematic Optimization

3. The Algorithm Of Query Optimization

- ① Single Table Scan Algorithm
- ② Two Table Join Algorithm
- ③ Multi-table Join Algorithm

4. Parallel Query Optimization

5. Distribute Query Optimization

The Query Optimization Technology Of PostgreSQL

1. Query reuse -----PostgreSQL 9.2.3 Not Support

- ① The query result reuse
- ② The query plan reuse

2. The Rule Of Query Rewrite

- ① Based on Relational Algebra and Heuristic Rule
- ② View Rewrite、Sub-query Optimization、Equivalent Predicate Rewrite、Condition Simplification、Outer Join Elimination、Join Elimination、Nest Join Elimination
- ③ Sematic Optimization

3. The Algorithm Of Query Optimization

- ① Single Table Scan Algorithm
- ② Two Table Join Algorithm
- ③ Multi-table Join Algorithm

4. Parallel Query Optimization

5. Distribute Query Optimization

How to implement the Query Optimization Of PostgreSQL ?

Core question 1 :

**How to implement the
Query Optimization of PostgreSQL ?**

3

How to implement the Query Optimization Of PostgreSQL ?

3.1

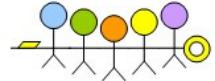
Query Optimizer Frame

3.2

The Sub-query Optimization Technology

3.3

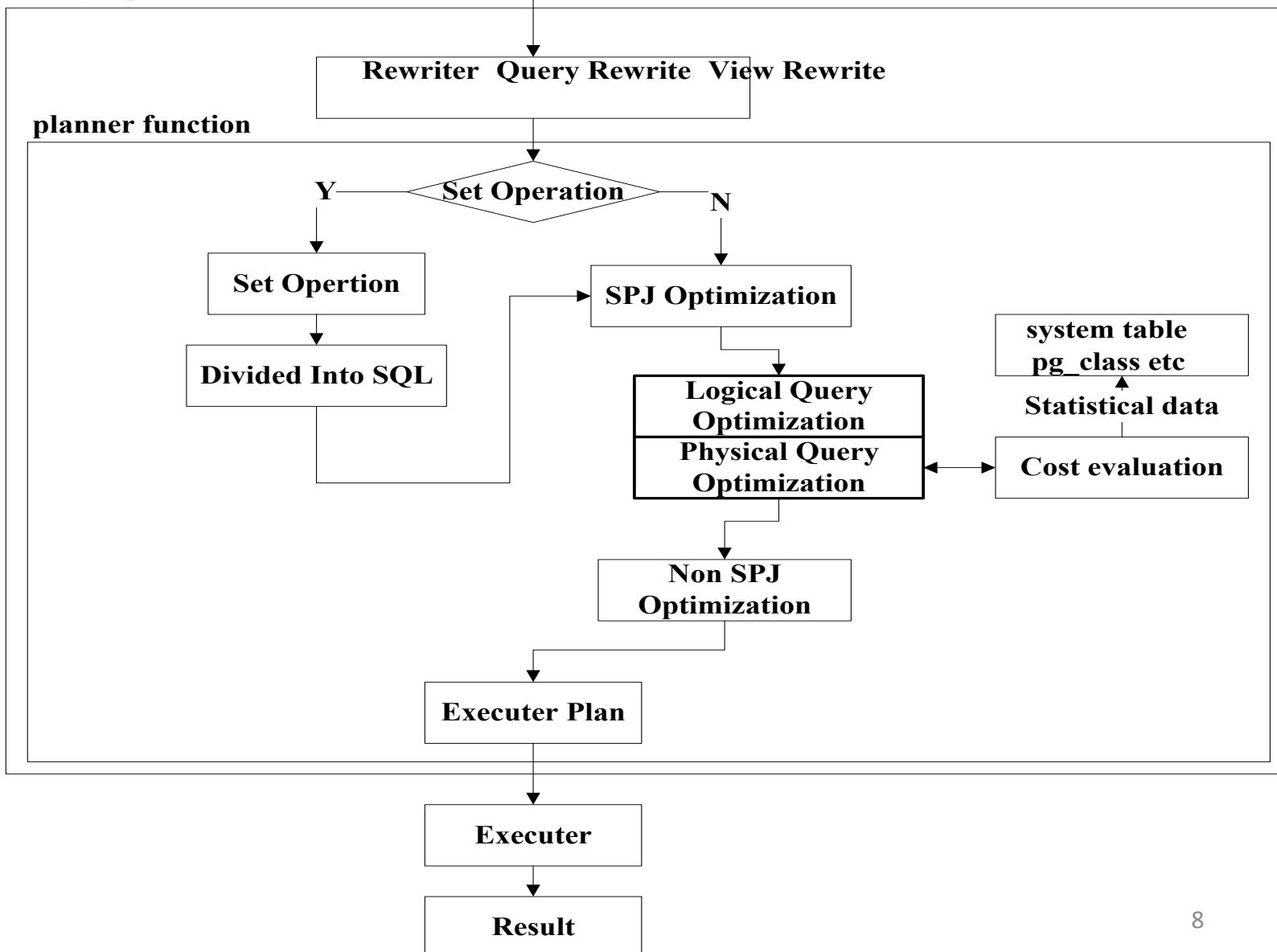
Other Query Optimization Technology

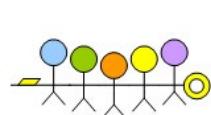


3.1

Query Optimizer Frame

Query Optimizer





3.1

Query Optimizer Main Function Process

第

第

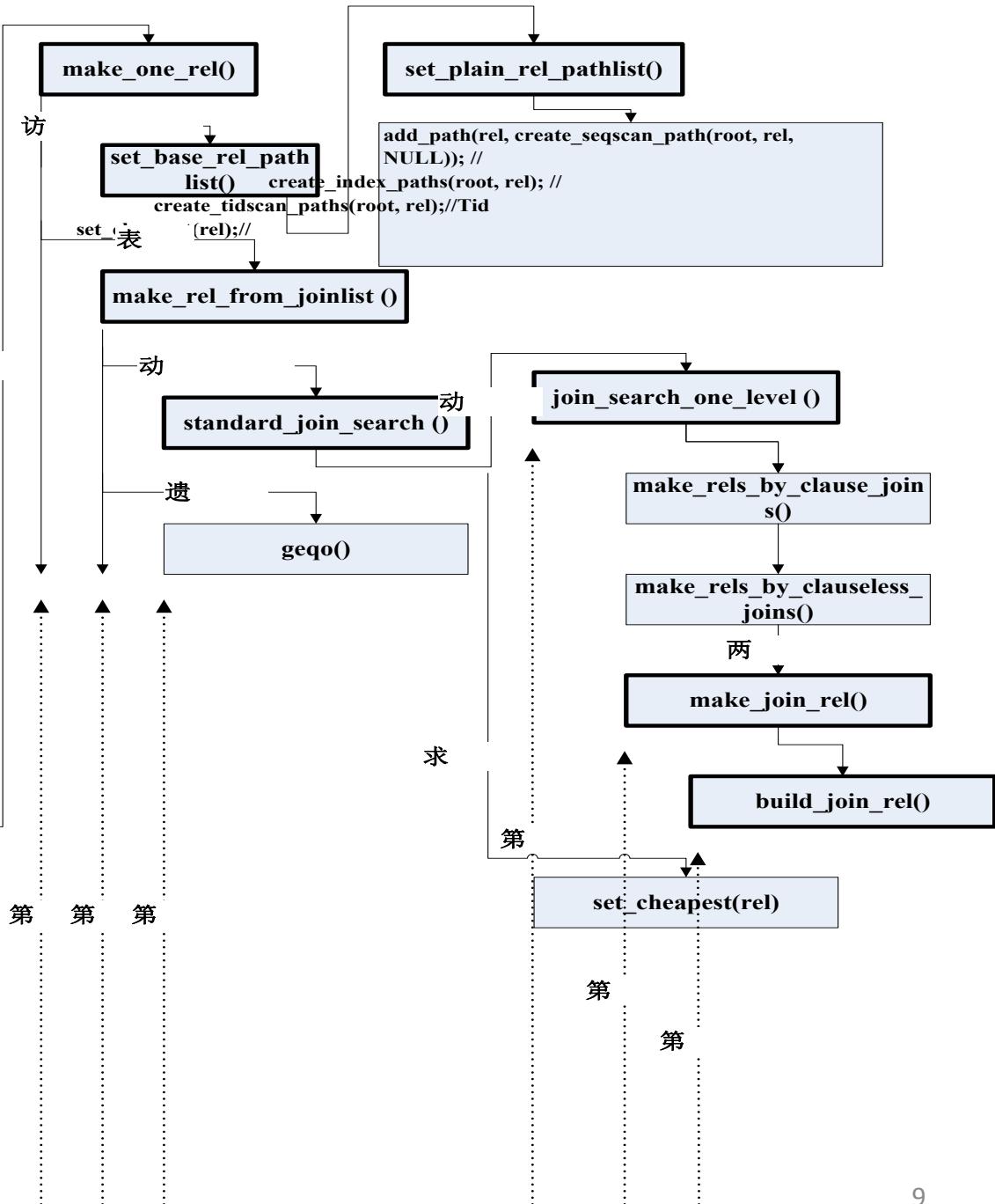
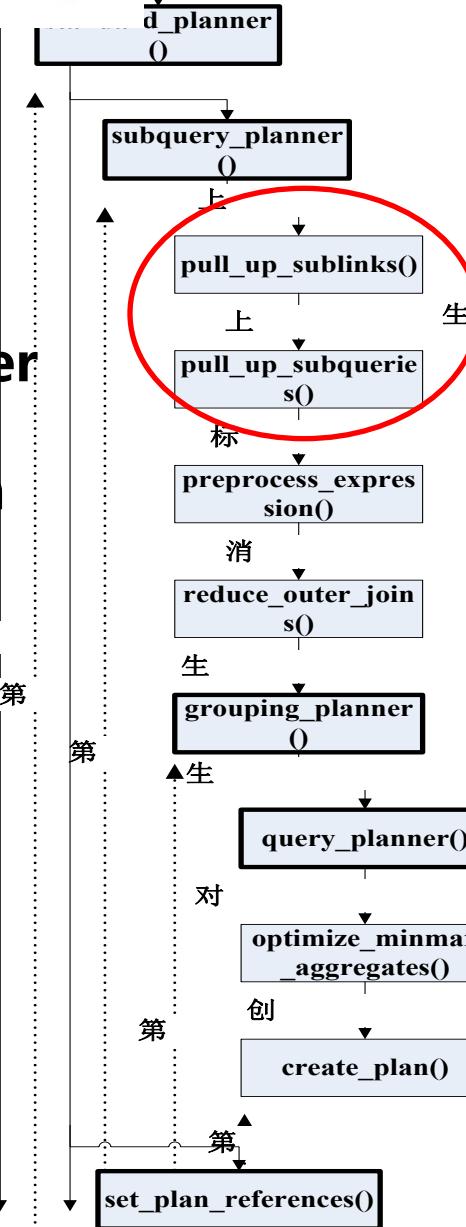
第

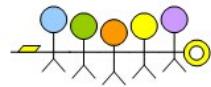
生

对

第

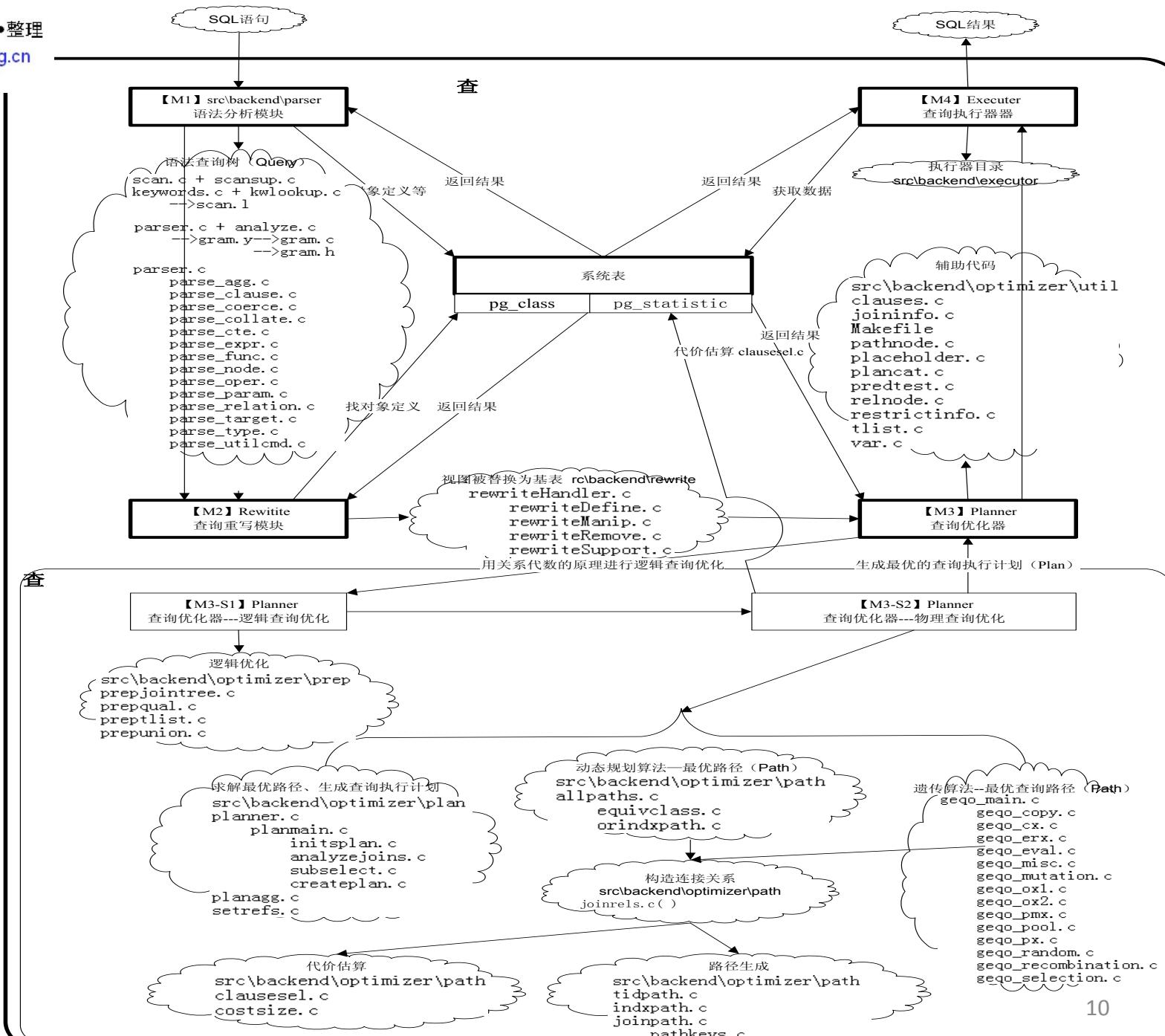
第





3.1

Query Optimizer Main File



How to implement the Subquery Optimization Of PostgreSQL ?

Core question 2 :

**How to implement the Sub-Query
Optimization of PostgreSQL ?**

How to implement the Sub-query Optimization Of PostgreSQL ?

Sub-Query : Subquery is an approach that provides the capability of embedding the first query into the other

PostgreSQL includes :

3.2.1 Sublink

3.2.2 Sub-query

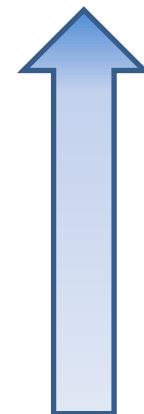
Sublink + Sub-query

==

Sub-query

PostgreSQL

Database



3.2 Sub-query Optimization Technology

```
01: Plan * //返回值是查询执行计划
02: subquery_planner(PlannerGlobal *glob, Query *parse, /*parse, 传入查询树*/
03:                      PlannerInfo *parent_root,
04:                      bool hasRecursion, double tuple_fraction,
05:                      PlannerInfo **subroot)
06: {
07:     .....
08:     /* 上拉WHERE和JOIN/ON中的ANY和EXISTS类型的子链接，作为半连接或反半连接操作 */
09:     if (parse->hasSubLinks)
10:         pull_up_sublinks(root);
11:
12:     .....
13:
14:     /*上拉子查询*/
15:     parse->jointree = (FromExpr *)
16:         pull_up_subqueries(root, (Node *) parse->jointree, NULL, NULL);
17:
18:     if (parse->setOperations)
19:         flatten_simple_union_all(root); //简化集合操作"UNION ALL"
20:
21: } ? end subquery planner ?
```

3.2.1

What is sublink ?

```
/*
 * SubLink
 *
 * A SubLink represents a subselect appearing in an expression, and in some
 * cases also the combining operator(s) just above it.      The subLinkType
 * indicates the form of the expression represented:
 *   * EXISTS_SUBLINK      EXISTS(SELECT ...)
 *   * ALL_SUBLINK          (lefthand) op ALL (SELECT ...)
 *   * ANY_SUBLINK          (lefthand) op ANY (SELECT ...)
 *   * ROWCOMPARE_SUBLINK  (lefthand) op (SELECT ...)
 *   * EXPR_SUBLINK         (SELECT with single targetlist item ...)
 *   * ARRAY_SUBLINK        ARRAY(SELECT with single targetlist item ...)
 *   * CTE_SUBLINK          WITH query (never actually part of an expression)
 *
 * .....
 */
```

数据库查询优化技术

3.2.1 How to optimize the Sublink Of PostgreSQL ?

```
01: pull_up_sublinks_qual_recurse(PlannerInfo *root, Node *node, Node **jtlink1,  
02:                                     Relids available_rels1, Node **jtlink2, Relids available_rels2)  
03: {  
04:     if (node == NULL)  
05:         return NULL;  
06:     if (IsA(node, SubLink))  
07:     {  
08:         if (sublink->subLinkType == ANY_SUBLINK) //上拉ANY_SUBLINK类型子链接  
09:         {  
10:             //处理方式同上，只是本处代码处理的是available_rels2对应的项  
11:             if (available_rels2 != NULL &&  
12:                 (j = convert_ANY_sublink_to_join(root, sublink,  
13:                                             available_rels2)) != NULL)  
14:             {  
15:             }  
16:         else if (sublink->subLinkType == EXISTS_SUBLINK) //上拉EXISTS_SUBLINK类型子链接  
17:         {  
18:             /* 处理逻辑基本等同"if (sublink->subLinkType == ANY_SUBLINK)"代码段 */  
19:             if ((j = convert_EXISTS_sublink_to_join(root, sublink, false,  
20:                                             available_rels1)) != NULL)  
21:             {  
22:             }  
23:         /* Else return it unmodified */  
24:         return node;  
25:     }  
26:     ....  
27:     return node;  
28: } end pull_up_sublinks_qual_recurse ?
```

数据库查询优化技术

(1)

convert_ANY_sublink_to_join

```
01: convert_ANY_sublink_to_join(PlannerInfo *root, SubLink *sublink, Relids available_rels)
02: {
03:     JoinExpr *result; // 把子链接转换为一个连接关系
04:     ...
05:     /* 1. 子链接操作符的右操作数：子链接不能包含上层中出现的任何var结构体表示的对象 */
06:     if (contain_vars_of_level((Node *) subselect, 1))
07:         return NULL;
08:
09:     /* 2. 子链接操作符的左操作数：
10:        a) 一定与上层中出现的"Var"结构体表示的对象有相同者：如果没有，则子链接与上层没有任何关联，可以直接求解，不用合并到上层作为一个关系对象和其他关系对象做连接操作
11:        b) 不能引用上层中出现的"关系"
12:        c) 不可包括易失函数 */
13:     upper_varnos = pull_varnos(sublink->testexpr); // 情况2.a
14:     if (bms_is_empty(upper_varnos))
15:         return NULL;
16:     if (!bms_is_subset(upper_varnos, available_rels)) // 情况2.b
17:         return NULL;
18:     if (contain_volatile_functions(sublink->testexpr)) // 情况2.c
19:         return NULL;
20:
21:
22:     /* 上拉子链接（右操作数subselect）到上层的范围表（RangeTblEntry）中，作为未来连接的对象 */
23:     rte = addRangeTableEntryForSubquery(NULL, subselect, makeAlias("ANY_subquery", NIL), false);
24:     ...
25:     /* 转换子链接中左右操作数为上拉后的连接关系（JoinExpr *result;）的连接条件 */
26:     quals = convert_testexpr(root, sublink->testexpr, subquery_vars);
27:     ...
28:     result = makeNode(JoinExpr); // result == JoinExpr
29:     result->jointype = JOIN_SEMI; /* 子链接转换后，是以半连接关系与上层其他关系进行连接 */
30:     ...
31:     return result;
```

数据库查询优化技术

(2)

convert_EXISTS_sublink_to_join

```
01: convert_EXISTS_sublink_to_join (PlannerInfo *root, SubLink *sublink,
02:                                bool under_not, Relids available_rels)
03: {
04:     /* 1. EXISTS类型子链接操作符的右操作数 (EXISTS子链接不存在左操作数) 的优化:
05:      a) 不支持带有WITH子句的格式
06:      b) 不支持带有集合操作、或带有聚集函数、CTE、HAVING、LIMIT/OFFSET等子句格式
07:      c) 不支持FROM或WHERE子句为空的优化 */
08:     if (subselect->cteList) //情况1.a
09:         return NULL;
10:     if (!simplify_EXISTS_query(subselect)) //情况1.b
11:         return NULL;
12:     if (subselect->jointree->fromlist == NIL) //情况1.c
13:         return NULL;
14:     ...
15:     /* 右操作数的subselect不能包含上层中出现的任何"Var"结构体表示的对象，这点同ANY
16:      子链接询。注意subselect是处理过的 (subselect->jointree->quals = NULL;) */
17:     if (contain_vars_of_level((Node *) subselect, 1))
18:         return NULL;
19:     /* WHERE子句需要存在上层中包含的"Var"，这样才可能构成连接 */
20:     if (!contain_vars_of_level(whereClause, 1))
21:         return NULL;
22:     //带有易失函数不优化
23:     if (contain_volatile_functions(whereClause))
24:         return NULL;
25:     ...
26:     /* 上拉子链接到顶层的范围表 (parse->rtable)。从技术上看，所谓上拉就是把子查询中的
27:      范围表合并到顶层的FROM子句中，把WHERE条件合并 */
28:     parse->rtable = list_concat(parse->rtable, subselect->rtable);
29:     /* EXISTS转为半连接，NOT EXISTS转为反半连接 */
30:     result->jointype = under_not ? JOIN_ANTI : JOIN_SEMI;
31:     ...
32:     return result;
33: } ? end convert EXISTS sublink to join ?
```

3.2.1

Conclusion of Sublink

1 ANY = SOME

src/backend/parser/gram.y :

```
sub_type: ANY { $$ = ANY_SUBLINK; }
               | SOME { $$ = ANY_SUBLINK; }
               ALL { $$ = ALL_SUBLINK; }
```

2 ANY → semi join

3 EXISTS → semi join

NOT EXISTS → anti-semi join

3.2.1

Conclusion of Sublink

Question:

- 1 Whether do PostgreSQL support IN Sub-query ?**
- 2 How to optimize In Sub-query in PostgreSQL ?**

数据库查询优化技术

3.2.1

How to optimize In Sub-query ?

In this file : src/backend/parser/gram.y

```
01: | a_expr IN_P in_expr
02: {
03:     if (IsA($3, SubLink))
04:     {
05:         /* generate foo = ANY (subquery) */
06:         SubLink *n = (SubLink *) $3;
07:         n->subLinkType = ANY_SUBLINK;
08:         ....
09:     }
10: else
11: {
12:     /* generate scalar IN expression */
13:     $s = (Node *) makeSimpleA_Expr(AEXPR_IN, "=",
14: );
15: }
16: | a_expr NOT IN_P in_expr
17: {
18:     if (IsA($4, SubLink))
19:     {
20:         /* Make an = ANY node */
21:         SubLink *n = (SubLink *) $4;
22:         n->subLinkType = ANY_SUBLINK;
23:         ....
24:     }
25: else
26: {
27:     /* generate scalar NOT IN expression */
28:     $s = (Node *) makeSimpleA_Expr(AEXPR_IN, "<>",
29: );
30: }
```

3.2.1

Conclusion of Sublink

1 ANY = SOME

src/backend/parser/gram.y :

```
sub_type:    ANY   { $$ = ANY_SUBLINK; }
              | SOME   { $$ = ANY_SUBLINK; }
              ALL   { $$ = ALL_SUBLINK; }
```

2 ANY → semi join IN ←→ =ANY
 IN → semi join

3 EXISTS → semi join

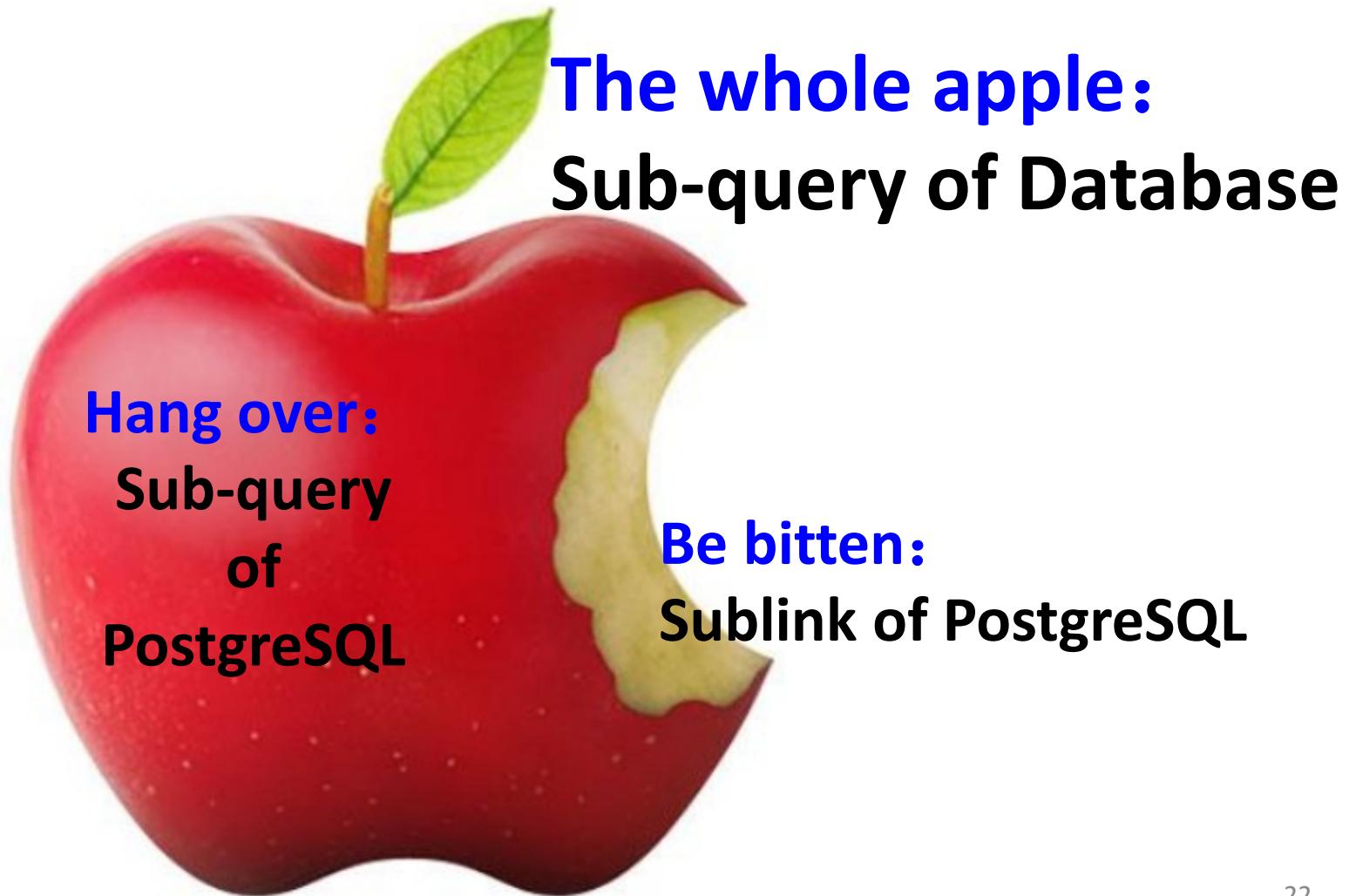
NOT EXISTS → anti-semi join

4 other sublink type:

.....

3.2.2

What is sub-query ?



数据库查询优化技术

3.2.2

How to optimize the Sub-query Of PostgreSQL ?

```
01: pull_up_subqueries(PlannerInfo *root, Node *jtnode,
02:                      JoinExpr *lowest_outer_join, AppendRelInfo *containing_appendrel)
03: {
04:     if (IsA(jtnode, RangeTblRef)) //处理RangeTblRef中的子查询
05:     {
06:         if (rte->rtekind == RTE_SUBQUERY &&
07:             is_simple_subquery(rte->subquery) && //如果是简单子查询，则上拉
08:             !rte->security_barrier && //没有数据安全的限制
09:             (containing_appendrel == NULL ||
10:              is_safe_append_member(rte->subquery)))
11:         return pull_up_simple_subquery(root, jtnode, rte, lowest_outer_join, conta
12:         .....
13:         if (rte->rtekind == RTE_SUBQUERY &&
14:             is_simple_union_all(rte->subquery)) //如果是简单的UNION操作，上拉
15:             return pull_up_simple_union_all(root, jtnode, rte);
16:     }
17:     else if (IsA(jtnode, FromExpr))
18:     {
19:         /* 递归处理From-list中的每个元素,如能上拉则依据上面的条件上拉*/
20:         foreach(l, f->fromlist)
21:             lfirst(l) = pull_up_subqueries(root, lfirst(l), lowest_outer_join, NULL);
22:     }
23:     else if (IsA(jtnode, JoinExpr))
24:     {
25:         /*递归处理Join-list中的每个元素,如能上拉则依据上面的条件上拉*/
26:         //对连接的左右子树分别调用pull_up_subqueries
27:     }
28:     else
29:         elog(ERROR, "unrecognized node type: %d", (int) nodeTag(jtnode));
30:     return jtnode;
31: } end pull_up_subqueries ?
```

3.2.2 Condition Of Sub-query Optimization

```
01: is_simple_subquery (Query *subquery)  
02: { .....  
03:     if (subquery->setOperations)  
04:         return false;  
05:  
06:     if (subquery->hasAggs ||  
07:         subquery->hasWindowFuncs ||  
08:         subquery->groupClause ||  
09:         subquery->havingQual ||  
10:         subquery->sortClause ||  
11:         subquery->distinctClause ||  
12:         subquery->limitOffset ||  
13:         subquery->limitCount ||  
14:         subquery->hasForUpdate ||  
15:         subquery->cteList)  
16:         return false;  
17:     if (expression_returns_set ((Node *) subquery->targetList) )  
18:         return false;  
19:     if (contain_volatile_functions ((Node *) subquery->targetList) )  
20:         return false;  
21:     if (subquery->jointree->fromlist == NIL)  
22:         return false;  
23:  
24:     return true;  
25: } ? end is simple subquery ?
```

3.2.2 Condition Of Sub-query Optimization

```
01: is_simple_union_all(Query *subquery)
02: {
03:     /* Let's just make sure it's a valid subselect ... */
04:     if (!IsA(subquery, Query) ||
05:         subquery->commandType != CMD_SELECT ||
06:         subquery->utilityStmt != NULL)
07:         elog(ERROR, "subquery is bogus");
08:
09:     /* Is it a set-operation query at all? */
10:     topop = (SetOperationStmt *) subquery->setOperations;
11:     if (!topop)
12:         return false;
13:
14:     /* can't handle ORDER BY, LIMIT/OFFSET, locking, or WITH */
15:     if (subquery->sortClause ||
16:         subquery->limitOffset ||
17:         subquery->limitCount ||
18:         subquery->rowMarks ||
19:         subquery->cteList)
20:         return false;
21:
22:     /* Recursively check the tree of set operations */
23:     return is_simple_union_all_recurse((Node *) topop, subquery,
24:                                         topop->colTypes);
25: } ? end is_simple_union_all ?
```

3.2.2

Method Of Sub-query Optimization

```
01: pull_up_simple_subquery(PlannerInfo *root, Node *jtnode, RangeTblEntry
02:                               JoinExpr *lowest_outer_join,
03:                               AppendRelInfo *containing_appendrel)
04: {
05:     .....
06:     /*
07:      * Adjust level-0 varnos in subquery so that we can append its rangetable
08:      * to upper query's. We have to fix the subquery's append_rel_list as
09:      * well.
10:     */
11:     rtoffset = list_length(parse->rtable);
12:     OffsetVarNodes((Node *) subquery, rtoffset, 0);
13:     OffsetVarNodes((Node *) subroot->append_rel_list, rtoffset, 0);
14:     .....
15:     parse->targetList = (List *)
16:         pullup_replace_vars((Node *) parse->targetList, &rvcontext);
17:     parse->returningList = (List *)
18:         pullup_replace_vars((Node *) parse->returningList, &rvcontext);
19:     .....
20:     /*
21:      * Now append the adjusted rtable entries to upper query. (We hold off
22:      * until after fixing the upper rtable entries; no point in running that
23:      * code on the subquery ones too.)
24:     */
25:     parse->rtable = list_concat(parse->rtable, subquery->rtable);
26:     .....
27:     return (Node *) subquery->jointree;
28: } ? end pull_up_simple_subquery ?
```

3.2.2

Conclusion of Sub-query

1 What kind of sub-query can be optimized (pull up)?

- 1.1 RangerTable
- 1.2 From clause
- 1.3 Join clause

2 What is the condition that sub-query can be optimized (pull up)?

a sub-query not include GROUP/ORDER/LIMIT/DISTINCT etc

3 How to optimize (pull up) subquery?

SELECT * FROM A, (SELECT * FROM B WHERE B.b<1) WHERE A.a>1

subquery's FROM --> parent's FROM

subquery's WHERE

parent's WHERE

SELECT * FROM A, B WHERE A.a>1 AND B.b<1

数据库查询优化技术

3.2.3

Example

```
CREATE TABLE t1 (id1 INT, a1 INT UNIQUE, b1 INT, PRIMARY KEY(id1));
```

```
CREATE TABLE t2 (id2 INT, a2 INT UNIQUE, b2 INT, , PRIMARY KEY(id2));
```

Statement 1:

```
EXPLAIN SELECT * FROM t1 WHERE id1 IN (SELECT id2 FROM t2);
```

Statement 2:

```
EXPLAIN SELECT * FROM t1 WHERE id1 IN (SELECT a2 FROM t2);
```

Statement 3:

```
EXPLAIN SELECT * FROM t1 WHERE id1 IN (SELECT b2 FROM t2);
```

Statement 4:

```
EXPLAIN SELECT * FROM t1 WHERE b1 N (SELECT id2 FROM t2);
```

Statement 5:

```
EXPLAIN SELECT * FROM t1 WHERE b1 N (SELECT a2 FROM t2);
```

Statement 6:

```
EXPLAIN SELECT * FROM t1 WHERE b1 N (SELECT b2 FROM t2);
```

3.2.3

Example

Statement 1 AND Statement 2 (Plan is similar but not identical) :

QUERY PLAN

Hash Semi Join (cost=85.17..181.71 rows=3030 width=12)

Hash Cond: (t1.id1 = t2.a2)

-> Seq Scan on t1 (cost=0.00..47.30 rows=3030 width=12)

-> Hash (cost=47.30..47.30 rows=3030 width=4)

-> Seq Scan on t2 (cost=0.00..47.30 rows=3030 width=4)

(5 行记录)

Statement	Left	Right
1	Primary Key	Primary Key
2	Primary Key	Unique Key
3	Primary Key	Common col

Statement 3:

QUERY PLAN

Hash Join (cost=77.38..145.94 rows=1000 width=12)

Hash Cond: (t1.id1 = t2.b2)

-> Seq Scan on t1 (cost=0.00..47.30 rows=3030 width=12)

-> Hash (cost=64.88..64.88 rows=1000 width=4)

-> HashAggregate (cost=54.88..64.88 rows=1000 width=4)

-> Seq Scan on t2 (cost=0.00..47.30 rows=3030 width=4)

(6 行记录)

3.2.3

Example

Statement 4 AND Statement 5 (Plan is similar but not identical) :

QUERY PLAN

Hash Semi Join (cost=85.17..181.31 rows=3000 width=12)

Hash Cond: (t1.b1 = t2.a2)

-> Seq Scan on t1 (cost=0.00..47.30 rows=3030 width=12)

-> Hash (cost=47.30..47.30 rows=3030 width=4)

 -> Seq Scan on t2 (cost=0.00..47.30 rows=3030 width=4)

(5 行记录)

Statement 6:

QUERY PLAN

Hash Join (cost=77.38..165.74 rows=3000 width=12)

Hash Cond: (t1.b1 = t2.b2)

-> Seq Scan on t1 (cost=0.00..47.30 rows=3030 width=12)

-> Hash (cost=64.88..64.88 rows=1000 width=4)

 -> HashAggregate (cost=54.88..64.88 rows=1000 width=4)

 -> Seq Scan on t2 (cost=0.00..47.30 rows=3030 width=4)

(6 行记录)

Statement	Left	Right
4	Common col	Primary Key
5	Common col	Unique Key
6	Common col	Common col

3.3

Please think about

- 1 Why can not IN sub-query statement 3 and statement 6 be converted into semi-join ?**
- 2 What does equivalent predicate rewrite include ?**
- 3 Why can outer join be optimized ?**
- 4 How to implement multi-table join optimization in PostgreSQL ?**
- 5 What is difference between PostgreSQL and MySQL query optimizer ?**
- 6**

?

Any Question