DTCC

2016中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2016

数 据 定 义 未 来

2016年5月12日-14日 | 北京国际会议中心

SequeMedia 盛拓传媒 | ITT168.com | ChinaUnix | ITPUB

数据库内核专场

2016年5月13日

揭开SQL优化的盖头来
——数据库查询优化器实现技术解密

李海翔 @那海蓝蓝

PostgreSQL, MySQL, Greenplum, Informix, etc

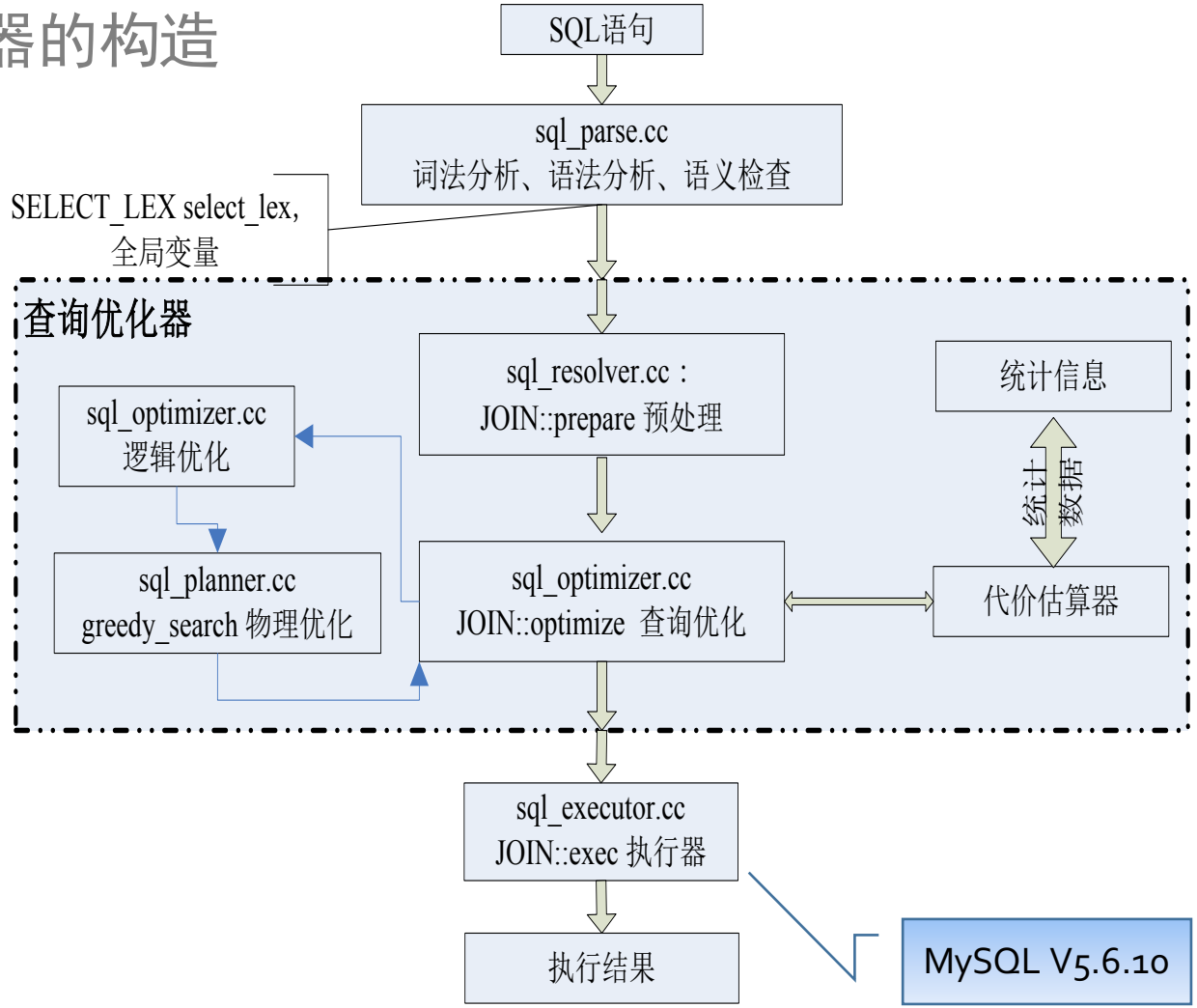
@那海蓝蓝 Blog : http://blog.163.com/li_hx/

《数据库查询优化器的艺术: 原理解析与SQL性能优化》

揭开SQL优化的盖头来--数据库查询优化器实现技术解密

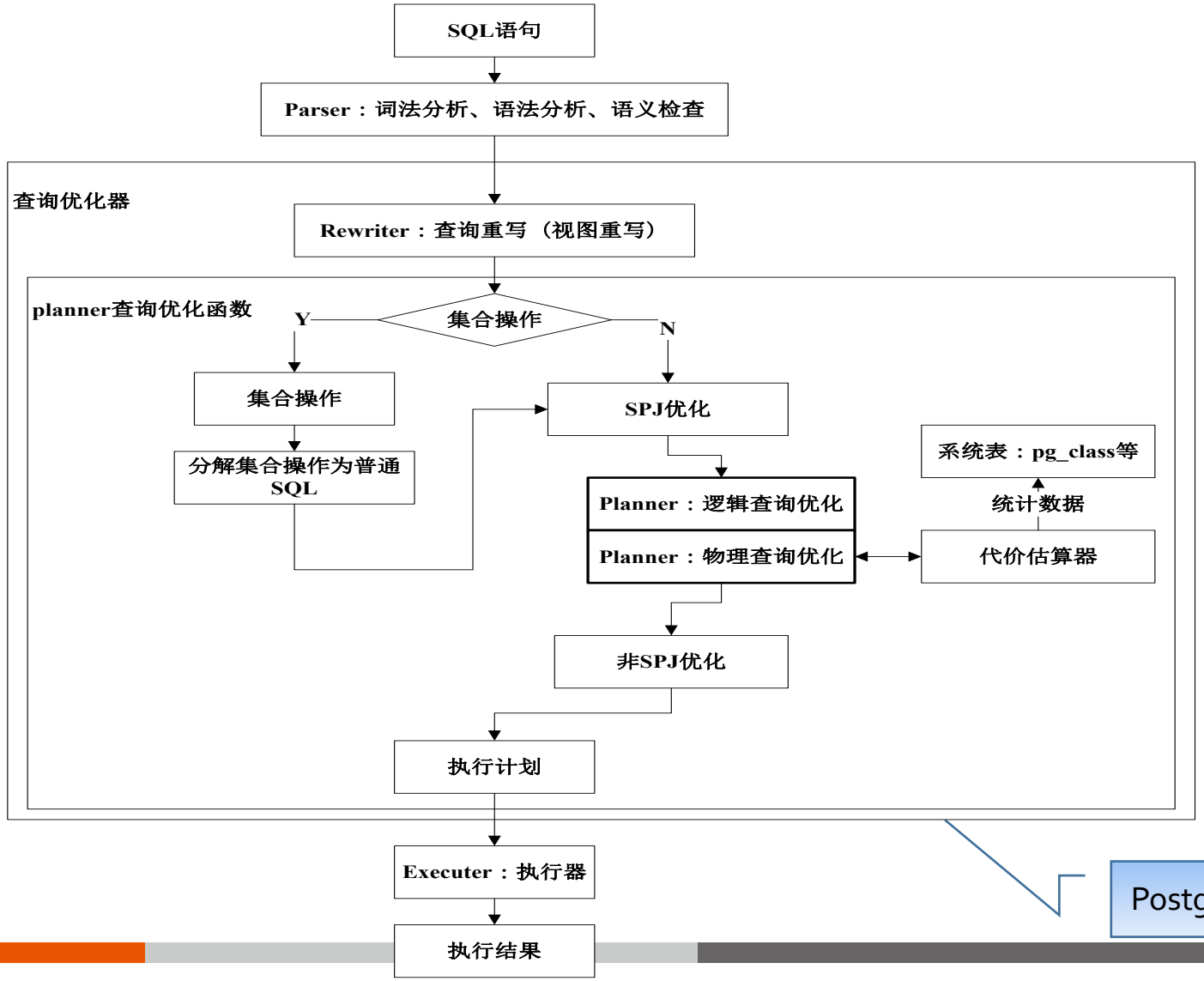
- **谁知我心——查询优化器的构造**
- 分析阶段——从拆卸到重新组装
- 优化计算——点点滴滴从小事做起
- 谁与争锋——主流数据库对比
  ---Oracle、PostgreSQL、MySQL、Informix

1 谁知我心—查询优化器的构造

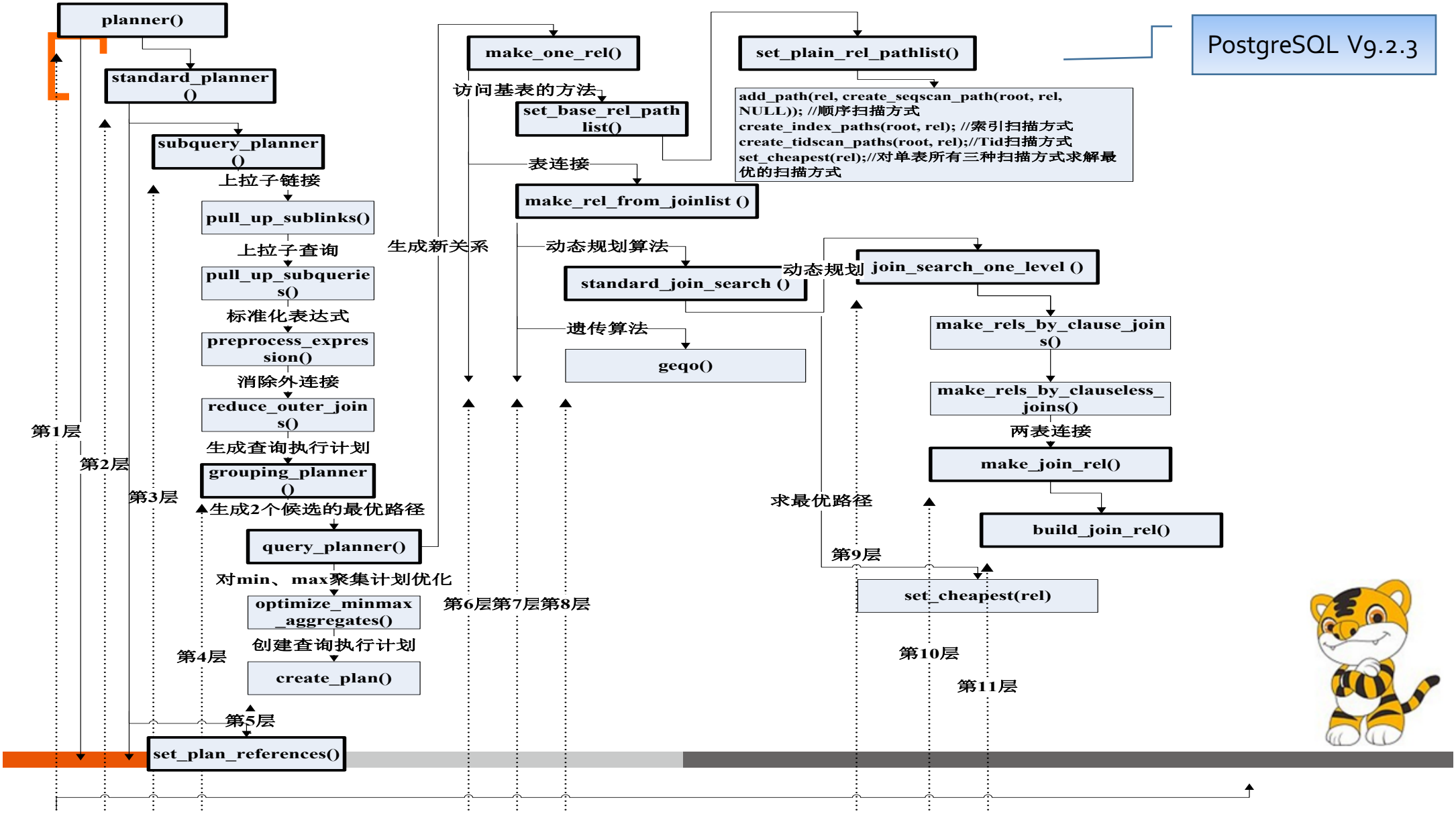


```
handle_select(){//第1层
    mysql_union(); //处理union操作
    mysql_select(){//第2层
        JOIN::prepare() {//第3层
            remove_redundant_subquery_clauses();//去除子查询中的冗余子句
            resolve_subquery(); //子查询优化
        }
        JOIN::optimize() {//第3层
            flatten_subqueries(); //把子查询转换为半连接操作，只支持IN格式的子查询转为半连接
            simplify_joins(); //消除外连接、消除嵌套连接
            optimize_cond(..., conds, ...); //优化WHRER子句
            optimize_cond(..., having, ...); //优化HAVING子句
            opt_sum_query(); //优化count(*), min() and max(), 适用于没有GROUPBY子句的情况
             make_join_statistics(); //确定多表的连接路径：单表是多表的特例
             {//第4层
                  update_ref_and_keys(); //获取索引信息，为快速定位数据、条件比较做准备
                  get_quick_record_count(); /*估算每个表中有多少元组可用（被循环调用计算多个表，包括了范围查询的代价计算） */
                  choose_plan()//多表连接以便得到最优的查询计划
                  {//第5层
                       //挑选二种多表连接的方式之一做最优的多表连接以便得到最优的查询计划
                       optimize_straight_join(); //方式一：用户指定表的连接次序
                       greedy_search(); //方式二：多表连接，贪婪算法
                       {//第6层
                           best_extension_by_limited_search(); //确定多表连接的最优查询执行计划
                           {//第7层
                               best_access_path()//第8层，估算指定表的最优访问路径（也包括花费等）
                           }//第7层结束
                       }//第6层结束
                  }//第5层结束
             }//第4层结束
             ......
             make_outerjoin_info(); //填充外连接的信息
             ......
             substitute_for_best_equal_field(); /* 循环遍历所有表达式，化简表达式（重复的等式能去掉则直接去掉，如：WHERE a=5 AND ((a=b AND b=c) OR c>4) 的条件将变为：“=(a) and (=(5, a, b, c) or c>4)”）*/
             ......
             set_access_methods(); //设置每个表的访问方式
             ......
             make_join_select();  /* 用于执行各种不同情况的join查询。该函数通过join时，连接表的不同搜索方式（唯一索引查找、ref查找、快速范围查找、合并索引查找、全表扫描等不同方式），进行join操作的处理 */

             //优化DISTINCT谓词相关的情况，如下多行代码，处理不同的DISTINCT情况
             ......
             //创建临时表
             //处理简单的IN子查询
        }//第3层结束，optimize()
        JOIN::exec() {//第3层
           do_select(); //执行连接，输出结果到客户端
        }
    }//第2层结束，mysql_select()
}//第1层结束，handle_select()
```



PostgreSQL V9.2.3


PostgreSQL V9.2.3
planner()
standard_planner()
subquery_planner()
上拉子链接
pull_up_sublinks()
上拉子查询
pull_up_subqueries()
标准化表达式
preprocess_expression()
消除外连接
reduce_outer_joins()
生成查询执行计划
grouping_planner()
生成2个候选的最优路径
query_planner()
对min、max聚集计划优化
optimize_minmax_aggregates()
创建查询执行计划
create_plan()
set_plan_references()
第1层
第2层
第3层
第4层
第5层
make_one_rel()
访问基表的方法
set_base_rel_pathlist()
set_plain_rel_pathlist()
add_path(rel, create_seqscan_path(root, rel, NULL)); //顺序扫描方式
create_index_paths(root, rel); //索引扫描方式
create_tidscan_paths(root, rel);//Tid扫描方式
set_cheapest(rel);//对单表所有三种扫描方式求解最优的扫描方式
表连接
make_rel_from_joinlist ()
生成新关系
动态规划算法
standard_join_search ()
遗传算法
geqo()
第6层第7层第8层
动态规划
join_search_one_level ()
make_rels_by_clause_joins()
make_rels_by_clauseless_joins()
两表连接
make_join_rel()
build_join_rel()
求最优路径
第9层
set_cheapest(rel)
第10层
第11层

1 谁知我心：内核开发心得分享

从广义的角度把握优化器

- 狭义优化器包括：逻辑优化、物理优化
- 广义优化器包括：分析、优化（逻辑优化、物理优化）、执行三个阶段

优化器的本质是实现了SQL语义的编译器

- 优化器的技术，基于编译器技术
- 优化器的技术，需要处理复杂的SQL语义（关系代数定义的语义）

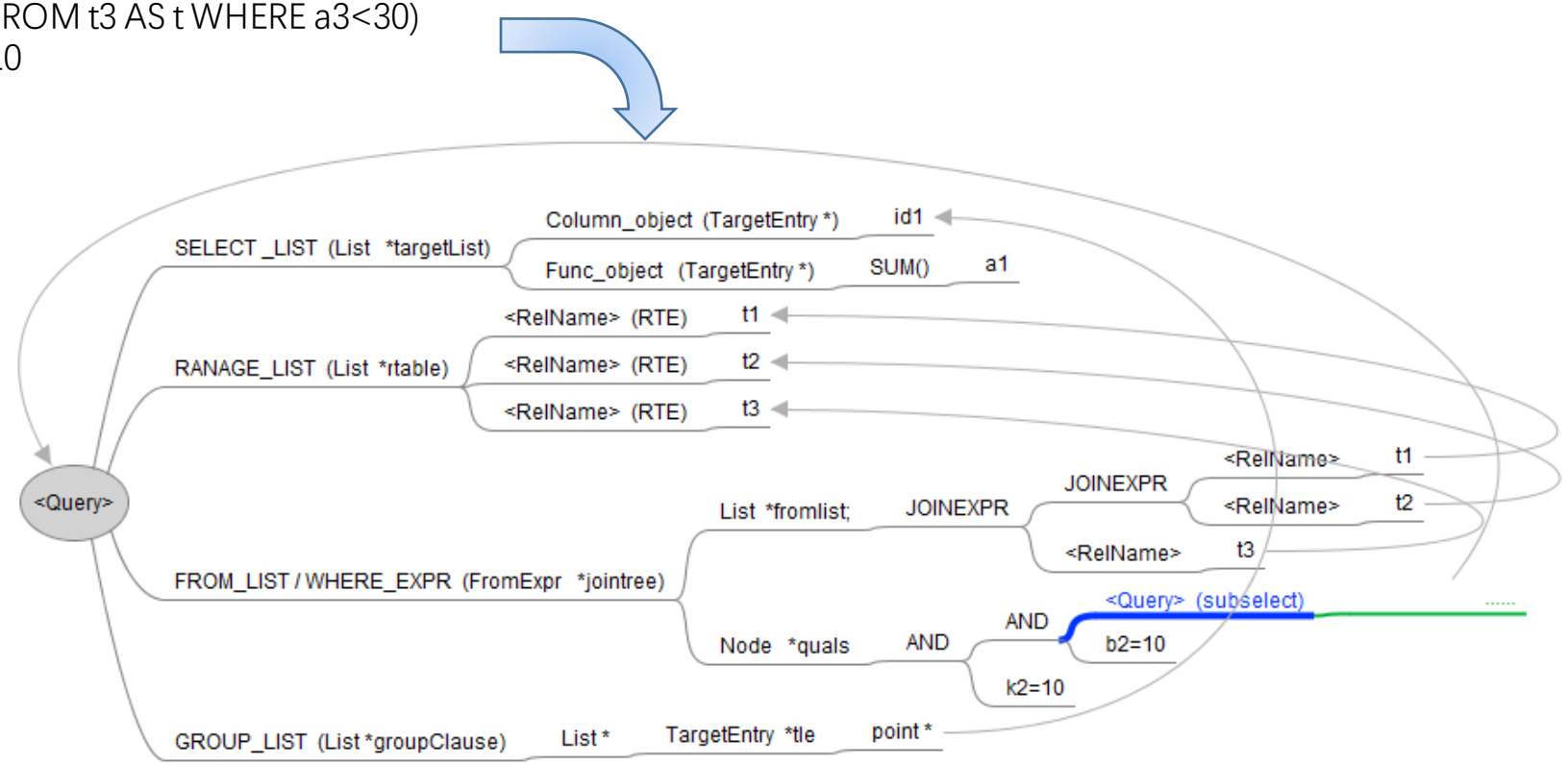
揭开SQL优化的盖头来--数据库查询优化器实现技术解密

- 谁知我心——查询优化器的构造
- **分析阶段——从拆卸到重新组装**
- 优化计算——点点滴滴从小事做起
- 谁与争锋——主流数据库对比
  ---Oracle、PostgreSQL、MySQL、Informix

2 分析阶段，从拆卸到重新组装

```
SELECT SUM(a1), id1
FROM t1 LEFT JOIN t2 on a1=a2 LEFT JOIN t3 on b2=b3
WHERE k1 IN (SELECT k3 FROM t3 AS t WHERE a3<30)
      AND b2=10 AND k2=10
GROUP BY id1;
```

从SQL语句 到 语法树（ PG）



2 分析阶段，从拆卸到重新组装

```
%token FROM  JOIN  SELECT  WHERE  GROUP …
```

```
SELECT SUM(a1), id1
FROM t1 LEFT JOIN t2 on a1=a2 LEFT JOIN t3 on b2=b3
WHERE k1 IN (SELECT k3 FROM t3 AS t WHERE a3<30)
    AND b2=10 AND k2=10
GROUP BY id1;
```

词法分析，SQL 被拆分为 token

DFA，确定有限状态自动机

```
opt_target_list: target_list { $$ = $1; }
   | /* EMPTY */ { $$ = NIL; }
   ;

target_list:
   target_el { $$ = list_make1($1); }
   | target_list ',' target_el { $$ = lappend($1, $3); }
   ;

target_el: a_expr AS ColLabel
   {
      $$ = makeNode(ResTarget);
…
   }
   | a_expr IDENT …
```

```
where_clause:
   WHERE a_expr { $$ = $2; }
   | /*EMPTY*/  { $$ = NULL; }
   ;
```

```
a_expr: c_expr { $$ = $1; }
     | a_expr TYPECAST Typename
     { $$ = makeTypeCast($1, $3, @2); }
...
c_expr: columnref { $$ = $1; }
     | '(' a_expr ')' opt_indirection

...
columnref: ColId
     {
          $$ = makeColumnRef($1, NIL, @1, yyscanner);
     }
     | ColId indirection
     {
          $$ = makeColumnRef($1, $2, @1, yyscanner);
     }
     ;
```

确定有限状态自动机：http://baike.baidu.com/link?url=lcdMGdKMOnbIlMYer02TQdw_UcUHxXYdC9o__w2_vULlotDpFoR7ism2Tsf_Bck0dBarXxKqfDhyR-qHtTrd__

2 分析阶段，从拆卸到重新组装

语法分析，token等变为一棵语法树

语义分析，让语法树变得“有含义”

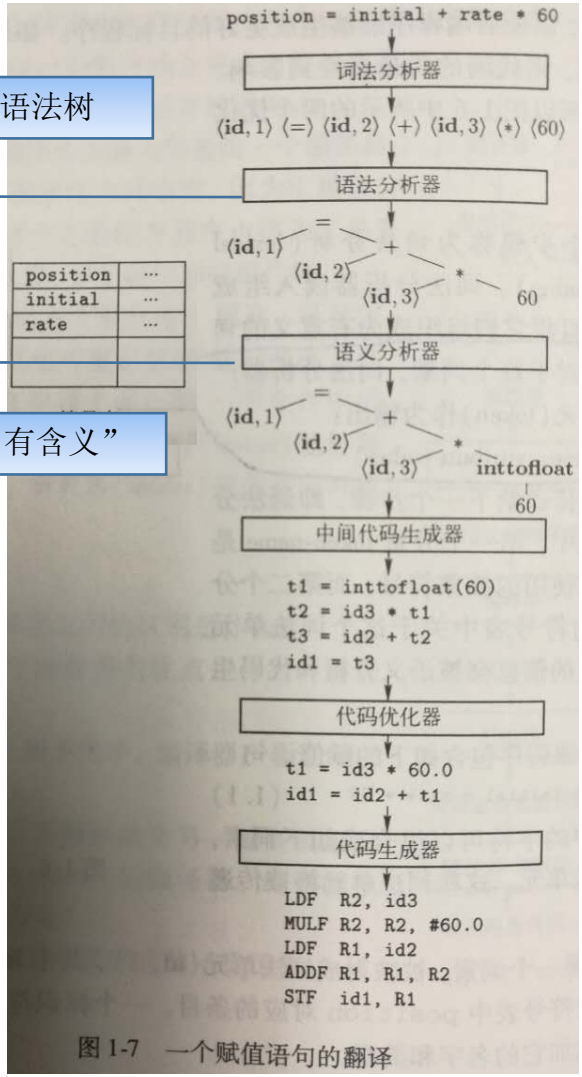


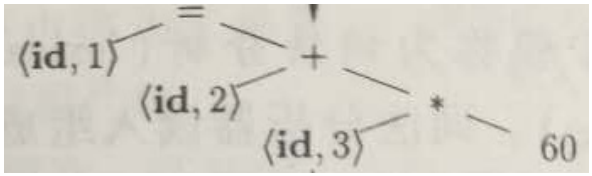
图 1-7 一个赋值语句的翻译

2 分析阶段，小结

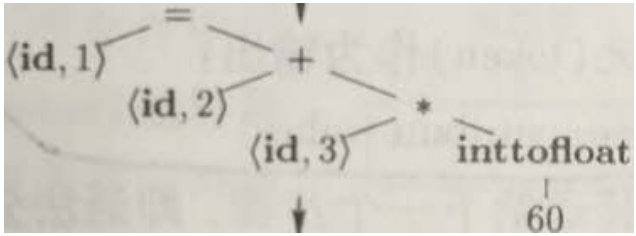
```
SELECT SUM(a1), id1
FROM t1 LEFT JOIN t2 on a1=a2 LEFT JOIN t3 on b2=b3
WHERE k1 IN (SELECT k3 FROM t3 AS t WHERE a3<30)
      AND b2=10 AND k2=10
GROUP BY id1;
```

1 词法分析

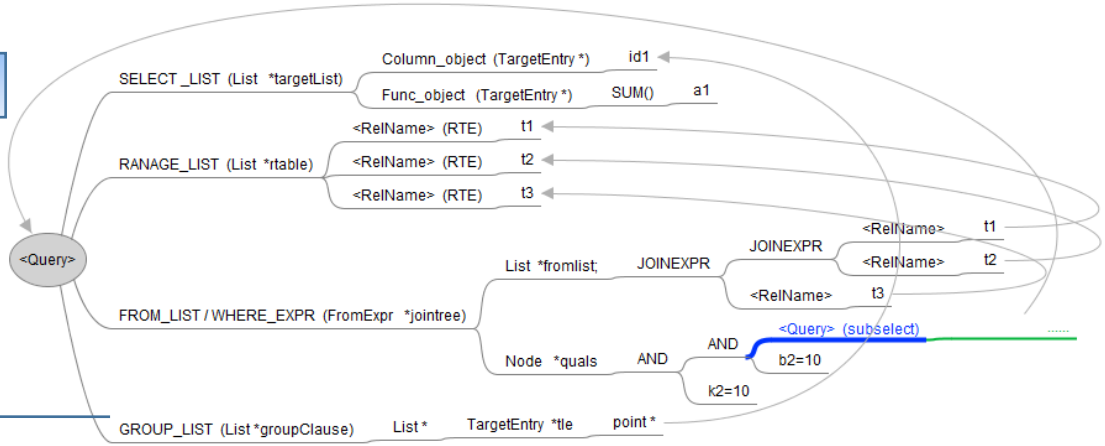
2 语法分析



3 语义分析



4 分析阶段产物：语法树



5 结论
SQL分析 == 编译器前半段

2 分析阶段：内核开发心得分享

看结构：深入理解SQL语句的结构

- 关键字是统领：整体、局部、隐含语义
- 对象间的关系：扁平化与嵌套

明语义：深入理解SQL语句的语义

- 各种连接方式之间的语义差异：笛卡尔集、等值连接、自然连接
- 语义之下的隐含关系：如外连接(外连接不符合关系代数定义的交换律)

揭开SQL优化的盖头来--数据库查询器实现技术解密

- 谁知我心——查询优化器的构造
- 分析阶段——从拆卸到重新组装
- **优化计算——点点滴滴从小事做起**
- 谁与争锋——主流数据库对比
  ---Oracle、PostgreSQL、MySQL、Informix

3 优化计算，点点滴滴从小事做起

优化的算法复杂

SQL的语义复杂

优化的技术多

优化的角度多

总之，优化点多而且零散，相互的关联关系不强

3 优化计算，点点滴滴从小事做起

优化器进行了多少个点的优化？

```
mysql> EXPLAIN SELECT SUM(a1), id1
    -> FROM t1 LEFT JOIN t2 on a1=a2 LEFT JOIN t3 on b2=b3
    -> WHERE k1 IN (SELECT k3 FROM t3 AS t WHERE a3<30)
    ->       AND b2=10 AND k2=10
    -> GROUP BY id1;
+----+-------------+-------+-------+------+--------+------+---------------------------------------------------+
| id | select_type | table | type  | key  | ref    | rows | Extra                                             |
+----+-------------+-------+-------+------+--------+------+---------------------------------------------------+
|  1 | SIMPLE      | t2    | const | k2   | const  |    1 | Using temporary; Using filesort                   |
|  1 | SIMPLE      | t     | ALL   | NULL | NULL   |   10 | Using where                                       |
|  1 | SIMPLE      | t1    | ref   | k1   | db.t.k3 |    1 | Using where                                       |
|  1 | SIMPLE      | t3    | ALL   | NULL | NULL   |   10 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------+-------+------+--------+------+---------------------------------------------------+
4 rows in set, 1 warning (0.00 sec)


mysql> SHOW WARNINGS;
...
/* select#1 */ select sum(`db`.`t1`.`a1`) AS `SUM(a1)`,`db`.`t1`.`id1` AS `id1`
from `db`.`t3` `t`
  join `db`.`t1` join `db`.`t2`
  left join `db`.`t3` on((`db`.`t3`.`b3` = 10))
where ((`db`.`t1`.`a1` = '10') and (`db`.`t1`.`k1` = `db`.`t`.`k3`) and (`db`.`t`.`a3` < 30))
group by `db`.`t1`.`id1`
...
```

3 优化计算，点点滴滴从小事做起

```
mysql> EXPLAIN SELECT SUM(a1), id1
    -> FROM t1 LEFT JOIN t2 on a1=a2 LEFT JOIN t3 on b2=b3
    -> WHERE k1 IN (SELECT k3 FROM t3 AS t WHERE a3<30)
    ->       AND b2=10 AND k2=10
    -> GROUP BY id1;
+----+-------------+-------+------+---------+---------+------+-----------------------------------------------------+
| id | select_type | table | type | key     | ref     | rows | Extra                                               |
+----+-------------+-------+------+---------+---------+------+-----------------------------------------------------+
|  1 | SIMPLE      | t2    | const| k2      | const   |    1 | Using temporary; Using filesort                     |
|  1 | SIMPLE      | t     | ALL  | NULL    | NULL    |   10 | Using where                                         |
|  1 | SIMPLE      | t1    | ref  | k1      | db.t.k3 |    1 | Using where                                         |
|  1 | SIMPLE      | t3    | ALL  | NULL    | NULL    |   10 | Using where; Using join buffer (Block Nested Loop)  |
+----+-------------+-------+------+---------+---------+------+-----------------------------------------------------+
4 rows in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
...
/* select#1 */ select sum(`db`.`t1`.`a1`) AS `SUM(a1)`,`db`.`t1`.`id1` AS `id1`
from `db`.`t3` `t` join `db`.`t1` join `db`.`t2`
  left join `db`.`t3` on((`db`.`t3`.`b3` = 10))
where ((`db`.`t1`.`a1` = '10') and (`db`.`t1`.`k1` = `db`.`t`.`k3`) and (`db`.`t`.`a3` < 30))
group by `db`.`t1`.`id1`
...
```

7 多表连接空间搜索
(t2 t t1)、(t2 t1 t)、(t t2 t1)、
(t t1 t2)、(t1 t2 t)、(t1 t t2)

8 GROUP BY利用索引

9 GROUP BY利用临时表

6 单表索引扫描时利用索引

4 “块”嵌套循环连接算法

4 嵌套循环两表连接算法

5 两表连接时利用索引

1 子查询上拉

3 外连接消除

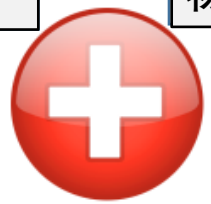
2 常量传递

3 优化器技术总结

逻辑优化

- 1 子查询上拉
- 2 常量传递
- 3 外连接消除
- x 视图优化
- x 等价谓词重写
- x 嵌套连接消除
- x 连接消除
- x 语义优化

逻辑优化：启发式规则



物理优化

- 4 “块”嵌套循环连接算法
- 5 两表连接时利用索引
- 6 单表索引扫描时利用索引
- 7 多表连接空间搜索
- 8 GROUP BY利用索引
- 9 对GROUP BY的优化
- x利用索引优化排序、MIN、MAX
- x 排序、DISTINCT、LIMIT等优化

物理优化：代价模型

基于规则优化的巨大好处：《2015 Pg用户大会--PostgreSQL视图优化》、《2015 Oracle技术嘉年华--MySQL视图优化》

3 优化计算：内核开发心得分享

逻辑优化：减少IO和CPU的消耗

- 下推操作减少元组数，减少了CPU消耗
- 子查询（视图、派生表）等优化消除了嵌套层次，减少了IO和CPU的消耗

物理优化：代价模型+利用索引

- 代价估算模型：各种物理操作的评估，实现细节是不同的
- 利用索引是各种物理算子操作的最好的优化手段

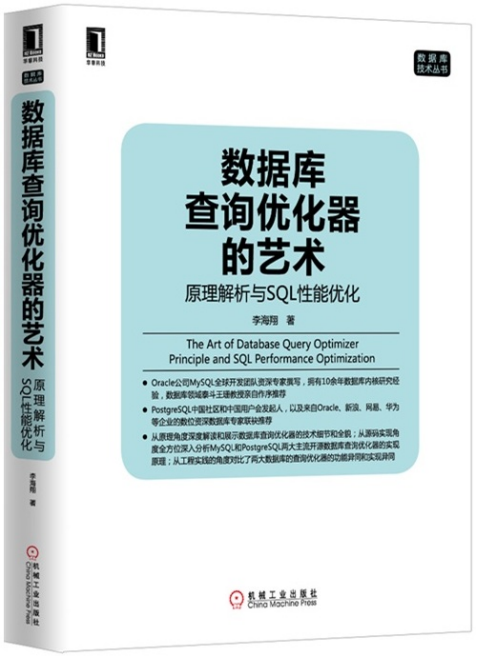
代价模型：2016年4月上海MySQL技术嘉年华大会--MySQL5.7代价模型

揭开SQL优化的盖头来--数据库查询优化器实现技术解密

- 谁知我心——查询优化器的构造
- 分析阶段——从拆卸到重新组装
- 优化计算——点点滴滴从小事做起
- **谁与争锋——主流数据库对比**
  ---Oracle、PostgreSQL、MySQL、Informix

4 谁与争锋，主流数据库对比---Oracle、Pg、MySQL、Informix?

PostgreSQL
9.2.3



MySQL®
5.6.10

Informix®
12.1 SOFTWARE

ORACLE®
11.1.0.6

4 谁与争锋

类别	内容	Oracle	Informix	PostgreSQL	MySQL
逻辑优化技术	子查询优化	支持	支持	支持	支持
	视图优化	支持	支持	支持	支持
	等价谓词重写	支持	支持	支持	支持
	条件化简	支持	支持	支持	支持
	外连接优化	支持	支持	支持	支持
	嵌套连接消除	支持	支持	支持	支持
	连接消除	支持	支持	支持	支持
物理优化技术	代价模型	支持	支持	支持	支持
	代价模型-直方图	支持	支持	支持	不支持
	利用索引进行多种优化	支持	支持	支持	支持
	全表扫描	支持	支持	支持	支持
	索引扫描	支持	支持	支持	支持
	嵌套循环连接	支持	支持	支持	支持
	hash连接	支持	支持	支持	不支持
	排序归并连接	支持	支持	支持	不支持
其他优化技术	自动创建索引	不支持	支持	不支持	支持
	并行查询计划与并行执行	支持	支持	不支持	不支持
	HINT改变执行计划	支持	支持	不支持	支持
	语义优化	部分支持	部分支持	部分支持	部分支持
	自动2阶段优化	不支持	支持	不支持	不支持
	自连接优化	不支持	支持	不支持	不支持

2016 DTCC 数据库内核技术专场 @那海蓝蓝

PostgreSQL, MySQL, Greenplum, Informix, etc

@那海蓝蓝 Blog: http://blog.163.com/li_hx/

《数据库查询优化器的艺术: 原理解析与SQL性能优化》

Database_xx@126.com

本次分享的Ppt位于:
http://pan.baidu.com/s/1jI6MBg6

更多的主流数据库优化器技术对比:
http://blog.163.com/li_hx/blog/static/183991413201632010369 32/

SequeMedia 盛拓传媒 IT168.com ChinaUnix ITPUB

附录

PostgreSQL中创建表和数据：

```
CREATE TABLE t1 (id1 INT, k1 INT UNIQUE, a1 INT, b1 INT, PRIMARY KEY(id1));
CREATE TABLE t2 (id2 INT, k2 INT UNIQUE, a2 INT, b2 INT, PRIMARY KEY(id2));
CREATE TABLE t3 (id3 INT, k3 INT UNIQUE, a3 INT, b3 INT, PRIMARY KEY(id3));

INSERT INTO t1 values(generate_series(1,5000), generate_series(1,5000), generate_series(1,1000), generate_series(1,1000));

INSERT INTO t2 values(generate_series(1,1000), generate_series(1,1000), generate_series(1,1000), generate_series(1,1000));
INSERT INTO t2 values(1001, 1001, NULL, NULL);

INSERT INTO t3 values(generate_series(1,10), generate_series(1,10), generate_series(2,20,2), generate_series(2,20,2));

ANALYZE;

//1 从Pg中导出数据到 “data.sql” 文件中
pg_dump --username=U --port=33333 --dbname=postgres --inserts –file=data.sql
//2 在其他数据库中，执行 “data.sql” 文件中的SQL
```