第十五讲：代价模型和优化模块的设计与实现

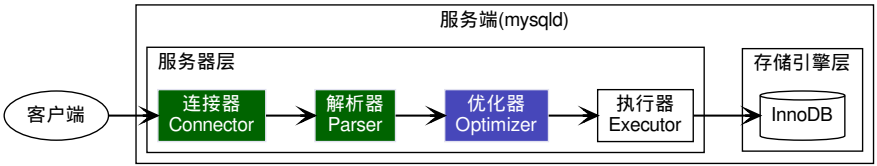
知春路遇上八里桥

<2024-06-27 Thu>

1

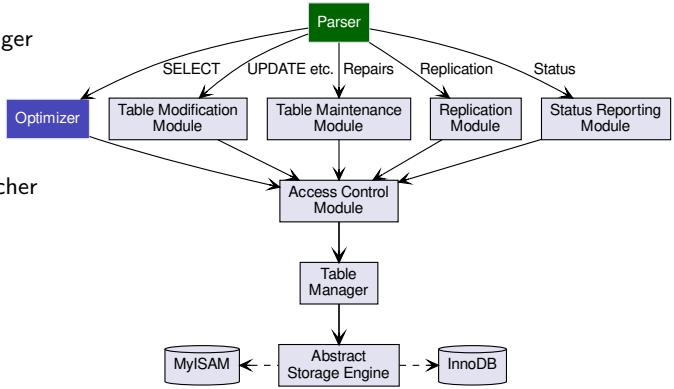
前情提要

执行流程


服务端(mysqld)
服务器层
客户端
连接器
Connector
解析器
Parser
优化器
Optimizer
执行器
Executor
存储引擎层
InnoDB

本节内容

- 连接器
  - ☑ 连接管理器 Connection Manager
  - ☑ 线程管理器 Thread Manager
  - ☑ 用户模块 User Module
- 解析器
  - ☑ 网络模块 Net Module
  - ☑ 派发模块 Commander Dispatcher
  - ☑ 词法分析 Lexical Analysis
  - ☑ 语法分析 Syntax Analysis
- 优化器
  - ☑ 准备模块 Prepare Module
  - ☑ 追踪日志 Optimizer Trace
  - ☐ 优化模块 Optimize Module



2

优化执行过程

JOIN::optimize() 核心函数说明

1. 代价优化准备
   - 窗口函数装配优化 `if (has_windows && Window::setup_windows2(thd, m_windows))`
   - 分区裁剪 `if (Query_block->partitioned_table_count && prune_table_partitions())`
   - 尝试把聚合函数 COUNT()、MIN()、MAX() 对应的值，替换成常量 `optimize_aggregated_query()`
   - 采用超图算法生成执行计划 [1] ，注意如果使用超图算法会直接返回

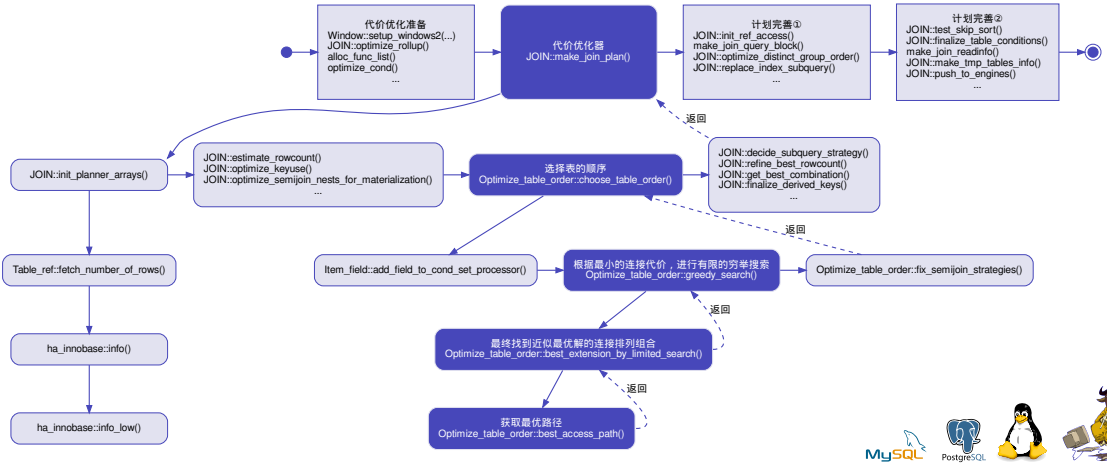
```
if (thd->lex->using_hypergraph_optimizer) {
  m_root_access_path = FindBestQueryPlan(thd, query_block, trace_ptr);
```

2. 代价优化器 `JOIN::make_join_plan()`
3. 计划完善部分
   - 语句块谓词条件下推，提升过滤性能 `make_join_query_block(this, where_cond)`
   - 优化 order by/distinct 语句 `optimize_distinct_group_order()`
   - 执行计划细化，优化子查询和半连接的情况 `make_join_readinfo(this, no_jbuf_after)`
     1. 关键代码是对半连接关联的策略进行装配 `setup_semijoin_dups_elimination()`
   - 为处理 group by/order by 创建开辟临时表空间 `make_tmp_tables_info()`
4. 生成访问路径 AccessPath `create_access_paths()`

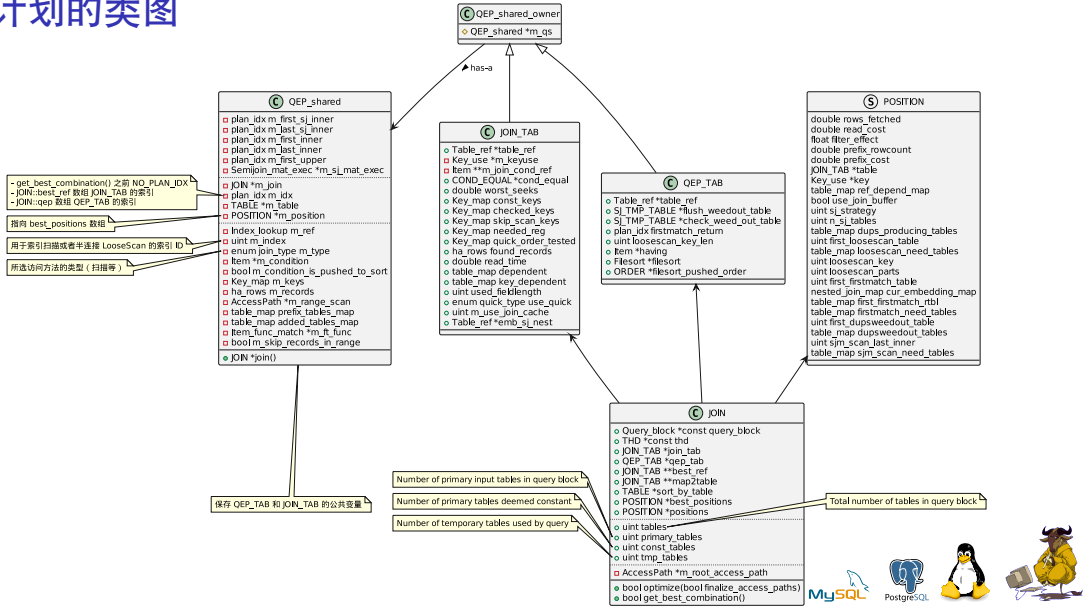
[1] 通过 `set optimizer_switch="hypergraph_optimizer=on"` 方式启用超图算法

JOIN::optimize() 流程图

- MySQL 优化器的好坏和背后的搜索策略、数学模型紧密相关
  - JOIN 优化器有 贪心算法 和 超图算法
  - 搜索策略有穷举搜索、贪婪搜索，整体优化过程基于 CBO 策略



执行计划的类图

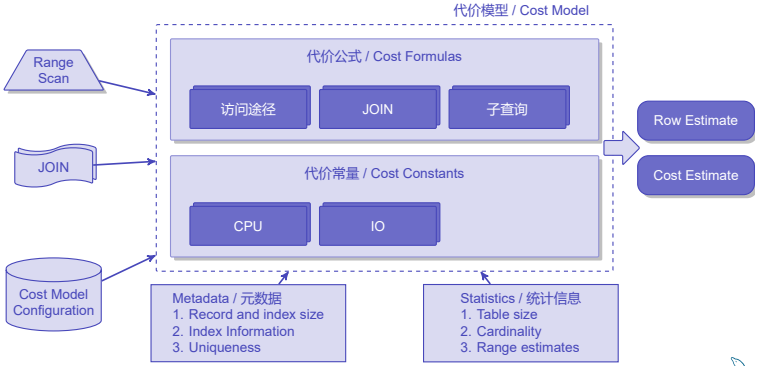

QEP_shared_owner
QEP_shared *m_qs
has-a
QEP_shared
plan_idx m_first_sj_inner
plan_idx m_last_sj_inner
plan_idx m_first_inner
plan_idx m_last_inner
plan_idx m_first_upper
Semijoin_mat_exec *m_sj_mat_exec
JOIN *m_join
plan_idx m_idx
TABLE *m_table
POSITION *m_position
Index_lookup m_ref
uint m_index
enum join_type m_type
Item *m_condition
bool m_condition_is_pushed_to_sort
Key_map m_keys
ha_rows m_records
AccessPath *m_range_scan
table_map prefix_tables_map
table_map added_tables_map
Item_func_match *m_ft_func
bool m_skip_records_in_range
JOIN *join()
- get_best_combination() 之前 NO_PLAN_IDX
- JOIN::best_ref 数组 JOIN_TAB 的索引
- JOIN::qep 数组 QEP_TAB 的索引
指向 best_positions 数组
用于索引扫描或者半连接 LooseScan 的索引 ID
所选访问方法的类型（扫描等）
保存 QEP_TAB 和 JOIN_TAB 的公共变量
JOIN_TAB
Table_ref *table_ref
Key_use *m_keyuse
Item **m_join_cond_ref
COND_EQUAL *cond_equal
double worst_seeks
Key_map const_keys
Key_map checked_keys
Key_map skip_scan_keys
Key_map needed_reg
Key_map quick_order_tested
ha_rows found_records
double read_time
table_map dependent
table_map key_dependent
uint used_fieldlength
enum quick_type use_quick
uint m_use_join_cache
Table_ref *emb_sj_nest
QEP_TAB
Table_ref *table_ref
SJ_TMP_TABLE *flush_weedout_table
SJ_TMP_TABLE *check_weed_out_table
plan_idx firstmatch_return
uint loosescan_key_len
Item *having
Filesort *filesort
ORDER *filesort_pushed_order
POSITION
double rows_fetched
double read_cost
float filter_effect
double prefix_rowcount
double prefix_cost
JOIN_TAB *table
Key_use *key
table_map ref_depend_map
bool use_join_buffer
uint sj_strategy
uint n_sj_tables
table_map dups_producing_tables
uint first_loosescan_table
table_map loosescan_need_tables
uint loosescan_key
uint loosescan_parts
uint first_firstmatch_table
nested_join_map cur_embedding_map
table_map first_firstmatch_rtbl
table_map firstmatch_need_tables
uint first_dupsweedout_table
table_map dupsweedout_tables
uint sjm_scan_last_inner
table_map sjm_scan_need_tables
JOIN
Query_block *const query_block
THD *const thd
JOIN_TAB *join_tab
QEP_TAB *qep_tab
JOIN_TAB **best_ref
JOIN_TAB **map2table
TABLE *sort_by_table
POSITION *best_positions
POSITION *positions
uint tables
uint primary_tables
uint const_tables
uint tmp_tables
AccessPath *m_root_access_path
bool optimize(bool finalize_access_paths)
bool get_best_combination()
Number of primary input tables in query block
Number of primary tables deemed constant
Number of temporary tables used by query
Total number of tables in query block
MySQL
PostgreSQL

3

代价模型实现

回顾 Cost 模型

- Cost 模型是一个基于代价估计的模型，每个查询被分解成若干操作 [2]
  1. 模型关注 m 种操作，其中操作 i 的 Cost 值为 C_i
  2. 假设查询可能有 n 个执行计划，执行计划 j 包含操作 i 的估计数量为 N_j
  3. 执行计划 j 的总 Cost 值为 $\text{cost}_j = \sum_{i=1}^{m} C_i * N_j$
  4. 在所有执行计划中选取 Cost 值最小的作为最优执行计划，最小的 Cost 值为 $min(\text{cost}_j), \forall 1 \leq j \leq n$



[2] 常见操作有 io_block_read/key_compare/row_evaluate 等

Cost 模型解释

- 基数 (Cardinality) 表示 Distinct Value 的个数
  - 比如性别字段取值只有男和女的话，则其基数为 2
- 选择度 (Selectivity) 计算公式为 选择度 = 基数 ÷ 行数
  - 选择度越高越好，最高为 100%
  - 像性别字段的例子，选择度一般就很低，作为索引列时区分度差
  - 男性筛选条件下面的数据依然是处于较大量级
- 访问途径 (Access Path)，获取数据行的方式，一般包含：
  - `ALL`: 全表扫描，对整个表的所有数据进行扫描
  - `index`: 一般 Index Scan 指的是二级索引扫描
  - `range`: 对于索引列的一些可能转化为范围查询的条件
  - `eq_ref`: Join Field 是索引且是主键或 Non-null Unique 索引
    意味着对于每个记录最多只会 Join 到右表的一行
  - `ref`: Join Field 是索引，但不是主键或 Non-null Unique 索引
  - ...
- JOIN 和子查询

Cost 模型常量的字典表

MySQL 8.0 的 Cost 配置常量如下所示 [3]

```
mysql> select * from mysql.engine_cost;
+-------------+-------------+------------------------+------------+---------------------+---------+---------------+
| engine_name | device_type | cost_name              | cost_value | last_update         | comment | default_value |
+-------------+-------------+------------------------+------------+---------------------+---------+---------------+
| default     |           0 | io_block_read_cost     |       NULL | 2024-05-09 08:39:29 | NULL    |             1 |
| default     |           0 | memory_block_read_cost |       NULL | 2024-05-09 08:39:29 | NULL    |          0.25 |
+-------------+-------------+------------------------+------------+---------------------+---------+---------------+
2 rows in set (0.00 sec)
mysql> select * from mysql.server_cost;
+------------------------------+------------+---------------------+---------+---------------+
| cost_name                    | cost_value | last_update         | comment | default_value |
+------------------------------+------------+---------------------+---------+---------------+
| disk_temptable_create_cost   |       NULL | 2024-05-09 08:39:29 | NULL    |            20 |
| disk_temptable_row_cost      |       NULL | 2024-05-09 08:39:29 | NULL    |           0.5 |
| key_compare_cost             |       NULL | 2024-05-09 08:39:29 | NULL    |          0.05 |
| memory_temptable_create_cost |       NULL | 2024-05-09 08:39:29 | NULL    |             1 |
| memory_temptable_row_cost    |       NULL | 2024-05-09 08:39:29 | NULL    |           0.1 |
| row_evaluate_cost            |       NULL | 2024-05-09 08:39:29 | NULL    |           0.1 |
+------------------------------+------------+---------------------+---------+---------------+
6 rows in set (0.01 sec)
```

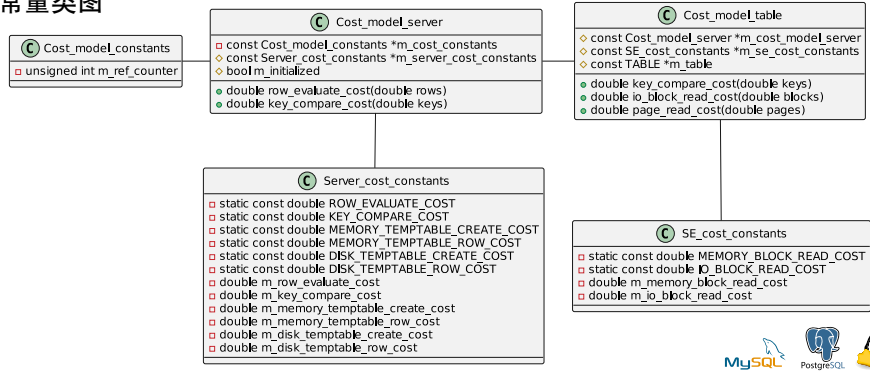
[3] https://dev.mysql.com/doc/refman/8.0/en/cost-model.html

Cost 模型中的常量值

- 代码中的使用场景

```
const Cost_model_server *const cost_model = thd->cost_model();
ha_rows records = table->file->stats.records;
if (!records) records++; /* purecov: inspected */
double scan_time =
    cost_model->row_evaluate_cost(static_cast<double>(records)) + 1;
```

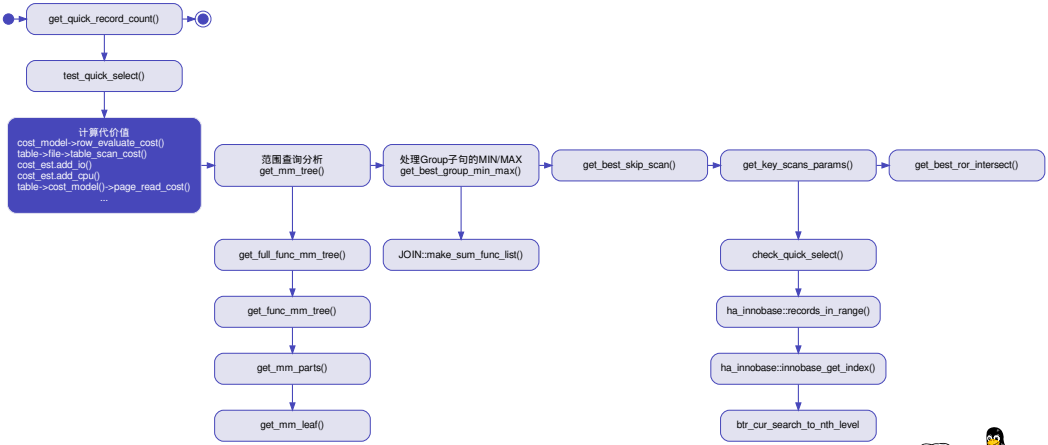
- Cost 模型常量类图



JOIN::estimate_rowcount() 流程

- 计算 CPU/IO 的 cost 值示例

```
select * from employees where emp_no < 10101;
```



结束