第十三讲：准备阶段 Rewrite 和 Prepare 设计与实现

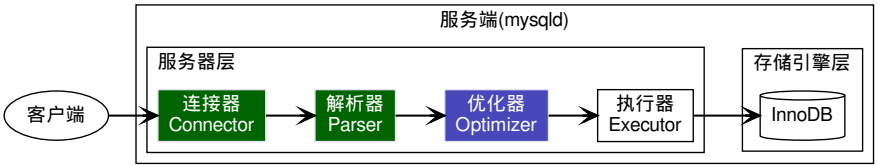
知春路遇上八里桥

<2024-06-20 Thu>

1

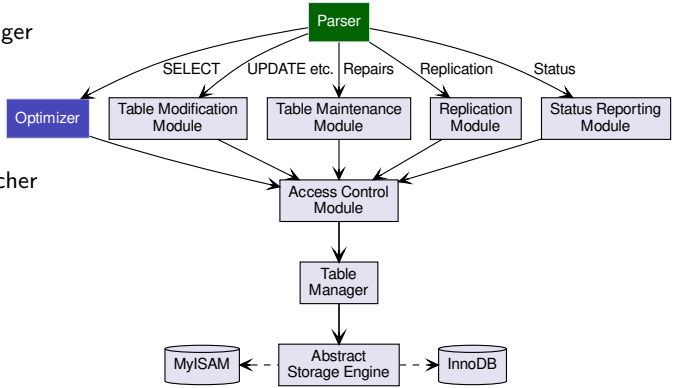
前情提要

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



2

Rewrite 过程

Rewrite 功能介绍

1. MySQL 支持查询 Rewrite [1] 插件，可以在执行之前检查并修改接收的语句
   - 该插件允许在服务器处理之前重写到达服务器的 SQL 语句
   - 该插件接收一个语句字符串，并可能返回一个不同的字符串
2. 这个设计可以满足 DBA 在不修改业务代码前提下在 Query 中添加 hits
3. Rewrite 的安装和卸载通过自带的脚本完成
   - 脚本位于 `$MYSQL_HOME/share` 目录下
   - 安装脚本 `install_rewriter.sql`
   - 卸载脚本 `uninstall_rewriter.sql`
4. Rewrite 的安装和卸载过程

```
mysql> source /opt/mysql/share/install_rewriter.sql
Query OK, 1 row affected (0.01 sec)
...
mysql> source /opt/mysql/share/uninstall_rewriter.sql
Query OK, 1 row affected (0.05 sec)
...
```

[1] https://dev.mysql.com/doc/refman/8.0/en/rewriter-query-rewrite-plugin.html

添加 Rewrite 规则

- 可以通过 Rewrite 将发送的查询语句添加一个 LIMIT 1 限制只返回一行

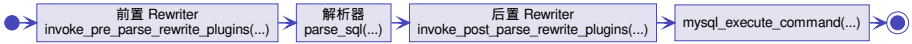
```
-- 插入规则
INSERT INTO query_rewrite.rewrite_rules
            (pattern_database, pattern, replacement)
VALUES (
  "employees",
  "SELECT * FROM employees WHERE emp_no > ? ",
  "SELECT * FROM employees WHERE emp_no > ? LIMIT 1"
);
-- 应用规则
CALL query_rewrite.flush_rewrite_rules();
```

规则生效示例

```
mysql> select * from query_rewrite.rewrite_rules\G
*************************** 1. row ***************************
                id: 1
           pattern: SELECT * FROM employees WHERE emp_no > ?
  pattern_database: employees
       replacement: SELECT * FROM employees WHERE emp_no > ? LIMIT 1
           enabled: YES
           message: NULL
    pattern_digest: 193b6eb2e5a0f8cc9351110a142f3bfa196f09b17cfb34bea295bd0bc77ec92a
normalized_pattern: select `*` from `employees`.`employees` where (`emp_no` > ?)
1 row in set (0.00 sec)
mysql> select count(*) from employees where emp_no > 11100;
+----------+
| count(*) |
+----------+
|   298924 |
+----------+
1 row in set (2.24 sec)
mysql> select * from employees where emp_no > 11100;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date  |
+--------+------------+------------+-----------+--------+------------+
|  11101 | 1961-10-31 | Manibrata  | Horswill  | M      | 1992-06-26 |
+--------+------------+------------+-----------+--------+------------+
1 row in set, 1 warning (0.01 sec)
mysql>
```

Rewrite 实现示例

- Rewrite 的实现包含两种模式：pre-parse 和 post-parse [2]



- Rewrite 的调用点在 dispatch_sql_command() 函数中，精简代码如下

```
void dispatch_sql_command(THD *thd, Parser_state *parser_state) {
  thd->reset_rewritten_query();

  thd->m_parser_state = parser_state;
  invoke_pre_parse_rewrite_plugins(thd);
  thd->m_parser_state = nullptr;

  if (!err) {
    err = parse_sql(thd, parser_state, nullptr);
    if (!err) err = invoke_post_parse_rewrite_plugins(thd, false);
```

- Rewrite 实现见文件 ☞ sql/sql_query_rewrite.cc
  - invoke_pre_parse_rewrite_plugins() 前置解析 Rewrite 的实现
  - invoke_post_parse_rewrite_plugins() 后置解析 Rewrite 的实现

[2] 上述 Rewrite 示例是 post-parse 模式实现

重写 Query 断点调试过程

```
(gdb) b THD::set_query
Breakpoint 8 at 0x555558c4959c: THD::set_query. (2 locations)
(gdb) c
Continuing.
Thread 51 "connection" hit Breakpoint 8, THD::set_query (this=0x7fff3807b2e0, query_arg=0x7fff381e7a08 "SELECT * FROM em
4142      void set_query(const char *query_arg, size_t query_length_arg) {
(gdb) bt
#0  THD::set_query (this=0x7fff3807b2e0, query_arg=0x7fff381e7a08 "SELECT * FROM employees WHERE emp_no > 11100 LIMIT 1"
#1  0x0000555558d8cd94 in alloc_query (thd=0x7fff3807b2e0, packet=0x7fff381ae130 "SELECT * FROM employees WHERE emp_no >
#2  0x00005555594b2997 in mysql_parser_parse (thd=0x7fff3807b2e0, query=..., is_prepared=0 '\000', handle_condition=0x0,
#3  0x00007fffe899e481 in services::parse (thd=0x7fff3807b2e0, query="SELECT * FROM employees WHERE emp_no > 11100 LIMIT
#4  0x00007fffe899a153 in rewrite_query_notify (thd=0x7fff3807b2e0, event_class=MYSQL_AUDIT_PARSE_CLASS, event=0x7fffe01
#5  0x0000555558ff0428 in plugins_dispatch (thd=0x7fff3807b2e0, plugin=0x7fff380a1820, arg=0x7fffe01ed620) at /opt/src/m
#6  0x0000555558ff0530 in event_class_dispatch (thd=0x7fff3807b2e0, event_class=MYSQL_AUDIT_PARSE_CLASS, event=0x7fffe01
#7  0x0000555558ff05fc in event_class_dispatch_error (thd=0x7fff3807b2e0, event_class=MYSQL_AUDIT_PARSE_CLASS, event_nam
#8  0x0000555558fee63d in mysql_audit_notify (thd=0x7fff3807b2e0, subclass=MYSQL_AUDIT_PARSE_POSTPARSE, subclass_name=0x
#9  0x0000555558de54dd in invoke_post_parse_rewrite_plugins (thd=0x7fff3807b2e0, is_prepared=false) at /opt/src/mysql-se
#10 0x0000555558d95048 in dispatch_sql_command (thd=0x7fff3807b2e0, parser_state=0x7fffe01ed970) at /opt/src/mysql-serve
#11 0x0000555558d8aa69 in dispatch_command (thd=0x7fff3807b2e0, com_data=0x7fffe01ee2c0, command=COM_QUERY) at /opt/src/
#12 0x0000555558d888d3 in do_command (thd=0x7fff3807b2e0) at /opt/src/mysql-server/sql/sql_parse.cc:1440
#13 0x0000555558fd61c5 in handle_connection (arg=0x5555610a30a0) at /opt/src/mysql-server/sql/conn_handler/connection_ha
#14 0x000055555b2a2410 in pfs_spawn_thread (arg=0x555561303860) at /opt/src/mysql-server/storage/perfschema/pfs.cc:3043
#15 0x00007ffff732fac3 in start_thread (arg=<optimized out>) at ./nptl/pthread_create.c:442
#16 0x00007ffff73c1850 in clone3 () at ../sysdeps/unix/sysv/linux/x86_64/clone3.S:81
```

3

Prepare 过程

准备阶段功能概述

- 准备阶段用于为整个查询做准备工作，具体划分成以下操作函数
  - Setup/Fix 设置和修复工作，也包含一部分语义检查
    1. 准备临时表
    2. 打开表
    3. 检查表是否可以访问
    4. 检查字段
    5. 检查非 group 函数
    6. 准备 procedure
    7. ...
  - Transformation 变换工作，通过恒等变换来降低后序优化分析代码的复杂度
    1. 代数变换
    2. 逻辑变换
- 准备阶段的大部分代码位于 ☞ sql/sql_resolver.cc 文件中
  - 历史遗留的变换方法，例如 `bool setup_order(...)`
  - Query_block 类是成员函数实现代码

Setup/Fix 部分举例

- Setup 设置函数
  - `setup_tables()` 设置 Query Block 所有表中的叶子节点的表
  - `setup_wild()` 将 `'*'` 设置成表中对应的所有列名

```
select * from employees; -- 将 * 通配符替换成 employees 表的列名
```

  - `setup_fields()` 检查表中是否存在所有给定的列，并将查找的列填充进去
  - `setup_conds()` 设置 `WHERE` 子句中的条件和 `JOIN` 子句中的条件
  - `setup_group()` 设置 `GROUP BY` 子句中的字段列表
  - `setup_order()` 设置 `ORDER BY` 子句
  - `setup_order_final()` 在 Query Block 全部解析完成后，再一次设置 `ORDER BY` 子句

```
select first_name, last_name
  from employees where emp_no <10010
 order by 2 -- 这里的 ORDER BY 必须等 Query Block 全部解析完成才能确定
```

  - `setup_ftfuncs()` 在解析 `HAVING` 子句后设置全文函数 (full-text functions)
  - ...
- Fix 修复函数
  - `m_having_cond->fix_fields()` 设置 `HAVING` 子句
  - `resolve_limits()` 设置 `OFFSET` 和 `LIMIT` 子句
  - `resolve_rollup()` 汇总 `SELECT` 和 `ORDER BY` 子句中的 `Item` 对象
  - ...

Transformation 变换举例（壹）

- `resolve_subquery()` 解析子查询
  - 将子查询转换成半连接 (semi-join)
  - 标记子查询执行时使用物化策略 (materialization)
  - 进行 `IN` → `EXISTS` 的变换
  - 进行 `ALL/ANY` → `MIN/MAX` 的重写
  - 使用具体值替换对应标量上下文 (scalar-context) 的子查询
- `simplify_joins()` 将 OUTER JOIN 尽可能地转化成 INNER JOIN，看代码注释中的例子

```
SELECT * FROM t1 LEFT JOIN t2 ON t2.a=t1.a WHERE t2.b < 5       -- 注意到 t2.b < 5 拒绝 NULL
SELECT * FROM t1 INNER JOIN t2 ON t2.a=t1.a WHERE t2.b < 5      -- 首次变换
SELECT * FROM t1, t2 ON t2.a=t1.a WHERE t2.b < 5 AND t2.a=t1.a -- 最终变换
```

- `apply_local_transforms()` 应用局部变换，例如：
  - 这里的 “局部” (local) 指转化对象限制在一个 Query Block
  - join nest simplification （嵌套 JOIN 的简化）
  - partition pruning （分区剪枝）
  - condition pushdown to derived tables （对派生表进行条件下推）
  - ...

Transformation 变换举例（贰）

- `remove_redundant_subquery_clause()` 删除子查询中的多余的字段
  - 对于包含 LIMIT 的子句，谓词 IN/ANY/ALL/EXISTS 等需要清理
  - 如果没有聚合函数和 HAVING 子句，清理 GROUP BY 多余的子句
  - 对于不包含 LIMIT 的标量查询，ORDER BY 是多余的，因为标量的列数为 1
- `flatten_subqueries()` 将 semi-join 子查询转换成 semi-join 嵌套 JOIN 子句

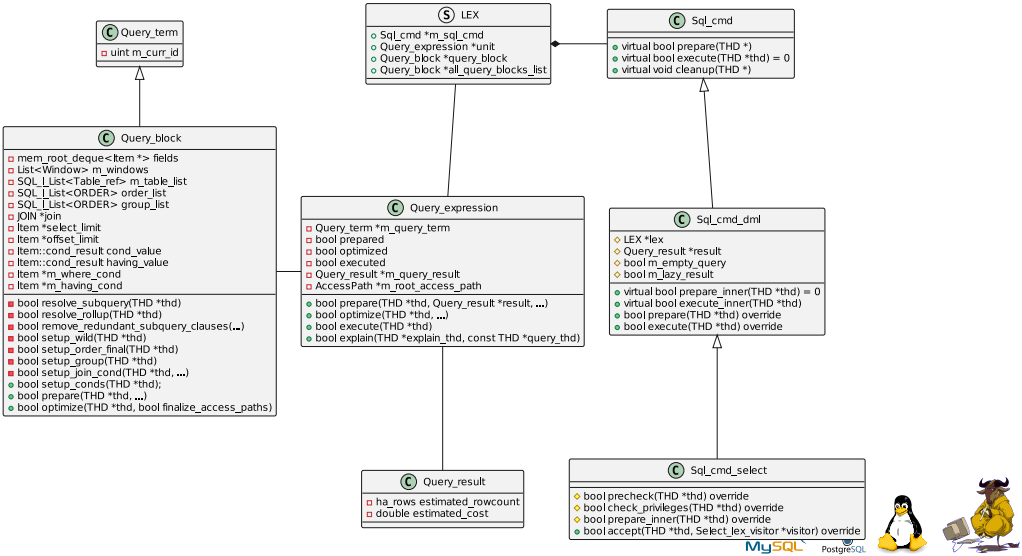
```
SELECT ...
FROM ot, ...
WHERE oe IN (SELECT ie FROM it1 ... itN WHERE subq_where) AND outer_where
-- 这里将子查询转换成半连接
SELECT ...
FROM ot SEMI JOIN (it1 ... itN), ...
WHERE outer_where AND subq_where AND oe=ie
```

- `delete_unused_merged_columns()` 合并表时删除没有使用的列
  - 对于嵌套 JOIN 进行递归调用来处理
  - 如果发现表合并，自动删除没有使用的列

4

代码分析

sql_cmd 类关系图


Query_term
uint m_curr_id
LEX
Sql_cmd *m_sql_cmd
Query_expression *unit
Query_block *query_block
Query_block *all_query_blocks_list
Sql_cmd
virtual bool prepare(THD *)
virtual bool execute(THD *thd) = 0
virtual void cleanup(THD *)
Query_block
mem_root_deque<Item *> fields
List<Window> m_windows
SQL_I_List<Table_ref> m_table_list
SQL_I_List<ORDER> order_list
SQL_I_List<ORDER> group_list
JOIN *join
Item *select_limit
Item *offset_limit
Item::cond_result cond_value
Item::cond_result having_value
Item *m_where_cond
Item *m_having_cond
bool resolve_subquery(THD *thd)
bool resolve_rollup(THD *thd)
bool remove_redundant_subquery_clauses(...)
bool setup_wild(THD *thd)
bool setup_order_final(THD *thd)
bool setup_group(THD *thd)
bool setup_join_cond(THD *thd, ...)
bool setup_conds(THD *thd);
bool prepare(THD *thd, ...)
bool optimize(THD *thd, bool finalize_access_paths)
Query_expression
Query_term *m_query_term
bool prepared
bool optimized
bool executed
Query_result *m_query_result
AccessPath *m_root_access_path
bool prepare(THD *thd, Query_result *result, ...)
bool optimize(THD *thd, ...)
bool execute(THD *thd)
bool explain(THD *explain_thd, const THD *query_thd)
Sql_cmd_dml
LEX *lex
Query_result *result
bool m_empty_query
bool m_lazy_result
virtual bool prepare_inner(THD *thd) = 0
virtual bool execute_inner(THD *thd)
bool prepare(THD *thd) override
bool execute(THD *thd) override
Query_result
ha_rows estimated_rowcount
double estimated_cost
Sql_cmd_select
bool precheck(THD *thd) override
bool check_privileges(THD *thd) override
bool prepare_inner(THD *thd) override
bool accept(THD *thd, Select_lex_visitor *visitor) override
MySQL
PostgreSQL

mysql_execute_command() 执行命令

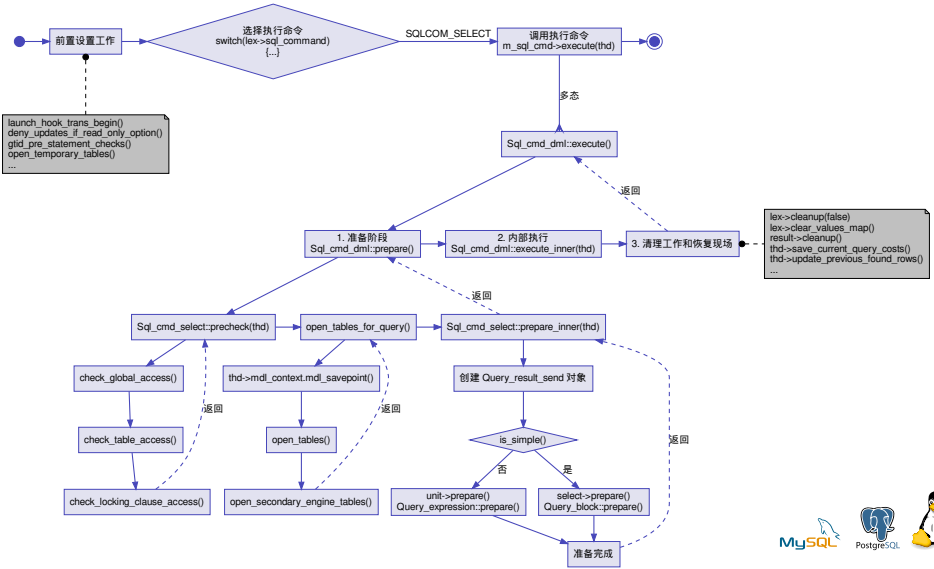
- mysql_execute_command() 的实现见 ☞ sql/sql_parse.cc ， 有超过 2000 行代码
  - 见 ☞ sql/sql_parse.cc

```
2946  int mysql_execute_command(THD *thd, bool first_level) {
 ⋮
5056    return res || thd->is_error();
5057  }
```

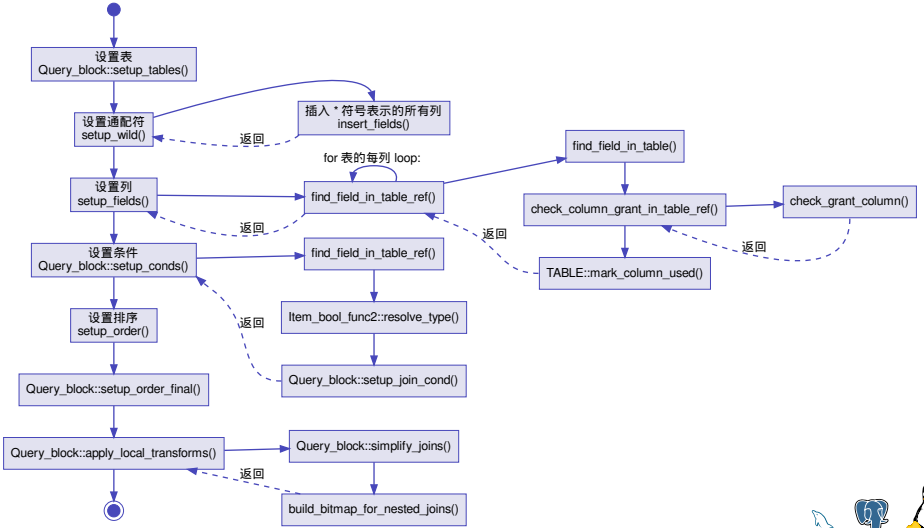
  - mysql_execute_command() 主要功能如下
    1. 执行 lex->m_sql_cmd 中的命令
    2. 根据 lex->sql_command 来调用不同的处理逻辑
    3. 如果是 SQLCOM_SELECT ， 根据多态调用 Sql_cmd_dml::execute() 函数
- Sql_cmd_dml::execute() 执行 SELECT 语句
  - 实现见文件 ☞ sql/sql_select.cc

```
672  bool Sql_cmd_dml::execute(THD *thd) {
673    DBUG_TRACE;
 ⋮
878    return thd->is_error();
879  }
```

从 mysql_execute_command() 进入准备阶段



查询块的准备过程 - Query_block::prepare() 函数



结束