

第十八讲：MySQL 执行器的设计与实现

知春路遇上八里桥

<2024-07-09 Tue>



① 前情提要

② 火山模型

③ 执行阶段

④ 代码调试

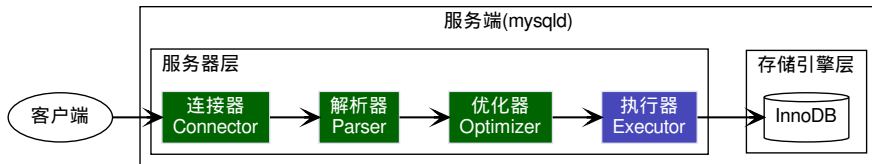


1

前情提要



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

- 执行器

- ▶ ☐ 火山模型 Volcano Model
- ▶ ☐ 执行模块 Execution Module



2

火山模型

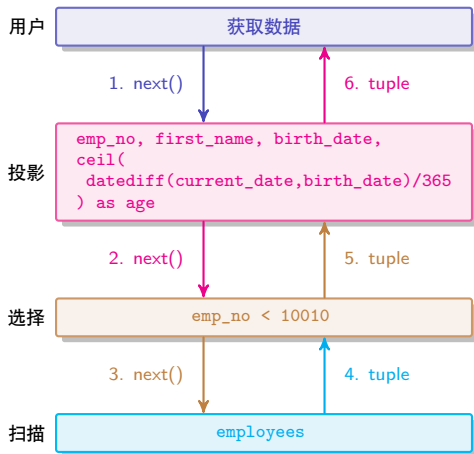


火山模型

- 数据库中的火山模型 (Volcano Model), 又称为迭代器模型
 - ▶ 最早由 Goetz Graefe 在 1990 年代提出
 - ▶ 旨在解决早期数据库系统中由于硬件资源限制所带来的性能问题
 - ▶ 是一种面向数据流的查询执行模型, 在各种数据库系统中应用最广泛
- 火山模型将关系代数中的每一种操作 (Operator) 抽象为一个迭代器 (Iterator)
 - ▶ 操作如扫描 (Scan)、选择 (Selection)、投影 (Projection)、连接 (Join) 等
 - ▶ 火山模型的执行树是通过 Operator 构成, 每个 Operator 是执行树的一个节点
 - ▶ 实际执行时, 根节点到叶子节点自上而下地递归调用 next() 函数, 实现数据的处理和传递
- 每个迭代器都遵循 open-next-close 协议进行工作, 分别实现以下接口
 - ▶ open 方法用于初始化迭代器
 - ▶ next 方法用于获取下一行数据
 - ▶ close 方法用于释放资源
- 除了火山模型, 还有以下执行器模型
 - ▶ Materialization Model (物化模型)
 - ▶ Vectorized / Batch Model (批式模型)



火山模型示例



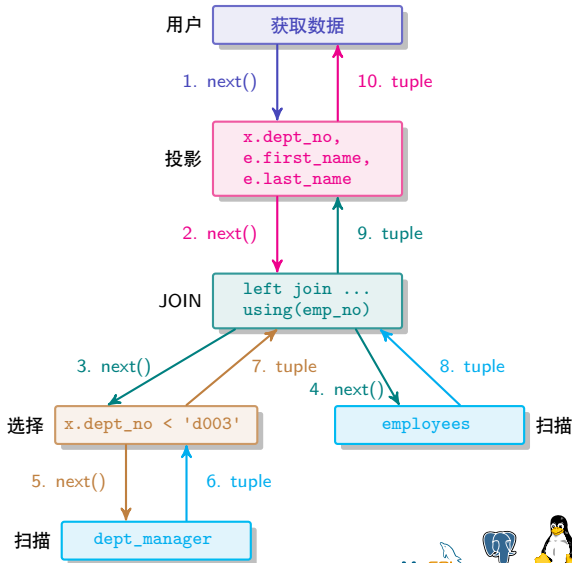
- 火山模型采用拉的形式驱动
- 每层调用 `next()` 函数, 返回 `tuple` 结果
- 右图是下面测试查询语句的执行示意图

```
1  select
2      emp_no,
3      first_name,
4      birth_date,
5      ceil(datediff(current_date, birth_date)/365)
6      as age
7  from
8      employees
9  where
10     emp_no < 10010;
```

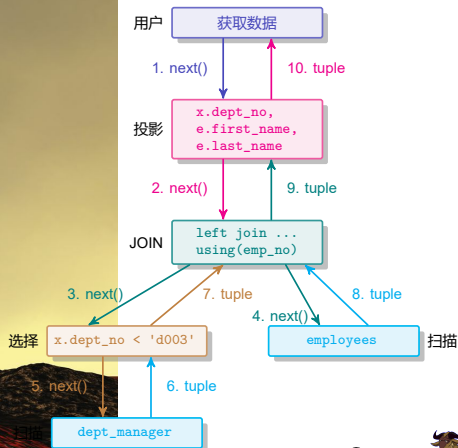
火山模型示例贰

● 两表 Join 的查询

```
1  select
2    x.dept_no,
3    e.first_name,
4    e.last_name
5  from
6    dept_manager x
7   left join
8     employees e
9     using (emp_no)
10 where
11   x.dept_no < 'd004';
```



火山模型示例



MySQL

PostgreSQL



迭代器实现

- Rowlterator 类是所有迭代器的基类¹，它通过纯虚函数声明必要的接口
- Rowlterator 类对的 open-next-close 协议的实现如下表

协议中的接口	Rowlterator 类的方法	功能说明
open	RowIterator:Init()	打开所有必须的资源
next	RowIterator:Read()	读取一行，将行放入记录缓存中
close	RowIterator:UnlockRow()	将一行过滤出结果集后

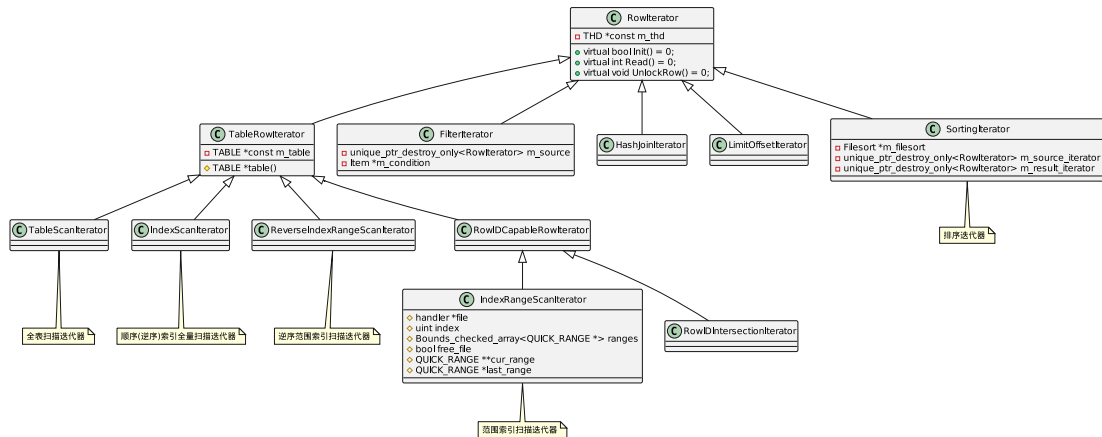
- Rowlterator 在 `sql/iterators/row_iterator.h` 中实现

```
82 class RowIterator {
83     public:
84         // NOTE: Iterators should typically be instantiated using NewIterator,
85         // in sql/iterators/timing_iterator.h.
86         explicit RowIterator(THD *thd) : m_thd(thd) {}
87         virtual ~RowIterator() = default;
```

¹<https://dev.mysql.com/doc/dev/mysql-server/8.0.37/classRowlterator.html>



Rowlterator 相关类继承关系




3


执行阶段



创建访问路径 AccessPath

- 优化器生成执行路径 JOIN::create_access_paths()  sql/sql_executor.cc

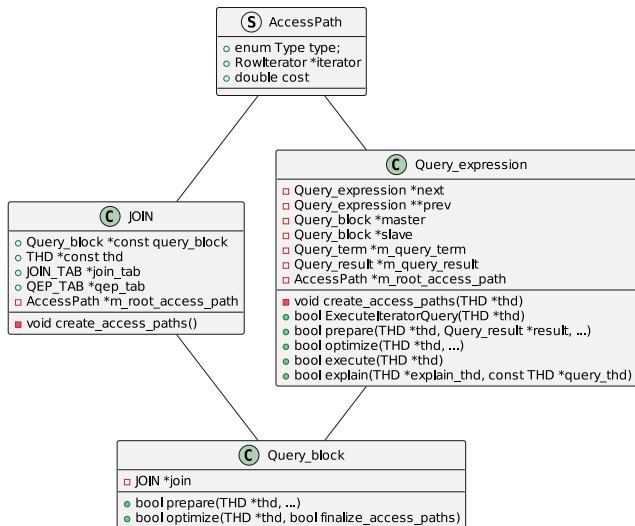
```
2959 void JOIN::create_access_paths() {
2960     assert(m_root_access_path == nullptr);
2961
2962     AccessPath *path = create_root_access_path_for_join();
2963     path = attach_access_paths_for_having_and_limit(path);
2964     path = attach_access_path_for_update_or_delete(path);
2965
2966     m_root_access_path = path;
2967 }
```

- 优化器生成执行路径 Query_expression::create_access_paths()  sql/sql_union.cc

```
1417 void Query_expression::create_access_paths(THD *thd) {
1418     if (is_simple()) {
1419         JOIN *join = first_query_block()->join;
1420         assert(join && join->is_optimized());
1421         m_root_access_path = join->root_access_path();
1422         return;
1423     }
1424     // ...
```



AccessPath 关系类图

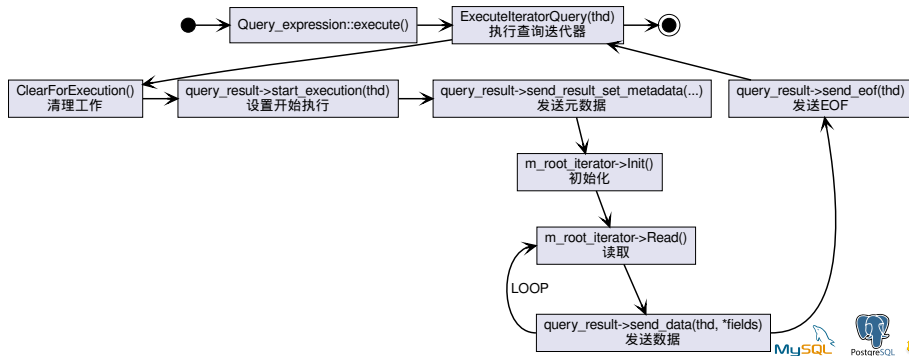


迭代执行过程

- 执行路径创建迭代器 `sql/join_optimizer/access_path.cc`

```
378 unique_ptr_destroy_only<RowIterator> CreateIteratorFromAccessPath(  
379     THD *thd, MEM_ROOT *mem_root, AccessPath *top_path, JOIN *top_join,  
380     bool top_eligible_for_batch_mode) {  
381     assert(IteratorsAreNeeded(thd, top_path));
```

- 然后通过火山模型执行并发送结果给客户端



迭代器执行细节

- Query_expression::ExecuteIteratorQuery() 在 `sql/sql_union.cc` 中实现

```
bool Query_expression::ExecuteIteratorQuery(THD *thd) {  
    // ...  
    if (m_root_iterator->Init()) {  
        return true;  
    }  
    for (;;) {  
        int error = m_root_iterator->Read();  
        // ...  
        if (query_result->send_data(thd, *fields)) {  
            return true;  
        }  
        thd->get_stmt_da()->inc_current_row_for_condition();  
    }  
}  
// ...  
return query_result->send_eof(thd);  
}
```



4

代码调试



调试语句 exe01.sql

```
mysql> \. exe01.sql
```

emp_no	first_name	birth_date	age
10001	Georgi	1953-09-02	71
10002	Bezalel	1964-06-02	61
10003	Parto	1959-12-03	65
10004	Chirstian	1954-05-01	71
10005	Kyoichi	1955-01-21	70
10006	Anneke	1953-04-20	72
10007	Tzvetan	1957-05-23	68
10008	Saniya	1958-02-19	67
10009	Sumant	1952-04-19	73

```
9 rows in set (0.01 sec)
```

运行 exe01.sql 查询语句并调试结果

```
1  select
2      emp_no,
3      first_name,
4      birth_date,
5      ceil(datediff(current_date, birth_date)/365)
6      as age
7  from
8      employees
9  where
10     emp_no < 10010;
```



添加调试的关键断点

● 在创建 AccessPath 和 Iterator 处添加断点

- b create_access_paths
- b CreateIteratorFromAccessPath
- b ExecuteIteratorQuery

● 查看断点

```
(gdb) i b
Num      Type           Disp Enb Address                What
4        breakpoint     keep y   0x0000555558d14072 in JOIN::create_access_paths()
          breakpoint     keep y   at /opt/src/mysql-server/sql/sql_executor.cc:2960
          breakpoint already hit 15 times
5        breakpoint     keep y   <MULTIPLE>
          breakpoint already hit 29 times
5.1      y             0x0000555558d14072 in JOIN::create_access_paths()
          at /opt/src/mysql-server/sql/sql_executor.cc:2960
5.2      y             0x0000555558ee0487 in Query_expression::create_access_paths(THD*)
          at /opt/src/mysql-server/sql/sql_union.cc:1417
6        breakpoint     keep y   <MULTIPLE>
          breakpoint already hit 26 times
6.1      y             0x0000555558d23b78 in CreateIteratorFromAccessPath(THD*, AccessPath*, JOIN*, bool)
          at /opt/src/mysql-server/sql/join_optimizer/access_path.h:1730
6.2      y             0x000055555938badc in CreateIteratorFromAccessPath(THD*, MEM_ROOT*, AccessPath*, JOIN*,
          at /opt/src/mysql-server/sql/join_optimizer/access_path.cc:380
```

(gdb)



Trace 日志片段分析

通过 `mysqld.trace` 日志^❶ 可以观察执行器执行的详细逻辑

```
T@10: | | | | | >Query_expression::execute <= 进入执行器
T@10: | | | | | THD::enter_stage: 'executing' /opt/src/mysql-server/sql/sql_union.cc:1670
...
T@10: | | | | | >InitIndexRangeScan <= 初始化火山模型
T@10: | | | | | | >handler::ha_index_init
T@10: | | | | | | | >int ha_innabase::index_init
T@10: | | | | | | | | >ha_innabase::change_active_index
T@10: | | | | | | | | | >dict_index_t* ha_innabase::innabase_get_index
...
T@10: | | | | | >int IndexRangeScanIterator::Read <= 读取一条数据，后续陆续读取 n 条
T@10: | | | | | | >handler::ha_multi_range_read_next
T@10: | | | | | | | >int handler::multi_range_read_next
T@10: | | | | | | | | >int handler::read_range_first
T@10: | | | | | | | | | >handler::ha_index_first
T@10: | | | | | | | | | | >int ha_innabase::index_first
T@10: | | | | | | | | | | >int ha_innabase::index_read
T@10: | | | | | | | | | | | >row_search_mvcc
...
T@10: | | | | | <int IndexRangeScanIterator::Read
T@10: | | | | | >bool Query_result_send::send_data
...
T@10: | | | | | | >bool Protocol_classic::end_row <= 发送数据结果
T@10: | | | | | | | net write: Memory: 0x7fff30006500 Bytes: (27)
05 31 30 30 30 31 06 47 65 6F 72 67 69 0A 31 39 35 33 2D 30 39 2D 30 32 02 37
31
```



结束

