# 第十二讲：MySQL 优化器功能概述及其观测工具

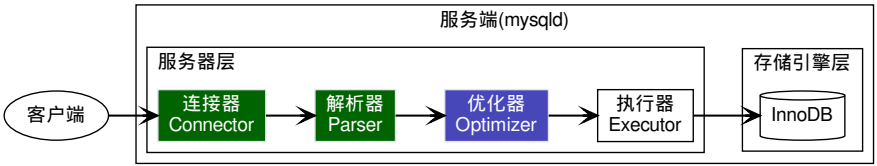
知春路遇上八里桥

<2024-06-15 Sat>

# 1

# 前情提要

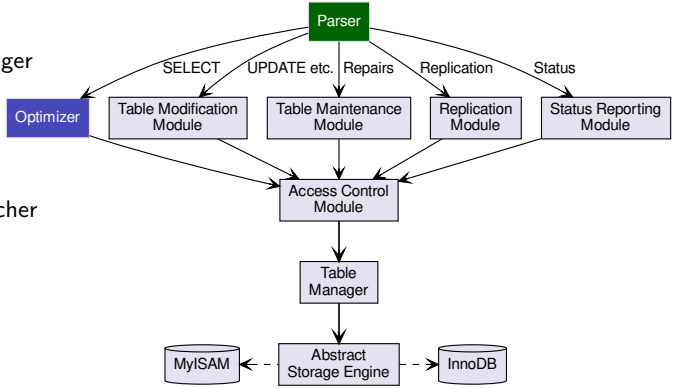
# 执行流程


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


MySQL
PostgreSQL

# 本节内容

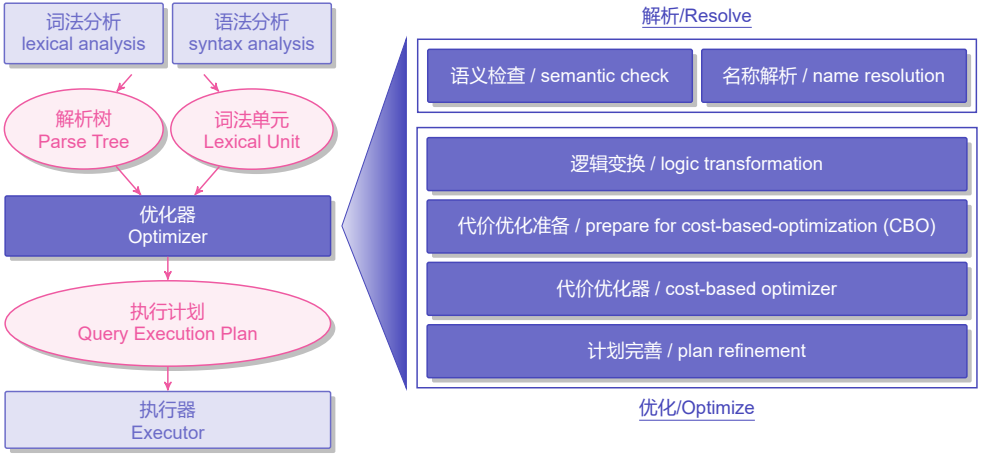
- 连接器
  - ☑ 连接管理器 Connection Manager
  - ☑ 线程管理器 Thread Manager
  - ☑ 用户模块 User Module
- 解析器
  - ☑ 网络模块 Net Module
  - ☑ 派发模块 Commander Dispatcher
  - ☑ 词法分析 Lexical Analysis
  - ☑ 语法分析 Syntax Analysis
- 优化器



2

# 优化器

# 优化器架构图


词法分析
lexical analysis
语法分析
syntax analysis
解析树
Parse Tree
词法单元
Lexical Unit
优化器
Optimizer
执行计划
Query Execution Plan
执行器
Executor
解析/Resolve
语义检查 / semantic check
名称解析 / name resolution
逻辑变换 / logic transformation
代价优化准备 / prepare for cost-based-optimization (CBO)
代价优化器 / cost-based optimizer
计划完善 / plan refinement
优化/Optimize

# Resolver

- 语义检查 (Semantic Check)
  - 通过数据字典如果找不到对应的表名，则直接返回报错

```
ERROR 1146 (42S02): Table 'employees.table1' doesn't exist
```

  - 表访问权限校验，在查询是如果用户只有数据库中部分表的权限是需要通过校验
    1. 设置权限

```
1 grant select on employees.departments to 'rtc'@'%';
2 flush privileges;
```

    2. 查看权限 [1]

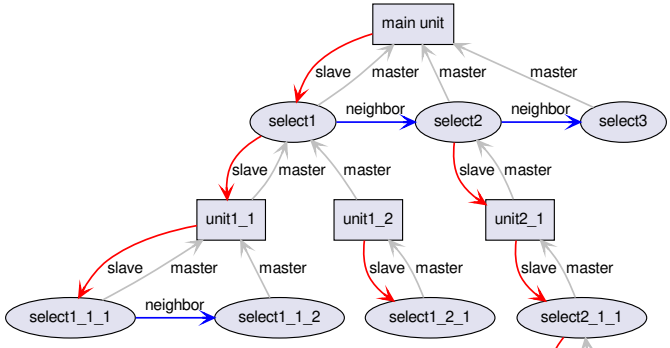
```
mysql> show grants\G
*************************** 1. row ***************************
Grants for rtc@%: GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD, SHUTDOWN,
PROCESS, FILE, REFERENCES, INDEX, ALTER, SHOW DATABASES, SUPER, CREATE TEMPORARY TABLES,
LOCK TABLES, EXECUTE, REPLICATION SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW,
CREATE ROUTINE, ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE TABLESPACE,
CREATE ROLE, DROP ROLE ON *.* TO `rtc`@`%`
1 rows in set (0.00 sec)
```

- 名称解析 (Name Resolution)
  - 主要包括将找到并补全对应语句的表名，库名等
  - 关联 `SELECT * FROM ...` 中 `*` 对应数据表的列名

[1] MySQL 的权限分成多个类别进行管理

# 语义检查举例

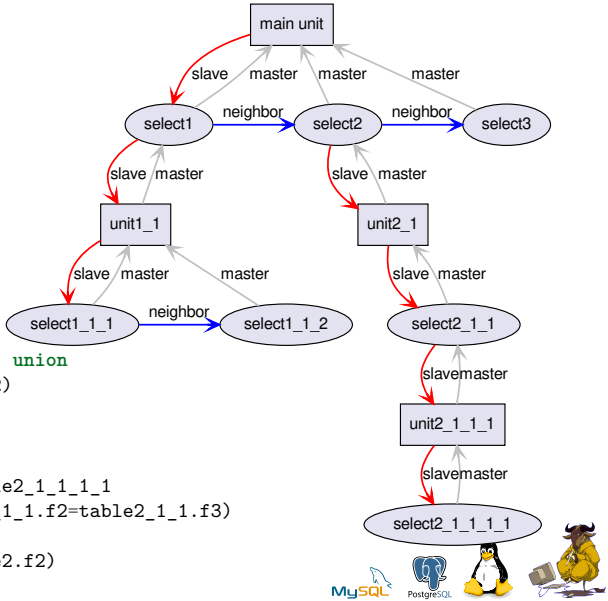
- unit1_2 是否正确?




```
select *
  from table1
  where table1.field IN (select * from table1_1_1 union
                         select * from table1_1_2)
  union
select *
  from table2
  where table2.field=(select (select f1 from table2_1_1_1_1
                              where table2_1_1_1_1.f2=table2_1_1.f3)
                      from table2_1_1
                      where table2_1_1.f1=table2.f2)
  union
select * from table3;
```

# 语义检查举例 (修复后)

```
select *
  from table1
  where table1.field IN (select * from table1_1_1 union
                         select * from table1_1_2)
  union
select *
  from table2
  where table2.field=(select (select f1 from table2_1_1_1_1
                              where table2_1_1_1_1.f2=table2_1_1.f3)
                      from table2_1_1
                      where table2_1_1.f1=table2.f2)
  union
select * from table3;
```



# 逻辑变换 (Logic transformation)

- 否定消除 (Negation elimination)
  - 对于多个表达式前面有否定的情况，将外面的 NOT 消除，例如
    1. 对于谓词 $p$ ，有 $p \Rightarrow \neg\neg p$
    2. 对于谓词 $p_1$ 和 $p_2$ ，有 $\neg(p_1 \vee p_2) \Rightarrow (\neg p_1) \wedge (\neg p_2)$
    3. 更多的参考离散数学或者数理逻辑中所介绍的知识
- 等值常量传递 (Equality and constant propagtion)
  - 利用了等值关系的传递特性，为了能够尽早执行下推运算
- 常量表达式计算 (Evaluation of constant expressions)
  - 对于能够立刻计算出结果的表达式，直接计算结果，并将结果与其他条件尽量提前化简
- 将 `OUTER JOIN` 转换成 `INNER JOIN` (Conversions of outer to inner join)
- 子查询变换 (Subquery transformation)
- MySQL 内部实现了很多逻辑变换并且也在持续更新中，请参考实际代码 ···

# 逻辑变换举例

```
select * from t1, t2 where
  t1.a=t2.a and t2.a=9 and (not (t1.a>10 or t2.b>3) or (t1.b=t2.b+7 and t2.b=5))
```

否定消除

```
select * from t1, t2 where
  t1.a=t2.a and t2.a=9 and (t1.a<=10 and t2.b<=3 or (t1.b=t2.b+7 and t2.b=5))
```

等值传递

```
select * from t1, t2 where
  t1.a= 9   and t2.a=9 and ( 9  <=10 and t2.b<=3 or (t1.b= 5  +7 and t2.b=5))
```

真值常量计算　　算术常量计算

```
select * from t1, t2 where
  t1.a= 9   and t2.a=9 and ( true    and t2.b<=3 or (t1.b= 13    and t2.b=5))
```

通过上述变换最终得到非常简单的查询语句

```
select * from t1, t2 where
  t1.a=9 and t2.a=9 and (t2.b<=3 or (t1.b=13 and t2.b=5))
```

# 代价优化准备 (Prepare for cost-based-optimization)

- 引用扫描分析 (Ref access analysis)
- 范围扫描分析 (Range access analysis) [2]
- 条件扇出估算 (Estimation of condition fan out)
  - 一个条件调用其他变量的个数，称为该条件的扇出
  - 扇出越大，计算该条件时需要考虑的问题就越多，因而复杂性越高
  - 下面是估算在给定 limit 时，需要从 outer table 中读取多少行数据的逻辑

```
11455  static double GetRowsNeededFromOuterTable(const AccessPath *join_path,
11456                                            const AccessPath *outer,
11457                                            double limit) {
11458    const double input_rows = outer->num_output_rows();
11459    const double output_rows = join_path->num_output_rows();
11460
11461    if (input_rows > 0 && output_rows > 0) {
11462      const double fanout = output_rows / input_rows;
11463      return ceil(limit / fanout);
11464    }
11465
11466    return kNoLimit;
11467  }
```

- 常量表检测 (Constant table detection)

[2] https://dev.mysql.com/doc/refman/8.0/en/range-optimization.html

# 代价优化器 (Cost-based Optimizer)

- 访问途径选取 (Access method selection)
  - 采用哪种索引一个表可能有主键，也可能有外键，需要根据条件确定使用哪个索引
- 确定 JOIN 次序 (Join order)
  - 不同的 JOIN 顺序对性能影响极大
  - 基本原则是小表驱动大表
- 确定子查询的执行策略 (Subquery)

# 计划完善 (Plan Refinement)

- 访问方式调整 (Access method adjustments)
- 排序避免 (Sort avoidance)
  - 排序对 CPU 和内存资源占用较大，优化器尽量避免排序操作
- 引擎条件下推 (Engine condition pushdown)
- 索引条件下推 (Index condition pushdown) [3]

[3] https://dev.mysql.com/doc/refman/8.0/en/index-condition-pushdown-optimization.html

3

# 观测工具

# Explain 工具

Explain 用于查看查询的执行计划，它包含列信息如下 [4]

- `id` 每个查询块都有一个唯一的标识符
- `select_type` 查询的类型（如 SIMPLE、SUBQUERY、UNION 等）
- `table` 输出行所引用的表的名称
- `type` 这是最重要的列之一，它描述了如何连接表。可能的值包括：
  - `system` 表只有一行（等于 const）
  - `const` 表中的一个匹配行是常量
  - `eq_ref` 所有部分都使用了唯一索引或主键
  - `ref` 非唯一索引或唯一索引的非唯一前缀
  - `fulltext` 使用全文索引
  - `ref_or_null` 与 ref 类似，但额外搜索了包含 NULL 值的行
  - `index_merge` 表示使用了索引合并优化方法
  - `unique_subquery` 用于 IN 子查询，该子查询返回唯一值
  - `index_subquery` 与 unique_subquery 类似，但返回非唯一值
  - `range` 对索引进行范围查找
  - `index` 全索引扫描
  - `ALL`（或 `full table scan`）全表扫描

[4] https://dev.mysql.com/doc/refman/8.0/en/explain.html

# Explain 工具 (贰)

- `possible_keys` 表示可能应用在这张表上的索引
  - 注意这并不意味着这些索引都将被实际使用
- `key` 实际使用的索引
- `key_len` 使用的索引的长度
  - 在不使用所有的索引列的情况下，可以计算该值
- `ref` 显示哪些列或常量被用作索引查找上的条件
- `rows` MySQL 估计为了找到所需的行而必须检查的
- `Extra` 包含不适合在其他列中显示的额外信息 [5]

```
mysql> explain select * from employees where emp_no < 10010;
+----+-------------+-----------+------------+-------+---------------+---------+---------+------+----------+-------------+
| id | select_type | table     | partitions | type  | possible_keys | key     | key_len | ref  | rows | filtered | Extra       |
+----+-------------+-----------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | employees | NULL       | range | PRIMARY       | PRIMARY | 4       | NULL |    9 |   100.00 | Using where |
+----+-------------+-----------+------------+-------+---------------+---------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select * from employees where first_name < 'a';
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra       |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------------+
|  1 | SIMPLE      | employees | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 299556 |    33.33 | Using where |
+----+-------------+-----------+------------+------+---------------+------+---------+------+--------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

[5] https://dev.mysql.com/doc/refman/8.0/en/explain-output.html

# Explain 举例

- 执行的查询语句

```
explain format=json select * from employees where emp_no < 10010\G
```

- 输出的 JSON 结果

```
EXPLAIN: {
  "query_block": { // 当前的 Query Block
    "select_id": 1,
    "cost_info": {
      "query_cost": "2.81"  // cost 数值
    },
    "table": {  // 当前查询表信息
      "table_name": "employees",
      "access_type": "range",
      "possible_keys": [
        "PRIMARY"
      ],
      "key": "PRIMARY",
      "used_key_parts": [
        "emp_no" // 命中主键索引
      ],
      "key_length": "4",
      "rows_examined_per_scan": 9,
      "rows_produced_per_join": 9,
      "filtered": "100.00",
      "cost_info": { // 详细的 cost 信息
        "read_cost": "1.91",
        "eval_cost": "0.90",
        "prefix_cost": "2.81",
        "data_read_per_join": "1K"
      },
      "used_columns": [ // 当前表中使用到的列
        "emp_no",
        "birth_date",
        "first_name",
        "last_name",
        "gender",
        "hire_date"
      ],
      "attached_condition": "(`employees`.`employees`.`emp_no` < 10010)"
    }
  }
}
1 row in set, 1 warning (0.00 sec)
```

# Optimizer Trace

Optimizer Trace [6] 可以帮助理解优化器如何为查询选择执行计划，它注意功能如下

1. 跟踪优化器在查询过程中所做的各种决策，例如
   - 逻辑转换过程
   - 访问表的方法
   - Cost 值计算过程的数值
2. 查看执行计划过程：通过 Optimizer Trace，用户可以查看优化器生成执行计划的整个过程
3. 优化 SQL 性能：基于跟踪结果，用户可以更好地优化 SQL 查询性能
4. 开启 Optimizer Trace 会影响 MySQL 的性能，因此只适合临时分析 SQL 使用，用完之后最好及时关闭
5. Explain 和 Optimizer Trace 都是用于分析查询性能的工具，但它们的侧重点不同
   - Explain 主要用于查看查询的执行计划
   - Optimizer Trace 则更深入地跟踪优化器的决策过程


[6] https://dev.mysql.com/doc/refman/8.0/en/information-schema-optimizer-trace-table.html

# Optimizer Trace 使用步骤

```
-- 开启 trace
set optimizer_trace="enabled=on";

-- 执行 Query
select emp_no from employees where emp_no < 11111;

-- Trace 日志表结构
```

```
mysql> desc information_schema.optimizer_trace;
+--------------------------------+----------------+------+-----+---------+-------+
| Field                          | Type           | Null | Key | Default | Extra |
+--------------------------------+----------------+------+-----+---------+-------+
| QUERY                          | varchar(65535) | NO   |     |         |       |
| TRACE                          | varchar(65535) | NO   |     |         |       |
| MISSING_BYTES_BEYOND_MAX_MEM_SIZE | int            | NO   |     |         |       |
| INSUFFICIENT_PRIVILEGES        | tinyint(1)     | NO   |     |         |       |
+--------------------------------+----------------+------+-----+---------+-------+
4 rows in set (0.03 sec)
```

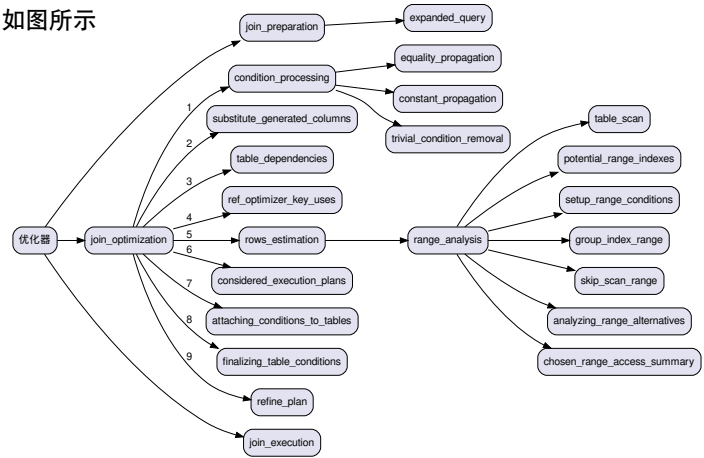
```
-- 查看结果
select * from information_schema.optimizer_trace\G
select trace from information_schema.optimizer_trace\G
```

# Optimizer Trace 举例

- 查询的语句示例

```
select * from employees where emp_no < 10010 and first_name like 'a%';
```

- 主要优化步骤 [7] 如图所示




[7] https://github.com/Jeanhwea/mysql-source-course/blob/master/assets/p12-opttrace-01.json

# 结束