第十七讲：子查询优化器的设计与实现

知春路遇上八里桥

<2024-07-04 Thu>

1. 前情提要

2. 子查询

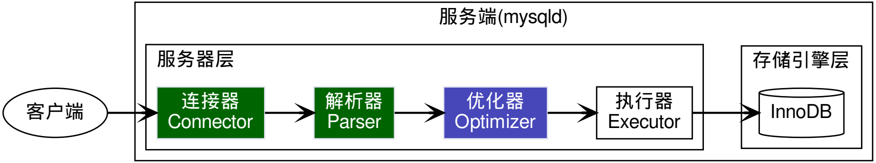
3. 半连接

4. 其他类型

1

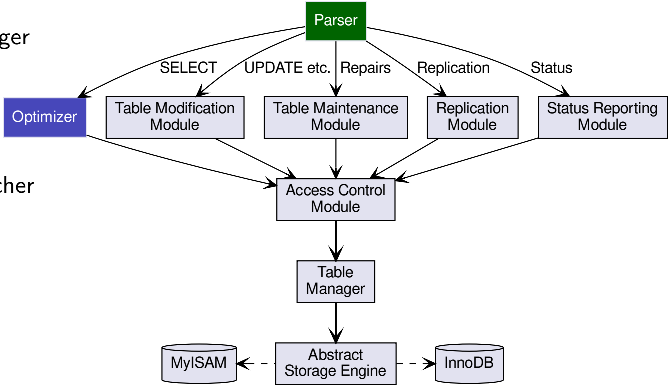
前情提要

执行流程


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

子查询

子查询介绍

- 子查询 (Subquery) 是一种嵌套在另一个外部查询中的查询 [1]
- 子查询有以下几种常见的形式
  - 标量子查询 (Scalar Subquery) 指只返回一条数据的子查询
  - 带有比较操作符号 (= <> > < >= <=) 的子查询

    ```
    select * from t1 where c1 = (select max(c2) from t2);
    ```
  - 带 IN/ANY/SOME/ALL 的子查询

    ```
    select s1 from t1 where s1 = any(select s1 from t2);
    select s1 from t1 where s1 in (select s1 from t2);
    select s1 from t1 where s1 > all(select s1 from t2);
    ```
  - 带 EXISTS/NOT EXISTS 的子查询

    ```
    select * from t1 where exists (select * from t2);
    select * from t1 where not exists (select * from t2);
    ```
  - 派生表 (Derived Table) 是在查询过程中由子查询生成的虚拟表

    ```
    select avg(sum1) from (select sum(c1) as sum1 from t1 group by c1) as t1;
    ```
- 子查询优化的示例 [2]
  - 例如：内部查询使用 union 合并，避免使用 or 连接两个子查询

[1] https://dev.mysql.com/doc/refman/8.0/en/subqueries.html

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

标量子查询 subq01.sql

```
mysql> select * from departments;
+---------+--------------------+
| dept_no | dept_name          |
+---------+--------------------+
| d009    | Customer Service   |
| d005    | Development        |
| d002    | Finance            |
| d003    | Human Resources    |
| d001    | Marketing          |
| d004    | Production         |
| d006    | Quality Management |
| d008    | Research           |
| d007    | Sales              |
+---------+--------------------+
9 rows in set (0.00 sec)

mysql> select * from dept_emp limit 3;
+--------+---------+------------+------------+
| emp_no | dept_no | from_date  | to_date    |
+--------+---------+------------+------------+
|  10001 | d005    | 1986-06-26 | 9999-01-01 |
|  10002 | d007    | 1996-08-03 | 9999-01-01 |
|  10003 | d004    | 1995-12-03 | 9999-01-01 |
+--------+---------+------------+------------+
3 rows in set (0.01 sec)
```

- 采用子查询统计员工数，同时满足下面条件
  - 员工号小于 10100
  - 部门属于研发 (Development)

```sql
select
  count(distinct emp_no) as emp_cnt
from
  dept_emp a
where
  a.emp_no < 10100
  and a.dept_no = (
    select
      d.dept_no
    from
      departments d
    where
      d.dept_name = 'Development');
```

标量子查询分析

select#1 中的附属条件使用到标量表

```
{"query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "6.69"
    },
    "table": {
      "table_name": "a",
      "access_type": "range",
      "possible_keys": ["PRIMARY", "dept_no"],
      "key": "dept_no",
      "used_key_parts": ["dept_no", "emp_no"],
      "key_length": "20",
      "rows_examined_per_scan": 32,
      "rows_produced_per_join": 32,
      "filtered": "100.00",
      "using_index": true,
      ...
      "used_columns": ["emp_no", "dept_no"],
      "attached_condition": "((`employees`.`a`.`dept_no` =
      (/* select#2 */ select 'd005' from
        `employees`.`departments` `d` where true))
      and (`employees`.`a`.`emp_no` < 10100))",
```

access_type 为 const 表示 select#2 是标量表

```
{"attached_subqueries": [{
    "dependent": false,
    "cacheable": true,
    "query_block": {
      "select_id": 2,
      "cost_info": {
        "query_cost": "1.00"
      },
      "table": {
        "table_name": "d",
        "access_type": "const",
        "possible_keys": ["dept_name"],
        "key": "dept_name",
        "used_key_parts": ["dept_name"],
        "key_length": "162",
        "ref": ["const"],
        "rows_examined_per_scan": 1,
        "rows_produced_per_join": 1,
        "filtered": "100.00",
        "using_index": true,
        ...
```

唯一性的 IN 子查询 subq02.sql

```
mysql> select emp_no, first_name, last_name
    ->   from employees limit 3;
+--------+------------+-----------+
| emp_no | first_name | last_name |
+--------+------------+-----------+
|  10001 | Georgi     | Facello   |
|  10002 | Bezalel    | Simmel    |
|  10003 | Parto      | Bamford   |
+--------+------------+-----------+
3 rows in set (0.00 sec)

mysql> select * from dept_manager limit 3;
+--------+---------+------------+------------+
| emp_no | dept_no | from_date  | to_date    |
+--------+---------+------------+------------+
| 110022 | d001    | 1985-01-01 | 1991-10-01 |
| 110039 | d001    | 1991-10-01 | 9999-01-01 |
| 110085 | d002    | 1985-01-01 | 1989-12-17 |
+--------+---------+------------+------------+
3 rows in set (0.01 sec)
```

- 查询 d001 部门所有经理的姓名

```sql
select
  e.first_name, e.last_name
from
  employees e
where
  e.emp_no in (
    select
      a.emp_no
    from
       dept_manager a
    where
      a.dept_no = 'd001');
```

非唯一性的 IN 子查询 subq03.sql

```
mysql> select * from departments limit 3;
+---------+------------------+
| dept_no | dept_name        |
+---------+------------------+
| d009    | Customer Service |
| d005    | Development      |
| d002    | Finance          |
+---------+------------------+
3 rows in set (0.01 sec)

mysql> select * from dept_manager limit 3;
+--------+---------+------------+------------+
| emp_no | dept_no | from_date  | to_date    |
+--------+---------+------------+------------+
| 110022 | d001    | 1985-01-01 | 1991-10-01 |
| 110039 | d001    | 1991-10-01 | 9999-01-01 |
| 110085 | d002    | 1985-01-01 | 1989-12-17 |
+--------+---------+------------+------------+
3 rows in set (0.01 sec)
```

- 查询 1995 年以来上任过经理的部门

```sql
select
  *
from
  departments d
where
  d.dept_no in (
    select
      a.dept_no
    from
      dept_manager a
    where
      a.from_date >= '1995-01-01');
```

IN 子查询转换半连接

- join_preparation 阶段会将 IN (SELECT) 转换成 semijoin
- subq02.sql 有唯一键走索引

```sql
select
  `e`.`first_name` as `first_name`,
  `e`.`last_name` as `last_name`
from
  `employees` `e` semi join (`dept_manager` `a`)
where ((`a`.`dept_no` = 'd001') and (`e`.`emp_no` = `a`.`emp_no`))
```

- subq03.sql 非唯一键构造物化表

```sql
select
  `d`.`dept_no` as `dept_no`,
  `d`.`dept_name` as `dept_name`
from
  `departments` `d` semi join (`dept_manager` `a`)
where ((`a`.`from_date` >= '1995-01-01')
  and (`d`.`dept_no` = `a`.`dept_no`))
```

EXISTS 子查询 subq04.sql

两个子查询的含义是一样的，最终都会变换成半连接进入 join_optimization 阶段

- 带有 IN 的子查询

```sql
select
  e.first_name,
  e.last_name
from
  employees e
where
  e.emp_no in (
    select
      a.emp_no
    from
      dept_manager a
    where
      a.dept_no = 'd001');

```

- 带有 EXISTS 的子查询

```sql
select
  e.first_name,
  e.last_name
from
  employees e
where
  exists (
    select
      *
    from
      dept_manager a
    where
      a.dept_no = 'd001'
      and a.emp_no = e.emp_no);
```

EXISTS 子查询变换过程

- EXISTS 子查询通过 `join_preparation` 转换成半连接，具体阶段如下：
  - `transformation_to_semi_join`
  - `transformations_to_nested_joins`

```
{
  "transformation": {
    "select#": 2,
    "from": "IN (SELECT)",
    "to": "semijoin",
    "chosen": true,
    "transformation_to_semi_join": {
      "subquery_predicate": "exists(/* select#2 */ select 1 from `dept_manager` `a`
         where ((`a`.`dept_no` = 'd001') and (`a`.`emp_no` = `e`.`emp_no`)))",
      "embedded in": "WHERE",
      "evaluating_constant_semijoin_conditions": [],
      "semi-join condition": "((`a`.`dept_no` = 'd001') and (`e`.`emp_no` = `a`.`emp_no`))",
      "decorrelated_predicates": [{"outer": "`e`.`emp_no`", "inner": "`a`.`emp_no`"}]}}},
{
  "transformations_to_nested_joins": {
    "transformations": [
      "semijoin"
    ],
    "expanded_query": "/* select#1 */ select `e`.`first_name` AS `first_name`,
      `e`.`last_name` AS `last_name` from
      `employees` `e` semi join (`dept_manager` `a`)
       where ((`a`.`dept_no` = 'd001') and (`e`.`emp_no` = `a`.`emp_no`))"}}
```

3

半连接

半连接

- 半连接（Semi-Join）是一种数据库的查询方式
  - 半连接用来从两个或多个表中获取匹配的行
  - 但只返回第一个表中的行，而不包括第二个表中的任何列或重复的行
- MySQL 不支持直接使用半连接查询，但是以下两种查询会转换成半连接[3]
  - EXISTS 子查询
  - 包含 IN 的子句
- MySQL 提供了多种半连接优化策略[4]，这些策略通过优化器自动选择，具体如下
  - DuplicateWeedout（重复剔除）
  - FirstMatch（首次匹配）
  - LooseScan（松散扫描）
  - MaterializeLookup（索引式物化）
  - MaterializeScan（扫描式物化）

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

[4] https://mariadb.com/kb/en/semi-join-subquery-optimizations/

半连接举例

以表 t1 和 t2 进行半连接操作，其中 t1 是外表，t2 是内表

表 t1 和 t2 中的数据

```
mysql> select * from t1;
+------+------+
| id   | c    |
+------+------+
|    1 | aaa  |
|    2 | bbb  |
|    4 | ccc  |
|    4 | ddd  |
+------+------+

mysql> select * from t2;
+------+------+
| id   | c    |
+------+------+
|    1 | xxx  |
|    3 | yyy  |
|    4 | uuu  |
|    4 | vvv  |
+------+------+
```

- 内连接 innerjoin

```
mysql> select * from t1 inner join t2 using(id);
+------+------+------+
| id   | c    | c    |
+------+------+------+
|    1 | aaa  | xxx  |
|    4 | ddd  | uuu  |
|    4 | ccc  | uuu  |
|    4 | ddd  | vvv  |
|    4 | ccc  | vvv  |
+------+------+------+
```

- 半连接 semijoin，仅返回单表，去重

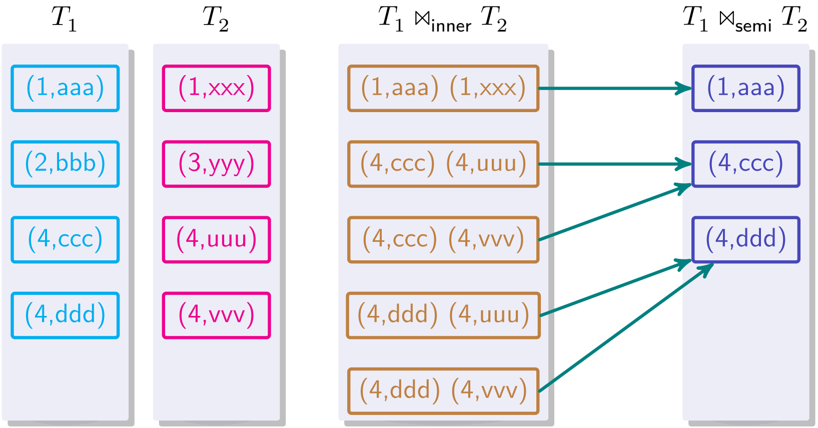
```
mysql> select * from t1 semi join t2 using(id);
mysql> select * from t1 where t1.id in (select id from t2);
+------+-----+
| id   | c   |
+------+-----+
|    1 | aaa |
|    4 | ddd |
|    4 | ccc |
+------+-----+
```

半连接算法

MySQL 提供了多种半连接优化策略，这些策略通过优化器自动选择，具体如下

1. DuplicateWeedout（重复剔除）
   - 执行普通的两表内连接操作，用临时表缓存结果
   - 并在临时表中通过主键或唯一索引去除重复的元组
2. FirstMatch（首次匹配）
   - 扫描内部表的行组合
   - 当有多个实例的给定值组时，只选择一个而不是全部返回
   - 这种快速扫描并消除了不必要的行的生成
3. LooseScan（松散扫描）
   - 在执行连接时，如果内表的元组有序（通常通过索引实现）
   - 则可以根据索引拿出每组重复元组中的第一个元组与外表进行连接，跳过后续相同的记录
4. MaterializeLookup（索引式物化）
   - 将子查询的结果物化到临时表，并为其创建索引
   - 执行连接时，使用临时表的索引来完成连接操作
5. MaterializeScan（扫描式物化）
   - 类似于 MaterializeLookup，但临时表的索引不能辅助加快连接
   - 只能通过全表扫描的方式完成半连接操作

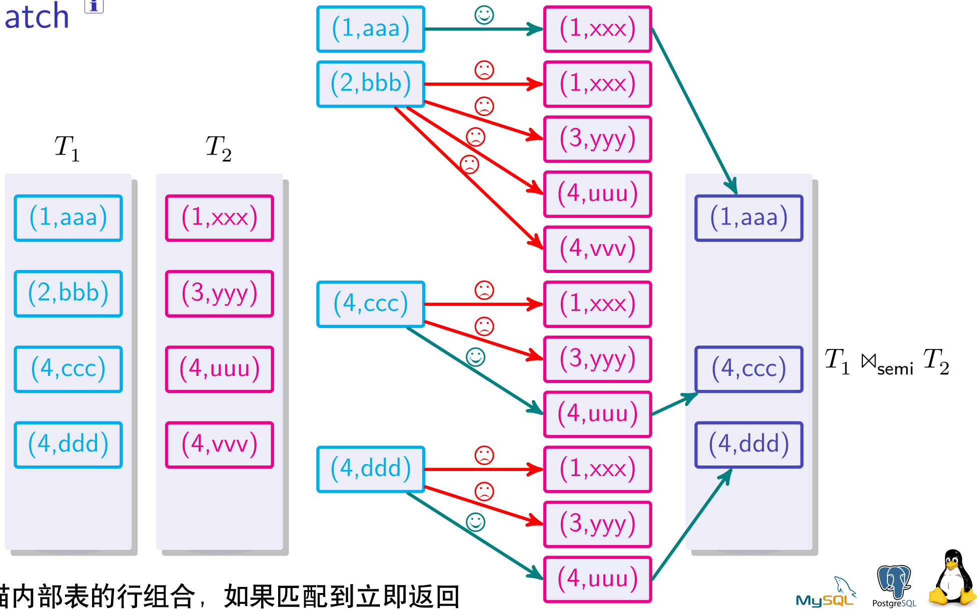
DuplicateWeedout



T_1	T_2	$T_1 \bowtie_{\text{inner}} T_2$	$T_1 \bowtie_{\text{semi}} T_2$
(1,aaa)	(1,xxx)	(1,aaa) (1,xxx)	(1,aaa)
(2,bbb)	(3,yyy)	(4,ccc) (4,uuu)	(4,ccc)
(4,ccc)	(4,uuu)	(4,ccc) (4,vvv)	(4,ddd)
(4,ddd)	(4,vvv)	(4,ddd) (4,uuu)	
		(4,ddd) (4,vvv)	

1. 先计算表 T_1 和 T_2 的内连接 $T_1 \bowtie_{\text{inner}} T_2$，放入临时表
2. 将临时表去重得到最终结果半连接表 $T_1 \bowtie_{\text{semi}} T_2$

FirstMatch



T_1

(1,aaa)
(2,bbb)
(4,ccc)
(4,ddd)

T_2

(1,xxx)
(3,yyy)
(4,uuu)
(4,vvv)

(1,aaa) ☺ → (1,xxx)

(2,bbb) ☹ → (1,xxx)
(2,bbb) ☹ → (3,yyy)
(2,bbb) ☹ → (4,uuu)
(2,bbb) ☹ → (4,vvv)

(4,ccc) ☹ → (1,xxx)
(4,ccc) ☹ → (3,yyy)
(4,ccc) ☺ → (4,uuu)

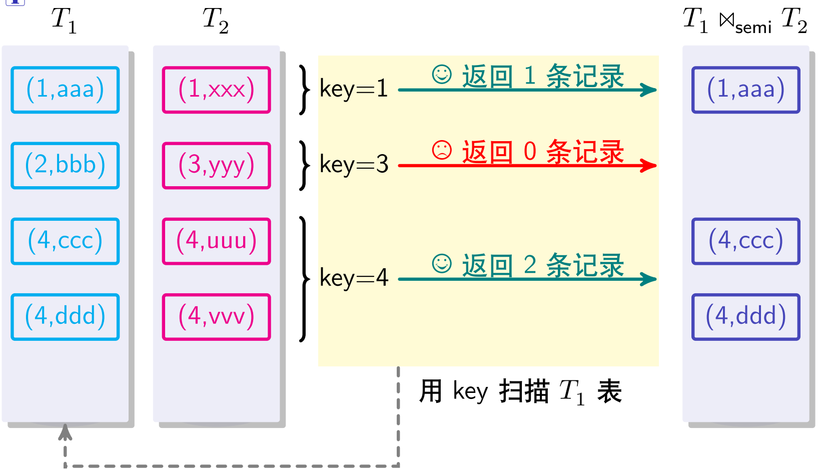
(4,ddd) ☹ → (1,xxx)
(4,ddd) ☹ → (3,yyy)
(4,ddd) ☺ → (4,uuu)

$T_1 \bowtie_{\text{semi}} T_2$

(1,aaa)
(4,ccc)
(4,ddd)

- 扫描内部表的行组合，如果匹配到立即返回

LooseScan



T_1

(1,aaa)

(2,bbb)

(4,ccc)

(4,ddd)

T_2

(1,xxx)

(3,yyy)

(4,uuu)

(4,vvv)

key=1 ☺ 返回 1 条记录

key=3 ☹ 返回 0 条记录

key=4 ☺ 返回 2 条记录

用 key 扫描 T_1 表

$T_1 \ltimes_{\text{semi}} T_2$

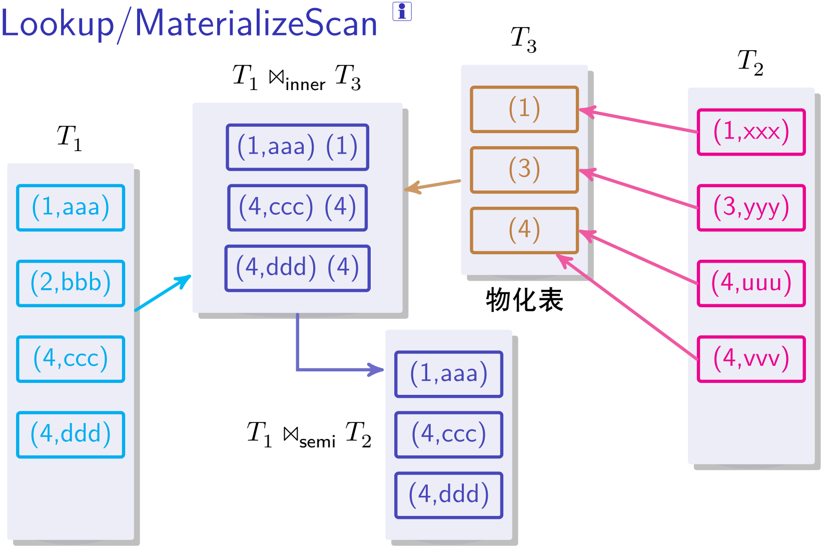
(1,aaa)

(4,ccc)

(4,ddd)

1. 在执行连接时，如果内表的元组有序（通常通过索引实现）
2. 则可以根据索引拿出每组重复元组中的第一个元组与外表进行连接
3. 跳过后续相同的记录

MaterializeLookup/MaterializeScan



T_1

(1,aaa)

(2,bbb)

(4,ccc)

(4,ddd)

$T_1 \bowtie_{\text{inner}} T_3$

(1,aaa) (1)

(4,ccc) (4)

(4,ddd) (4)

T_3

(1)

(3)

(4)

物化表

T_2

(1,xxx)

(3,yyy)

(4,uuu)

(4,vvv)

$T_1 \ltimes_{\text{semi}} T_2$

(1,aaa)

(4,ccc)

(4,ddd)

- 将 T_2 物化到临时表 T_3，然后计算 T_1 内连接 T_3 得到结果
- MaterializeLookup 对 T_2 创建索引，MaterializeScan 不建索引直接扫描

不同策略 Cost 值对比

- 通过 hints 可以控制选取对于的半连接策略 [5]
- 下面是两个查询优化值的对比

- subq10 查询 ℹ

```
select * from departments d where
  d.dept_no in (
    select a.dept_no
    from dept_manager a
    where a.from_date >= '1995-01-01');
```

- subq10 策略对比

策略	Cost	说明
DuplicateWeedout	8.05	
FirstMatch	8.95	
LooseScan	-	不支持
Materialize	6.50	命中 ☺

- subq20 查询 ℹ

```
select * from employees e where
  e.emp_no in (
    select a.emp_no
    from dept_manager a
    where a.dept_no >= 'd003')
```

- subq20 策略对比

策略	Cost	说明
DuplicateWeedout	16.59	
FirstMatch	124440.10	代价太大
LooseScan	10.39	命中 ☺
Materialize	16.59	

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

优化器配置开关

- 优化器配置开关默认值通过数据库系统变量来控制

```
mysql> show variables like '%optimizer_switch%'\G
*************************** 1. row ***************************
Variable_name: optimizer_switch
        Value: index_merge=on, index_merge_union=on, index_merge_sort_union=on,
               index_merge_intersection=on, engine_condition_pushdown=on,
               index_condition_pushdown=on, mrr=on, mrr_cost_based=on,
               block_nested_loop=on, batched_key_access=off, materialization=on,
               semijoin=on, loosescan=on, firstmatch=on, duplicateweedout=on,
               subquery_materialization_cost_based=on, use_index_extensions=on,
               condition_fanout_filter=on, derived_merge=on, use_invisible_indexes=off,
               skip_scan=on, hash_join=on, subquery_to_derived=off,
               prefer_ordering_index=on, hypergraph_optimizer=off,
               derived_condition_pushdown=on
1 row in set (0.02 sec)
```

- 可以使用下面方式开启/关闭

```
set optimizer_switch='semijoin=on'
set optimizer_switch='semijoin=off'
```

半连接优化函数

- 半连接优化决策函数 advance_sj_state()，计算不同半连接策略的 cost 值
  - ☞ sql/sql_planner.cc

```
void Optimize_table_order::advance_sj_state(table_map remaining_tables,
⋮
  if (sj_strategy != SJ_OPT_NONE)
    pos->set_prefix_cost(best_cost, best_rowcount);
}
```

- 半连接优化计算伪代码，对每种策略分别计算，最后保存最优

```
for each semi-join strategy {
  update strategy's state variables;
  if (join prefix has all the tables that are needed to consider
      using this strategy for the semi-join(s)) {
    calculate cost of using the strategy
    if ((this is the first strategy to handle the semi-join nest(s)  ||
        the cost is less than other strategies)) {
      // Pick this strategy
      pos->sj_strategy= ..
      ..
    }
  }
}
```

4

其他类型

NOT IN 子查询 subq05.sql

```
mysql> select * from departments limit 3;
+---------+------------------+
| dept_no | dept_name        |
+---------+------------------+
| d009    | Customer Service |
| d005    | Development      |
| d002    | Finance          |
+---------+------------------+
3 rows in set (0.01 sec)

mysql> select * from dept_manager limit 3;
+--------+---------+------------+------------+
| emp_no | dept_no | from_date  | to_date    |
+--------+---------+------------+------------+
| 110022 | d001    | 1985-01-01 | 1991-10-01 |
| 110039 | d001    | 1991-10-01 | 9999-01-01 |
| 110085 | d002    | 1985-01-01 | 1989-12-17 |
+--------+---------+------------+------------+
3 rows in set (0.01 sec)
```

- 使用 NOT IN 的查询语句会被变化成 antijoin

```
select
  *
from
  departments d
where
  d.dept_no not in (
    select
      a.dept_no
    from
      dept_manager a
    where
      a.from_date > '1995-01-01');
```

派生表 subq06.sql

```
mysql> select emp_no, first_name, last_name
    ->   from employees limit 3;
+--------+------------+-----------+
| emp_no | first_name | last_name |
+--------+------------+-----------+
|  10001 | Georgi     | Facello   |
|  10002 | Bezalel    | Simmel    |
|  10003 | Parto      | Bamford   |
+--------+------------+-----------+
3 rows in set (0.00 sec)

mysql> select * from titles limit 3;
+--------+-----------------+------------+------------+
| emp_no | title           | from_date  | to_date    |
+--------+-----------------+------------+------------+
|  10001 | Senior Engineer | 1986-06-26 | 9999-01-01 |
|  10002 | Staff           | 1996-08-03 | 9999-01-01 |
|  10003 | Senior Engineer | 1995-12-03 | 9999-01-01 |
+--------+-----------------+------------+------------+
3 rows in set (0.01 sec)
```

- 使用派生表使用 merge 策略来优化

```sql
select
  *
from
  titles t,
  (
    select
      *
    from
      employees
    where
      emp_no <= 10100) emp_100
where
  t.emp_no = emp_100.emp_no
  and t.from_date > '2000-01-01';
```

结束