# SQL 优化对比：驱动表 vs Hash 关联

原创 何文超 爱可生开源社区 2025年07月02日 16:30 江苏


作者：何文超，分享 MySQL 和 OceanBase 相关技术博文。 个人博客【CSDN | 雅俗数据库】



本文约 2000 字，预计阅读需要 6 分钟。


## 1. 问题背景

### 1.1 问题描述

在 SQL 优化的过程中，经常会通过 **指定驱动表** 或 **修改表的关联方式** 来实现。下面将以案例的形式来介绍他们的不同之处以及使用场景需要满足的条件。

**SQL 耗时：11.25s**

```
-- SQL中 IN 的条件很多，内容中已简化
SELECTDISTINCT
    STORE_ID,
    PRODUCT_ID
FROM (
    SELECTDISTINCT
        ASP.PRODUCT_ID,
        t.STORE_ID
    FROM
        CT_ACT A
    JOIN
        CT_ACT_STAGE CAS ON A.ACT_ID = CAS.ACT_ID
    JOIN
    JOIN
        CT_ACT_STAGE_PRODUCT ASP ON CAS.STAGE_ID = ASP.STAGE_ID
            AND ASP.PRODUCT_STATUS = '1'
            AND ASP.PRODUCT_ID IN (
                '10185219',
                '10382854'
            )
    JOIN
        CT_STORE_PRODUCT_REL t ON ASP.PRODUCT_ID = t.PRODUCT_ID
            AND t.RELATIONSHIP_STATUS = '01'
            AND t.STORE_ID IN (
                '299800000149313',
                '299800000148811',
                'a2f162ae0fbe47c9b7b762ed27deb9b1',
```

```
                '7787a5cb102744088f46b381ee667fd9'
            )
);
```

### 1.2 查看执行计划

```
==========================================================================================
|ID|OPERATOR            |NAME                                                  |EST. ROWS
------------------------------------------------------------------------------------------
|0 |MERGEDISTINCT       |                                                      |1        |
|1 | SORT               |                                                       |1
|2 |  NESTED-LOOPJOIN   |                                                      |1        |
|3 |   NESTED-LOOPJOIN  |                                                      |1        |
|4 |    NESTED-LOOPJOIN |                                                      |1        |
|5 |     TABLESCAN      |ASP(IDX_CT_ACT_STAGE_PRODUCT_PRODUCT_ID)              |1        |
|6 |     TABLEGET       |CAS                                                   |1        |
|7 |    TABLESCAN       |t(IDX_CT_STORE_PRODUCT_REL_STATUS_STORE_ID_PRODUCT_ID)|1        |
|8 |   TABLESCAN        |A(UK_CT_ACT_ACT_ID)                                   |1        |
==========================================================================================
OutlineData:
      BEGIN_OUTLINE_DATA
      NO_USE_HASH_AGGREGATION(@"SEL$2")
      LEADING(@"SEL$2" (((("nctmbasedb.ASP"@"SEL$2""nctmbasedb.CAS"@"SEL$2" )"nctmbasedb.t
      USE_NL(@"SEL$2" ("nctmbasedb.A"@"SEL$2" ))
      PQ_DISTRIBUTE(@"SEL$2" ("nctmbasedb.A"@"SEL$2" ) LOCALLOCAL)
      NO_USE_NL_MATERIALIZATION(@"SEL$2" ("nctmbasedb.A"@"SEL$2" ))
      USE_NL(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ))
      PQ_DISTRIBUTE(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ) LOCALLOCAL)
      NO_USE_NL_MATERIALIZATION(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ))
      USE_NL(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ))
      PQ_DISTRIBUTE(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ) LOCALLOCAL)
      NO_USE_NL_MATERIALIZATION(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ))
      INDEX(@"SEL$2""nctmbasedb.ASP"@"SEL$2""IDX_CT_ACT_STAGE_PRODUCT_PRODUCT_ID")
      FULL(@"SEL$2""nctmbasedb.CAS"@"SEL$2")
      INDEX(@"SEL$2""nctmbasedb.t"@"SEL$2""IDX_CT_STORE_PRODUCT_REL_STATUS_STORE_ID_PRODU
      INDEX(@"SEL$2""nctmbasedb.A"@"SEL$2""UK_CT_ACT_ACT_ID")
      END_OUTLINE_DATA
Optimization Info:
------------------------------------
ASP:table_rows:9737755, physical_range_rows:1, logical_range_rows:1, index_back_rows:1, o
CAS:table_rows:116467, physical_range_rows:1, logical_range_rows:1, index_back_rows:0, ou
t:table_rows:6563720, physical_range_rows:1, logical_range_rows:1, index_back_rows:0, out
A:table_rows:9912, physical_range_rows:1, logical_range_rows:1, index_back_rows:0, output
```

### 1.3 检查表数据量

| 表名 | 别名 | 数据量 |
| --- | --- | --- |
| CT_ACT | A | 9912 |
| CT_ACT_STAGE | CAS | 116467 |
| CT_ACT_STAGE_PRODUCT | ASP | 9737755 |
| CT_STORE_PRODUCT_REL | t | 6563720 |

### 1.4 分析过程

- 表都走了索引， CAS 表走的主键索引，故先不检查关联字段情况
- ASP 表是驱动表，大表作为驱动表，非最优
- 两张大表： ASP 、 t 表， where 条件过滤性都较高

**查看 Outline Data:**

- USE_NL(@"SEL$2" ("nctmbasedb.A"@"SEL$2" ))
- USE_NL(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ))
- USE_NL(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ))

注意： ASP 表是驱动表，所以不显示关联。

```
LEADING(@"SEL$2" (("nctmbasedb.ASP"@"SEL$2" "nctmbasedb.CAS"@"SEL$2" )"nctmbasedb.t"@"SE
```

1. 首先是 ("nctmbasedb.ASP"@"SEL$2" "nctmbasedb.CAS"@"SEL$2" ) ，这表明优化器应该先对 nctmbasedb 模式下的 ASP 表和 CAS 表进行连接操作。
2. 接着是 ("nctmbasedb.t"@"SEL$2" ("nctmbasedb.ASP"@"SEL$2" "nctmbasedb.CAS"@"SEL$2" )) ，意味着将 nctmbasedb 模式下的 t 表与前面连接好的 ASP 表和 CAS 表的结果进行连接。
3. 最后是 ((("nctmbasedb.t"@"SEL$2" ("nctmbasedb.ASP"@"SEL$2" "nctmbasedb.CAS"@"SEL$2" ))"nctmbasedb.A"@"SEL$2" ) ，也就是把 nctmbasedb 模式下的 A 表和前面连接得到的结果再进行连接。

**小结**

- A ， t ， CAS 三张表走的 LNESTED-LOOP JOIN 关联，其中 A 表, CAS 表数据量不大， NLJ 关联 符合预期；
- t 表是大表，且 where 过滤条件中， t.STORE_ID 是有效的过滤条件，故考虑让 t 表走 hash 关联；
- ASP 表是驱动表，大表作为驱动表，非最优。

## 2. SQL 优化

### 2.1 方案一：指定小表（A表）为驱动表

#### 2.1.1 指定驱动表

```
/*+leading(A) use_nl(A,CAS,ASP,t) */
```

SQL 执行时间超过 30s，人为中断。

执行计划等价于：

```
LEADING(@"SEL$2" (("nctmbasedb.A"@"SEL$2" "nctmbasedb.ASP"@"SEL$2" )"nctmbasedb.CAS"@"SE
USE_NL(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ))
USE_NL(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ))
USE_NL(@"SEL$2" ("nctmbasedb.ASP"@"SEL$2" ))
```

- 表 t 是大表，走了 USE_NL 关联，故 SQL 执行超时；
- 且表 A 虽然是小表，但是无直接 WHERE 过滤条件，故不能通过索引快速匹配，不适合作为驱动表；

#### 2.1.2 查看执行计划

```
==========================================================================================
|ID|OPERATOR |NAME |EST. ROWS|COST |
------------------------------------------------------------------------------------------
|0 |MERGEDISTINCT | |1 |127753|
|1 | SORT | |1 |127753|
|2 | NESTED-LOOPJOIN | |1 |127753|
|3 | NESTED-LOOPJOIN | |1 |127474|
|4 | NESTED-LOOPJOIN CARTESIAN| |4956 |5911 |
|5 | TABLESCAN |A(UK_CT_ACT_ACT_ID) |9912 |3835 |
|6 | MATERIAL | |1 |264 |
|7 | TABLESCAN |ASP(IDX_CT_ACT_STAGE_PRODUCT_PRODUCT_ID) |1 |264 |
|8 | TABLEGET |CAS |1 |24 |
|9 | TABLESCAN |t(IDX_CT_STORE_PRODUCT_REL_STATUS_STORE_ID_PRODUCT_ID)|1 |552 |
==========================================================================================

Used Hint:
------------------------------------
LEADING(@"SEL$2" ((("nctmbasedb.A"@"SEL$2" ))))
USE_NL(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ))
USE_NL(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ))
USE_NL(@"SEL$2" ("nctmbasedb.ASP"@"SEL$2" ))
```

### 2.2 方案二：指定大表（t表）为驱动表

#### 2.2.1 指定 t 表为驱动表

**执行耗时：0.19s**

```
/*+leading(t (ASP,CAS)A) use_nl(t,A,CAS,ASP) */
```

如上等价于如下：

```
LEADING(@"SEL$2" (("nctmbasedb.t"@"SEL$2" ("nctmbasedb.ASP"@"SEL$2" "nctmbasedb.CAS"@"SEL
USE_NL(@"SEL$2" ("nctmbasedb.A"@"SEL$2" ))
USE_NL(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ))
```

表 `t` 虽然是大表，但存在有效的过滤条件。

#### 2.2.2 查看执行计划

```
==========================================================================================
|ID|OPERATOR |NAME |EST. ROWS|COST|
------------------------------------------------------------------------------------------
|0 |MERGEDISTINCT | |1 |4624|
|1 | SORT | |1 |4624|
|2 | NESTED-LOOPJOIN | |1 |4624|
|3 | HASHJOIN | |1 |4612|
|4 | TABLESCAN |t(IDX_CT_STORE_PRODUCT_REL_STATUS_STORE_ID_PRODUCT_ID)|3414 |1848|
|5 | NESTED-LOOPJOIN| |1 |276 |
|6 | TABLESCAN |ASP(IDX_CT_ACT_STAGE_PRODUCT_PRODUCT_ID) |1 |264 |
|7 | TABLEGET |CAS |1 |24 |
|8 | TABLESCAN |A(UK_CT_ACT_ACT_ID) |1 |24 |
==========================================================================================

Used Hint:
------------------------------------
LEADING(@"SEL$2" (("nctmbasedb.t"@"SEL$2" ("nctmbasedb.ASP"@"SEL$2""nctmbasedb.CAS"@"SEL$
USE_NL(@"SEL$2" ("nctmbasedb.A"@"SEL$2" ))
USE_NL(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ))
```

### 2.3 方案三：修改表的关联方式

#### 2.3.1 改为 hash 关联方式

添加 `/*+use_hash(t,ASP) */` ，耗时 0.06S。

查看 Used Hint 发现:

```
/*+use_hash(t,ASP) */ 等价于 USE_HASH(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ))
```

```
可考虑加    /*+ USE_HASH(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ))*/                t 表大表
可考虑加    /*+ USE_HASH(@"SEL$2" ("nctmbasedb.ASP"@"SEL$2" ))*/          ASP 表大表，但是驱动表
可考虑加    /*+ USE_HASH(@"SEL$2" ("nctmbasedb.t"@"SEL$2" )) USE_HASH(@"SEL$2" ("nctmbasedb
```

#### 2.3.2 查看执行计划

```
==========================================================================================
|ID|OPERATOR |NAME |EST. ROWS|COST|
------------------------------------------------------------------------------------------
|0 |MERGEDISTINCT | |1 |4351|
|1 | SORT | |1 |4351|
|2 | HASHJOIN | |1 |4351|
|3 | NESTED-LOOPJOIN | |1 |289 |
|4 | NESTED-LOOPJOIN| |1 |276 |
|5 | TABLESCAN |ASP(IDX_CT_ACT_STAGE_PRODUCT_PRODUCT_ID) |1 |264 |
|6 | TABLEGET |CAS |1 |24 |
|7 | TABLESCAN |A(UK_CT_ACT_ACT_ID) |1 |24 |
|8 | TABLESCAN |t(IDX_CT_STORE_PRODUCT_REL_STATUS_STORE_ID_PRODUCT_ID)|3414 |1848|
==========================================================================================
Used Hint:
------------------------------------
USE_HASH(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ))

OutlineData:
------------------------------------
BEGIN_OUTLINE_DATA
NO_USE_HASH_AGGREGATION(@"SEL$2")
LEADING(@"SEL$2" (("nctmbasedb.ASP"@"SEL$2""nctmbasedb.CAS"@"SEL$2" )"nctmbasedb.A"@"SEL
USE_HASH(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ))
PQ_DISTRIBUTE(@"SEL$2" ("nctmbasedb.t"@"SEL$2" ) LOCALLOCAL)
USE_NL(@"SEL$2" ("nctmbasedb.A"@"SEL$2" ))
PQ_DISTRIBUTE(@"SEL$2" ("nctmbasedb.A"@"SEL$2" ) LOCALLOCAL)
NO_USE_NL_MATERIALIZATION(@"SEL$2" ("nctmbasedb.A"@"SEL$2" ))
USE_NL(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ))
PQ_DISTRIBUTE(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ) LOCALLOCAL)
NO_USE_NL_MATERIALIZATION(@"SEL$2" ("nctmbasedb.CAS"@"SEL$2" ))
INDEX(@"SEL$2""nctmbasedb.ASP"@"SEL$2""IDX_CT_ACT_STAGE_PRODUCT_PRODUCT_ID")
FULL(@"SEL$2""nctmbasedb.CAS"@"SEL$2")
INDEX(@"SEL$2""nctmbasedb.A"@"SEL$2""UK_CT_ACT_ACT_ID")
INDEX(@"SEL$2""nctmbasedb.t"@"SEL$2""IDX_CT_STORE_PRODUCT_REL_STATUS_STORE_ID_PRODUCT_ID"
END_OUTLINE_DATA
```

### 2.4 综上所述

| 优化方式 | hint | 效果 |
| --- | --- | --- |
| 指定小表（A表）为驱动表 | /*+leading(A) use_nl(A,CAS,ASP,t) */ | SQL执行超过30s,人为中断 |
| 指定大表（t表）为驱动表 | /*+leading(t (ASP,CAS)A) use_nl(t,A,CAS,ASP) */ | 执行耗时 0.19s |
| 改为hash关联方式 | /*+use_hash(t,ASP) */ | 耗时 0.06S |

## 3. 总结

### 3.1 hash join 使用场景

- 大表；
- 存在有效的过滤条件，过滤后数据量很小；

以上两个条件需要都满足。

### 3.2 驱动表区分

```
LEADING(@"SEL$2" (((("nctmbasedb.ASP"@"SEL$2" "nctmbasedb.CAS"@"SEL$2" )"nctmbasedb.t"@"SE
```

在 `LEADING` 提示所定义的连接顺序里，最外层括号中处于最左边的表就是驱动表。

### 3.3 驱动表使用场景有关疑问

#### 3.3.1 小表是否一定适合作为驱动表？

**回答：**小表通常适合作为驱动表，但并非在所有情况下都绝对适合。

以下是一些需要考虑的因素：

- 无索引或索引不佳
- 数据分布不均匀
- 连接条件复杂

#### 3.3.2 大表是否一定不适合作为驱动表？

**回答：**大表不一定不适合作为驱动表。

在一些特定情况下，大表作为驱动表也能获得较好的查询性能

- 存在强过滤条件
- 使用哈希连接且数据分布均匀
- 大表的索引设计合理
- 基于成本的优化器选择

本文关键字：#OceanBase #SQL优化


故障分析 | MySQL 8.0 中多字段虚拟列引发的宕机
故障分析 | 如何解决由触发器导致 MySQL 内存溢出？
故障分析 | 查询 ps.data_locks 导致 MySQL hang 住
故障分析 | TCP 缓存超负荷导致的 MySQL 连接中断
生产运维脚本引发的 MDL 锁故障排查之旅

✨ Github：https://github.com/actiontech/sqle
📚 文档：https://actiontech.github.io/sqle-docs/
💻 官网：https://opensource.actionsky.com/sqle/
👥 微信群：请添加小助手加入 ActionOpenSource
🔗 商业支持：https://www.actionsky.com/sqle


OceanBase · 目录

‹ 上一篇
多场景 OceanBase 并发参数调整方案

下一篇 ›
如何计算 OceanBase 最大/最小工作线程数？