TDSQL Boundless

性能调优

腾讯云

文档目录

性能调优

性能调优概述

最近更新时间：2026-02-05 11:01:13

性能调优目标

数据库系统性能调优的最终目标是充分利用服务器软硬件资源，使数据库软件能够提供高效的数据服务。具体衡量指标包括：

- QPS/TPS（每秒查询数/每秒事务数）：反映系统处理请求的能力。
- RT（响应时间）：指一个请求从发出到完成所需的时间。

提升 QPS/TPS 可以充分利用单台服务器资源，提升性价比，并降低总体拥有成本（TCO）。而降低 RT 能够改善用户体验，同时进一步提升系统的处理能力。

性能调优步骤

性能调优通常分为三个关键步骤：

1. 确定优化方向：分析当前系统的负载特征和业务需求，明确是以提升吞吐量为主还是降低延迟为主。
2. 定位瓶颈：通过监控工具和执行计划分析，识别系统中的性能瓶颈，如 CPU 使用率过高、内存不足、网络延迟或热点行竞争等。
3. 制定优化方案：根据瓶颈原因采取相应措施，例如调整参数配置、优化 SQL 执行路径、修改数据分布策略等。

系统级调优

系统的调优关注的是整个系统的运行效率，而非单一 SQL 的性能。其主要手段包括：

- 综合分析多条 SQL 的执行计划与系统负载特征；
- 关注全局性问题，如：
  - 热点行竞争
  - Buffer Cache 命中率
- 分区表设计合理性
  - 通过调整访问路径、执行顺序、逻辑改写等方式进行优化；
  - 在分区数量较多时，可考虑提高查询并行度以换取更高的性能，但需权衡资源消耗。

SQL 调优
SQL 调优概述

最近更新时间：2026-02-05 11:01:13

SQL 是一种声明式语言 (Declarative Language)，它只表达用户想要的是什么，具体怎么实现用户的目的则由数据库的优化器模块自行决定，即对查询选择相应的执行计划。对于同一条用户 SQL，在数据库中可能对应多种不同的实现方式，优化器面临的常见选择包括：

- 查询中的表访问是选择全表扫描，还是索引扫描，以及应该选择哪个索引。
- 查询中的 JOIN 操作应该选择什么 JOIN 顺序，以及每个 JOIN 应该选择什么 JOIN 算法。
- 查询中的 GROUP BY 和 ORDER BY 选择什么策略来实现。
- 查询中的子查询选择什么策略来处理。

这些选择会产生大量的组合结果，即候选执行计划，优化器会从这些候选执行计划中选择它认为最优的一个用于执行。

大部分情况下，优化器都能够选中执行效率较高的执行计划，但不可避免地在某些场景下，它不能选中最优执行计划，因为以下两个可能的原因：

- 优化器没能枚举到最优的那个候选执行计划。
- 优化器在评估所有候选执行计划时出现了偏差，最优执行计划没能胜出。

在这些场景下，如果优化器默认选择的执行计划执行效率很低，就需要 DBA 或者用户进行 SQL 调优，提升查询性能。

要做 SQL 调优，首先需要理解优化器选中的执行计划所代表的执行逻辑，从中识别出执行开销高的那部分，进而分析为什么优化器会这样选择，然后干涉优化器对执行计划的选择，使得期望中更好的执行计划能被选中。

理解执行计划

最近更新时间：2026-02-05 11:01:13

通过 EXPLAIN 可以查看查询的执行计划，TDSQL Boundless 和 MySQL 一样支持三种展示执行计划的格式：TRADITIONAL，TREE 和 JSON。

TRADITIONAL 格式

默认的展示格式，将执行计划输出为一个表格，表格中每一行代表 SELECT 语句中的一张表，行的顺序代表执行查询语句时读取表和做 JOIN 的顺序。
以如下查询为例：

```
tdsql> explain select * from t1, t2 where t1.a = t2.a;
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------------------------------
------+
| id | select_type | table | partitions | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra
|
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------------------------------
------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL |
NULL    | NULL |    3 |   100.00 | NULL
|
|  1 | SIMPLE      | t2    | NULL       | ALL  | NULL          | NULL |
NULL    | NULL |    3 |    33.33 | Using where; Using join buffer (hash
join) |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------------------------------
------+
```

执行计划可以解读为：对 t1 表和 t2 表都执行全表扫描，预期每张表返回 3 行数据，然后按照 t1 JOIN t2 的顺序执行连接，产生 3 * 3 * 33.33 / 100 = 3 行连接结果。
更详细的表格字段具体含义可以参考 MySQL 文档 。

TREE 格式

将执行计划展示为一棵树，能更直观地看到查询执行的逻辑，上面的例子对应的 TREE 格式为：

```
tdsql> explain format=tree select * from t1, t2 where t1.a = t2.a;
+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----------------+
| EXPLAIN
|
+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----------------+
| -> Inner hash join (t2.a = t1.a)  (cost=6.28 rows=3)
    -> Table scan on t2  (cost=0.88 rows=3)
    -> Hash
        -> Table scan on t1  (cost=2.84 rows=3)
|
+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
----------------+
```

相比于 TRADITIONAL 格式提供的信息，还能看到对 t1 JOIN t2 选择的算法是 Hash Join，其中 t1 作为 build 端，t2 作为 probe 端，并且还展示了执行计划的 cost 信息，以及 JOIN 使用的连接条件表达式 t2.a = t1.a。

JSON 格式

将执行计划以 JSON 格式输出，上面的例子对应 JSON 输出为：

```
tdsql> explain format=json select * from t1, t2 where t1.a = t2.a;
+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------+
| EXPLAIN
|
+---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------------------------------------------------------
---------------------------+
| {
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "6.27"
    },
    "nested_loop": [
      {
        "table": {
          "table_name": "t1",
          "access_type": "ALL",
          "rows_examined_per_scan": 3,
          "rows_produced_per_join": 3,
          "filtered": "100.00",
          "cost_info": {
            "read_cost": "2.54",
```

```
            "eval_cost": "0.30",
            "prefix_cost": "2.84",
            "data_read_per_join": "48"
          },
          "used_columns": [
            "a",
            "b"
          ]
        }
      },
      {
        "table": {
          "table_name": "t2",
          "access_type": "ALL",
          "rows_examined_per_scan": 3,
          "rows_produced_per_join": 3,
          "filtered": "33.33",
          "using_join_buffer": "hash join",
          "cost_info": {
            "read_cost": "2.54",
            "eval_cost": "0.30",
            "prefix_cost": "6.28",
            "data_read_per_join": "48"
          },
          "used_columns": [
            "a",
            "b"
          ],
          "attached_condition": "(`test`.`t2`.`a` = `test`.`t1`.`a`)"
        }
      }
    ]
  }
} |
+-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
---------------------------+
```

它可以看作是 TRADITIONAL 格式和 TREE 格式展示内容的混合，还额外提供更丰富的一些信息，包括更细分的 cost 以及每张表具体返回哪些列等。

EXPLAIN FOR CONNECTION

如果想要查看某个正在执行中查询的执行计划，可以使用 EXPLAIN FOR CONNECTION 语句，前提是知道该查询所在的 connection 号，这个 connection 号可以通过在该连接中使用 CONNECTION_ID() 获取，或者在任何连接中通过 SHOW PROCESSLIST 获取，比如：

```
tdsql> select connection_id();
+-----------------+
| connection_id() |
+-----------------+
|         1048611 |
+-----------------+

tdsql> show processlist;
+---------+------------+-------+-------------------+--------------------
-+---------------------+---------+------+-------------------------+-------
----------+---------+-----------+---------------+
| Id      | Tid        | Mem   | User              | Host
| db                  | Command | Time | State                   | Info
| Time_ms | Rows_sent | Rows_examined |
+---------+------------+-------+-------------------+--------------------
-+---------------------+---------+------+-------------------------+-------
----------+---------+-----------+---------------+
| 1048611 | 4294970988 | 16384 | test              | 127.0.0.1:40318
| test                | Query   |    0 | init                    | show
```

```
processlist |       0 |         0 |              0 |
+---------+-----------+-------+------------------+--------------------
-+--------------------+---------+------+-----------------------+-------
----------+---------+----------+--------------+

tdsql> explain for connection 1048611;
ERROR 3012 (HY000): EXPLAIN FOR CONNECTION command is supported only for
SELECT/UPDATE/INSERT/DELETE/REPLACE. txid: 0. sql-node: node-1-001.
error-store-node: nil
```

例子里因为当前连接并没有在执行可以 EXPLAIN 展示执行计划的 DML 语句，因此 EXPLAIN FOR CONNECTION 报错。

注意可能会有一种情况，通过 EXPLAIN 看到的执行计划，和不带 EXPLAIN 时查询执行时真正用的执行计划并不一样，有以下几种可能的原因：

- 不同连接的参数设置不同，尤其是优化器参数；
- 不同连接因为时间差异看到的统计信息不完全相同，因此做出的行数估算不同导致执行计划不同；
- 优化器代码中个别地方对 EXPLAIN 语句和非 EXPLAIN 语句会走不同路径；

这种情况下，通过 EXPLAIN FOR CONNECTION 查看查询执行真正使用的执行计划就会有帮助；

EXPLAIN ANALYZE

通过 EXPLAIN 用户可以解读出执行器会按照什么方式执行这条查询，但 EXPLAIN 只是展示优化器会对查询选择的执行计划，并不会真正执行它，想要识别出这个执行逻辑中耗时高导致整个查询慢的部分，需要用到 EXPLAIN ANALYZE，它会按照展示的执行计划真正执行这条查询，并统计各个部分的耗时和返回的结果行数，上面例子对应的 EXPLAIN ANALYZE 结果为：

```
tdsql> explain analyze select * from t1, t2 where t1.a = t2.a;
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------+
| EXPLAIN
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------+
| -> Inner hash join (t2.a = t1.a)  (cost=6.28 rows=3) (actual
time=1.459..1.481 rows=3 loops=1)
```

```
    -> Table scan on t2  (cost=0.88 rows=3) (actual time=0.528..0.548
rows=3 loops=1)
    -> Hash
        -> Table scan on t1  (cost=2.84 rows=3) (actual
time=0.841..0.863 rows=3 loops=1)
|
+------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------+
```

它展示了 t1 表和 t2 表的扫描都确实返回了和估算一致的3行数据，t1 JOIN t2 的结果也是3行，产生第一行 JOIN 结果耗时1.459ms，执行完整个 JOIN 耗时 1.481ms。通过分析 EXPLAIN ANALYZE 中各部分的耗时，用户可以找出查询的瓶颈在哪里，进而做针对性的调优。

除此之外，TDSQL 还提供信息更丰富的 EXPLAIN ANALYZE VERBOSE 功能，能够额外展示查询涉及的 RPC 类型以及执行情况，以及查询的内存使用情况。上面查询对应的 EXPLAIN ANALYZE VERBOSE 输出为：

```
tdsql> explain analyze verbose select * from t1, t2 where t1.a = t2.a;
+------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
----------------------------------------------------+
| EXPLAIN
|
+------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
----------------------------------------------------+
| -> Inner hash join (t2.a = t1.a)  (cost=6.28 rows=3) (actual
time=0.511..0.533 rows=3 loops=1)
    Chunk pair files: 0, memory usage: 16kB
```

```
    -> Table scan on t2  (cost=0.88 rows=3) (actual time=0.167..0.185
rows=3 loops=1)
    -> Hash
        -> Table scan on t1  (cost=2.84 rows=3) (actual
time=0.262..0.283 rows=3 loops=1)
  RPC statistics: leader
    -> LocalScanRecord=latency(ms): 2,0.266323,0.081208...0.185115,
retry_count: 0, retry_interval_all(ms): 0.000000, failure_count: 0 |
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------+
```

它展示的信息是：查询的内存开销是16kB，并且包含两次 LocalScanRecord RPC，总共耗时0.266323ms，其中最快的一次耗时0.081208ms，最慢的一次耗时0.185115ms，RPC 执行过程中都没有遇到错误和重试。这些运行时信息能帮助用户在更细粒度上定位查询的具体开销在什么地方。

有时候慢查询的 EXPLAIN ANALYZE 看起来并不慢，此时查询的时间开销可能并不在执行上；或者 EXPLAIN ANALYZE 确实慢，但它的执行计划可能并没有问题，而是被别的慢查询拖累才变慢的；这些情况下就需要在更大的维度上分析查询的耗时，此时可以用到 SHOW PROFILE 工具，比如：

```
tdsql> set profiling = 1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

tdsql> select * from t1 where a > 0;
+------+------+
| a    | b    |
+------+------+
|    1 |    1 |
|    2 |    2 |
|    3 |    3 |
+------+------+
3 rows in set (0.00 sec)

tdsql> show profiles;
+----------+------------+------------------------------+
| Query_ID | Duration   | Query                        |
```

```
+----------+------------+-----------------------------+
|        1 | 0.00195300 | select * from t1 where a > 0 |
+----------+------------+-----------------------------+
1 row in set, 1 warning (0.00 sec)

tdsql> show profile for query 1;
+--------------------------------+----------+
| Status                         | Duration |
+--------------------------------+----------+
| starting                       | 0.000204 |
| Executing hook on transaction  | 0.000004 |
| starting                       | 0.000031 |
| checking permissions           | 0.000014 |
| Opening tables                 | 0.000095 |
| init                           | 0.000009 |
| System lock                    | 0.000035 |
| optimizing                     | 0.000020 |
| statistics                     | 0.000076 |
| Wait gts rsp                   | 0.000354 |
| preparing                      | 0.000082 |
| executing                      | 0.000849 |
| end                            | 0.000007 |
| query end                      | 0.000005 |
| waiting for handler commit     | 0.000028 |
| closing tables                 | 0.000061 |
| freeing items                  | 0.000079 |
| cleaning up                    | 0.000004 |
+--------------------------------+----------+
18 rows in set, 1 warning (0.01 sec)
```

它能提供查询在数据库服务端经历的各个步骤的耗时，不仅限于执行；
此外，其他分析查询耗时和瓶颈可用的工具还包括 SPAN_TRACE 和 慢查询日志 等，综合使用它们可以帮助找出真正的慢查询，以及它耗时的地方。

OPTIMIZER TRACE

以上的信息都是关于优化器最终选择的执行计划以及其执行情况，它们能帮助定位慢查询到底慢在哪里，在此之上用户可能还需要知道为什么优化器选择了这个慢的执行计划，是因为它真的就没有其他更好的选择了，还是因为它的评估偏差导致选错了执行计划，或者是因为它根本就没有搜索到更优的某个执行计划，这就需要用到 Optimizer Trace 功能，它能回溯优化器对执行计划的具体决策过程。

以一个简单例子解释 Optimizer Trace 的使用：

```
tdsql> set optimizer_trace='enabled=on';
Query OK, 0 rows affected (0.00 sec)

tdsql> explain select * from t1 where a > 0;
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL |
NULL    | NULL |    3 |    33.33 | Using where |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

tdsql> select * from information_schema.optimizer_trace;
+----------------------------------------+---------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
```

```
+-------------------------------------+-------------------------------------+--
----------------------+
| QUERY                               | TRACE
| MISSING_BYTES_BEYOND_MAX_MEM_SIZE | INSUFFICIENT_PRIVILEGES |
+-------------------------------------+---------------------------------
```

```
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
---------------------------------+-----------------------------------+--
-----------------------+
| explain select * from t1 where a > 0 | {
```

```
  "steps": [
    {
      "join_preparation": {
        "select#": 1,
        "steps": [
          {
            "expanded_query": "/* select#1 */ select `t1`.`a` AS
`a`,`t1`.`b` AS `b` from `t1` where (`t1`.`a` > 0)"
          }
        ]
      }
    },
    {
      "join_optimization": {
        "select#": 1,
        "steps": [
          {
            "condition_processing": {
              "condition": "WHERE",
              "original_condition": "(`t1`.`a` > 0)",
              "steps": [
                {
                  "transformation": "equality_propagation",
                  "resulting_condition": "(`t1`.`a` > 0)"
                },
                {
                  "transformation": "constant_propagation",
                  "resulting_condition": "(`t1`.`a` > 0)"
                },
                {
                  "transformation": "trivial_condition_removal",
                  "resulting_condition": "(`t1`.`a` > 0)"
                }
              ]
            }
          },
          {
            "substitute_generated_columns": {
            }
          },
```

```
        {
          "table_dependencies": [
            {
              "table": "`t1`",
              "row_may_be_null": false,
              "map_bit": 0,
              "depends_on_map_bits": [
              ]
            }
          ]
        },
        {
          "ref_optimizer_key_uses": [
          ]
        },
        {
          "rows_estimation": [
            {
              "table": "`t1`",
              "table_scan": {
                "rows": 3,
                "cost": 2.5375
              }
            }
          ]
        },
        {
          "considered_execution_plans": [
            {
              "plan_prefix": [
              ],
              "table": "`t1`",
              "best_access_path": {
                "considered_access_paths": [
                  {
                    "rows_to_scan": 3,
                    "filtering_effect": [
                    ],
                    "final_filtering_effect": 0.333333,
                    "access_type": "scan",
```

```
                "resulting_rows": 1,
                "cost": 2.8375,
                "chosen": true
              }
            ]
          },
          "condition_filtering_pct": 100,
          "rows_for_plan": 1,
          "cost_for_plan": 2.8375,
          "chosen": true
        }
      ]
    },
    {
      "attaching_conditions_to_tables": {
        "original_condition": "(`t1`.`a` > 0)",
        "attached_conditions_computation": [
        ],
        "attached_conditions_summary": [
          {
            "table": "`t1`",
            "attached": "(`t1`.`a` > 0)"
          }
        ]
      }
    },
    {
      "force_batched_key_access": [
        {
          "table": "`t1`",
          "batched_key_access": true
        }
      ]
    },
    {
      "finalizing_table_conditions": [
        {
          "table": "`t1`",
          "original_table_condition": "(`t1`.`a` > 0)",
          "final_table_condition   ": "(`t1`.`a` > 0)"
```

```
              }
            ]
          },
          {
            "refine_plan": [
              {
                "table": "`t1`"
              }
            ]
          },
          {
            "engine_push_conditions": [
              {
                "table": "`t1`",
                "total_rows": 3,
                "index": "hidden pk",
                "condition_push": {
                },
                "single_table_push": {
                  "projection_push": {
                    "enabled": false,
                    "cause": "read field pct is less than
tdsql_max_projection_pct"
                  }
                },
                "condition_pushed": false,
                "cause": "scan_rows < tdsql_push_down_threshold_rows"
              }
            ]
          }
        ]
      }
    },
    {
      "parallel_plan": {
        "select#": 1,
        "steps": [
          {
            "considering": {
              "chosen": false,
```

```
              "cause": "plan_cost_less_than_threshold"
            }
          }
        ]
      }
    },
    {
      "local_access_optimize": {
        "check_local": true,
        "tables": [
          {
            "table": "`t1`",
            "type": "scan",
            "strategy": "condidate_for_local",
            "info": "not support direct local"
          }
        ]
      }
    },
    {
      "join_explain": {
        "select#": 1,
        "steps": [
        ]
      }
    }
  ]
} |                                    0 |                          0 |
+--------------------------------------+------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------------------------------------------------
----------------------------------+-----------------------------------+--
----------------------+
1 row in set (0.01 sec)

tdsql> set optimizer_trace='enabled=off';
Query OK, 0 rows affected (0.00 sec)
```

它展示了优化器在每个阶段做了些什么操作和决策，以及背后的原因，比如上面这个例子里，优化器没有选择将过滤条件 a > 0 下推到存储层，以及没有选择对 t1 表做并行扫描，因为它预估 t1 的行数很少没有超过相应优化的触发阈值，它也没有选择在 tdstore 中做列裁剪，因为查询需要的列数量占主键索引中列数量的比例超过了 tdsql_max_projection_pct。

SQL调优

最近更新时间：2026-02-11 14:13:01

在使用 理解执行计划 章节中的工具找出真正的慢查询以及它具体慢在哪里后，用户可能需要采取相应的措施来优化该查询的执行效率，这些措施大致可以分为三类：

- 调整查询涉及的表的 SCHEMA。
- 修改执行器相关参数。
- 干预优化器对执行计划的选择。

调整 SCHEMA

当优化器选中的执行计划不优时，用户可以首先检查执行计划中各个部分的行数估算和真实值是否有很大的偏差，如果是则进一步检查查询涉及的列是否存在相应的统计信息，这些统计信息是否已经过期不能体现最新的数据分布，或者统计信息没有过期但准确度本身就很低；这些情况下需要重新收集统计信息，或者调大统计信息采样率相关的参数再收集统计信息，比如 `sample_sst_blocks`，但需要注意调大采样率会导致 ANALYZE 语句开销更高。示例：

```
# 示例 1：统计信息缺失
tdsql> set global tdsql_get_index_stats_from_tdstore = on;
Query OK, 0 rows affected (0.01 sec)

tdsql> create table t1(a int, b int);
Query OK, 0 rows affected (0.21 sec)

tdsql> insert into t1 values(1,1),(2,2),(3,3);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

tdsql> analyze table t1;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t1 | analyze | status   | OK       |
+---------+---------+----------+----------+
1 row in set (0.01 sec)

# a 列没有合适的统计信息，估算行数 1 有偏差
tdsql> explain select * from t1 where a >= 2;
```

```
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL |
NULL    | NULL |    3 |    33.33 | Using where |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

tdsql> analyze table t1 update histogram on a;
+---------+-----------+----------+----------------------------------------
--------+
| Table   | Op        | Msg_type | Msg_text
|
+---------+-----------+----------+----------------------------------------
--------+
| test.t1 | histogram | status   | Histogram statistics created for
column 'a'. |
+---------+-----------+----------+----------------------------------------
--------+
1 row in set (0.01 sec)

# 对 a 列搜集直方图后，估算行数 2 准确
tdsql> explain select * from t1 where a >= 2;
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL |
NULL    | NULL |    3 |    66.67 | Using where |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)

# 示例 2：统计信息过期
```

```
tdsql> create table t1(a int, b int);
Query OK, 0 rows affected (0.26 sec)

tdsql> insert into t1
    -> with recursive nrows(n) as (
    -> select 1 union all
    -> select n+1 from nrows where n < 200
    -> )
    -> select n, n from nrows;
Query OK, 200 rows affected (0.02 sec)
Records: 200  Duplicates: 0  Warnings: 0

tdsql> analyze table t1;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t1 | analyze | status   | OK       |
+---------+---------+----------+----------+
1 row in set (0.01 sec)

tdsql> explain select * from t1;
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL |
NULL    | NULL |  200 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

tdsql> insert into t1
    -> with recursive nrows(n) as (
    -> select 1 union all
    -> select n+1 from nrows where n < 10
    -> )
    -> select n, n from nrows;
Query OK, 10 rows affected (0.01 sec)
```

```
Records: 10  Duplicates: 0  Warnings: 0

# 新插入 10 条数据不足以触发 auto analyze，统计信息过期
tdsql> explain select * from t1;
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL |
NULL    | NULL |  200 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)

tdsql> analyze table t1;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t1 | analyze | status   | OK       |
+---------+---------+----------+----------+
1 row in set (0.01 sec)

tdsql> explain select * from t1;
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL |
NULL    | NULL |  210 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

此外，可以考虑为查询涉及的表增加或者删除索引：增加索引可以用于加速查询中的过滤条件执行，或者改变 JOIN 的算法提升效率，或者提供查询需要的顺序属性避免 Filesort 操作；删除索引一般适用于该索引干扰了优化器对执

行计划的选择这种情况。注意，增删索引可能会影响到其他的查询，需要谨慎操作，可以通过将索引标记为 VISIBLE / INVISIBLE 预先验证效果。示例：

```
tdsql> create table t1(a int, b int);
Query OK, 0 rows affected (0.19 sec)

tdsql> insert into t1 with recursive nrows(n) as ( select 1 union all
select n+1 from nrows where n < 200 ) select n, n from nrows;
Query OK, 200 rows affected (0.03 sec)
Records: 200  Duplicates: 0  Warnings: 0

tdsql> analyze table t1;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t1 | analyze | status   | OK       |
+---------+---------+----------+----------+
1 row in set (0.01 sec)

tdsql> create table t2(a int, b int);
Query OK, 0 rows affected (0.20 sec)

tdsql> insert into t2 values(1,1),(2,2),(3,3);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

tdsql> analyze table t2;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t2 | analyze | status   | OK       |
+---------+---------+----------+----------+
1 row in set (0.01 sec)

tdsql> explain format=tree select * from t1, t2 where t1.a = t2.a;
+-------------------------------------------------------------------------
-------------------------------------------------------------------------
--------------------+
| EXPLAIN
|
```

```
+-------------------------------------------------------------------------
-------------------------------------------------------------------------
--------------------+
| -> Inner hash join (t1.a = t2.a)  (cost=67.84 rows=60)
    -> Table scan on t1  (cost=2.33 rows=200)
    -> Hash
        -> Table scan on t2  (cost=2.84 rows=3)
|
+-------------------------------------------------------------------------
-------------------------------------------------------------------------
--------------------+
1 row in set (0.00 sec)

tdsql> alter table t1 add key(a);
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0

tdsql> analyze table t1;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t1 | analyze | status   | OK       |
+---------+---------+----------+----------+
1 row in set (0.01 sec)

# join 算法从 hash join 变更成了 bka
tdsql> explain format=tree select * from t1, t2 where t1.a = t2.a;
+-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-----------------------------------------------+
| EXPLAIN
|
+-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-----------------------------------------------+
| -> Batched key access inner join  (cost=1.33 rows=3)
    -> Batch input rows
        -> Filter: (t2.a is not null)  (cost=2.84 rows=3)
```

```
            -> Table scan on t2  (cost=2.84 rows=3)
    -> Multi-range index lookup on t1 using a (a=t2.a)  (cost=0.28
rows=1)
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-------------------------------------------------+
1 row in set (0.02 sec)

tdsql> alter table t1 alter index a invisible;
Query OK, 0 rows affected (0.12 sec)
Records: 0  Duplicates: 0  Warnings: 0

# 索引 a 不可见，join 算法变回 hash join
tdsql> explain format=tree select * from t1, t2 where t1.a = t2.a;
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
------------------+
| EXPLAIN
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
------------------+
| -> Inner hash join (t1.a = t2.a)  (cost=67.84 rows=3)
    -> Table scan on t1  (cost=1.70 rows=200)
    -> Hash
        -> Table scan on t2  (cost=2.84 rows=3)
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
------------------+
1 row in set (0.02 sec)
```

然后，还可以考虑的是根据业务和查询特点，将查询涉及的表从普通表调整为分区表，或者从分区表调整为普通表，或者考虑调整分区键，目的都是为了充分利用查询的过滤条件，扫描更少的数据。示例：

```
tdsql> create table t1(a int, b int);
Query OK, 0 rows affected (0.19 sec)
```

```
tdsql> insert into t1 with recursive nrows(n) as ( select 1 union all
select n+1 from nrows where n < 200 ) select n, n from nrows;
Query OK, 200 rows affected (0.02 sec)
Records: 200  Duplicates: 0  Warnings: 0

tdsql> analyze table t1 update histogram on a;
+---------+-----------+----------+----------------------------------------
--------+
| Table   | Op        | Msg_type | Msg_text
|
+---------+-----------+----------+----------------------------------------
--------+
| test.t1 | histogram | status   | Histogram statistics created for
column 'a'. |
+---------+-----------+----------+----------------------------------------
--------+
1 row in set (0.02 sec)

# 非分区表扫描全表
tdsql> explain select * from t1 where a <= 10;
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL |
NULL    | NULL |  200 |     5.00 | Using where |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
1 row in set, 1 warning (0.01 sec)

tdsql> drop table t1;
Query OK, 0 rows affected (0.11 sec)

tdsql> create table t1(a int, b int) partition by range(a) (
    -> partition p0 values less than (51),
    -> partition p1 values less than (101),
    -> partition p2 values less than (151),
```

```
    -> partition p3 values less than (maxvalue));
Query OK, 0 rows affected (0.22 sec)

tdsql> insert into t1 with recursive nrows(n) as ( select 1 union all
select n+1 from nrows where n < 200 ) select n, n from nrows;
Query OK, 200 rows affected (0.04 sec)
Records: 200  Duplicates: 0  Warnings: 0

tdsql> analyze table t1;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t1 | analyze | status   | OK       |
+---------+---------+----------+----------+
1 row in set (0.01 sec)

tdsql> analyze table t1 update histogram on a;
+---------+-----------+----------+-------------------------------------
--------+
| Table   | Op        | Msg_type | Msg_text
|
+---------+-----------+----------+-------------------------------------
--------+
| test.t1 | histogram | status   | Histogram statistics created for
column 'a'. |
+---------+-----------+----------+-------------------------------------
--------+
1 row in set (0.01 sec)

# 分区表只需要扫 p0
tdsql> explain select * from t1 where a <= 10;
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key  |
key_len | ref  | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
|  1 | SIMPLE      | t1    | p0         | ALL  | NULL          | NULL |
NULL    | NULL |   50 |     5.00 | Using where |
```

```
+----+-------------+-------+------------+------+---------------+------+-
--------+------+------+----------+-------------+
1 row in set, 1 warning (0.01 sec)
```

修改执行器参数

在不变更执行计划的前提下，如果想要加速查询执行，可以考虑修改部分执行器相关的参数，比如适当调大 join_buffer_size 和 temptable_max_ram 等参数。注意这些参数的生效范围，避免对预期外的查询产生影响。

干预优化器

大部分慢查询情况下，用户需要的还是干预优化器过程使得查询可以选择预期中更高效的执行计划，可以用以下几种方式对优化器做干预：

- 控制优化器对 JOIN 顺序的穷举度：通过设置 `optimizer_prune_level` 和 `optimizer_search_depth` 这两个参数，用户可以控制优化器搜索的所有候选 JOIN 顺序的数量；如果慢查询选择的 JOIN 顺序较差，并且通过 Optimizer Trace 发现优化器并没有搜索到想要的更优 JOIN 顺序，则可以考虑调整这两个参数；注意更大的 JOIN 顺序搜索空间意味着更大的优化器开销，具体如何设置这两个参数可以参考 MySQL 文档，一般情况下这两个值保持默认值就好；
- 通过 `optimizer_switch` 或者其他系统参数控制是否开启某些优化操作：会存在一些优化操作，它们在某些情况下能导致更好的执行计划结果，但某些情况下也会导致执行计划变差，通常这些操作会对应 `optimizer_switch` 中的一个开关，常用的比如 `subquery_to_derived` 选项，它控制是否考虑将子查询转化为 derived table 再做 JOIN，这个操作默认是关闭的，但在某些查询下开启它能大幅度提升查询执行效率；更详细的 `optimizer_switch` 选项可以参考 MySQL 文档；示例：

```
tdsql> create table t1(a int, b int);
Query OK, 0 rows affected (0.21 sec)

tdsql> create table t2(a int, b int);
Query OK, 0 rows affected (0.19 sec)

# 默认执行计划里，子查询是用 EXISTS 方式执行，会扫描 t2 多次
tdsql> explain format=tree select * from t1 where t1.a > (select
count(1) from t2 where t2.b = t1.b);
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------------------------+
```

```
| EXPLAIN
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------+
| -> Filter: (t1.a > (select #2))  (cost=2.61 rows=1)
    -> Table scan on t1  (cost=2.61 rows=1)
    -> Select #2 (subquery in condition; dependent)
        -> Aggregate: count(1)  (cost=2.84 rows=1)
            -> Filter: (t2.b = t1.b)  (cost=2.61 rows=1)
                -> Table scan on t2  (cost=2.61 rows=1)
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------+
1 row in set, 1 warning (0.01 sec)

tdsql> set optimizer_switch='subquery_to_derived=on';
Query OK, 0 rows affected (0.00 sec)

# 打开 subquery_to_derived 功能后，t2 只需要扫描一次，中间结果用临时表
derived_1_2 保存，它和 t1 做 join
tdsql> explain format=tree select * from t1 where t1.a > (select
count(1) from t2 where t2.b = t1.b);
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------+
| EXPLAIN
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------+
| -> Filter: (t1.a > coalesce(derived_1_2.`count(1)`,0))  (cost=3.21
rows=1)
    -> Nested loop left join  (cost=3.21 rows=1)
        -> Table scan on t1  (cost=2.61 rows=1)
        -> Index lookup on derived_1_2 using <auto_key0> (b=t1.b)
(cost=0.70 rows=2)
            -> Materialize  (cost=10.07..10.07 rows=1)
                -> Table scan on <temporary>  (cost=5.12..5.12 rows=1)
                    -> Aggregate using temporary table
(cost=9.14..9.14 rows=1)
                        -> Table scan on t2  (cost=2.61 rows=1)
|
+-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------------------------------------------
-----------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

- 通过 hint 干预候选执行计划的选择：TDSQL 和 MySQL 一样支持以注释形式使用的 Optimizer Hints 以及在查询中使用的 Index Hints ，用户可以通过它们控制优化器对索引的选择，对 JOIN 顺序和算法的选择，对子查询处理方式的选择，对 SEMI JOIN 的执行策略的选择等；此外，TDSQL 还支持用 Hint 控制优化器对 PARALLEL 执行方式的选择；推荐使用注释形式的 Optimizer Hints，它们可以搭配 TDSQL 的 Outline 功能一起使用，使得 DBA 可以通过添加 Outline 规则对查询自动添加 hint，而不用更改业务中的具体查询；示例：

```
# 上面示例里的 optimizer_switch 可以通过 hint 指定，只在单条语句上生效
tdsql> set optimizer_switch='subquery_to_derived=off';
Query OK, 0 rows affected (0.00 sec)
```

```
tdsql> explain format=tree select * from t1 where t1.a > (select
count(1) from t2 where t2.b = t1.b);
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------------------------+
| EXPLAIN
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------------------------+
| -> Filter: (t1.a > (select #2))  (cost=2.61 rows=1)
    -> Table scan on t1  (cost=2.61 rows=1)
    -> Select #2 (subquery in condition; dependent)
        -> Aggregate: count(1)  (cost=2.84 rows=1)
            -> Filter: (t2.b = t1.b)  (cost=2.61 rows=1)
                -> Table scan on t2  (cost=2.61 rows=1)
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------------------------+
1 row in set, 1 warning (0.00 sec)

tdsql> explain format=tree
select/*+set_var(optimizer_switch='subquery_to_derived=on')*/ * from
t1 where t1.a > (select count(1) from t2 where t2.b = t1.b);
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------------------------------+
```

```
| EXPLAIN
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------------------------------+
| -> Filter: (t1.a > coalesce(derived_1_2.`count(1)`,0))  (cost=3.21
rows=1)
    -> Nested loop left join  (cost=3.21 rows=1)
        -> Table scan on t1  (cost=2.61 rows=1)
        -> Index lookup on derived_1_2 using <auto_key0> (b=t1.b)
(cost=0.70 rows=2)
            -> Materialize  (cost=10.07..10.07 rows=1)
                -> Table scan on <temporary>  (cost=5.12..5.12 rows=1)
                    -> Aggregate using temporary table
(cost=9.14..9.14 rows=1)
                        -> Table scan on t2  (cost=2.61 rows=1)
|
+-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
-----------------------------------------------------------------------
----------------------------------------+
1 row in set, 1 warning (0.00 sec)

# 对这条语句的 hint 创建 outline 规则
tdsql> call dbms_admin.statement_outline_add_rule("test",
"select/*+set_var(optimizer_switch='subquery_to_derived=on')*/ * from
t1 where t1.a > (select count(1) from t2 where t2.b = t1.b)");
+------+
| ID   |
+------+
|    1 |
```

```
+------+
1 row in set (0.01 sec)

# 现在查询不再需要 hint 也能触发 subquery_to_derived 改写
tdsql> explain format=tree select * from t1 where t1.a > (select
count(1) from t2 where t2.b = t1.b);
+----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
-----------------------------------------+
| EXPLAIN
|
+----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
-----------------------------------------+
| -> Filter: (t1.a > coalesce(derived_1_2.`count(1)`,0))  (cost=3.21
rows=1)
    -> Nested loop left join  (cost=3.21 rows=1)
        -> Table scan on t1  (cost=2.61 rows=1)
        -> Index lookup on derived_1_2 using <auto_key0> (b=t1.b)
(cost=0.70 rows=2)
            -> Materialize  (cost=10.07..10.07 rows=1)
                -> Table scan on <temporary>  (cost=5.12..5.12 rows=1)
                    -> Aggregate using temporary table
(cost=9.14..9.14 rows=1)
                        -> Table scan on t2  (cost=2.61 rows=1)
|
+----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
```

```
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
----------------------------------------+
1 row in set, 2 warnings (0.02 sec)

# warnings 会告知有 outline 规则被应用了
tdsql> show warnings;
+-------+------+-----------------------------------------------------------
--------------------+
| Level | Code | Message
|
+-------+------+-----------------------------------------------------------
--------------------+
| Note  | 8579 | Statement outline rule 1 was applied, it may have
changed the query plan. |
| Note  | 1276 | Field or reference 'test.t1.b' of SELECT #2 was
resolved in SELECT #1     |
+-------+------+-----------------------------------------------------------
--------------------+
2 rows in set (0.01 sec)
```

以上方式本质上都是在两个维度上对优化器做干预：

- **干预优化器搜索候选执行计划的空间，使得优化器能看到最优的候选执行计划。**
- **干预优化器评估候选执行计划，使得最优的候选执行计划能被选中。**