**TiDB SQL 性能稳定性分析及最佳实践**

北京平凯星辰科技发展有限公司

TiDB SQL 性能稳定性分析及最佳实践

20xx 年 xx 月 xx 日

文档编号：DOC-223

# 文档控制

|  |  |  |  |
| --- | --- | --- | --- |
| **项目名称** |  | | |
| **文档名称** |  | | |
|  | | | |
| **1. 文档属性** | | | |
| **文档编号** |  | | |
| **2. 修改记录** | | | |
| 日期 | 修改人 | 修改记录 | 备注 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **3. 审阅记录** | | | |
| 日期 | 审阅人 | 公司名称及职务 | 备注 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**版权声明**

**平凯星辰（北京）科技有限公司版权所有，保留一切权利。未经本公司许可，任何单位和个人不得擅自摘抄、复制本文档内容，并以任何形式传播。**

目录

[xx 用户分布式核心系统项目 1](#_Toc645843851)

[文档控制 2](#_Toc232021604)

[一. TiDB的SQL执行过程 4](#_Toc1428194714)

[1. SQL的解析及执行过程 4](#_Toc54331245)

[2. 点查与非点查的执行计划区别 6](#_Toc630318919)

[3. 谓词下推 6](#_Toc1026447236)

[二. TiDB索引类型及扫描方式 7](#_Toc935845403)

[1. 索引类型 7](#_Toc959982317)

[2. 算子的介绍 8](#_Toc873561648)

[3. 索引的算子扫描方式 9](#_Toc33322434)

[三. SQL的join方式 11](#_Toc1579807174)

[四. TiDB表和索引的统计信息管理 14](#_Toc1872661786)

[1. 表的统计信息 16](#_Toc1070371382)

[2. 表的健康度判断 17](#_Toc587962171)

[3. 分区表的快速收集 18](#_Toc1242552243)

[4. 列的统计信息 19](#_Toc2073710832)

[5. 直方图信息 20](#_Toc1660865682)

[6. 索引的统计信息收集策略 22](#_Toc1500850408)

[7. 23](#_Toc1826713497)

[7.1自动更新触发条件 23](#_Toc1061905616)

[7.2控制并发度(和cpu的关系，多少region与IO是否有关系) 24](#_Toc587858766)

[7.3 ANALYZE 配置持久化 24](#_Toc382176363)

[7.4 ANALYZE运行状态检查 25](#_Toc1525322228)

[7.5 表统计信息状态检查 26](#_Toc1990026995)

[7.6 表健康度检查 27](#_Toc871014438)

[7.7 统计备份与恢复 27](#_Toc13860017)

[五. TiDB执行计划的管理 28](#_Toc1939840806)

[1. 影响CBO的因素分析 28](#_Toc1269635356)

[2. hint的支持方式 28](#_Toc925327032)

[3. SPM 执行计划管理 28](#_Toc321530694)

[六. 慢SQL的管理 30](#_Toc1264238743)

[1. 如何判断慢SQL 30](#_Toc1571170883)

[a. 通过grafana判断query的duration的分布情况 30](#_Toc553552298)

[b. 慢日志分析 31](#_Toc544949809)

[2. Dashboard判断关键业务SQL 32](#_Toc1625502129)

[3. 从数据字典中获取SQL 33](#_Toc1183871217)

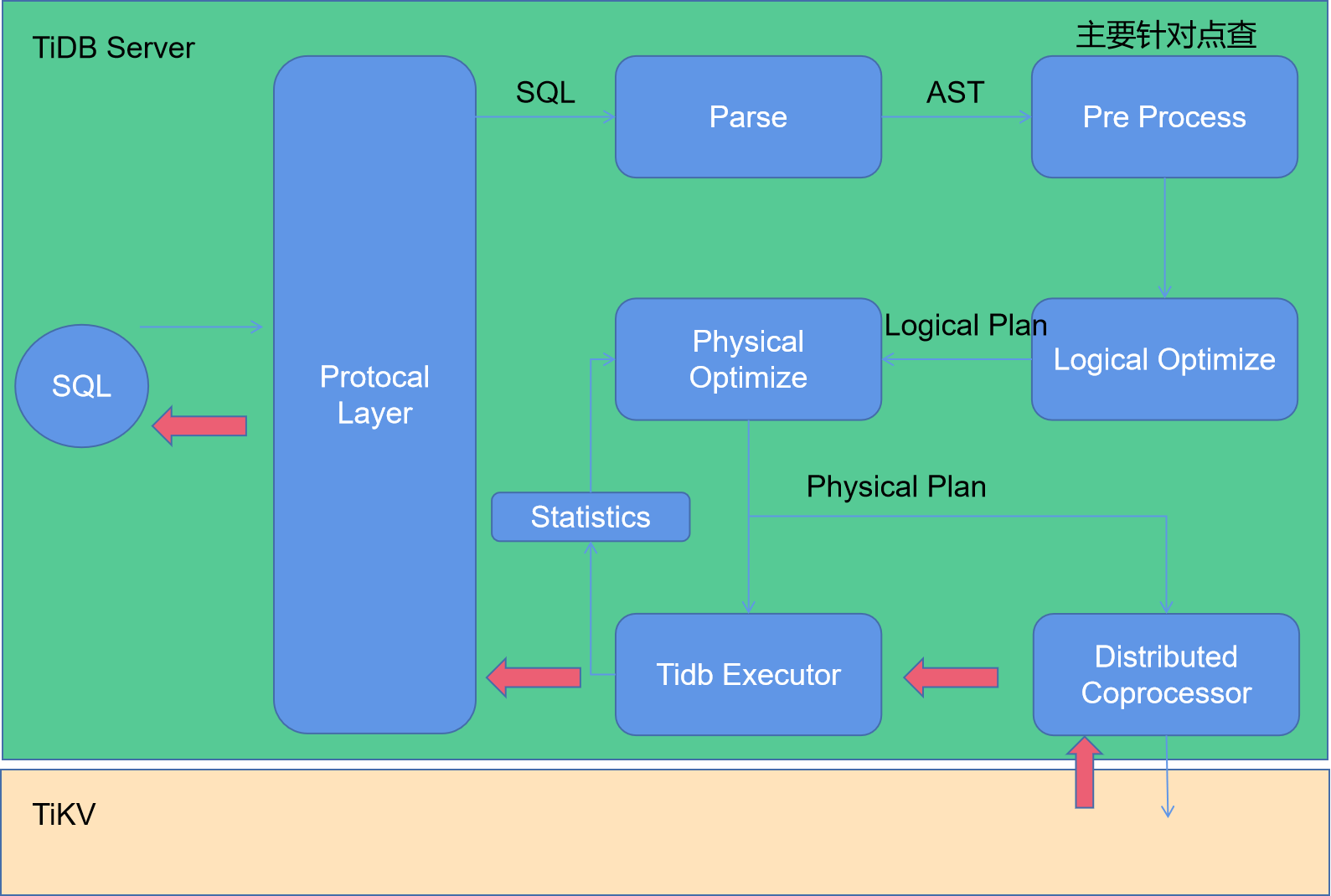
[4. 从TiDB.log中获取性能差的SQL 34](#_Toc1571397046)

[七. 监控 36](#_Toc413863884)

**一. TiDB的SQL执行过程**

**1. SQL的解析及执行过程**

https://docs.pingcap.com/zh/TiDB/stable/sql-tuning-overview



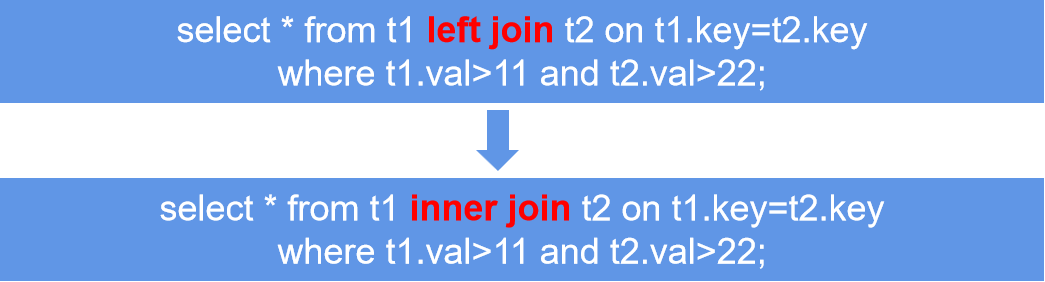
* Protocal Layer:协议层。
* Parse:将SQL解析为AST的方式。
* Pre Porcess:当判断SQL为点查(where条件为主键字段或唯一性索引字段返回1行)，则将SQL发送给TiDB Executor去TiKV获取数据，同时还要完成表达式简化,子查询简化处理等。
* Logical Optimize:根据AST的结果判断权限，表字段，表名等信息是否正确(数据字典信息是否准确)。
* Logical Plan:根据SQL的where条件对SQL逻辑进行一些判断和将复杂的SQL拆解为逻辑简单的SQL，比如：分区裁剪判断，列裁剪，谓词下推，表join顺序的调整等。
* Physical Optimize:参考统计信息对Logical Plan的结果进行基于cost的计算和判断，得出cost最低的SQL执行计划(Physical Plan),包括表join的方式，索引扫描的方式，表的扫描方式，算子是否下推到TiKV等，并交由TiDB Executor执行。

备注:Physical Plan和SQL的digest会被缓存在Plan Cache中，每个SQL的digest只保留最近一次的Physical Plan。

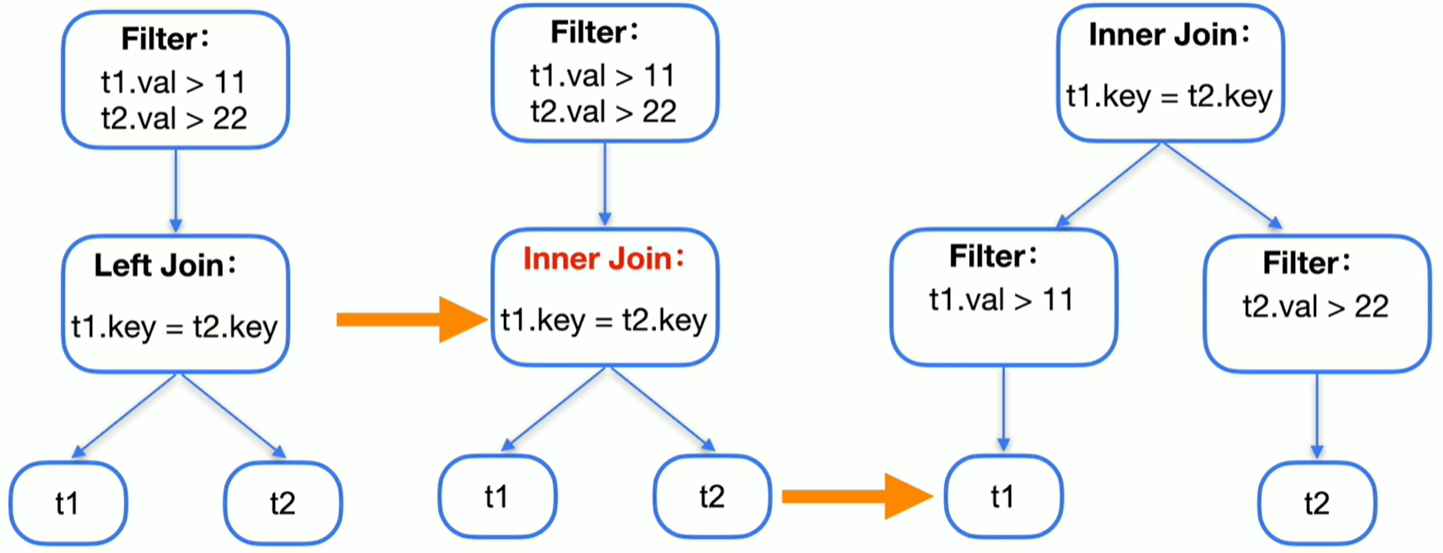
**2. 点查与非点查的执行计划区别**

* 点查：当SQL 经过Parse阶段为AST以后,Pre Porcess阶段会根据where条件判断是否该SQL为访问主键或唯一索引只返回一行数据的查询，如果是则该SQL为点查类。点查是直接跳过逻辑优化和物理优化，直接执行。
* 非点查：当SQL 经过Parse阶段为AST以后，Pre Porcess阶段会根据where条件判断不是访问主键和唯一索引的查询。

**3. 谓词下推**

****

谓词下推属于逻辑优化阶段，当一个SQL需要访问的数据分散在多个TiKV节点时,SQL将多个TiKV节点的数据统一拉到TiDB Server节点过滤和汇聚的性能是远低于在TiKV节点层完成各自节点数据的过滤然后再将结果传到计算层TiDB Server聚合，该操作就叫谓词下推。该操作的好处在于SQL数据访问的过程中，TiKV与TiDB server数据传输量降低，且降低了TiDB server侧数据缓存的资源开销。



1. **TiDB索引类型及扫描方式**

**1. 索引类型**

* 聚簇表

Key: tablePrefix{TableID}\_recordPrefixSep{Col1}

Value: [col2,col3,col4....]

* 非聚簇表

Key: tablePrefix{TableID}\_recordPrefixSep{\_TiDB\_RowID}

Value: [col1,col2,col3,col4....]

TiDB的索引与聚簇表与非聚簇表的key有关，通过value部分可以定位到行。

* 唯一索引

Key: tablePrefix{TableID}\_recordPrefixSep{indexID}\_indexedColumnsValue

Value: [RowID]

* 二级索引

Key: tablePrefix{TableID}\_recordPrefixSep{indexID}\_indexedColumnsValue\_RowID

Value: [null]

* 覆盖索引

SQL使用的索引直接可以扫描到所需要的数据，无需回表操作，减少IO的消耗。

* 表达式索引

索引列带一个函数的索引。

* 不可见索引

该索引只对优化器不可见。

**2. 算子的介绍**

算子是为返回查询结果而执行的特定步骤，真正执行扫表（读盘或者读 TiKV Block Cache），操作的算子有如下几类：

* TableFullScan：全表扫描。
* TableRangeScan：带有范围的表数据扫描。
* TableRowIDScan：根据上层传递下来的 RowID 扫描表数据。时常在索引读操作后检索符合条件的行。
* IndexFullScan：另一种“全表扫描”，扫的是索引数据，不是表数据。
* IndexRangeScan：带有范围的索引数据扫描操作。

TiDB 会汇聚 TiKV/TiFlash 上扫描的数据或者计算结果，这种“数据汇聚”算子目前有如下几类：

* TableReader：将 TiKV 上底层扫表算子 TableFullScan 或 TableRangeScan 得到的数据进行汇总。
* IndexReader：将 TiKV 上底层扫表算子 IndexFullScan 或 IndexRangeScan 得到的数据进行汇总。
* IndexLookUp：先汇总 Build 端 TiKV 扫描上来的 RowID，再去 Probe 端上根据这些 RowID 精确地读取 TiKV 上的数据。Build 端是 IndexFullScan 或 IndexRangeScan 类型的算子，Probe 端是 TableRowIDScan 类型的算子。
* IndexMerge：和 IndexLookupReader 类似，可以看做是它的扩展，可以同时读取多个索引的数据，有多个 Build 端，一个 Probe 端。执行过程也很类似，先汇总所有 Build 端 TiKV 扫描上来的 RowID，再去 Probe 端上根据这些 RowID 精确地读取 TiKV 上的数据。Build 端是 IndexFullScan 或 IndexRangeScan 类型的算子，Probe 端是 TableRowIDScan 类型的算子。

**3. 索引的算子扫描方式**

https://book.TiDB.io/session3/chapter1/sql-execution-plan.html

TiDB 支持以下使用索引的算子来提升查询速度：

* [IndexLookup](https://docs.pingcap.com/zh/tidb/stable/explain-indexes#indexlookup)

TiDB 从二级索引检索数据时会使用 IndexLookup 算子。例如，以下所有查询均会在 intkey 列的索引上使用 IndexLookup 算子。

IndexLookup 算子有以下两个子节点：

├─IndexRangeScan\_8(Build) 算子节点对 intkey 列的索引执行范围扫描，并检索内部的 RowID 值（对此表而言，即为主键）。

└─TableRowIDScan\_9(Probe) 算子节点随后从表数据中检索整行。

IndexLookup 任务分以上两步执行。如果满足条件的行较多，SQL 优化器可能会根据[统计信息](https://docs.pingcap.com/zh/tidb/stable/statistics)选择使用 TableFullScan 算子。在以下示例中，很多行都满足 intkey > 100 这一条件，因此优化器选择了 TableFullScan：

EXPLAIN SELECT \* FROM t1 WHERE intkey > 100;

+-------------------------+---------+-----------+---------------+-------------------------+| id | estRows | task | access object | operator info |+-------------------------+---------+-----------+---------------+-------------------------+| TableReader\_7 | 898.50 | root | | data:Selection\_6 || └─Selection\_6 | 898.50 | cop[TiKV] | | gt(test.t1.intkey, 100) || └─TableFullScan\_5 | 1010.00 | cop[TiKV] | table:t1 | keep order:false |+-------------------------+---------+-----------+---------------+-------------------------+3 rows in set (0.00 sec)

IndexLookup 算子能在带索引的列上有效优化 LIMIT：

EXPLAIN SELECT \* FROM t1 ORDER BY intkey DESC LIMIT 10;

+--------------------------------+---------+-----------+--------------------------------+------------------------------------+| id | estRows | task | access object | operator info |+--------------------------------+---------+-----------+--------------------------------+------------------------------------+| IndexLookUp\_21 | 10.00 | root | | limit embedded(offset:0, count:10) || ├─Limit\_20(Build) | 10.00 | cop[TiKV] | | offset:0, count:10 || │ └─IndexFullScan\_18 | 10.00 | cop[TiKV] | table:t1, index:intkey(intkey) | keep order:true, desc || └─TableRowIDScan\_19(Probe) | 10.00 | cop[TiKV] | table:t1 | keep order:false, stats:pseudo |+--------------------------------+---------+-----------+--------------------------------+------------------------------------+4 rows in set (0.00 sec)

以上示例中，TiDB 从 intkey 索引读取最后 10 行，然后从表数据中检索这些行的 RowID 值。

* [IndexReader](https://docs.pingcap.com/zh/tidb/stable/explain-indexes#indexreader)

TiDB 支持覆盖索引优化 (covering index optimization)。如果 TiDB 能从索引中检索出所有行，就会跳过 IndexLookup 任务中通常所需的第二步（即从表数据中检索整行）。

* [Point\_Get和Batch\_Point\_Get](https://docs.pingcap.com/zh/tidb/stable/explain-indexes#point_get-%E5%92%8C-batch_point_get)

TiDB 直接从主键或唯一键检索数据时会使用 Point\_Get 或 Batch\_Point\_Get 算子。这两个算子比 IndexLookup 更有效率。

* [IndexFullScan](https://docs.pingcap.com/zh/tidb/stable/explain-indexes#indexfullscan)

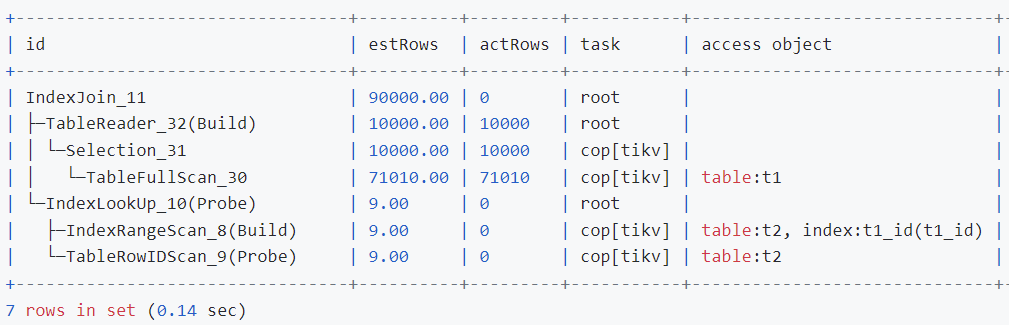
索引是有序的，所以优化器可以使用 IndexFullScan 算子来优化常见的查询，例如在索引值上使用 MIN 或 Max 函数。

**建议：对于执行计划中的索引扫描方式，Point\_Get和Batch\_Point\_Get的效率是最高的，其次是IndexReader，IndexLookup，比较差的是[IndexFullScan](https://docs.pingcap.com/zh/tidb/stable/explain-indexes" \l "indexfullscan)。**

**三. SQL的join方式**

[https://docs.pingcap.com/zh/TiDB/stable/explain-joins](https://docs.pingcap.com/zh/tidb/stable/explain-joins)

1. **Index Join**



如果预计需要连接的行数较少，推荐使用 Index Join 算法。这个算法与 MySQL 主要使用的 Join 算法类似。

如果使用 Hint [INL\_JOIN](https://docs.pingcap.com/zh/tidb/stable/optimizer-hints#inl_joint1_name--tl_name-) 进行 Index Join 操作，TiDB 会在连接外表之前创建一个中间结果的 Hash Table。TiDB 同样也支持使用 Hint [INL\_HASH\_JOIN](https://docs.pingcap.com/zh/tidb/stable/optimizer-hints#inl_hash_join) 在外表上建 Hash Table。以上所述的 Index Join 相关算法都由 SQL 优化器自动选择。

Index Join 算法对内存消耗较小，但如果需要执行大量探查操作，运行速度可能会慢于其他 Join 算法。

Index Join 算法的性能受以下系统变量影响：

* [TiDB\_index\_join\_batch\_size](https://docs.pingcap.com/zh/tidb/stable/system-variables#tidb_index_join_batch_size)（默认值：25000）- index lookup join 操作的 batch 大小。
* [TiDB\_index\_lookup\_join\_concurrency](https://docs.pingcap.com/zh/tidb/stable/system-variables#tidb_index_lookup_join_concurrency)（默认值：4）- 可以并发执行的 index lookup 任务数。

**建议：Join的过程可采用并行多任务方式提升性能,每一个并行任务的驱动batch size 建议25000以内，当驱动表小于25000的时候，依然会触发,为了触发NL且并行，才需要对其进行调整,调整的目的是为了尽可能触发并行。**

****

**对比explain analyze 结果中的 concurrency 和 task**

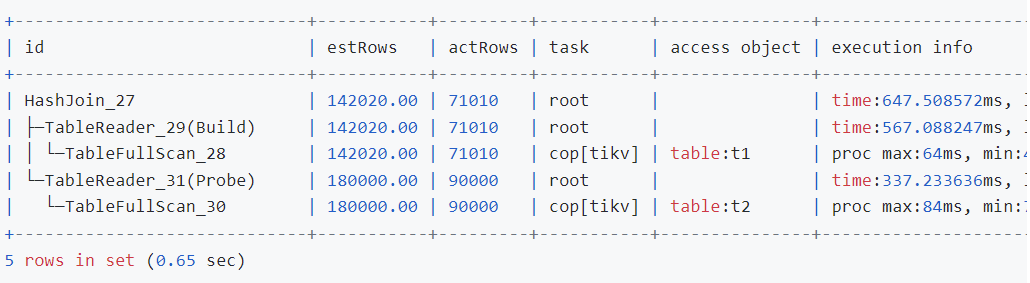
**若 concurrency < task，则并发度最高是 concurrency 的值**

**若 concurrency >= task，则并发度最高是 task 的值**

**Task=驱动表行数/[TiDB\_index\_join\_batch\_size](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_index_join_batch_size)**

1. **Hash Join**

在 Hash Join 操作中，TiDB 首先读取 Build 端的数据并将其缓存在 Hash Table 中，然后再读取 Probe 端的数据，使用 Probe 端的数据来探查 Hash Table 以获得所需行。与 Index Join 算法相比，Hash Join 要消耗更多内存，但如果需要连接的行数很多，运行速度会比 Index Join 快。TiDB 中的 Hash Join 算子是多线程的，并且可以并发执行。

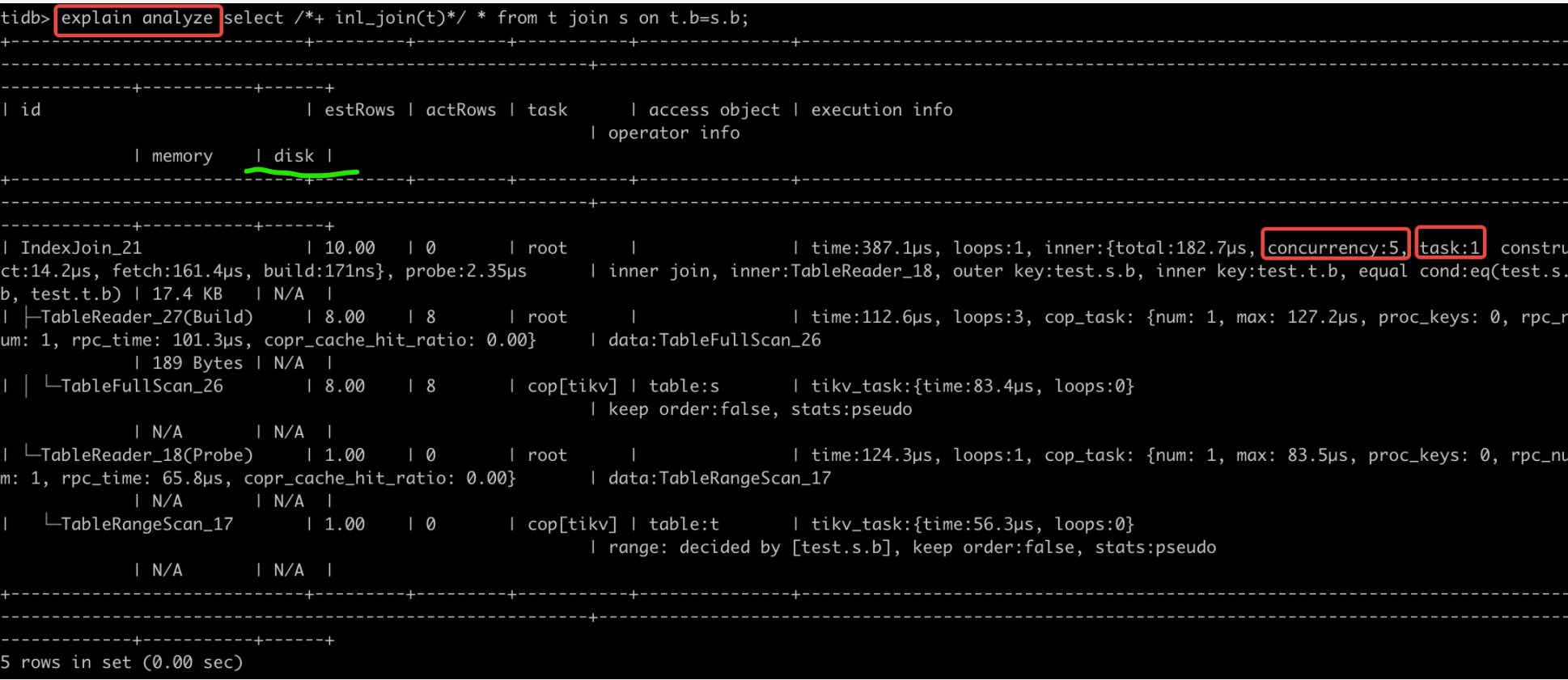


如果在执行操作时，内存使用超过了 [TiDB\_mem\_quota\_query](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_mem_quota_query) 规定的值（默认为 1GB），且 oom-use-tmp-storage 的值为 true （默认为 true），那么 TiDB 会尝试使用临时存储，在磁盘上创建 Hash Join 的 Build 端。

Hash Join 算法的性能受以下系统变量影响：

* [TiDB\_mem\_quota\_query](https://docs.pingcap.com/zh/tidb/stable/system-variables#tidb_mem_quota_query)（默认值：1GB）- 如果某条查询的内存消耗超出了配额，TiDB 会尝试将 Hash Join 的 Build 端移到磁盘上以节省内存。
* [TiDB\_hash\_join\_concurrency](https://docs.pingcap.com/zh/tidb/stable/system-variables#tidb_hash_join_concurrency)（默认值：4）- 可以并发执行的 Hash Join 任务数量。

**建议：[TiDB\_hash\_join\_concurrency](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_hash_join_concurrency)参数控制并发执行的 Hash Join 任务数量,[TiDB\_hash\_join\_concurrency](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_hash_join_concurrency)的所有任务计算过程当中所申请的内存使用量总和不超过[TiDB\_mem\_quota\_query](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_mem_quota_query)，否则超过部分会移到磁盘从而导致性能降低。**

****

**当disk>0则需要优化[TiDB\_mem\_quota\_query](https://docs.pingcap.com/zh/tidb/stable/system-variables" \l "tidb_mem_quota_query)参数。**

**备注：Hash batch size=1024**

1. **Merge Join**

Merge Join 是一种特殊的 Join 算法。当两个关联表要 Join 的字段需要按排好的顺序读取时，适用 Merge Join 算法。由于 Build 端和 Probe 端的数据都会读取，这种算法的 Join 操作是流式的，类似“拉链式合并”的高效版。Merge Join 占用的内存要远低于 Hash Join，但 Merge Join 不能并发执行。

**建议：**

**对于小表的join或者小表与大表的join，且扫描行数少，推荐index join**

**（注意：对于小于与大表的join,大表的关联字段上必须要有索引）。**

**表上无索引则建议最好根据where条件建立索引;**

**对于大表的join或扫描行数多的SQL，推荐hash join，通过消耗内存提高执行效率。**

**四. TiDB表和索引的统计信息管理**

[https://docs.pingcap.com/zh/TiDB/dev/statistics](https://docs.pingcap.com/zh/tidb/dev/statistics)

两种版本中，TiDB 维护的统计信息如下：



Version 2 的统计信息避免了 Version 1 中因为哈希冲突导致的在较大的数据量中可能产生的较大误差，并保持了大多数场景中的估算精度。

**建议：在 5.3以上（5.4以上更好）的版本使用 Version 2,analyze 的时候不要指定太高的 SAMPLERATE 防止内存消耗过大。**

* **十万行级别的表SAMPLERATE 按100%采样；**
* **百万行及以上的表SAMPLERATE 按默认值采样，在分析统计信息时间窗口足够的情况下，建议采样比按默认值，如果时间窗口不够则按SAMPLERATE=round(100000/rowscount)收集；**
* **千万行及以上的表SAMPLERATE 按默认值采样，在分析统计信息时间窗口足够的情况下，建议采样比按默认值，如果时间窗口不够则按SAMPLERATE=round(100000\*10/rowscount)收集；**
* **Disable auto analyze手动执行analyze table内部采样比计算采集 10 万行所对应的采样率;**

**对于平安Ubsql，Ubsql2.0.0对应TiDB 5.4.0内核，Ubsql1.0.1对应TiDB 4.0.15内核，该建议都适用。**

**1. 表的统计信息**

在 TiDB 中执行 ANALYZE TABLE 语句比在 MySQL 或 InnoDB 中耗时更长。InnoDB 采样的只是少量页面，但 TiDB 会完全重构一系列统计信息。适用于 MySQL 的脚本会误以为执行 ANALYZE TABLE 耗时较短。

如需更快的分析速度，可将 TiDB\_enable\_fast\_analyze 设置为 1 来打开快速分析功能。该参数的默认值为 0。

快速分析功能开启后，TiDB 会随机采样约 10000 行的数据来构建统计信息。因此在数据分布不均匀或者数据量比较少的情况下，统计信息的准确度会比较差。可能导致执行计划不优，比如选错索引。**如果可以接受普通 ANALYZE 语句的执行时间，则推荐关闭快速分析功能。**

TiDB\_enable\_fast\_analyze 为实验性功能，目前与 TiDB\_analyze\_version=2 的统计信息不完全匹配。因此开启 TiDB\_enable\_fast\_analyze 时需要将 TiDB\_analyze\_version 的值设置为 1。

收集 TableNameList 中所有表的统计信息：

ANALYZE TABLE TableNameList [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];

* WITH NUM BUCKETS 用于指定生成直方图的桶数量上限。
* WITH NUM TOPN 用于指定生成 TOPN 数目的上限。
* WITH NUM SAMPLES 用于指定采样的数目。
* WITH FLOAT\_NUM SAMPLERATE 用于指定采样率。

WITH NUM SAMPLES 与 WITH FLOAT\_NUM SAMPLERATE 这两种设置对应了两种不同的收集采样的算法。

* WITH NUM SAMPLES 指定了采样集的大小，**在 TiDB 中是以蓄水池采样的方式实现。当表较大时，不推荐使用这种方式收集统计信息。因为蓄水池采样中间结果集会产生一定的冗余结果，会对内存等资源造成额外的压力。**

**建议：尽量不要启用快速分析功能**

**v5.3及以后版本,WITH FLOAT\_NUM SAMPLERATE可以提升收集效率TiDB\_build\_stats\_concurrency 控制 analyze 的并发度，但是只对手动 analyze 生效。对于平安当前3.0和4.0版本只能通过调大TiDB\_build\_stats\_concurrency参数提高收集的效率，建议业务低峰期定时执行。**

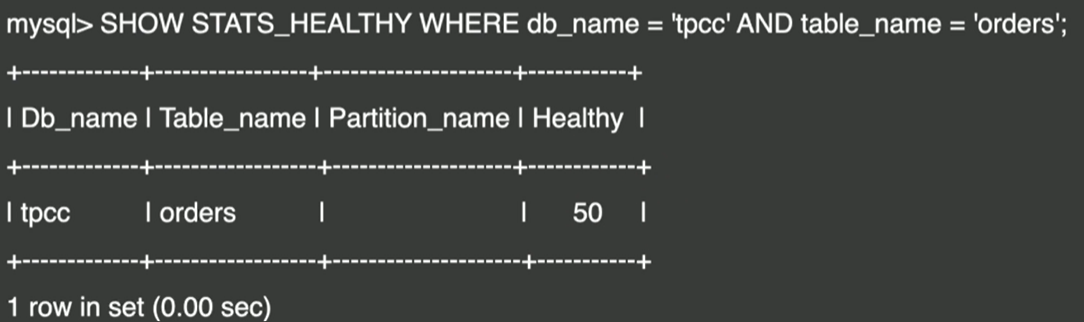
**version2 加索引的时候，表和索引就是一起收集的。**

**Version1 加索引的时候只会收集索引统计信息，表统计不会自动更新，需要手工收集。**

**2. 表的健康度判断**

**通过 SHOW STATS\_HEALTHY 可以查看表的统计信息健康度，并粗略估计表上统计信息的准确度。当 modify\_count >= row\_count 时，健康度为 0；当 modify\_count < row\_count 时，健康度为 (1 - modify\_count/row\_count) \* 100。**





**建议:每天定时扫描健康度问题,变化量超过10即healthy<90则判定为不健康，要提醒启用收集(是否自动收集)。**

**备注：**

**3.0/4.0版本触发自动收集统计信息的时候，收集队列的表的顺序是按表名排序，所以存在部分表在指定的收集时间窗口没有完成收集的情况发生。**

**6.1改为当前随机，后续按优先级优化。**

**对于平安当前3.0和4.0版本,建议通过获取非健康的表清单后，根据各个库表大小的不同编写手动收集统计信息的策略的代码定时执行。**

**3. 分区表的快速收集**

[https://book.TiDB.io/session4/chapter6/partition-table-info.html](https://book.tidb.io/session4/chapter6/partition-table-info.html)

* 如果要收集 TableName 中所有的 PartitionNameList 中分区的统计信息，请使用以下语法：

ANALYZE TABLE TableName PARTITION PartitionNameList [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];

* 如果要收集 TableName 中所有的 PartitionNameList 中分区的索引统计信息，请使用以下语法：

ANALYZE TABLE TableName PARTITION PartitionNameList INDEX [IndexNameList] [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];

* 当收集分区的统计信息时，如果只[收集部分列的统计信息](https://docs.pingcap.com/zh/tidb/dev/statistics" \l "%E6%94%B6%E9%9B%86%E9%83%A8%E5%88%86%E5%88%97%E7%9A%84%E7%BB%9F%E8%AE%A1%E4%BF%A1%E6%81%AF)，请使用以下语法：

ANALYZE TABLE TableName PARTITION PartitionNameList [COLUMNS ColumnNameList|PREDICATE COLUMNS|ALL COLUMNS] [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];

a,按分区设定并行和设定采样比收集；

**建议：对于分区的统计信息收集采样比设定。**

**十万行级别的分区SAMPLERATE 按100%采样；**

**百万行及以上的分区SAMPLERATE 按默认值采样，在分析统计信息时间窗口足够的情况下，建议采样比按默认值，如果时间窗口不够则按SAMPLERATE=round(100000/rowscount)收集；**

**千万行及以上的分区SAMPLERATE 按默认值采样，在分析统计信息时间窗口足够的情况下，建议采样比按默认值，如果时间窗口不够则按SAMPLERATE=round(100000\*10/rowscount)收集；**

b,按分区copy统计信息；

**当生产环境统计信息收集无法完成的情况下，建议：**

1. **对于双集群部署通过binlog/cdc同步的架构，可以将下游集群同分区收集的统计信息备份恢复到上游集群的同分区中。**
2. **将某个分区的统计信息备份，修改备份中信息修改后恢复到指定的分区。**

**4. 列的统计信息**

如果一个表有很多列，收集所有列的统计信息会有较大的开销。为了降低开销，你可以只收集指定列或者 PREDICATE COLUMNS 的统计信息供优化器使用。

如果要收集指定列的统计信息，请使用以下语法：

ANALYZE TABLE TableName COLUMNS ColumnNameList [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];

其中，ColumnNameList 表示指定列的名称列表。如果需要指定多列，请使用用逗号 , 分隔列名。例如, ANALYZE table t columns a, b。该语法除了收集指定表中指定列的统计信息，将同时收集该表中索引列的统计信息以及所有索引的统计信息。

如果你想查看一个表中哪些列是 PREDICATE COLUMNS，哪些列的统计信息已经被收集，可以使用以下语法：

SHOW COLUMN\_STATS\_USAGE [ShowLikeOrWhere];

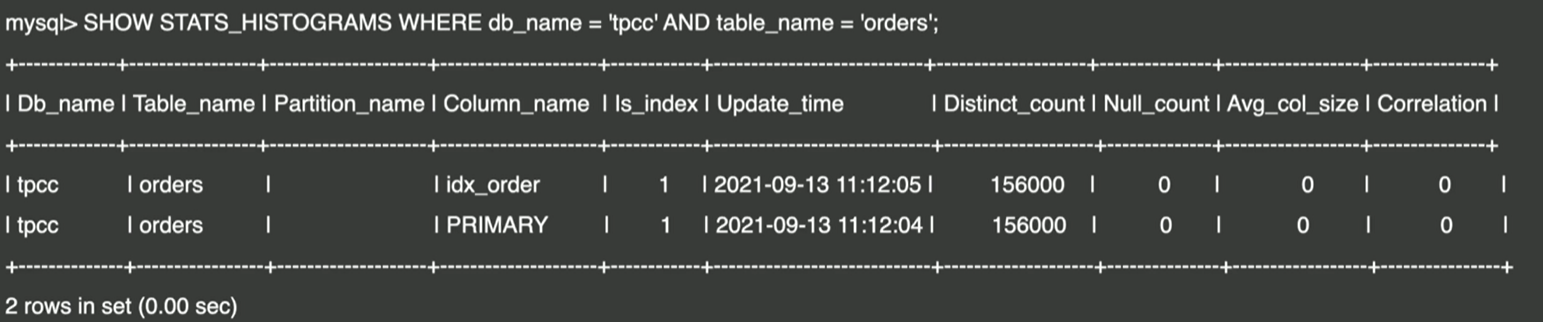
SHOW COLUMN\_STATS\_USAGE 会输出 6 列，具体如下：



备注：收集 PREDICATE COLUMNS 的统计信息目前为实验特性，不建议在生产环境中使用。

**建议：不要在生产环境使用。**

**5. 直方图信息**



Count-Min Sketch

判断一个列的数据在所有行里出现的频率,通过hash函数计算出值的计数器，获得数据的分布情况。

* WITH NUM CMSKETCH DEPTH 用于指定 CM Sketch 的长。
* WITH NUM CMSKETCH WIDTH 用于指定 CM Sketch 的宽。

修改[统计信息的收集-手动收集](https://docs.pingcap.com/zh/tidb/dev/statistics" \l "%E6%89%8B%E5%8A%A8%E6%94%B6%E9%9B%86)中提到的 WITH NUM CMSKETCH DEPTH 和 WITH NUM CMSKETCH WIDTH 两个参数，这两个参数会影响哈希的桶数和碰撞概率，可是适当调大来减少冲突概率，同时它会影响统计信息的内存使用，可以视具体情况来调整。在 TiDB 中，DEPTH 的默认值是 5(4)，WIDTH 的默认值是 2048。

**建议：由于CMSKETCH 存在误差问题，建议版本允许且对统计信息准确度要求比较高的用 version2，DEPTH  \* WIDTH 尽可能接近distinct，优先调整DEPTH\*2，WIDTH=distinct/DEPTH。百万行级别的表 按DEPTH 的默认值是5,WIDTH 的默认值是 2048；**

**千万行及以上的 按DEPTH 的默认值是5\*N(N为2的幂等),WIDTH=distinct/DEPTH；**

**对于5.3之前版本:**

**Point\_Get: WITH NUM CMSKETCH DEPTH 和 WITH NUM CMSKETCH WIDTH 两个参数有效，且存放直方图信息；**

**Index range scan:按bucket=256存放直方图信息；**

**对于5.3的version 2版本:按bucket=256存放直方图信息；**

**6. 索引的统计信息收集策略**

如果要收集 TableName 中 IndexNameList 里所有索引的统计信息，请使用以下语法：

ANALYZE TABLE TableName INDEX [IndexNameList] [WITH NUM BUCKETS|TOPN|CMSKETCH DEPTH|CMSKETCH WIDTH]|[WITH NUM SAMPLES|WITH FLOATNUM SAMPLERATE];

注意：

为了保证前后统计信息的一致性，当设置 TiDB\_analyze\_version=2 时，该语句也会收集整个表的统计信息（包括所有列和所有索引的统计信息）而不限于索引的统计信息。

https://docs.pingcap.com/zh/TiDB/dev/statistics

**建议：**

**version2 以后为了保证一致性，不会单独收集索引统计信息，即使执行 analyze table t index ... 也是会整张表一起收集，并发数通过 TiDB\_build\_stats\_concurrency 控制，但这个并发度是指单个 analyze 内部的并发，并没有参数控制同时可以跑多少个 analyze，用户主动执行的 analyze 不限个数，auto-analyze 同一时间只有一个。**

**5.3以后**

**version 2: 并发度起两个作用,1，analyze命令后面跟多张表或分区时，同时可以有多个表和分区同时收集task运行；2，对于同一张表采集数据来自多个region的时候，做merge操作的task数量。**

**version 1: 多个表和分区时候，每张表和分区分配各自的task执行收集动作，索引也是单独分配task执行收集动作。**

**对于平安当前3.0和4.0版本,根据各个库表索引大小的不同编写手动收集统计信息的策略的代码定时执行。**

**统计信息的管理对于平安当前3.0和4.0版本,根据各个库表索引大小的不同编写手动收集统计信息的策略的代码定时执行。**

**7.**

**7.1自动更新触发条件**

在发生增加，删除以及修改语句时，TiDB 会自动更新表的总行数以及修改的行数。这些信息会定期持久化下来，**更新的周期是 20 \* stats-lease，stats-lease 的默认值是 3s，如果将其指定为 0，那么将不会自动更新。**

|  |  |  |
| --- | --- | --- |
| **系统变量名** | **默认值** | **参数说明** |
| TiDB\_auto\_analyze\_ratio | 0.5 | 自动更新阈值 |
| TiDB\_auto\_analyze\_start\_time | 0:00 | 一天中能启动自动收集统计信息的起始时间 |
| TiDB\_auto\_analyze\_end\_time | 23:59 | 一天中能启动自动收集统计信息的结束时间 |

**建议：**

**开发脚本，auto analyze关闭，按表为单位手动收集统计信息。**

**1，关闭自动收集统计信息。**

**2，根据需要收集统计信息的表大小，为每个表单独指定采样比和多个脚本并行同时收集多张表的统计信息并控制好完成时间。**

**3，analyze统计信息的持久化(5.4版本以后)。**

****

**对于平安版本3.0/4.0,由于无法指定samplerate,所以需要根据实际数据库表大小进行统计信息的任务队列编排进行并发多任务收集。**

**1，统计信息收集中断,modify\_count和统计信息的一致性?**

**异常中断的analyze,modify count不会清零。analyze version1确实会存在不一致的情况,v5.3+以后的 analyze version2就没有这种情况，所以对于version1如果因为表和索引统计信息不一致对执行计划影响则建议重新收集表和索引统计信息；**

**7.2控制并发度(和cpu的关系，多少region与IO是否有关系)**

|  |  |  |
| --- | --- | --- |
| **系统变量名** | **默认值** | **参数说明** |
| TiDB\_build\_stats\_concurrency | 4 | 当前analyze执行的并行度,每个并行任务负责一个列或者一个索引，该参数控制同时执行的任务总数。 |
| TiDB\_distsql\_scan\_concurrency | 15 | 控制SQL表的任务一次读取的Region的数量,同样适用于analzye,session level配置 |
| TiDB\_index\_serial\_scan\_concurrency | 1 | 控制SQL索引的任务一次读取Region的数量 |

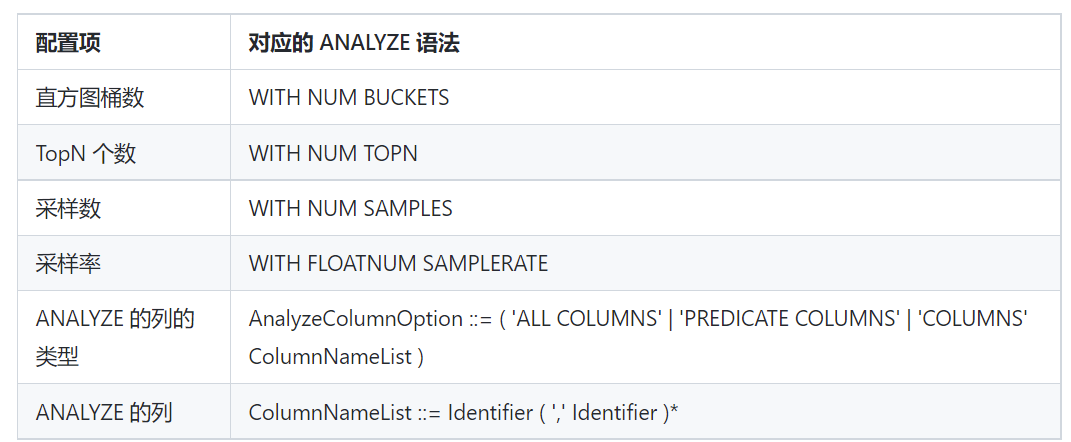
**建议：**

**在统计信息任务编排后，为了进一步优化和提升单个对象的收集效率，需要调整该参数。**

**对于任务队列中的大表，通过调整TiDB\_build\_stats\_concurrency，TiDB\_distsql\_scan\_concurrency和TiDB\_index\_serial\_scan\_concurrency**

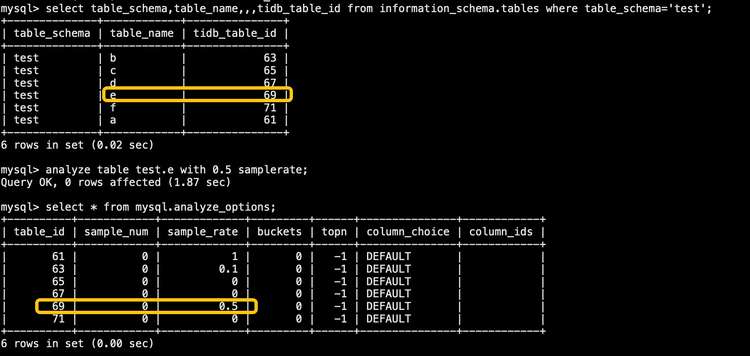
**7.3 ANALYZE 配置持久化**

从v5.4.0起,TiDB支持ANALYZE配置持久化功能,方便后续收集统计信息时沿用已有配置项。TiDB 支持持久化的配置项包括：



开启 ANALYZE 配置持久化功能ANALYZE配置持久化功能默认开启（系统变量TiDB\_analyze\_version为默认值2,TiDB\_persist\_analyze\_options为默认值ON),用于记录手动执行ANALYZE语句时指定的持久化配置项.记录后,当 TiDB下一次自动更新统计信息或者你手动收集统计信息但未指定配置项时,TiDB 会按照记录的配置项收集统计信息。

多次手动执行ANALYZE语句并指定持久化配置项时,TiDB会使用最新一次ANALYZE指定的配置项覆盖上一次记录的持久化配置。

****

**建议：对于v5.4.0及以后版本，建议针对不同的表配置不同的收集策略，统一适用TiDB的统计信息调度程序调度。**

**对于平安当前3.0和4.0版本,根据各个库表索引大小的不同编写手动收集统计信息的策略的代码定时执行。**

**7.4 ANALYZE运行状态检查**

在执行 ANALYZE 时，可以通过 SQL 语句来查看当前 ANALYZE 的状态。

SHOW ANALYZE STATUS [ShowLikeOrWhere];



**建议：**

**监控到达时间没有完成的要结束任务，并记录并调整采样比和任务编排策略。**

**对于平安当前3.0和4.0版本,建议将这些表或分区根据当时的手动收集统计信息的策略的优化排成规则。**

**7.5 表统计信息状态检查**

可以通过 SHOW STATS\_META 来查看表的总行数以及修改的行数等信息。

SHOW STATS\_META [ShowLikeOrWhere];



没有出现在这个清单中的表均为需要收集统计信息的表。

**建议：**

**监控24小时更新时间没有更新的表需要人工收集统计信息。**

**对于平安当前3.0和4.0版本,建议将这些表或分区根据当时的手动收集统计信息的策略的优化排成规则。**

**7.6 表健康度检查**

通过 SHOW STATS\_HEALTHY 可以查看表的统计信息健康度，并粗略估计表上统计信息的准确度。当 modify\_count >= row\_count 时，健康度为 0；当 modify\_count < row\_count 时，健康度为 (1 - modify\_count/row\_count) \* 100。



**建议：**

**监控healthy低于90的为需要收集统计信息的。**

**对于平安当前3.0和4.0版本,建议将这些表或分区根据当时的手动收集统计信息的策略的优化排成规则。**

**7.7 统计备份与恢复**

导出统计信息

通过以下接口可以获取数据库 ${db\_name}中的表${table\_name}的JSON格式的统计信息:

http://${TiDB-server-ip}:${TiDB-server-status-port}/stats/dump/${db\_name}/${table\_name}

语法如下:

curl -s http://127.0.0.1:10080/stats/dump/test/t1 -o /tmp/t1.json

通过以下接口可以获取数据库${db\_name}中的表${table\_name}在指定时间上的JSON格式统计信息。（历史统计信息保存)

http://${TiDB-server-ip}:${TiDB-server-status-port}/stats/dump/${db\_name}/${table\_name}/${yyyyMMddHHmmss}

导入统计信息

注意:启动MySQL客户端时,请使用--local-infile=1参数。

导入的统计信息一般是通过统计信息导出接口得到的json文件。

语法如下:

LOAD STATS '/path/file\_name';

file\_name 为要导入的统计信息的文件名。

**建议：由于统计信息统一为认为脚本控制，所以在收集统计信息前备份统计信息。**

**可以手动修改json文件里面的modify count和修改统计信息，再导入统计信息。**

**备注：产品版本有规划后期实现统计信息历史保存的功能。**

**五. TiDB执行计划的管理**

**1. 影响CBO的因素分析**

a,表,列和索引统计信息,决定表的join方式,索引的选择及索引扫描方式。

b,直方图信息,列的数据的分布情况决定索引的扫描方式。

c,modify\_count,基于最近一次统计信息的增量数据的统计。

**2. hint的支持方式**

[https://docs.pingcap.com/zh/TiDB/stable/optimizer-hints#optimizer-hints](https://docs.pingcap.com/zh/tidb/stable/optimizer-hints#optimizer-hints)

TiDB 目前支持的 Optimizer Hints 根据生效范围的不同可以划分为两类：

第一类是在查询块范围生效的 Hint，例如 [/\*+ HASH\_AGG() \*/](https://docs.pingcap.com/zh/tidb/stable/optimizer-hints#hash_agg)；

第二类是在整个查询范围生效的 Hint，例如 [/\*+ MEMORY\_QUOTA(1024 MB)\*/](https://docs.pingcap.com/zh/tidb/stable/optimizer-hints#memory_quotan)。

**3. SPM 执行计划管理**

[https://docs.pingcap.com/zh/TiDB/stable/sql-plan-management](https://docs.pingcap.com/zh/tidb/stable/sql-plan-management)

执行计划管理，又称 SPM (SQL Plan Management)，是通过执行计划绑定，对执行计划进行人为干预的一系列功能，包括执行计划绑定、自动捕获绑定、自动演进绑定等。

执行计划绑定是 SPM 的基础。在优化器 Hints?中介绍了可以通过 Hint 的方式选择指定的执行计划，但有时需要在不修改 SQL 语句的情况下干预执行计划的选择。执行计划绑定功能使得可以在不修改 SQL 语句的情况下选择指定的执行计划。

a,创建绑定

CREATE [GLOBAL | SESSION] BINDING FOR BindableStmt USING BindableStmt;

该语句可以在 GLOBAL 或者 SESSION 作用域内为 SQL 绑定执行计划。目前支持的可创建执行计划绑定的 SQL 类型 (BindableStmt) 包括：SELECT，DELETE，UPDATE和带有SELECT子查询的INSERT/REPLACE。

b,自动捕获绑定 (Baseline Capturing)

通过将TiDB\_capture\_plan\_baselines的值设置为on（其默认值为off）可以打开自动捕获绑定功能。

**注意**

自动绑定功能依赖于Statement Summary，因此在使用自动绑定之前需打开 Statement Summary 开关。

开启自动绑定功能后，每隔?bind-info-lease（默认值为?3s）会遍历一次 Statement Summary 中的历史 SQL 语句，并为至少出现两次的 SQL 语句自动捕获绑定。绑定的执行计划为 Statement Summary 中记录执行这条语句时使用的执行计划。

对于以下几种 SQL 语句，TiDB 不会自动捕获绑定：

EXPLAIN 和 EXPLAIN ANALYZE 语句；

TiDB 内部执行的 SQL 语句，比如统计信息自动加载使用的 SELECT 查询；

存在手动创建的执行计划绑定的 SQL 语句；

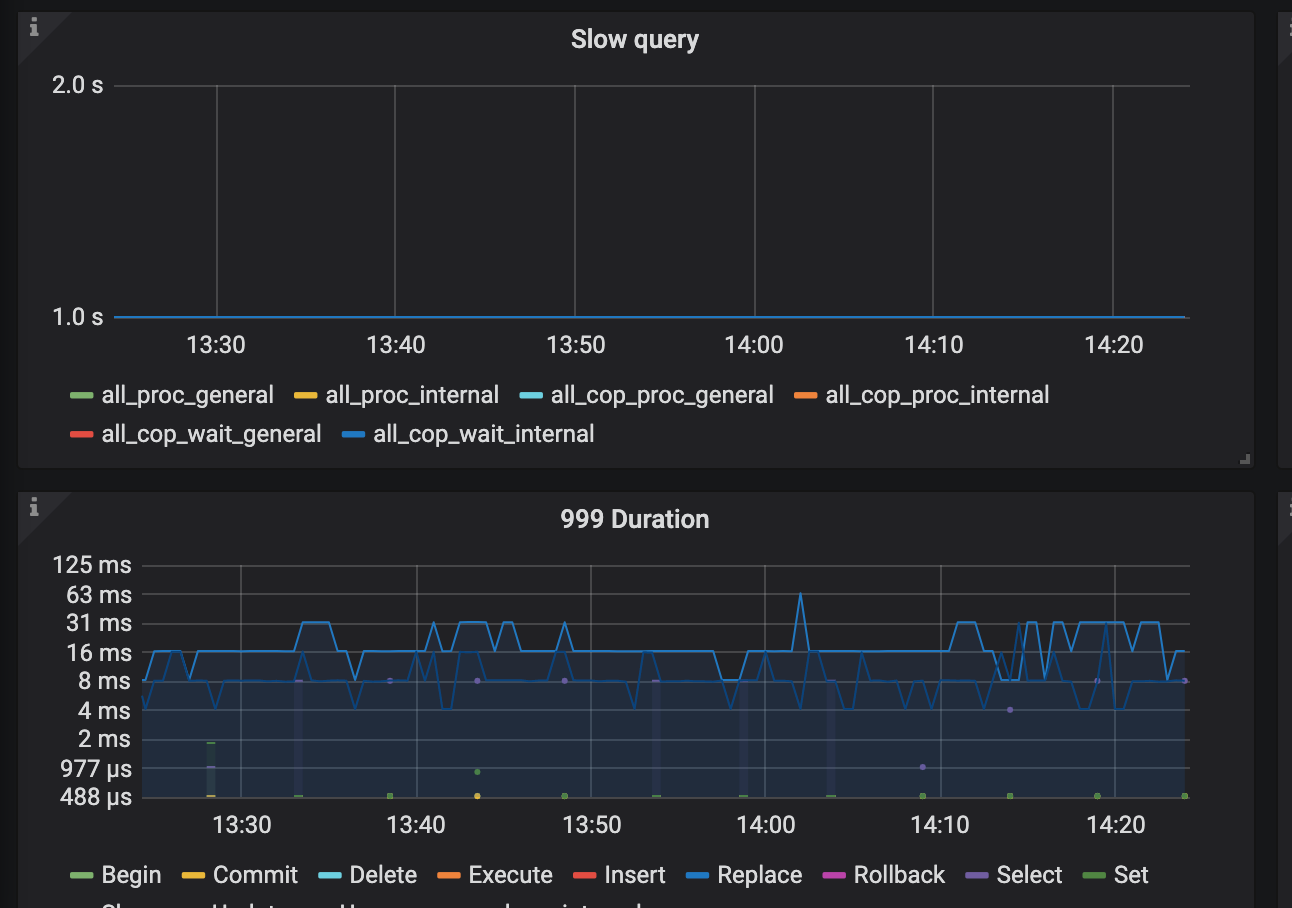
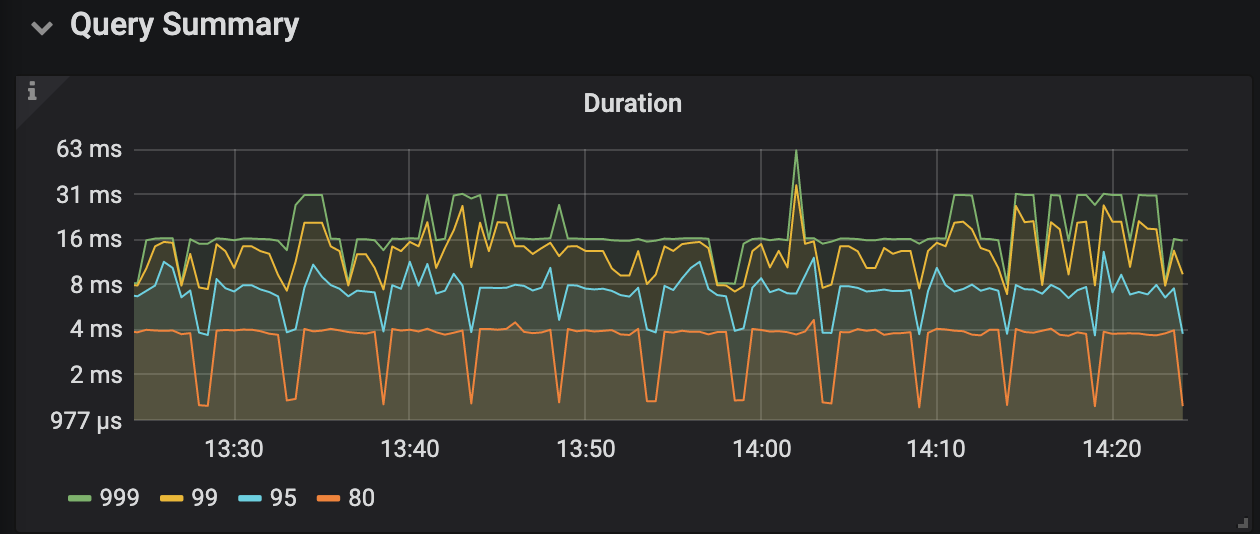
对于PREPARE/EXECUTE语句组，或通过二进制协议执行的查询，TiDB 会为真正的查询（而不是PREPARE/EXECUTE语句）自动捕获绑定。

**六. 慢SQL的管理**

[https://docs.pingcap.com/zh/TiDB/stable/identify-slow-queries#%E6%85%A2%E6%9F%A5%E8%AF%A2%E6%97%A5%E5%BF%97](https://docs.pingcap.com/zh/tidb/stable/identify-slow-queries#%E6%85%A2%E6%9F%A5%E8%AF%A2%E6%97%A5%E5%BF%97)

**1. 如何判断慢SQL**

a. 通过grafana判断query的duration的分布情况



查看Dashboard或手动执行如下SQL查看慢的SQL

SELECT SUMMARY\_BEGIN\_TIME,SUMMARY\_END\_TIME,digest,QUERY\_SAMPLE\_TEXT,

CAST(SUM(exec\_count\*avg\_latency)/SUM(exec\_count)/1000/1000 AS SIGNED ) AS agg\_avg\_latency\_ms

FROM INFORMATION\_SCHEMA.CLUSTER\_STATEMENTS\_SUMMARY\_HISTORY p

where SUMMARY\_BEGIN\_TIME > date\_add(now(), interval -2 hour)

GROUP BY digest,QUERY\_SAMPLE\_TEXT,SUMMARY\_BEGIN\_TIME,SUMMARY\_END\_TIME HAVING agg\_avg\_latency\_ms>5000;

b. 慢日志分析

https://docs.pingcap.com/zh/TiDB/stable/identify-slow-queries#%E6%85%A2%E6%9F%A5%E8%AF%A2%E6%97%A5%E5%BF%97

TiDB 会将执行时间超过 [slow-threshold](https://docs.pingcap.com/zh/tidb/stable/tidb-configuration-file#slow-threshold)（默认值为 300 毫秒）的语句输出到 [slow-query-file](https://docs.pingcap.com/zh/tidb/stable/tidb-configuration-file#slow-query-file)（默认值："TiDB-slow.log"）日志文件中，用于帮助用户定位慢查询语句，分析和解决 SQL 执行的性能问题。

https://book.TiDB.io/session3/chapter3/slow-query-table.html

检索当前节点 Top N 慢查询

以下 SQL 用于检索当前TiDB节点的 Top 2 慢查询：

select query\_time, query

from information\_schema.slow\_query -- 检索当前 TiDB 节点的慢查询

where is\_internal = false -- 排除 TiDB 内部的慢查询

order by query\_time desc

limit 2;

+--------------+------------------------------------------------------------------+

| query\_time | query |

+--------------+------------------------------------------------------------------+

| 12.77583857 | select \* from t\_slim, t\_wide where t\_slim.c0=t\_wide.c0; |

| 0.734982725 | select t0.c0, t1.c1 from t\_slim t0, t\_wide t1 where t0.c0=t1.c0; |

+--------------+------------------------------------------------------------------+

检索全部节点上指定用户的 Top N 慢查询

以下 SQL 会检索全部 TiDB 节点上指定用户 test 的 Top 2 慢查询：

select query\_time, query, user

from information\_schema.cluster\_slow\_query -- 检索全部 TiDB 节点的慢查询

where is\_internal = false

and user = "test"

order by query\_time desc

limit 2;

+-------------+------------------------------------------------------------------+----------------+

| Query\_time | query | user |

+-------------+------------------------------------------------------------------+----------------+

| 0.676408014 | select t0.c0, t1.c1 from t\_slim t0, t\_wide t1 where t0.c0=t1.c1; | test |

+-------------+------------------------------------------------------------------+----------------+

**2. Dashboard判断关键业务SQL**

[https://docs.pingcap.com/zh/TiDB/dev/top-sql](https://docs.pingcap.com/zh/tidb/dev/top-sql)

通过选择一段时间范围查询的SQL,按平均耗时,内存消耗,执行次数获取关键和敏感的SQL。



1,如何在Dashboard上直接绑定SPM？(开发需求)，对于这些SQL通过手动SPM固定。

2,目前产品还不支持从grafana的指标数据中下钻到Dashboard看SQL的执行情况，所以只能通过Dashboard的SQL延迟情况判断慢的SQL。

**3. 从数据字典中获取SQL**

[https://book.TiDB.io/session3/chapter2/statements.html](https://book.tidb.io/session3/chapter2/statements.html)

针对 SQL 性能相关的问题，TiDB Dashboard 提供了 Statements 用来监控和统计 SQL。



例如页面上提供了丰富的列表信息，包括延迟、执行次数、扫描行数、全表扫描次数等，用来分析哪些类别的 SQL 语句耗时过长、消耗内存过多等情况，帮助用户定位性能问题。

https://book.TiDB.io/session3/chapter3/statements.html

TiDB\_enable\_stmt\_summary

Statements功能默认开启,也可以通过设置系统变量打开,例如：

set global TiDB\_enable\_stmt\_summary = true;

TiDB\_stmt\_summary\_refresh\_interval

设置performance\_schema.events\_statements\_summary\_by\_digest表的清空周期,单位是秒 (s),默认值是 1800,例如:

set global TiDB\_stmt\_summary\_refresh\_interval = 1800;

TiDB\_stmt\_summary\_history\_size

设置performance\_schema.events\_statements\_summary\_by\_digest\_history表保存每种SQL的历史的数量,默认值是24,例如:set global TiDB\_stmt\_summary\_history\_size = 24;以上配置生效后，statements\_summary每 30分钟清空一次，所以statements\_summary\_history保存最近 12 小时的历史数。statements\_summary\_evicted保存最近 24 个发生了 evict 的时间段记录；statements\_summary\_evicted则以 30 分钟为一个记录周期，表容量为 24 个时间段。

由于Statements信息是存储在内存表中,为了防止内存溢出等问题,需要限制保存的SQL条数和SQL的最大显示长度。这两个参数需要在config.toml的[stmt-summary]类别下配置：

通过max-stmt-count更改保存的SQL种类数量,默认200条.当SQL种类超过max-stmt-count时,会移除最近没有使用的SQL通过max-sql-length更改DIGEST\_TEXT和QUERY\_SAMPLE\_TEXT的最大显示长度,默认是4096

注意：

TiDB\_stmt\_summary\_history\_size,max-stmt-count,max-sql-length几项配置影响内存占用,建议根据实际情况调整,不宜设置得过大。

**4. 从TiDB.log中获取性能差的SQL**

如果一条语句在执行过程中达到或超过资源使用阈值时（执行时间/使用内存量）则会即时将这条语句写入到日志文件（默认文件为：TiDB.log）

**4.1 限制 SQL 内存消耗**

https://book.TiDB.io/session3/chapter1/memory-quota-execution-time-limit.html

通过配置文件参数修改

oom-action

默认值："log"

当 TiDB 中单条 SQL 的内存使用超出 mem-quota-query 限制且不能再利用临时磁盘时的行为。

目前合法的选项为 ["log", "cancel"]。

如果配置项使用的是 "log"，那么当一条 SQL 的内存使用超过一定阈值后，TiDB 会在 log 文件中打印一条 LOG，然后这条 SQL 继续执行，之后如果发生了 OOM 可以在 LOG 中找到对应的 SQL。

如果上面的配置项使用的是 "cancel"，那么当一条 SQL 的内存使用超过一定阈值后，TiDB 会立即中断这条 SQL 的执行并给客户端返回一个 error，error 信息中会详细写明这条 SQL 执行过程中各个占用内存比较多的物理执行算子的内存使用情况。

mem-quota-query

默认值：34359738368（32GB）

单条 SQL 语句可以占用的最大内存阈值。

超过该值的请求会被 oom-action 定义的行为所处理。

**4.2 限制 SQL 执行时间**

https://book.TiDB.io/session3/chapter1/memory-quota-execution-time-limit.html

修改session/global变量

max\_execution\_time

单位为：ms

目前对所有类型的 statement 生效，并非只对 SELECT 语句生效。实际精度在 100ms 级别，而非更准确的毫秒级别。

**建议：借助SPM将以上关键SQL的固定hint和执行计划。**

**备注：同时参考<<SPM固化执行计划指引 v1.0>>**

**七. 监控**

**-- 监控sql语句有两个以上的执行计划,不同plan的执行耗时超1s**

SELECT digest,plan\_digest,

CAST(

SUM(exec\_count \* avg\_latency) / SUM(exec\_count)/1000/1000 AS SIGNED

) AS agg\_avg\_latency\_ms

FROM

`INFORMATION\_SCHEMA`.`CLUSTER\_STATEMENTS\_SUMMARY\_HISTORY` p

WHERE

digest in ( SELECT

digest

FROM

`INFORMATION\_SCHEMA`.`CLUSTER\_STATEMENTS\_SUMMARY\_HISTORY` t

GROUP BY

digest

HAVING

-- SQL语句有两个以上的 plan

COUNT(DISTINCT plan\_digest) > 1)

GROUP BY plan\_digest,digest

-- plan 平均执行时间超过 1s

HAVING agg\_avg\_latency\_ms > 1000;

**--查看执行计划发生变更的慢查询**

SELECT COUNT(distinct plan\_digest) AS count ,digest ,MIN(query)

FROM cluster\_slow\_query

GROUP BY digest

HAVING count > 1

LIMIT 3;

**-- 监控，两小时内执行时间超过 5s 的SQL**

SELECT SUMMARY\_BEGIN\_TIME,SUMMARY\_END\_TIME,digest,QUERY\_SAMPLE\_TEXT,

CAST(

SUM(exec\_count \* avg\_latency) / SUM(exec\_count)/1000/1000 AS SIGNED

) AS agg\_avg\_latency\_ms

FROM

`INFORMATION\_SCHEMA`.`CLUSTER\_STATEMENTS\_SUMMARY\_HISTORY` p

where SUMMARY\_BEGIN\_TIME > date\_add(now(), interval -2 hour)

GROUP BY digest,QUERY\_SAMPLE\_TEXT,SUMMARY\_BEGIN\_TIME,SUMMARY\_END\_TIME

HAVING agg\_avg\_latency\_ms > 5000;

**建议：这些SQL的输出结果对接到监控平台。**