# 概述

# 优化器

## TDSQL

## PolarDB

## GoldenDB

和单机数据库相比，GoldenDB分布式数据库系统中多了一层全局概念模式到本地概念模式的映射。表数据也被水平切分到多个数据节点。分布式查询优化器的优劣直接影响着业务性能。GoldenDB分布式数据库查询优化器主要朝着如下三个方向努力：

1. 最大程度使数据操作本地化、局部化，减少网络通信的交互次数和交换的数据量，提升数据节点的并行计算是分布式数据库系统优化器致力的重要方向之一；
2. 如何降低数据全局一致性保证的开销，也是分布式数据库系统查询优化努力的方向；
3. 在分布式数据库系统中使用单机数据库的优化手段或者其变形，从而使存储节点承担更多的优化工作，在全局层面仅作少量的启发式优化。

### 影响因素

优化器的优化工作主要体现在计划树的生成上，GoldenDB的查询优化器设计实现主要考虑以下两个方面：

1. 代价模型的选择。

GoldenDB采用分布式系统代价估算模型，考虑节点间传输数据的代价，以减少数据传输的次数和数据量作为查询优化的目标，提高数据节点之间计算的并行度、减少计算节点的计算量。这主要考虑在分布式数据库系统环境中，表结构被水平或垂直拆分到多个数据节点，因此需要考虑语句如何分拆、分片之间数据如何移动、结果如何计算与合并的问题，网络通信开销不可忽视。

1. 考虑数据一致性开销。

在分布式数据库系统中，数据全局一致性机制相较于单机数据库需要更为复杂的控制。因此，如何降低数据全局一致性保证的开销，也是GoldenDB查询优化器的设计要求。

总体来讲，GOldenDB的分布式查询优化器遵循了上述的设计原则，以基于规则的优化为主，基于成本的优化为辅，在提升系统的灵活性的同时控制系统实现的复杂性。优化器内部内置大量的优化规则，通过查询重写的方式进行经验性优化。在优化规则的选择上，重点分析分片剪枝、并行执行、合并下压、条件下推、条件繁殖、排序消除、去重消除、排序下推等。

内置大量的优化规则，对上百个场景进行优化，复杂SQL语句兼容性和处理性能好，同时支持prepare预处理、执行计划缓存、数据集透传等功能，保证数据一致性条件下实现高性能SQL处理。

支持的典型优化包括：

1. 分片剪枝
2. 合并下压
3. 并行执行
4. 条件下推及条件繁殖
5. 排序下推、limit下推等
6. 聚合函数优化

得益于完善的优化器设计，使得GoldenDB对单节点、跨节点的复杂SQL的兼容支持程度很高，包括跨节点SUM、COUNT、AVG等汇聚类操作，跨节点WHERE、FROM等子查询，跨节点JOIN，跨节点GROUP BY、ORDER BY、LIMIT等。这是GoldenDB将Proxy命名为计算节点而非中间件的原因之一，也是其和很多分布式数据库产品中间件的主要区别。

### 合并下压优化

#### 表层次

#### 主子查询

### SQL引擎优化

#### 条件繁殖

条件繁殖是指优化器对已知条件进行推断，从而衍生出其他条件进行改造，缩小数据检索的范围。繁殖后的条件，或推入基表、或下压到数据节点执行。

SQL示例1：

select T1.col,T1.col1 from T1 join T2 on T1.col=T2.col where T1.col>100 UR;

优化后执行计划示例：

优化器会识别出本例中的条件传递从而推断出T2.col>100，并将该条件推入基表。语句被重写：

select T1.col,T1.col1 from {T1 where col>100} join {T2 where col>100} on T1.col=T2.col UR;

注：在等价语句改造的时候，也可以利用ON条件等做等值的传递。

SQL示例2：

select T1.col, T1.col1 from T1 join T2 on T1.col=T2.col where T1.col1 in (100,200) and T2.col1=T1.col1 UR;

优化器执行计划示例：

优化器会识别出本例中的条件传递从而推出T2.col1 in (100,200)，并将该条件推入基表。语句被重写为：

select T1.col, T1.col1 from {T1 where col1 in (100,200)} join {T2 where col1 in (100,200)} von T1.col=T2.col UR;

#### 并行执行

并行执行是指执行计划在各个分区间进行并行执行，从而提升执行效率。

当SQL查询在分区剪裁后，仍然涉及多个分区时，会生成一个分布式执行计划，该分布式计划会被调度到分区所在不同机器上进行执行。GoldenDB在判断语句需要下发到多个分区时，会将语句拆分成多个同时下发到对应的节点并行执行。

SQL示例：

假定T1、T2在g1,g2,g3三个分区上：

select T1.col,T1.col1 from T1 join T2 on T1.col=T2.col where T1.col1=200 UR;

优化后执行计划，语句被拆分如下：

select T2.col from T2 order by T2.col ASC;

select T1.col,T1.col1 from T1 where T1.col1=200 order by T1.col ASC;

同时下发到对应的三个节点上并行执行后，将结果汇总到proxy层做join。

#### AVG优化

在GoldenDB中，AVG被自动重写成SUM和COUNT两个计算，在每个数据节点上，只返回本数据节点的SUM、COUNT；在计算节点层面，再对各数据节点返回的SUM、COUNT进行累计，然后再用SUM/COUNT得到AVG的最终结果。

SQL示例：

select avg(col) from T where col>100 UR;

优化后执行计划示例：

select sum(col),count(col) from T where col>100;

#### WHERE条件下推

在分布式数据库系统实现中，为了尽量减少数据节点向计算节点移动的数据量，系统被设计为尽可能将where条件下推到数据节点。

SQL示例：

假定T为单分发键表，且分发键为col：

select col,col1 from T where col=100 and col1>10 UR;

优化后执行计划示例：

根据where子句可以将查询数据定位到某（几）个GROUP，执行时将语句直接下发到对应的group节点上执行：

select col,col1 from T where col=100 and col1>10;

#### order by下推

GoldenDB对于排序处理通常会优先考虑推入数据节点完成。利用数据节点的计算能力并行完成排序操作；涉及结果合并的，计算节点再对有序数据集进行合并排序。

SQL示例：

select col,col1 from T where col1>100 order by col UR;

优化后执行计划示例：

将语句下发到各个节点上并行执行，并在proxy层汇总结果，如果下发为多节点，则需要执行sort merge排序操作。

select col,col1 from T where col1>100 order by col;

#### distinct下推

GoldenDB中，遇到不能合并下发的SQL语句，如果其中含有distinct，则计算节点在拆分语句时，会考虑将distinct下推入数据节点执行。以减少从数据节点提取到计算节点的数据量。

UR场景下，如果查询数据分布在同一个节点上或者select list为分发键的情况下，将distinct下推入节点执行，汇总结果不需要在proxy层再做distinct。

SQL示例：

假定T为多分发键表，且分发键为col1,col1：

select distinct col,col1 from T where col>100 and col1=20 UR;

优化后执行计划示例：

select distinct col,col1 from T where col>100 and col1=20;

#### limit下推

在GoldenDB中，limit下推的主要目的是在需要计算节点进一步计算的场景下，尽量减少从数据节点提取到计算节点的数据量。

优化原则：

1. SQL语句能下发到一个db group执行的，limit子句不用调整
2. SQL语句下发到多个db group执行的，需要在proxy层汇总数据，做limit操作：
3. SQL语句能下发，但是需要下发到多个db group执行的，limit子句需要调整，调整格式如下：limit x,y --> limit 0,x+y
4. SQL语句不能下发，需要把数据拉到db proxy层计算的，limit子句不变

SQL示例：

查询数据分布在多个节点上，且查询语句能下发：

select col,col1 from T where col1>100 limit 2,2 UR;

优化后执行计划：

select col,col1 from T where col1>100 limit 0,4;

#### 常数折叠

在GoldenDB中，为了减少对确定值的反复计算而先进行计算的优化方法。此过程一般发生在S（Select）F（From）W（Where）中SW阶段。

SQL示例：

假定T为单分发键表，且分发键为col：

select col,col1 from T where col=50+50 and col1>10 UR;

优化后执行假话示例：

在此查询中，50+50会被先计算橙100。查询重写后，条件变为where col=100 and col1>10。根据重写后的where条件可以将查询语句定位到某（几）个group上执行，执行时下发原始条件。

select col,col1 from T where col=50+50 and col>10;

#### 非逻辑优化

在GoldenDB中，会针对NOT运算进行处理，通常是将其下推，将表达式整体取反变为表达式分量补集的运算。

|  |  |
| --- | --- |
| 处理前 | 处理后 |
| NOT (col!=5) | col=5 |
| NOT(col1<=4 OR col2>0) | col1>4 AND col2<=0 |
| NOT(col1<=4 AND col2>0) | col1>4 OR col2<=0 |

经过变换，可以减少一次逻辑运算并在一定条件下使范围扫描可用。

#### 死代码消除

GoldenDB分布式优化器中的处理逻辑通过判断出为恒指或者逻辑冗余的条件，然后在运行时减少不必要的逻辑判断，从而提升执行效率。

SQL示例：

假定T为range表，col为单分发键：

select col,col1 from T where col>0 and col > 200 UR;

优化后执行计划：

此例子中，根据where条件确定下发group（col>200所在group）：

select col,col1 from T where col>0 and col>200;

SQL示例：

select col,sol1 from T where col<0 and col>200 UR;

优化后执行计划：

此例中，根据where条件获取下发group为0，选择一个group下发执行：

select col,col1 from T where col<0 and col>200;

#### 合并下发

GoldenDB的分布式优化，很重要的一个努力方向就是尽量利用数据节点的计算能力进行计算，避免不必要的从数据节点向计算节点的数据移动，并减少和数据节点交互的次数。计算节点分析语句后，尽量把能够一起执行的语句下发到数据节点。

注：这里涉及到表层次合并、主子查询合并以及JOIN等合并下发规则的判断。

SQL示例：

假定T1和T2分发属性相同：range表、分发键为col、分布在g1,g2,g3节点上，将“条件繁殖”部分的示例语句改写如下：

select T1.col,T1.col1 from T1 join T2 on T1.col=T2.col where T1.col=200 UR;

见“where条件下推”部分，根据where子句可以将查询数据定位到某个group上，则可以直接将语句下发到group节点上执行。

优化后执行计划：

select T1.col,T1.col1 from T1 join T2 on T1.col=T2.col where T1.col=200;

### 执行器

## OceanBase

## TiDB