SQL Server调优系列玩转篇（如何利用查询提示（Hint）引导语句运行）

前言

前面几篇我们分析了关于SQL Server关于性能调优的一系列内容，我把它分为两个模块。

第一个模块注重基础内容的掌握，共分7篇文章完成，内容涵盖一系列基础运算算法，详细分析了如何查看执行计划、掌握执行计划优化点，并一一列举了日常我们平常所写的T-SQL语句所会应用的运算符。我相信你平常所写的T-SQL语句在这几篇文章中都能找到相应的分解运算符。

第二个模块注重SQL Server执行T-SQL语句的时候一些内幕解析，共分为5篇文章完成，其中包括：查询优化器的运行方式、运行时几个优化指标值检测，统计信息、利用索引等一系列内容。通过这块内容让我们了解SQL Server为我们所写的T-SQL语句如何进行优化及运行的。

从本篇进入第三个模块的内容，该篇为第一篇，该模块主要让我们来指导SQL Server进行定向调整，达到优化的目的。本模块的内容是以前面一系列内容为前提的，希望充分掌握了前面基础内容，方能进入本模块内容。

技术准备

数据库版本为SQL Server2012，利用微软的以前的案例库（Northwind）进行分析，部分内容也会应用微软的另一个案例库AdventureWorks。

相信了解SQL Server的朋友，对这两个库都不会太陌生。

概念理解

谈到hint，其实概念很简单，正如词义理解：提示，也就是说让我们通过给予SQL Server提示（hint）让数据库运行时按照我们的思路进行，我估计很多不怎么了解SQL Server的童鞋都不怎么知道，因为一般应用的不多。

其实，SQL Server本身的查询优化器已经做到很好了，所以大部分情况下不需要我们人工干预，自己就能运行的很好，并且最大限度的优化运行项。但是，俗话说：老虎也有打盹的时候，所以，在有些场景下，就需要我们来给数据库指导一个方向，让其运行的更流畅。

但是，记住了：你所应用的hint是在现在的场景中基于现有的环境下，相对是一个好的方式，不能确保你所给予的提示（Hint）永久有效，并且随着时间推移，数据量的变更，你所发出的提示（Hint）有可能会成为数据库优化的绊脚石。所以没有充分的把握不要轻易使用Hint，并且最好采用目标导向Hint。

Hint主要分为三类应用：查询Hint、表Hint、连接Hint。查询Hint影响整个查询，主要应用于查询语句优化，本篇主要分析查询Hint。

表Hint影响查询引用的单个表，而连接Hint影响一个单独的连接。

Hint应用方式分为两类：目标导向Hint和物理运算符Hint。

目标导向Hint传递逻辑的目标给优化器，而不会具体指定优化器应该如何达到这个目标，应该使用什么物理运算符，或者如何排列这些运算符。所以这种运算符使我们所推荐的，原因很简单：**我告诉丫按照这个思路执行就可以，至于怎么达到，自己想办法！**这种方式从长期看对于数据库的影响会小很多。

另外一个就是物理运算符，此方式就更直接了：**直接告诉丫的步骤，你按照这个去做就行。**这种方式不推荐，原因很简单：你的思路暂时会是好的，但是过段时间就不好了。

一、查询提示（Hint）

首先，查询提示（Hint）是我们在调优中应用最广泛的，因为大部分时间我们是在调整查询的性能。

关于查询中的优化选项就是在指导SQL Server的连接类型、聚合类型、联合类型等物理连接运算符。关于此块的详细解析，可以参照我调优系列中前几篇文章，分析的相当的详细。

a、FAST N Hint提示

关于此方式的提示，我在前面的文章中已经有使用到，在介绍索引那篇文章中，可以点击这里查看。

首先，这个Hint是一个目标导向hint。提示目标很简单：告诉数据库给我速度出前N行数据就可以，而其它的数据你爱咋地咋地。

这个提示最优的应用环境就是：应用系统中的分页查询，当然其它环境可以用。有点类似于SELECT TOP N....

其次，在我们的应用环境中，尤其数据量多的情况下，如果这时候我们的场景是：我想速度的看到前面的部分数据，其它的数据你可以稍后再显示，但是在执行T-SQL的时候，SQL Server会多方面的考虑耗费（cost），然后再平衡各种利弊选择出它认为相对好的执行计划去执行，显然这种方式获取数据的方式是很浪费的，并且速度就会相对慢很多。

所以，我们利用FAST N Hint提示，这样，SQL Server会阻止优化器使用哈希连接、哈希聚合、排序、甚至是并行这些大消耗的动作，而转变成为这N条数据做快速的优化并输出。这在大数据量的情况下，是一种非常高明的方式。

来个例子：

```
SELECT OrderID,CustomerID,OrderDate
FROM Orders
ORDER BY OrderDate
```

简单的查询，并且按照OrderDate排序，不看执行计划，我们就已经推测出这个执行计划中最耗损的就是这个OrderDate了，排序永远是高耗损，这也是为什么各种类型的索引都要提前排序的原因。

然后，我们再来看一下加上这个FAST N Hint提示的执行

```
SELECT OrderID,CustomerID,OrderDate
FROM Orders
ORDER BY OrderDate
OPTION(FAST 1)
```

为了快速获取这一行数据，利用HINT后，改为了索引扫描+书签查找，因为这是获取一条数据的最优的一种方式。

因为数据量的关系，所以我上述演示没能很好的表现出FAST 提示的优越性来，其实在实际生产中，在面临庞大的数据量的时候，一般利用FAST N提示获取出部分数据之后，就不再继续运行了，因为我们关注的就是这一部分数据。

当然，此HINT也有弊端：在快速获取前N行结果之后，可能会延迟整个查询的总体相应时间。也就是说，尽管FAST N HINT可能会使优化器快速产生前N个输出计划。但是它会使优化器产生一个在结束最后一行前花费更多时间，消耗更多CPU，甚至于更多IO。

b、OPTIMIZE FOR Hint提示

此HINT是一种非常有用的提示，也是我们在日常中经常使用的。

这个HINT目标很简单：告诉优化器目标以Hint值进行分配或者执行。此Hint提示是从SQL Server2005版本以上开始支持，能够根据指定的参数值产生一个计划，尤其适用于非对称数据集中，因为这种数据集中数据分布不均匀，不同的参数值可能导致不同的基数评估和不同的查询计划，我们可以从不同的参数中选择一个最优的执行计划，作为后续不同参数的执行计划，避免了SQL Server的重新评估和重编译的耗费的动作。

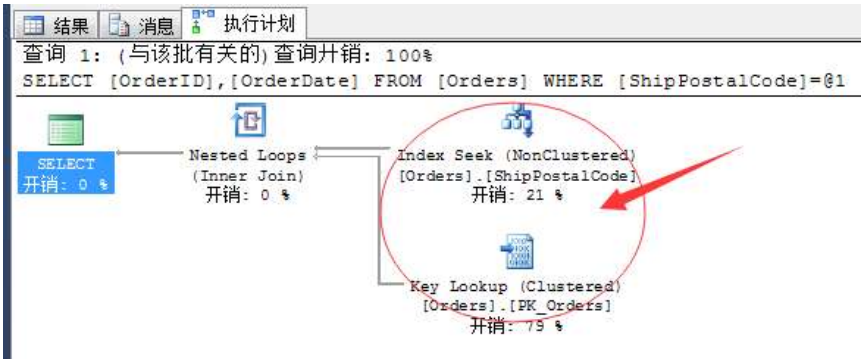
来个例子：

```
SELECT OrderID,OrderDate
FROM Orders
WHERE ShipPostalCode=N'51100'
```

此语句很简单，就是通过查询邮政编码（ShipPostalCode），获取出订单ID和订单日期。

来看这个查询语句，最理想的情况就是直接通过索引查找（index seek）动作获取出数据。其实最好的方式也是通过INCLUDE将两列值包含进去。

我们来看一下实际的执行计划：



SQL Server通过了索引查找+书签查找方式获取，这种方式也凑合吧，其实我们还可以继续优化。

但是，这不是问题重点，问题重点是该段T-SQL一般我们会利用参数进行查询或者包装成存储过程通过传参调用。是吧？？不会你永远只查询一个固定值吧....来看语句

```
DECLARE  @ShipPostalCode NVARCHAR(50)
SET @ShipPostalCode=N'51100'

SELECT OrderID,OrderDate
FROM Orders
WHERE ShipPostalCode=@ShipPostalCode
```

是吧，这种方式才能做到重用嘛，不过包装成一个存储过程或者一个函数等，估计核心代码肯定就这样子了。

来看看生成的执行计划：

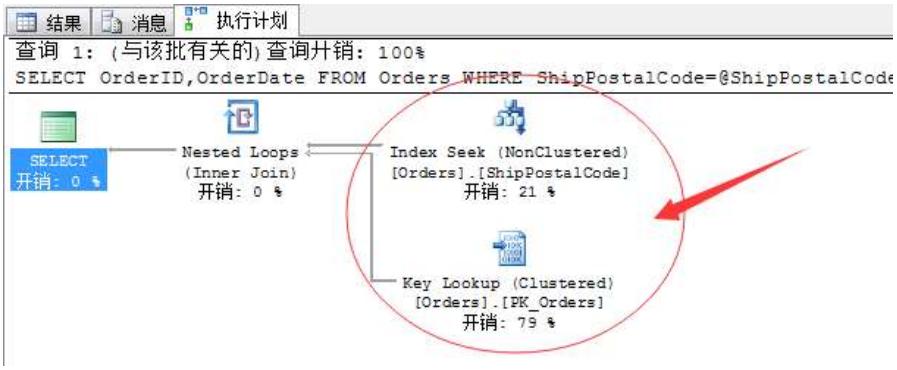
本来很爽的非聚集索引查找（Seek），通过我加了一个参数之后变成了聚集索引扫描（Scan）了，聚集索引扫描的性能跟表扫描基本一样，没有啥质的提高！

如果该表数据量特别大的话，我们为该语句设计的**非聚集索引就失效了**。只能通过依次扫描获取数据了。有意思吗？？？没意思！！！

怎么解决呢？这就是我们此处提到Hint出场的时候了，告诉数据库：丫就按照执行 “51100” 的查询一样去执行我传过来的参数。

```
DECLARE  @ShipPostalCode NVARCHAR(50)
SET @ShipPostalCode=N'51100'

SELECT OrderID,OrderDate
FROM Orders
WHERE ShipPostalCode=@ShipPostalCode
OPTION(OPTIMIZE FOR( @ShipPostalCode=N'51100'))
```



看到了，这里又回归了快速的非聚集索引查找（Seek）状态，并且不受限制于传过来的参数是啥。

这个提示只是告诉SQL Server查询按照这个目标值进行操作，并不会实际影响结果值。

当然上面的问题，如果封装成存储过程的时候，可以采用重编译的方式解决，但是相比利用Hint的方式，重编译带来的消耗远大的多。尤其高并发的环境下重编译所带来CPU消耗是非常高的。

c、物理连接提示（Hint）

关于物理连接我们在前面的文章中已经详细的分析了，在SQL Server中共分为三种物理连接方式：嵌套循环、合并、哈希连接。

详细的内容可以参照我的基础篇中的链接：SQL Server调优系列基础篇（常用运算符总结——三种物理连接方式剖析）

文章中对三种连接的利弊进行了详细的对比，并且对三种连接的使用环境进行了详细的介绍。但是，有时候SQL Server为我们评估的连接并不是最优的，或者说并不是符合我们的要求，这时候，就要利用我们的物理连接提示进行指导。

总共分为三种查询级别的连接Hint，正好对应三种物理连接运算符，依次是:LOOP JOIN、MERGE JOIN 和 HASH JOIN

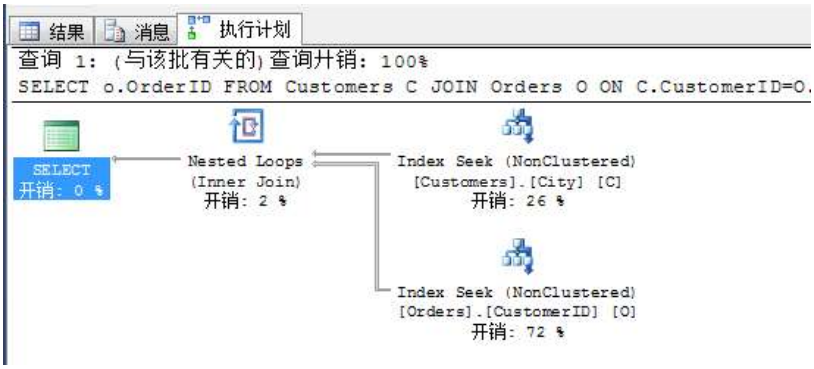
在应用时候，可以指定一个或者多个，如果指定一个，那么查询计划中的全部连接使用指定的连接类型，如果指定两个，SQL Server会在这两个连接类型中选择最好的一个，也就是毙掉了第三个。

应用场景蛮多的，根据三种连接的特性，我们可以有选择的进行提示，比如我们想一个查询不消耗内存，那么就可以指定OPTION(LOOP JOIN，MERGER JOIN)，这样就去掉消耗内存的哈希连接，当然这是减小内存消耗但会增加执行时间。如果采用了合并连接（MERGER JOIN）方式不会消耗内存，但是合并连接需要提前排序（sort），排序会消耗大量的内存。

当然，有时候嵌套循环连接执行的时间不理想，就可以指定为哈希连接（hash join）进行连接。

来看个例子:

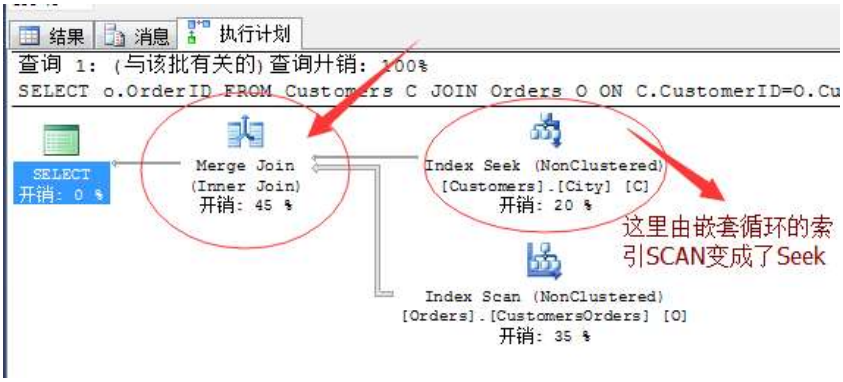
```
SELECT o.OrderID
FROM Customers C JOIN Orders O
ON C.CustomerID=O.CustomerID
WHERE C.City=N'London'
```



上面的查询计划采用了嵌套循环的连接方式，两张表依次进行循环嵌套执行。

如果，经过测试这里发现采用合并连接的方式更好一点，我们可以采用如下Hint进行提示操作

```
SELECT o.OrderID
FROM Customers C JOIN Orders O
ON C.CustomerID=O.CustomerID
WHERE C.City=N'London'
OPTION(MERGE JOIN)
```

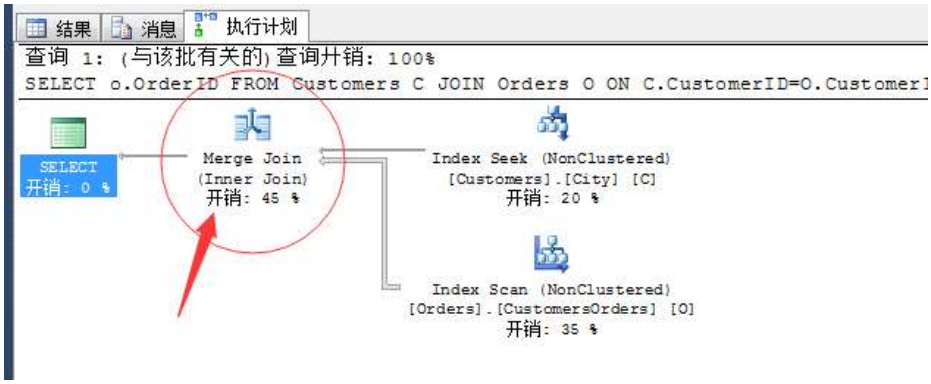


经过调整之后，这时候该语句就利用到了我们设计的非聚集索引，并且由原来的索引SCAN变成了索引Seek运算。

通过如下方式，可以指导SQL Server在哈希连接和合并连接之间做出选择，但是一定要放弃嵌套循环连接。

```
SELECT o.OrderID
FROM Customers C JOIN Orders O
ON C.CustomerID=O.CustomerID
WHERE C.City=N'London'
OPTION(HASH JOIN,MERGE JOIN)
```

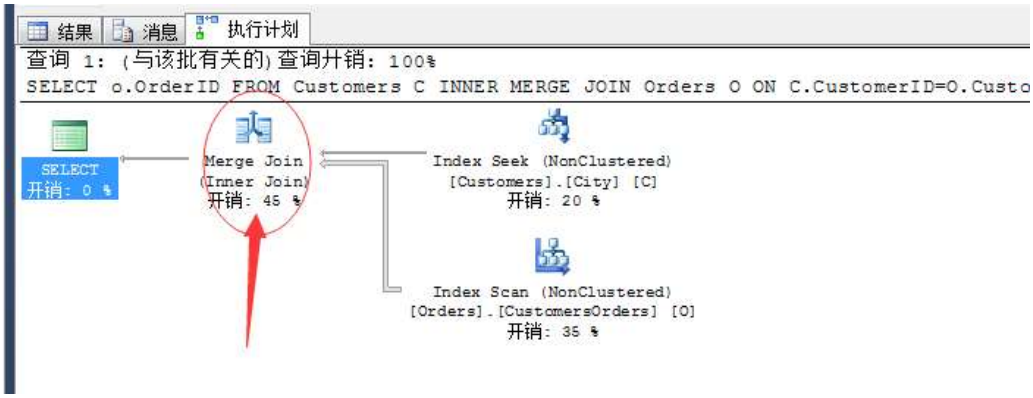
看以看到，经过评估SQL Server还是依然的选择了合并连接



其实，这个很正常，首先数据量不大，其次是在City列上存在非聚集索引，所以要充分利用，并且在两张表的CustomerID是都为索引所覆盖，这就保证了两张表在这列上都是预先排序(sort)了，这完全满足了合并连接的条件。当然，默认选择嵌套循环连接的原因，我估计的原因就一个：两张表数据量不大。

当然，出来上面的HINT方式可以指定连接的物理连接方式，还有另外更为粗暴的一种方式，**强制执行**。如下：

```
SELECT o.OrderID
FROM Customers C INNER MERGE JOIN Orders O
ON C.CustomerID=O.CustomerID
WHERE C.City=N'London'
```



当然，这种方式也手动的达到了指定采用合并连接的方式。

但是，此种方式有严重的弊端：

1、通过采用这种方式貌似暂时解决问题了，但是经过一段时间，此连接方式可能会严重阻碍数据库的优化，而要解决此问题就不得不更改代码。

2、只能粗暴的指定一种物理连接方式，不能顺应SQL Server本身自己的优化策略。

上述的方式是非常不推荐的一种，大部分新手会选择这种方式。

当然，利用Hint的方式是并非一种万全之策，但在当前基本能解决问题，当运行到一段周期之后，如果当前的HINT干预了SQL Server数据库的正常运行，我们也可以采用适当的方式予以停用Hint。使数据库得到完美的平稳的正常运行。后续文章我们依次介绍。

关于Hint这块的使用，内容还是挺多的，其中一部分还包含锁提示等，后续文章我们依次介绍，有兴趣的童鞋提前关注。

其实Hint是平常我们调优时候一种重要的工具。但是，这个工具的正确的使用则要依靠牢靠的基础知识掌握和经验累积。正所谓：厚积薄发！ 不要轻易的看到了使用场景就妄自的进行盲目的使用。如果使用不当，还会扰乱SQL Server数据库本身正常的生态环境，得不偿失，越调越乱。

所以：施主，三思而行呀.....

参考文献

- 微软联机丛书逻辑运算符和物理运算符引用
- 参照书籍《SQL.Server.2005.技术内幕》系列

结语

此篇文章先到此吧，关于SQL Server调优工具Hint的使用还有很多内容，后续依次介绍，有兴趣的童鞋可以提前关注。

有问题可以留言或者私信，随时恭候有兴趣的童鞋加入SQL SERVER的深入研究。共同学习，一起进步。

文章最后给出前面几篇的连接，以下内容基本涵盖我们日常中所写的查询运算的分解以及调优内容项，皆为原创，看来有必要整理一篇目录了.....

SQL Server调优系列基础篇

SQL Server调优系列基础篇（常用运算符总结）

SQL Server调优系列基础篇（联合运算符总结）

SQL Server调优系列基础篇（并行运算总结）

SQL Server调优系列基础篇（并行运算总结篇二）

SQL Server调优系列基础篇（索引运算总结）

SQL Server调优系列基础篇（子查询运算总结）

----------以下进阶篇----------

SQL Server调优系列进阶篇（查询优化器的运行方式）

SQL Server调优系列进阶篇（查询语句运行几个指标值监测）

SQL Server调优系列进阶篇（深入剖析统计信息）

SQL Server调优系列进阶篇（如何索引调优）

SQL Server调优系列进阶篇（如何维护数据库索引）

分类: SQLSERVER

window_net
关注 - 97
粉丝 - 328
+加关注

« 上一篇： JavaWeb+SVN+Maven+Tomcat +jenkins实现自动化部署
» 下一篇： 关于SQLSERVER数据库连接池

posted @ 2019-04-19 16:55 window_net 阅读(72) 评论(0) 编辑 收藏