SQL Server 群集与非群集索引设计指南

SQL Server 的群集索引与非群集索引设计指南

针对 SQL Server 中的索引设计我们将分成群集索引与非群集索引两个部分分别介绍，分析如何通过索引的设计来提升性能和优化查询。并且在本技术专题中，我们还讲解了如何维护 SQL Server 索引以实现优化查询。

SQL Server 的群集索引设计

SQL Server 中群集索引设计对 SQL Server 数据库系统性能和未来的维护十分重要。在本文中你将了解到为什么群集索引应该是静态、随着时间推移而增长、了解它们是如何使用多对多表的。此外，在文中你还会知道在 SQL Server 2005 中分区表概念是怎样影响群集索引的。

- 设计 SQL Server 群集索引以提升性能（一）
- 设计 SQL Server 群集索引以提升性能（二）

SQL Server 的非群集索引设计

非群集索引是书签，它们让 SQL Server 找到我们所查询的数据的访问捷径。非群集索引是很重要的，因为它们允许我们只查询一个特定子集的数据，而不需要扫描整个表。对于这个重要主题的探讨，我们首先从了解基础开始，比如，聚簇索引与非聚簇索引如何互相作用，如何选择域，何时使用复合索引以及统计是如何影响非聚簇索引的。

- 设计 SQL Server 非群集索引优化查询（上）
- 设计 SQL Server 非群集索引优化查询（下）

维护 SQL Server 索引以实现查询优化

维护 SQL Server 索引是一个不寻常的实践。如果查询不使用索引，那么往往会有一个新的非群集索引被创建，它只是包含一个不同的或是相同的字段组合。但现在并没有发布一个关于为什么 SQL Server 会忽略这些索引的详细分析。

- 如何维护 SQL Server 索引以实现查询优化（一）
- 如何维护 SQL Server 索引以实现查询优化（二）

设计 SQL Server 群集索引以提升性能（一）

SQL Server 的集簇索引是数据库整体架构的一个非常重要的方面。它们经常被忽视、误解，或者如果数据库很小，它们会被认为是不重要的。

本文阐述了集簇索引对于整个系统性能以及数据库增大时维护的重要性。我将主要说明 SQL Server 集簇索引是如何存储在硬盘中的，为什么它们应该一直随着时间增加以及为什么静态的集簇索引是最好的。我同时也将探讨多对多表，为什么它们会被使用，以及集簇索引如何能够让这些表效率更高。

最后，很重要的是我们会讨论新的 SQL Server 2005 分割表概念，并探讨分割表是如何影响集簇索引的。这将有助于你以后作为出正确的决定性。

集簇索引默认是与匹配主键相匹配的，而主键是定义在 SQL Server 表上的。然而，你可以在任何字段上创建一个集簇索引，然后在另一个字段或多个字段上定义一个主键。这时，这个主键将会作为一个唯一的非集簇索引被创建。典型地，一个集簇索引会与主键相匹配，但这并不是必须的，所以要仔细考虑。对于各种可能出现的情况，我将讨论集簇索引本身，而不管你是否选择将它与主键相匹配。

集簇索引实际上装载了 SQL Server 的数据记录行，所以你的集簇索引存储的地方就是你的数据存储的地方。集簇索引是按数据范围而组织的。比如，1 到 10 之间的值存储为一个范围，而 90 到 110 是另一个范围。因为集簇索引是按范围存储的，如果你需要在一个范围中搜索一个审计日志，使用基于日期字段的集簇索引效率会更高，其中日期字段会用于返回日期范围。非集簇索引更适用于具体值的搜索，比如“等于 DateValue 的日期”，而不是范围搜索，比如“在 date1 和 date2 之间的日期”。

集簇索引的不断增加值

集簇索引必须是基于值不断增加的字段。在前面的例子中，我使用了审计日志的日期字段，审计日志的日期值是不断增加的，而且旧的日期将不会再插入到数据表中。这就是一个“不断增加”字段。另一个不断增加值的好例子就是标识字段，它也是从创建后就持续恒定增加的。

为什么我在这里花这么多时间讨论集簇索引的不断增加值呢？这是因为集簇索引的最重要的属性就是它们是不断增加的并且本质上是静止的。不断增加之所以重要的原因与我之前提到的范围架构有关。如果值不是不断增加的，SQL Server 就必须在现有记录的范围内分配位置，而不是直接将它们放到索引后面的新的范围中。

如果值不是不断增加的，那么当范围的值用完后再出现一个已经用索引范围的值时，SQL Server 将做一个页拆分插入一个索引。在实现时，SQL Server 会将已填满的页拆分成两个单独的页，这两个页此时会有更多的值空间，但这需要更多的资源去处理。你可以通过设置填充参数为 70%来作好预备工作，这样就可以有 30%的自由空间来为后来的值使用。

这个方法的问题是你必须不断地“再索引”集簇索引使它能维持 30%的自由空间。对集簇索引进行再索引会带来繁重的 I/O 负载，因为它必须移动它的实际数据，并且任何非集簇索引都必须重建，这会增加许多的维护时间。

如果集簇索引是不断增加的，你将不需要重建集簇索引。你可以将集簇索引的填充因数设置为 100%，这样随着时间的推移，你就只需要对于不集中的、非集簇索引进行再索引，这样就可以增加数据库在线时间。

不断增加的值将只会在索引的尾部添加新值，并且只在需要的时候才创建新的索引范围。由于新的值都只会不断地添加到索引的尾部而且填充因数为 100%，所以将不再会有逻辑碎片出现。填充因数越高，每一页所填充的记录行就越多。更高的填充因数使得查询时会需要更少的 I/O、RAM 和 CPU 资源。你查询集簇索引中越少的数据类型，JOIN/查询操作速度会更快。同时，因为每一个非集簇索引都要求包括集簇索引键，所以集簇索引键和非集簇索引也会更小。

集簇索引的最佳数据类型是非常狭窄的。对于数据类型大小，它通常是 smallint、int、bigint 或 datetime。当 datetime 值用作集簇索引时，它们是唯一的字段并且通常是不断增加的日期值，这些值通常是作为范围数据查询的。通常，你应该避免组合（多字段）集簇索引，除了以下情况：多对多数据表和 SQL Server 2005 分割表，这种分割表有分割的字段，它包含了集簇索引而允许索引排列。

（作者：Matthew Schroeder 译者：陈柳/曾少宁 来源：TT 中国）

设计 SQL Server 群集索引以提升性能（二）

多对多表和集簇索引

多对多表有非常快速的 JOIN 并允许快速的从一个记录到另一个记录的重新联合。设想有下面这样的数据结构：

Customer

CustomerID (bigint identity)	Name	Fieldn+

CustomerOrder

CustomerID	OrderID

Orders

OrderID (bigint identity)	Date	Fieldn+

这些结构中的集簇索引是 CustomerID 和 OrderID。组合键是 CustomerID/OrderID。下面这个结构的优点：

JOIN 都是基于集簇索引（比非集簇索引 JOIN 快很多）。

将一个 Order 赋给另一个 Customer 只需要对 CustomerOrder 表作一个 UPDATE 操作，这是改动非常少的，只影响到一个集簇索引。因此，它减少了你在更新一个大表时的数据库锁的时间，如 Orders 表。

使用多对多表就不需要大表中的一些非集簇索引，如 Customer/Orders。因此，它减少大表的维护时间。

这个方法的一个缺点是 CustomerOrder 表的碎片（不是连续的）。然而，这并不是一个大问题，因为这个表是相对较小的，它只有 2 个字段，数据类型也很少，并且只有一个集簇索引。这些本来是在包括 CustomerID 的 Orders 表的非集簇索引的减少，带来的好处是大于额外开销的。

SQL Server 2005 的集簇索引和分割表

SQL Server 2005 中的分割表是一些表面上为一个独立的表，但实际上——在存储子系统中——它们包含能够存储在许多文件组（Filegroup）的多个部分。表的部分是根据一个字段的值来分割成不同文件组的。这种方式的分割表会有几个缺点。这里我会说明几个基本的缺点，希望能让你对相关的原因有一些了解。我建议你使用分割表时先学习它的使用方法。

你可以在这个环境中基于一个字段创建一个集簇索引。但是，如果这个字段不是表用于分割的那个字段，那么集簇索引就被称为非对齐的。如果一个集簇索引是非对齐的，那么任何分区的数据进/出（或合并）都需要你删除集簇索引以及非集簇索引，然后再重建这些索引。这是必要的，因为 SQL Server 并不知道集簇/非集簇索引的哪部分属于哪个表的分区。毫无疑问，这会带来一定的系统停机时间。

分割表上的集簇索引应该总包含常规的集簇字段，它是不断增加和静态的，并且它是用于分割数据库表的。如果集簇索引包含用于分割表的字段，那么 SQL Server 就知道集簇/非集簇索引的哪个部分属于哪个分区。当一个集簇索引包含了用于分割表的字段时，那么这个集簇索引就是“对齐的”。这时表的分区就可以在数据进/出（和合并）而不需要重建集簇/非集簇索引，这样就不会带来额外的系统停机时间。表的 INSERT/UPDATE/DELETE 操作也会更快，因为这些操作只需要关注处于它们自己分区的索引。

总结

SQL Server 集簇索引是数据体系结构的一个重要部分，我希望你已经从本文学习中知道了为什么你需要在一开始就仔细设计好集簇索引。集簇索引应该是窄小的、静态的和不断增加的，这对于将来数据库的健壮性是非常重要的。集簇索引可能帮你实现更快速的 JOIN 和 IUD 操作，并最小化系统的忙时拥塞时间。

最后，我们讨论了 SQL Server 2005 的分割表是如何影响你对集簇索引的使用，集簇索引与分区的“对齐”的意思是什么，以及为什么集簇索引必须按顺序对齐以使分割表正常工作。请继续关注关于将在二月发表的文章《非集簇索引》（第二部分）和三月发表的文章《最优索引维护》（第三部分）。

Matthew Schroeder 是一位高级软件工程师，从事于 SQL Server 数据库系统开发工作，规模从 2GB 到 3+TB、2k 到 40+k 的每秒事务。Matt 目前在游戏供应商 IGT 工作，它为游戏公司提供服务。他也是一位独立顾问，专门为游戏、汽车、电子商务、娱乐、银行和非营利性行业提供 SQL Server、Oracle 和.NET 技术的咨询服务。Matt 精通 OLTP/OLAP DBMS 系统，以及用.NET 实现的高可扩展处理系统。他是一个 Microsoft 认证 MCITP 数据库开发人员，拥有计算机科学的博士学位，以及超过 12 年的 SQL Server/Oracle 工作经验。你可以通过电子邮件与他联系：cyberstrike@aggressivecoding.com。

（作者：Matthew Schroeder 译者：陈柳/曾少宁 来源：TT 中国）

设计 SQL Server 非群集索引优化查询（上）

非聚簇索引是书签，它们让 SQL Server 找到我们所查询的数据的访问捷径。非聚簇索引是很重要的，因为它们允许我们只查询一个特定子集的数据，而不需要扫描整个表。对于这个重要主题的探讨，我们首先从了解基础开始，比如，聚簇索引与非聚簇索引如何互相作用，如何选择域，何时使用复合索引以及统计是如何影响非聚簇索引的。

SQL Server 中的非聚簇索引基础

非聚簇索引由所选择的域和聚簇索引值所组成。如果聚簇索引不是定义为唯一的，那么 SQL Server 将使用一个聚簇索引值及一个唯一值。一定要将聚簇索引定义为唯一的——如果它们实际上就是唯一的——因为它会带来更小的聚簇索引/非聚簇索引。如果我们的唯一聚簇索引由一个 INT 构成，并且我们还在一个年字段（定义为 SAMLLINT）上创建了一个非聚簇索引，这样的这个非聚簇索引相对于表中每行都将包含一个 INT 和 SAMLLINT。索引的大小将随着所选数据类型不同而增长。因此，聚簇索引/非聚簇索引数据类型越小，索引也就会越小，这样就提高了可维护性。

选择非聚簇索引的域

第一条规则是不要在非聚簇索引中包含聚簇索引键的域。由于域已经是聚簇索引的一部分，因此，它总是用来查询的。在非聚簇索引中包含任意聚簇索引键的唯一情况是，当聚簇索引是一个复合索引并且查询是指向复合索引中的第二、第三或更高域时。

假设我们有如下表：

ID (identity, clustered unique)	DateFrom	DateTo	Amt	DateInserted	Description

现在假设我们总是运行如下查询：

Example 1:

```
Select *
```

```
From tbl [t]
where t.datefrom = '12/12/2006' and
t.DateTo = '12/31/2006' and t.DateInserted
= '12/01/2006'
```

在这里，在 DateFrom、DateTo 和 DateInserted 上定义非聚簇索引是合理的，因为它总能提供最佳的唯一结果。

现在假设我们运行如下多个查询：

Example 2:

```
Select *
From tbl [t]
where t.datefrom = '12/12/2006' and
t.DateInserted = '12/01/2006'
Select *
From tbl [t]
where t.datefrom = '12/12/2006'
Select *
From tbl [t]
where t.DateTo = '12/31/2006'
Select *
From tbl [t]
where t.DateInserted = '12/01/2006'
Select *
From tbl [t]
where t.DateTo = '12/31/2006' and
t.DateInserted = '12/01/2006'
Select *
From tbl [t]
where t.id = 5 and t.DateTo = '12/31/2006'
and t.DateInserted = '12/01/2006'
```

在这里，很多人都会尝试创建如下非聚簇索引：

1. DateFrom
2. DateTo
3. DateInserted
4. DateTo and DateInserted
5. DateFrom and DateInserted
6. ID, DateTo and DateInserted

在这里，可能我们都预计索引的大小会明显增大，因为我们将 DateFrom 存储在两个不同的位置，DateTo 存储在三个位置，以及 DateInserted 存储在四个位置。在此之前，

我们已经将聚簇索引键存储在七个位置存储了。这个方法提高了 I/O 的插入、更新和删除操作（也称为 IUD 操作）。记录更新必须首先写入聚簇索引数据行。然后，将对非聚簇索引进行更新以使它们可写。

（作者：Matthew Schroeder 译者：陈柳/曾少宁 来源：TT 中国）

设计 SQL Server 非群集索引优化查询（下）

我们应该经常问自己这些问题：

改进的查询时间是否抵得过 IUD 操作增加的 I/O 和维护开销？

我们在查询上获得的任何性能提高是否抵得过额外的 I/O 和增加的维护时间？

哪种方式在尽可能低开销的情况下能为我们提供最准确的结果？

在这种情况下，最佳解决方案将是下面三个非聚簇索引：

1. DateFrom

2. DateTo

3. DateInserted

在这种情况下，除了在三个非聚簇索引中都存储的主键，其余每个域都只存储一次。因此，索引大小将非常小，并且所需要的 I/O 和维护也将更少。SQL Server 将根据所选择的查询条件，查询每个非聚簇索引，然后将结果集中在一起。虽然它不如例 1 一样高效，但是比起定义五个不同的非聚簇索引，它的效率要高得多。实际查询中，会更多使用例 2 的结构，而不是例 1。

SQL Server 统计

统计告诉 SQL Server 最可能有多少行与一个给定值相匹配。它告诉 SQL Server 匹配值有多“精确”，这个信息将用以确认是否该使用索引。默认情况下，当 SQL Server 发现有接近 20%的记录发生改变时，它将自动更新统计。在 SQL Server 2000 中，它是与 IUD 操作同步进行的，同时在抽取行样本时它会延迟所进行的 IUD 操作。在 SQL Server

2005 中，我们可以设置抽样与 IUD 操作同步，或者与 IUD 操作异步进行。后一种方法更好并且可以减少阻塞，因为锁会更快地被释放。我推荐关闭数据库设置的“自动更新统计（Auto Update Statistics）”。这个设置将在服务器最忙的时候增加服务器的负荷。取代设置 SQL Server 自动保持统计更新的方法是，我们可以创建一个任务来调用命令“更新统计”，并使它在服务器最空闲的时候运行。我们可以根据我们所希望的统计精确性来选择自己的取样频率。

统计只在在任何非聚簇索引的第一个字段上进行。而这在复合非聚簇索引中意味着什么呢？它意味着 SQL Server 将使用第一个域来确认是否应该使用索引。即使复合索引的第二个域匹配 50%的记录行，并且该域仍然需要被用来返回结果（如例 3）。现在，如果非聚簇索引被分成两个非聚簇索引，那么 SQL Server 可能选择使用索引 1 而不是索引 2。这是因为在索引 2 上的统计可能显示它不适合查询（如例 4）。

Example 3

假设我们在 DateFrom 和 Amt 上定义了一个复合非聚簇索引。

统计将仅会在索引中的 DateFrom 域中进行，而 SQL Server 将同时查找（或扫描）DateFrom 和 Amt。由于 SQL Server 必须检查更多的数据，因此，查询将比较缓慢。

Example 4

假设我们有两个非聚簇索引：第一个定义在 DateFrom 上，而第二个定义在 Amt 上。

由于它们是不同的索引，因此这两个域都会被统计。SQL Server 将检查 DateFrom 上的统计并决定使用哪个索引。接着，它还将检查 Amt 字段并可能决定——根据统计——该索引不够精确，应该被忽略。在这种情况下，SQL Server 将仅需要检查 DateFrom 域，而非 DateFrom 和 Amt，这样就可以实现更快的查询。

通过使用 SQL Server 中的非聚簇索引，我们将可以关注于数据子集中的查询。使用本文中描述的规则来确定是应该创建多非聚簇索引还是复合非聚簇索引。同时，记住统计

的作用，以及它们是如何影响非聚簇索引的：在 SQL Server 中，统计影响在多非聚簇索引和合并非聚簇索引之间进行的选择。

（作者：Matthew Schroeder 译者：陈柳/曾少宁 来源：TT 中国）

如何维护 SQL Server 索引以实现查询优化（一）

维护 SQL Server 索引是一个不寻常的实践。如果查询不使用索引，那么往往会有一个新的非聚簇索引被创建，它只是包含一个不同的或是相同的字段组合。但现在并没有发布一个关于为什么 SQL Server 会忽略这些索引的详细分析。

让我们来看看是如何选择聚簇索引和非聚簇索引，以及为何查询优化器可能选择扫描一个表而不是非聚簇索引。在本文中，我们将学习页拆分、碎裂索引、表分区和统计更新是如何影响索引使用的。最后，我们还将学习如何维护 SQL Server 索引以便查询优化器能使用这些索引，以及这些索引能够被快速查询。

索引选择

在索引选择中，聚簇索引是目前最容易理解的。聚簇索引基本上是唯一指向每行记录的键。即使定义一个聚簇索引而未声明它为唯一的，SQL Server 仍然通过添加一个 4 字节的“唯一符”到聚簇索引来使它实际上是唯一的。附加的“唯一符”将增加聚簇索引的宽度，从而导致维护时间增加和查找速度减慢。由于聚簇索引是一个标识每个记录行的键，因此它们被应用在每个查询中。

对于非聚簇索引来说，情况会有一点复杂。由于以下原因，查询会忽略非聚簇索引：

1、高碎片率——当索引有超过 40%的碎片时，优化器可能会忽略该索引，因为查找一个碎裂索引比扫描一个表的开销要高。

2、唯一性——如果优化器确定一个非聚簇索引不是唯一的，那么它会认为扫描表会比使用非聚簇索引要效率高些。比如：如果一个查询引用一个比特字段（这里的 bit=1）而且字段的统计显示 75%的记录行都是 1，那么优化器可能认为表扫描比非聚簇索引扫描更快获得结果。

3、过时的统计——如果字段的统计过期了，那么 SQL Server 会对聚簇索引的好处作出错误的判断。自动更新统计不仅减缓数据修改脚本，而且随着时间的推移，它还会变得与实际的记录统计不同步。有时，最好运行一下 sp_updatestats 或 UPDATE STATISTICS。

4、方法使用——当查询条件带有一个方法时，SQL Server 就无法使用索引。如果我们引用了一个非聚簇索引字段，但又使用一个方法，如 convert(varchar, Col1_Year) = 2004，那么 SQL Server 也无法使用 Col1_Year 上的索引。

5、错误字段——如果一个非聚簇索引是定义在(col1, col2, col3)，而我们的查询中有一个 WHERE 子句，如“where col2 = 'somevalue”，那么就不会使用索引。只有当索引中的第一个字段有在 WHERE 子句中引用时，才会使用该索引。对于这样的 WHERE 子句，如“where col3 = 'someval”，将不会使用索引中，但是，如“where col1 = 'someval”或“where col1='someval and col3 = 'someval2”的 WHERE 子句则会使用索引。

索引不会在查询中使用 col3，因为这个字段在索引定义中不并在 col1 之后。如果想要在类似的这种情况下使用 col3 来查找，那么最佳的方法就是定义两个单独的非聚簇索引，一个在 col1，另一个在 col3。

页拆分

为了存储数据，SQL Server 使用有 8kb 数据块的页。而填充到页中的数据量则被称为填充因数，并且填充因数越高，8kb 页面就越满。越高的填充因数就意味着需要越少的页，从而 IO/CPU/RAM 使用也就越少。因此，我们可能会想将索引设置为 100%填充因数；然而，这里有个问题：一旦页面填满了，而又有在已填充的索引范围内的新值到达时，SQL Server 将通过“页拆分”来为索引提供空间。本质上，SQL Server 是将把填满的页拆分成两个页，这样就会有更多的存储空间了。我们可以通过设置填充因数为 70%左右来解决这个问题。这样就总是有 30%的可用空间预留给将输入的值。这个方法的问题是我们必须不断的“重建”索引，才可以维持 30%的可用空间。

（作者：Matthew Schroeder 译者：陈柳/曾少宁 来源：TT 中国）

如何维护 SQL Server 索引以实现查询优化（二）

聚簇索引维护

静止的或“不断增长的”聚簇索引都必须有 100%的填充因数。因为值是不断增长的，因此只有添加到将最后的索引才不会出现碎片。更详细的探讨，可以阅读这一系列的第一部分《设计 SQL Server 集簇索引以提升性能》。这个索引分类不需要重建，因为它没有碎裂。

非静止或不断增长的聚簇索引都会在数据页中数据记录移动时出现碎片或页拆分。这一类的索引必须重建以将碎片保持在较低的比例，从而使查询能有效地使用索引。

当重建这些聚簇索引时，我们必须决定填充因数是多少。正常情况是 70%到 80%，这可以为新进入页面的记录保留 20%到 30%的可用空间。最理想的环境配置必须根据记录移动的频率，插入数据数量以及发生索引重建的次数决定。目标是设置足够低的填充因数，使得在下次循环维护时，页中数据可以达到 95%而仍不发生拆分，而只有它们达到 100%时才会出现页拆分。

非聚簇索引维护

非聚簇索引总会在页上出现数据移动。这个问题并不像发生在聚簇索引上一样难解决——在聚簇索引上发生的实际数据记录移动，在非聚簇索引中只是记录指针的移动。换言之，相同的规则将应用在非聚簇索引上，直到设置填充因数。再次，目标是设置足够低的填充因数，这样在下次循环维护时，页中数据就只达到 95%。

非聚簇索引总是会出现碎片，为了避免这个情况，我们必须对它进行不断地监控和维护。

拆分表索引要考虑的方面

拆分表可以根据字段中的数据将数据分割到不同的分区中。大多数表是根据日期范围拆分的。比如，将订单表按年分区。假设聚簇索引是对齐的（如本系列的第 1 部分），那么我们可以重新索引 2000 年 100%填充因数的非聚簇索引，因为从技术上来说，这个数据

是不能移动的。在这种情况下，2008 年分区的非聚簇索引上的填充因数可能是 70%，这样它就是允许移动数据的，但是 2000 年的分区将不能发生任何移动，而且它可以用 100%填充因数进行重新索引，因此可以优化索引查找。

相同的概念可以应用到非静止或不断增长的聚簇索引上。需要移动数据的聚簇索引在 2008 年分区上设置 70%填充因数而在 2000 年分区设置 100%填充因数。

SQL Server 统计

统计维护在字段和索引上，并且它们会用于帮助 SQL Server 确定某些值可能的“唯一”性——比如，如果统计显示某个值可以匹配接近 80%的记录行，那么 SQL Server 将通过扫描表来代替索引。如果统计显示某一个值可能匹配接近 10%的行，那么查询优化器将选择影响数据库最小的查询。

SQL Server 统计可以自动地维护或手动运行。由于重新索引会改变统计结果，因此我推荐，在重新索引之后，我们手动运行 sp_updatestats 或 T-SQL UPDATE STATISTICS 命令。统计只在任何组合合索引的第一字段上维护，因此无法确定索引的其它字段的“唯一性”。

总结

索引维护对于保证查询能够总是受益于索引使用并减少 IO/RAM/CPU 是至关重要的，同时这也可以减少阻塞。

在打开选项“show execution plan”时运行查询。如果查询没有使用我们的索引，那么要进行以下的检查：

1、运行 dbcc showcontig('tablename')来检查表是否有碎片。

2、检查“where clause”来查看是否它引用了索引的第一个字段。

3、保证“where clause”的查询条件中没有针对索引的第一个字段的方法。

4、只当统计过期时才更新统计。如果表有碎片，那么在重新索引之后更新统计。

5、确保所使用的查询条件是足够唯一的，这样 SQL Server 更好地查找数据。

（作者：Matthew Schroeder 译者：陈柳/曾少宁 来源：TT 中国）