SQL Server 索引设计与调优

SQL Server 索引技巧设计与调优

如果你想极大提高 SQL Server 性能，本篇指南中提到的索引将是您最佳选择之一。在本文指南中你将了解如何设计最佳 SQL Server 索引、如何调整 SQL Server 索引等一系列内容，让你现存的 SQL Server 索引能够发挥最佳效能。

SQL Server索引设计

SQL Server 集簇索引的设计

SQL Server 中集群索引设计对 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集群索引应该是静态、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集群索引的。

- 设计 SQL Server 集簇索引以提升性能（一）
- 设计 SQL Server 集簇索引以提升性能（二）

如何创建 SQL Server 索引

索引的作用应该是确保主要性能。本节你将会学到如何清除那些没有价值的索引并识别推荐索引保证你的 SQL Server 索引能发挥它的最大效能。

- SQL Server 索引创建技巧（上）
- SQL Server 索引创建技巧（下）

如何优化索引

索引 SQL Server 数据库既是艺术也是技术。我们必须根据设计和编码来选择正确的索引。但是，当测试索引设计时，我们可能发现它对系统性能的提高并没有达到我们的要求。我们必须通过学习索引字段、聚簇索引、主键以及索引配置来创建最佳设计的 SQL Server 索引。文中介绍了一些设计索引时的常见问题。

- 专家详解 SQL Server 2000 创建和优化索引

索引的能与不能

在这一系列的问题和答案中，我们将了解索引列和数据库的正确含义，避免出现页面拆分的情况并了解 SQL Server 2000 的能与不能。

- SQL Server 2000 索引的能与不能（DO 和 DON’T）

改进性能的分区索引

SQL Server 2005 索引分区允许你将特定索引符合分散到多个文件。本文中还介绍了如何用分区数据创建索引的方法。

- 改进 SQL Server 2005 性能的分区索引（上）
- 改进 SQL Server 2005 性能的分区索引（下）

聚簇索引和非聚簇索引的区别

什么时候使用聚簇索引或非聚簇索引呢？回答这个问题有点难度，坦白地说，我即将给出的答案是一个流传已久的标准数据库管理员的回答：“具体问题具体分析”。有大量因素影响何时以及何地进行索引创建。幸好只有两个选择，但分析这两个选择的优缺点都相当复杂。

- SQL Server 中的聚簇索引和非聚簇索引（一）
- SQL Server 中的聚簇索引和非聚簇索引（二）

SQL Server 非聚簇索引设计

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。

- 设计查询优化的 SQL Server 非聚簇索引（上）
- 设计查询优化的 SQL Server 非聚簇索引（下）

如何添加非聚簇索引

增加非聚簇索引到我们经常要查询的 SQL Server 列表是很有可能的。本文就介绍了这方面的知识。

- 添加非聚簇索引到 SQL Server 字段

创建索引的更好办法

为文本数据（varchar、nvarchar、char 等）创建索引是一种很好的实现更快数据查询的方法。然而，这些索引会给存储索引的磁盘以及服务器内存带来压力。这是因为索引上存有大量的数据。

- 为文本数据创建索引的更好方法

SQL Server 索引调优

SQL Server Index Tuning Wizard 的使用技巧

回答如何全面提高 SQL Server 2000 性能、如何使用 SQL Server Index Tuning Wizard 技巧更好地理解基本数据库索引。

- SQL Server Index Tuning Wizard 的使用技巧

处理 SQL Server 2000 索引碎片技巧

当遇到数据库的性能问题时，其中一个最大的性能提升方法可以通过优化索引来实现。索引可以改善数据访问，这样我们就不需要扫描整个表，因为这会消耗大量的 CPU、IO 和内存资源。随着时间的推移，索引可能会产生碎片，从而导致 SQL Server 性能下降、事务时间处理时间变长、阻塞和低吞吐量。

- 处理 SQL Server 2000 索引碎片技巧（一）
- 处理 SQL Server 2000 索引碎片技巧（二）
- 处理 SQL Server 2000 索引碎片技巧（三）

最佳 SQL Server 索引策略

恰当的索引能创建完全不同的性能。对于大多数的数据类型，SQL Server 只支持两种索引类型——聚簇索引和非聚簇索引。同时，SQL Server 也支持全文索引和 XML 索引，但是它们只与特定数据类型相关。

- 最佳 SQL Server 索引策略

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。

- 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）
- 如何维护 SQL Server 索引以实现查询优化（三）

SQL Server 中用于查找索引碎片的存储过程

由于数据修改，SQL Server 表和索引会逐渐出现数据碎片。在大型的 I/O 操作中，在 SQL Server 中用到这些碎片索引和表，可能对应用性能产生不利影响。

- SQL Server 中用于查找索引碎片的存储过程（上）
- SQL Server 中用于查找索引碎片的存储过程（下）

设计 SQL Server 集簇索引以提升性能（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 集簇索引以提升性能（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）

SQL Server 索引创建技巧（上）

在开始一个应用时，我们必须根据一套合理的规则来识别多个索引。随着应用的增长和修改，我们必须检查索引以保证不会忽略任何好的候选索引。这些都必须依据应用的使用方式而不是按照理论来处理。同样，我们必须保证错误的、重复的或者无用的索引已经被删除。这样可以确保我们的 SQL Server 不用管理不需要的索引。在本文中，我们将学习推荐索引、创造索引以及进行索引验证。

通用索引字段

对于许多 DBA 和开发人员来说，索引的传统推荐字段是一个常见问题。常见的推荐是：

- Primary Key's 主键
- Foreign Key's 外键
- 支持 SELECT、INSERT、UPDATE 和 DELETE 命令的字段

1、INNER JOIN

2、RIGHT | LEFT OUTER JOIN

3、WHERE

4、ORDER BY

5、GROUP BY

6、HAVING

索引的其它考虑因素是：

- 数据量 ——与遍历索引相比，记录行很小的表格可以一样快速地访问数据并更节省资源。
- 数据选择比——当数据的选择比低时，如在字段中存储相同的数据，索引值可以是最小的。
- SQL Server 2000 索引 Q&A：创建和优化索引
- SQL Server 2000 索引的能与不能（DO 和 DON’T）

如何确定应用是否需要索引？

一旦确认了通用的推荐索引，我们就可以决定应用的最佳索引。下面的列表是可以帮助完成这个确认过程的 SQL Server 工具。

处理类型——通过事务处理系统，索引必须最小化以便尽可能快地插入，这与以新的不同方式检索数据的报告系统不同，它是可以用大规模的索引提升查询性能。

ID	Tool	Purpose	SQL Version	Additional Resources
1	Profiler	Identify poorly performing queries as a means to identify potential indexes 识别性能差的查询以及潜在的索引	SQL Server 2000 SQL Server 2005	Tracking query execution with SQL Server 2005 Profiler SQL Profiler: Features, functions and setup in SQL Server 2005
2	Database Engine	Analyze data	SQL	Database Engine

	Tuning Advisor	from Profiler or in real time to offer beneficial indexes or partitions based 分析 Profiler 数据或实时提供有用的索引或存储分区	Server 2005	Tuning Adviser: How to tune your new SQL Server 2005
3	Index Tuning Wizard	Analyze data from Profiler or in real time to offer beneficial indexes 分析 Profiler 得到的数据或实时提供有用的索引	SQL Server 2000	Tricks for using the Index Tuning Wizard
4	sys.dm_db_missing_index_columns (Dynamic Management View)	Identifies columns that are missing indexes 识别缺少索引的字段	SQL Server 2005	sys.dm_db_missing_index_columns

(作者：Jeremy Kadlec 译者：曾少宁／陈柳 来源：TT 中国)

SQL Server 索引创建技巧（下）

如何建立索引?

一旦确认了通用的推荐索引，我们就可以决定应用的最佳索引。下面的列表是可以帮助完成这个确认过程的 SQL Server 工具。

ID	Command\Directions	Example
1	CREATE INDEX T-SQL command (SQL Server 2000)	Create a non-clustered index CREATE INDEX [IDX_Job_Desc] ON [dbo].[jobs] ([job_desc]) GO
2	CREATE INDEX T-SQL command (SQL Server 2005)	

3	Enterprise Manager (SQL Server 2000)	
4	Management Studio (SQL Server 2005)	

如何验证索引是否有益？

建立索引是一项非常有意义的事情，但是如果我们没有确定索引是否有益，那么就不值得花时间去建立和维护索引。如果建立一个索引，虽然它一方面有利于应用，但另一方面却对应用有害，这就是一个我们不愿意看到的常见情形。一个避免这种问题的方法是建立一个基线，当数据库修改时，我们可以运行数据收集脚本以及将结果与基线作比较。更多关于基线的信息，可以阅读 Developing a performance baseline。

在我们的环境中添加一些便捷的索引将显著地提高性能。其中的挑战是保证包含所有常见推荐索引，然后，我们再进一步决定选择哪些索引。如果你已经花费时间来分析和建设需要的索引了，那么一定要确保使用可量化的测量方法来验证性能提高。

关于作者：Jeremy Kadlec 是 Edgewood Solutions 主要的数据库工程师。该公司是一个技术服务公司，专门提供 Microsoft SQL Server 相关的专业服务和产品解决方案。他在地方性 SQL Server Users Groups 和全国性 SQL PASS 上发表了大量的文章并多次发表演讲。他是 Rational Guide to IT Project Management 的作者。同时，Jeremy 还是 SearchSQLServer.com Performance Tuning 专家。

(作者：Jeremy Kadlec 译者：曾少宁／陈柳 来源：TT 中国)

专家详解 SQL Server 2000 创建和优化索引

索引 SQL Server 数据库既是艺术也是技术。我们必须根据设计和编码来选择正确的索引。但是，当测试索引设计时，我们可能发现它对系统性能的提高并没有达到我们的要求。我们必须通过学习索引字段、聚簇索引、主键以及索引配置来创建最佳设计的 SQL Server 索引。

让我们先来看一些设计索引时的常见问题：

用户如何访问数据？

- 指定一个值访问一行记录
- 指定一个值访问多行记录
- 指定多个值访问多行记录
- 访问不同范围的记录，如指定一段时间

哪些是常见索引字段？

- 主键
- 外键
- 用于 JOIN、WHERE、ORDER BY、GROUP BY、HAVING 和其它子句中的字段

应该选择哪种索引？

- 聚簇索引——这是一种当数据在表格中物理排序时使用的索引。表格不可能有超过一种排序方式。一个很好的候选簇索引是主键、唯一识别每一行的字段，或者支持范围的字段，如一个日期。

- 非聚簇索引——它是用于基于聚簇索引的有序数据上使用，或者在没有簇索引时单独基于数据使用。一个好的候选非聚簇索引可以是外键，或者在 JOIN、WHERE、ORDER BY、GROUP BY、HAVING 和其它子句中使用的字段。
- 不使用索引——在只有固定少数行的表中不使用索引，如查找表，在 SQL Server 中直接查询会比使用索引更快。

我总是需要使用聚簇索引吗？

- 不，我们并不“总”得使用它。使用的比例是 80:20：虽然不是所有，但大多数环境下我们都必须使用聚簇索引来对数据进行物理排序。典型的，当表中有大量的事务，并且当使用 SQL Server 来维护聚簇索引消耗过多资源时，我们就不使用它们。虽然如此，我也必须提示一点，我个人见过许多按照惯例不使用聚簇索引的大型表会因为聚簇索引而提升性能。其中实质好处是数据访问时间改进和最小化 I/O 资源。这是一个伟大的胜利！

我们必须总使用主键吗？

- 不，我们并不“总”得使用它。根据我的个人经验，90%以上的情况下我们需要一个主键来维持引用完整性，或支持第三方的数据比较工具。

索引中应用有多少个字段呢？

- 如果不确定确切的索引数，最佳的方法可能就是每个索引用一个字段。
- 如果索引中的字段匹配多个检索或频繁进行的关键检索中的字段顺序，那么每个索引包含多个字段是有用的。唯一的说明是索引的统计信息只基于第一个字段，而不是整组索引。

索引还有其它什么配置？

- 索引顺序——我们可以创建递增或递减顺序的索引。

- 填充指数——为每一个索引指定填充因数，以确定索引在创建或重建时在每个索引页预留多少空余空间。
- 统计信息——保证手动创建索引统计信息，或者允许 SQL Server 根据数据库大小自动创建和更新它们。

(作者：Jeremy Kadlec 译者：曾少宁/陈柳 来源：TT 中国)

SQL Server 2000 索引的能与不能（DO 和 DON’T）

在这三部分特性的第二部分，Edgewood Solutions 公司 Jeremy Kadlec 概述了 SQL Server 2000 中最好和最差的索引实践。

索引能够做什么？

Kadlec:

- 没有系统是一成不变的，因此我们必须不断地修改索引以支持使用变化。
- 高选择性的（不同的）数据的索引字段。如果数据选择性低下，那么 SQL Server 将无法在索引中生成大量值。
- 如果是非常大型的数据库（VLDB），那么可以将数据和聚簇索引与非聚簇索引分开，并存储在其它物理硬盘上的独立文件组中。
- 从整体角度上处理索引以便保证索引对应用一部分有益时不会对另一部分有害。
- 平衡每个表的索引数目以便减少 SQL Server 在执行事务处理时的工作量。这样将减少整体存储空间支持，同时在时间和 I/O 方面仍然实现高效的处理。
- 同时，平衡还有助于保证新索引不会在对应用的一部分有益的同时损害另一部分的性能。要对索引进行全面的测试，并重新检查一个位置的修改不会损害应用另一部分。
- 在测试阶段中检查查询计划，以保证索引可以改进查询时间以及预期的资源使用率。
- 为每个索引选择恰当的填充因数。如果表内数据只会有很少的修改，那么索引就要配置一个高填充因数，如，接近于 100%，这样将节省存储空间。如果表内数据会有许多的修改，那么我们可以选择一个较低的填充因数，如 65%到 85%，这样当数据添加到页时，索引重新建立的页划分将是最小的。

- 定期使用 DBCC SHOWCONTIG 命令来检查表、索引或数据的碎片。根据碎片程度，定期重建索引。
- 如果在短时期内，索引出现大量的碎片，那么我们需要检查数据是如何插入、更新以及删除的，以便确认我们是否需要一个较低的填充因数。同时，确定是否可以修改代码来减少碎片的数量。
- 在索引创建、删除和/或重建时捕捉性能基线，以确定每个操作对性能的影响。
- 在变更管理过程之后引进新索引测试和部署。保证用文档记录所有修改。
- 按周、月或季度执行数据库维护。只有不断地维护，SQL Server 才可以完美地运行。

索引不能做什么？

Kadlec:

- 避免表出现热点。当所有事务都访问表的相同位置（如，表格末端）以及引发线路争夺问题时，就会出现。
- 不要在查询中使用索引提示来替代 SQL Server 优化器索引选择，除非所有的选项都用完了。
- 避免页切分。它是用在现有数据页空间无法再存储新记录行的时候。当出现这种情况时，SQL Server 将把一半的数据转移到新的页面。最终结果是，原先的页面存储了一半数据而新的页面存储了另一半数据。这是一个非常耗费资源的操作，因为 SQL Server 必须完成记录行的存储及后续处理。
- 不要建立或维护不必要的重复索引。

（作者：Jeremy Kadlec 译者：曾少宁／陈柳 来源：TT 中国）

改进 SQL Server 2005 性能的分区索引（上）

索引分区是 SQL Server 2005 所引进的多个新特性的其中一个。它是将特定索引负荷分散到多个文件的一种方式，同时，它还可以提高并行性和索引性能。

SQL Server 2000 分区视图和 SQL Srver 2005 索引分区

较早版本的 SQL Server 使用分割视图来实现索引分区。表的查询和修改也可以通过视图来进行某些方面的限制，这样，就只有需要的物理文件会被查询或修改。比如（这只是一个随意的例子，但它可以满足我们的需要），如果我们的客户数据库中有 26 个表，对应字母表的 26 个字母，那么我们可以使用分区视图来汇总所有表的结果，同时使用 WITH CHECK 来约束只更新所需要的表。我们可以查询所有“B”开头的客户，并且分区视图也会知道只需要查“B”表。

分区视图的缺点是我们必须手动进行创建和管理。在 SQL Server 2005 对分区、表和数据之间有更大的抽象，因此它们可以被单独操作。

同时，索引分区的新特性是，特定表的索引被分区或限制在多文档和文件组中。在此，我收集了一些关于如何建立和使用索引分区的基本指导原则；详细的信息可以阅读 SQL Server 2005 Books Online。

创建分区数据索引

有两种方式可以创建分区数据索引：根据数据分区方式来分区索引或单独分区索引。选择何种分区方式，取决于我们访问和更新数据的方式。

首先，索引是“按分区排列的”。默认情况下，在一个分区表上的任意新建立的索引都与表有一样的分区。在下面的情况中，这将会是最佳的方式：

- 表将被插入的大量数据；
- 预计会需要添加的分区；

索引分区对表的数据建立非常重要。

例如，可以将一年的数据根据月数分区，其中以日期为主键。由于 SQL Server 可以快速确定指定关键词在索引中的位置，因此这种方式的索引分区可以加快数据查找速度。

有些时候我们并不想使用分区对齐的方式。特别是在唯一的索引键不包含表分区字段时。比如，如果我们使用的是上面 A 到 Z 分区模式，而表的索引键却是 GUID 或自动递增数而非客户名称，那么我们可以将索引保存在索引本身的分区中，这样它就不是与表对齐。任何情况下，如果我们显示地将索引放到一个不同的文件组，那么分区将与表不匹配。

如果我们分区有唯一索引的数据，那么用于分区的字段必须与唯一索引键相同。例如，如果我们的唯一分区索引是一个客户 ID 号，那么它也是用于分区索引键的相同字段。

（作者：Serdar Yegulalp 译者：曾少宁／陈柳 来源：TT 中国）

改进 SQL Server 2005 性能的分区索引（下）

理解分区方法和分区模式

分区由两部分组成：分区方法和分区模式。第一个表示的是数据本身是如何分区在不同的分区的。比如，以 A 到 Z 为例，数据是将字母表的每个字母作为 26 个单独的分区功能来分区的。

模式表示分区方法中的各个分区是如何映射到文件组的。如果我们的 A 到 Z 表中的“A”数据存储在文件组的一个物理文件中，而“A”索引存储在相同文件组的另一个物理文件中，那么按分区排列的索引将有助于加速和并行数据和索引的访问。这样，多 CPU 可以同时在不同的分区或物理文件上操作。（如果需要的话，我们可以通过将索引和数据放到单独的物理设备上来实现更进一步的数据并行访问。）

TEMPDB 空间

建立按分区排列索引需要消耗内存并占用 TEMPDB 空间。许多数据库管理员在安装 SQL Server 时，都保持 TEMPDB 空间分配的默认值，而没有设置它的大小，而用于自动扩展 TEMPDB 的很消耗时间和资源，从而降低性能。同时，不同分区的索引是根据不同内存分配方法建立的：按分区排列的索引每次都与一个排序表一起建立，而非排列索引则同时与它所有的排序表同时建立。

Microsoft 在 Books Online 中规定每个分区区的排序表最小是 40 页，并且每页 8KB，因此一个有 26 个分区的非排列分区索引（以上面的 A 到 Z 为例）将需要 1，040 页——接近 4.25 MB 内存。一个非排列索引则只需要 163,840 字节。对于大多数健壮的 SQL Server 设置这都不是问题，但是当我们处理超大型分区模式并且同时运行多个分区模式时，我们还是要留心的。

(作者：Serdar Yegulalp 译者：曾少宁/陈柳 来源：TT 中国)

SQL Server 中的聚簇索引和非聚簇索引（一）

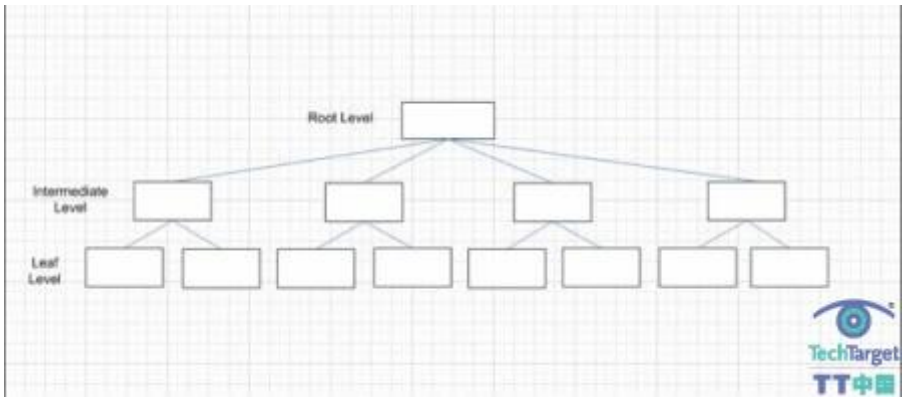
概述

什么时候使用聚簇索引或非聚簇索引呢？回答这个问题有点难度，坦白地说，我即将给出的答案是一个流传已久的标准数据库管理员的回答：“具体问题具体分析”。有大量因素影响何时以及何地进行索引创建。幸好只有两个选择，但分析这两个选择的优缺点都相当复杂。

基础

在此，我并不打算对最低层的索引功能进行详细的探讨。关于这个主题的书已经有一整套书籍了，同时有一群人都专注于索引创建。现在只需要知道：有一个聚簇索引总比没有索引强。聚簇索引和非-聚簇索引之间最大的不同是，当我们使用一个聚簇索引时，包含聚簇索引的表格部分的数据页与包含非-聚簇索引的数据页之间的数据传输是不同。

关于 SQL Server 2005，我们大概都知道“堆或 B-Tree”。在此需要指出的是，一个没有聚簇索引的表一般指的是一个堆。而“B-Tree”或“平衡树”是聚簇索引所采用的通用结构。它们有点类似于电话簿。我们知道 SQL Server 2005 有 8K 的数据页。同时还有 8 个不同类型的数据页。索引数据页面有指向较小数据子集的指针，同时这些数据子集也有指向更小数据子集的指针，以此类推。比如，当我们打开电话簿时，在反面的左上或右角我们可以看到什么呢？我们可以看到页面的范围。这就是 B-Tree 的功能了。



那么此处它们有何不同呢？当应用一个聚簇索引时，在“叶级”的数据包含了实际的数据页，这里存储了我们所要查询的数据。在非聚簇索引中，在叶级的数据页仅包含指向所查询的实际数据的数据页的指针。因此，一个聚簇索引的叶级数据页只有一种排序方式，并且是“有序的”。

比如，如果一个字段加了一个 IDENTITY 约束并创建了一个聚簇索引，那么构成 IDENTITY 约束的数字总是有序的。但是，它并不总是连续的，因为我们可能会 DELETE 记录行。但是它们总是有序的。这样就可以实现非常快速的查询，特别是在用于诸如订单或发票 ID 的应用中。

注意事项

关于使用聚簇索引或非聚簇索引的权衡是所谓“给和取”概念。我所知道的最重要的一点是，由于聚簇索引将它们的所有数据都保留在 B-Tree 的叶级中，因此，任何数据修改都要求重新排列数据页。这就意味着，如果我们添加一个聚簇索引到一个有大量插入、更新或删除操作的表中，那么，比起使用一个非聚簇索引，我们将可能需要更频繁地重建索引或清除索引碎片。这是所有数据页面移动所造成的。再次，这样做的好处是由于数据的有序排列，我们可以得到更快的数据读取速度。其中一个最大的不同是，每个表只能有一个聚簇索引。但是，每个表可以应用 249 个非聚簇索引。

记住，虽然在每个表中只能有一个聚簇索引，但是它并不一定是由一个字段组成的。更常用的做法是将它应用到多个字段中来创建成覆盖索引。考虑表的搜索条件：正在查询

什么？如果在同一时间查询多个字段，那么覆盖索引可能就是我们要的结果。另外一个折衷的办法是先评估正在查询的内容，然后将一个聚簇索引应用到 WHERE 子句用到最多的一个字段中。然后在 SELECT 语句中的其余字段中应用一个覆盖非聚簇索引。

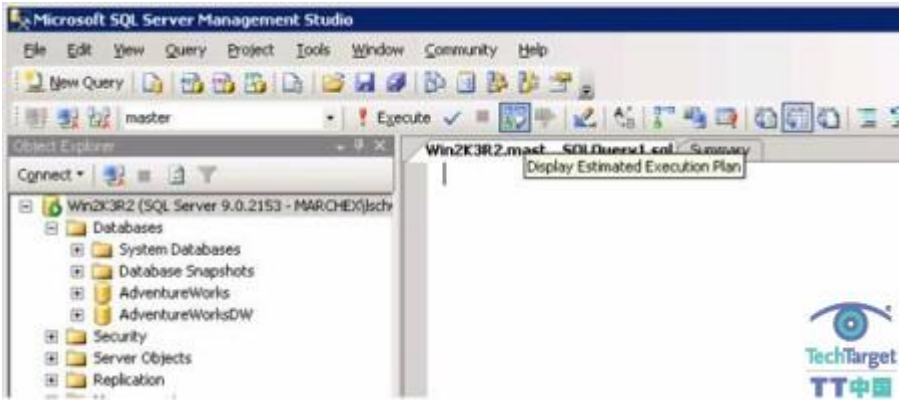
最后，我们还可以使用索引视图。实际上，这是 Microsoft 实现的所谓“物化视图”。注意——当我们将一个索引应用到视图时，我们此时是在创建一个新的数据库对象，然而，来自非聚簇索引视图的结果集只在会话打开的时间内存在，并且它是完全虚拟的。而索引视图则不是这样的。

（作者：Laurence Schwarz 译者：曾少宁 / 陈柳 来源：TT 中国）

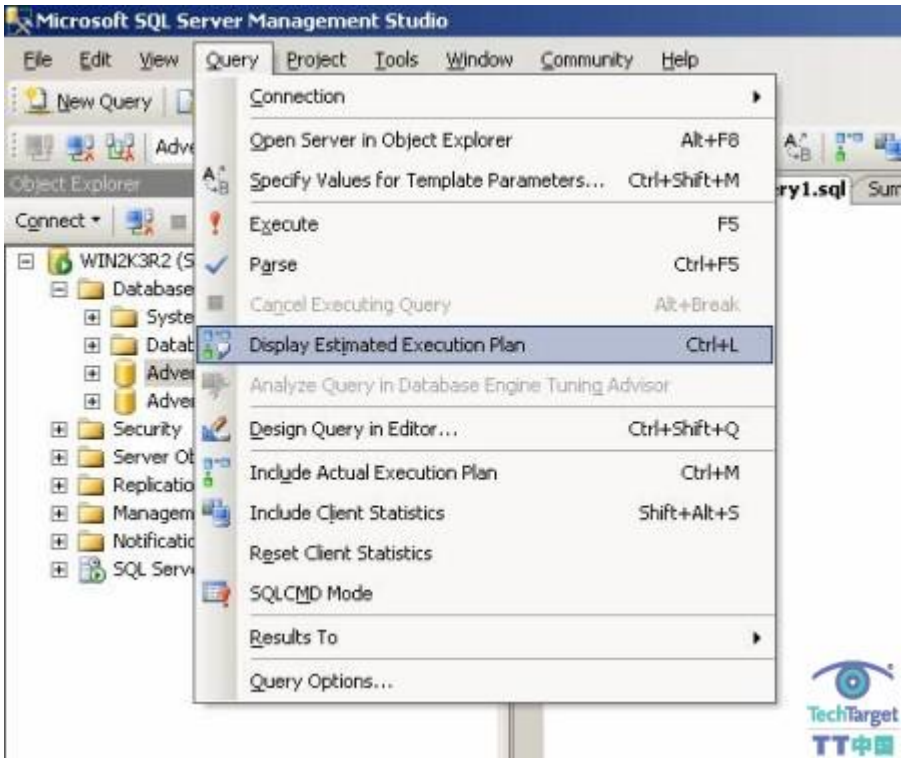
SQL Server 中的聚簇索引和非聚簇索引（二）

最佳方案

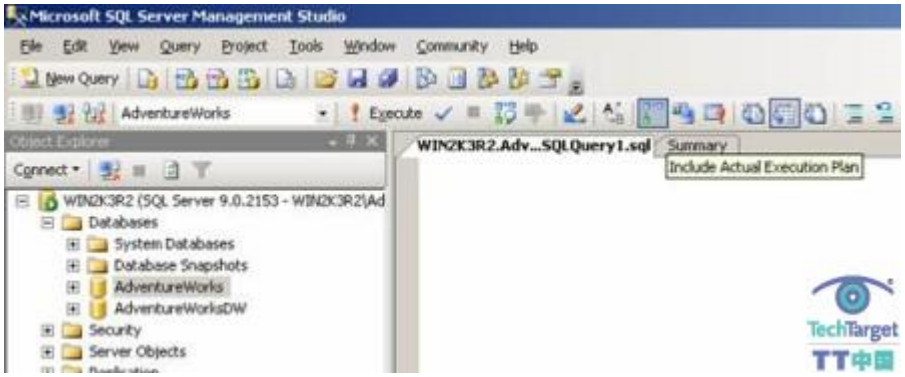
测试，测试，反复地测试。在 SQL Server 2005 中有大量的工具可以用来帮助我们做出最佳选择。其中一个就是“Display Estimated Execution Plan”。通过按 CTRL+L 键，我们可以在 SQL Server Management Studio 上方的 Query Menu 中找到 Display Estimated Execution Plan，



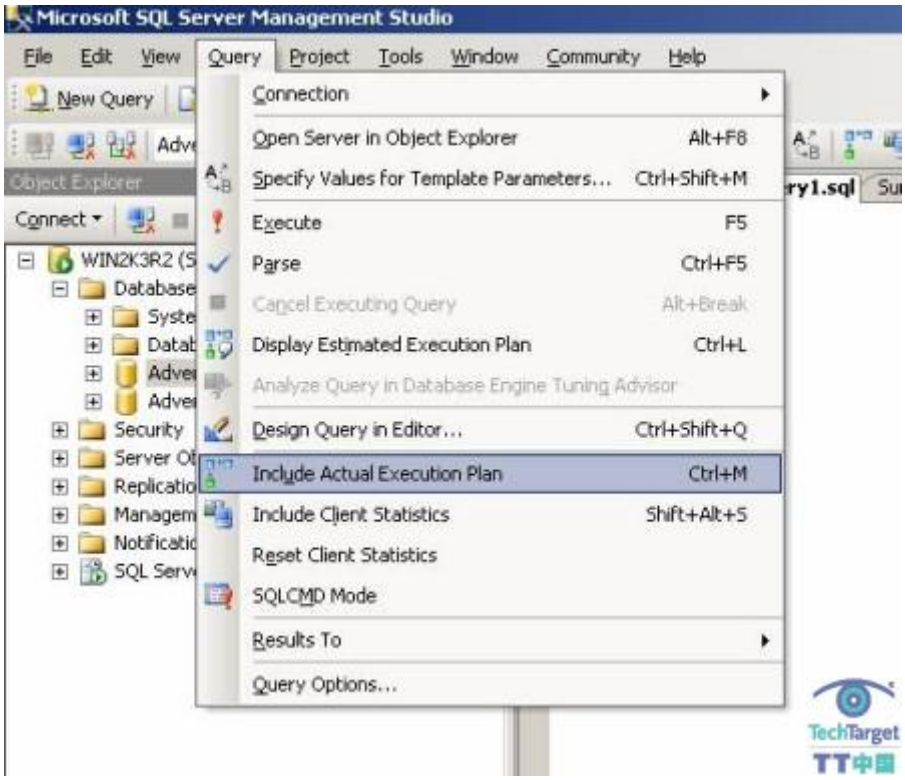
或者在 SSMS 的查询方框中的工具栏上。



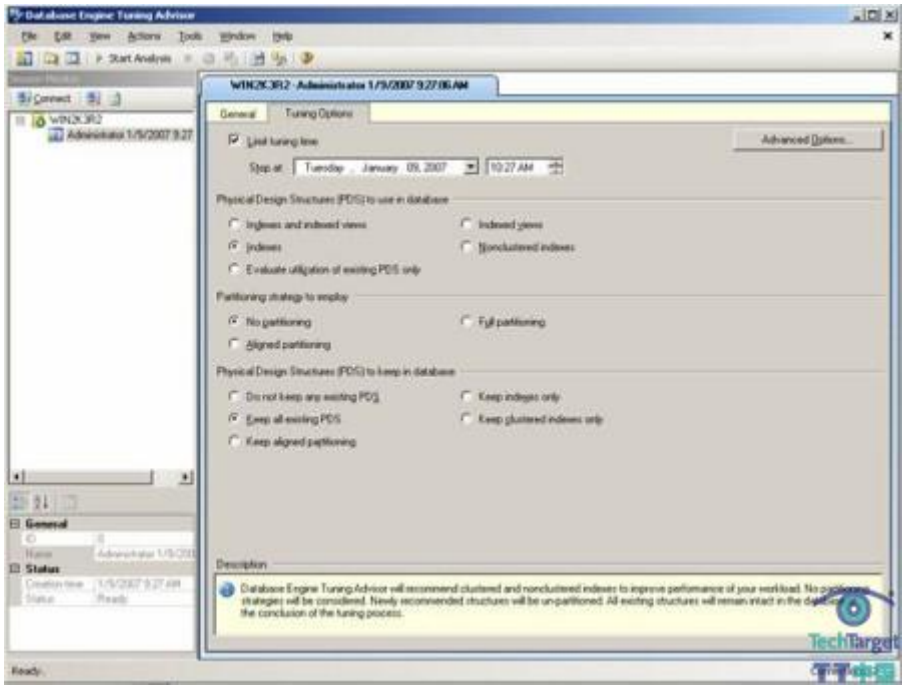
通过执行这个工具，SQL Server 告诉我们它预计可能完成的任务。不用担心，它的结果通常都是正确的。在应用一个索引之前先查看一个查询估计的执行计划，然后再应用这个索引来看看我们所选择的索引是否提高了性能。



我们同时也可以执行 Actual Execution Plan。我们也可以在查询菜单中找到它，通过按 CTRL+M 键，



或者点击工具栏的按键打开。



另外一个很好的工具是 Tools 菜单中的 Database Engine Tuning Advisor。这个工具不仅有助于我们选择正确的索引实现，同时，还有助于我们决定执行 SQL Server 2005 新特性中的其中一个最佳查找：分区。但是，记住，分区只可以用在 SQL Server 2005 Enterprise Edition 中的生产系统中，并且与 Standard Edition 相比，它有一些附加费用。

提高索引性能的最后一个可以考虑的最佳方案是：是否将一个索引结构移到不同的存储阵列中。如果我们目前的所有数据库对象都在同一个阵列或者即使是同一个阵列的多文件组上，那么将索引结构转移到 RAID 1 阵列会带来性能的大幅提升。我个人曾经成功地使用这个技术。

总结

总之，在这方面并没有硬性的规定。如果我们在刚刚应用了一个索引的表上显示实际执行计划，并且它显示表扫描仍在进行中，那么我们有以下选择：

•在查询上保留索引并等待统计信息的逐渐提高
•删除索引
•在查询本身多花点时间进行优化

在提高数据库性能上，我们要不断尝试新的东西。可惜的是，人们总是习惯性认为数据库是相对静态的，并且有时他们都非常不理解：“发生了什么？上周还是工作得很顺利的？？？”当数据库中的数据越来越多时，它就会发生变化。当数据库的功能越来越多时，它就会发生变化。当删除越来越多的数据时，它也发生变化。由于这些变化发生时，我们必须重新评估索引结构、性能调优的可能性。

在 Microsoft 出版社出版上的 Kalen Delaney 的“Inside SQL Server 2005: The Storage Engine”有许多学习索引组件和页结构的好资料。同时，我还是 Itzik Ben-Gan 的超级粉丝。关于这一系列的内容，他出版了两本新书，即“Inside SQL Server 2005:

T-SQL Querying”和“Inside SQL Server 2005: T-SQL Programming”。这两本书也在 Microsoft 出版社的，并且这两人都对这个主题进行了深入地分析。

（作者：Laurence Schwarz 译者：曾少宁／陈柳 来源：TT 中国）

设计查询优化的 SQL Server 非聚簇索引（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	DateFrom	DateTo	Amt	DateInserted	Description

unique)					

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：曾少宁/陈柳 来源：TT 中国)

设计查询优化的 SQL Server 非聚簇索引（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：曾少宁／陈柳 来源：TT 中国）

添加非聚簇索引到 SQL Server 字段

问：我有一个非常大的顾客表。它的 ID 是主键，并且 SQL Server 在上面建立了一个聚簇索引。同时，表上没有其它的索引。那么，我可以添加一个索引到诸如年龄、国家和性别的域上，以便实现更快的查询吗？我们在 WHERE 子句中有很多对于这些字段的表的页查询，因此，表可以进行频繁的更新和写入。我所担心的是，在这些域上创建一个非聚簇索引将会影响性能。你能给我一些建议吗？

答：可以，如果你有根据这些字段查询（WHERE、ORDER BY、GROUP BY 等）或 JOIN，那么你可以添加一个非聚簇索引到这些其它的字段中。但你要记住的是，你只可以有一个聚簇索引和最多 255 个非聚簇索引。

在 SQL Server 2005 中，你也可以添加字段到索引，从而在通常不可加索引的字段上添加索引，如文本数据类型字段。此外，在某种情况下，添加太多索引到有大量插入、更新或删除事务的数据库上将降低性能，因此要小心地选择。

（作者：Jeremy Kadlec 译者：曾少宁／陈柳 来源：TT 中国）

为文本数据创建索引的更佳方法

为文本数据（varchar、nvarchar、char 等）创建索引是一种很好的实现更快数据查询的方法。然而，这些索引会给存储索引的磁盘以及服务器内存带来压力。这是因为索引上存有大量的数据。

例如，下面这个表:

```
CREATE TABLE Employee
(EmployeeID INT,
FirstName VARCHAR(50),
LastName VARCHAR(50),
EmailAddress VARCHAR(255))
```

现在，假设我们要基于EmailAddress域查找数据。那么我们将使用一个非聚簇索引来索引EmailAddress域。如果我们在诸如AMD的公司工作，那么，我们的邮件地址就会相当的短(f.lastname@amd.com)。然而，如果我们在一个像我所工作的公司工作，那么邮件地址就会稍微有点长(flastname@awarenesstechnologies.com)。现在，当我们对这个字段创建索引时，我们就会将整个邮件地址存放到索引上，它就会在这个索引中占用大量的空间；特别相对数值，如一个整数。毫无疑问，如果我们使用一个双字节编码数据类型，那么它的每个字符会需要两个字节的存储空间，而不是只有通常的一个字节。

如果我们需要对系统中带有 URL 的域创建索引，这也会是一个问题。 由于 URL 的长度较长，值的长度可能比索引的值所允许的长度要更长些，这样会带来索引存储问题。

我所知道的这个技术有好几种。我最常用的一种是使用 CHECKSUM 方法作为计算字段的一部分，然后在这个计算字段上创建索引。这样，我们简单地获得我们要查找的字段的 CHECKSUM 值，然后就可以查找计算字段。现在我们就有一个由整数组成的索引，这个索引可以填入比每个物理数据页更多的数据，从而减少了索引查找的 IO 开销并节省磁盘空间。

这样，我们的表变成这样:

```
CREATE TABLE Employee
(EmployeeID INT,
FirstName VARCHAR(50),
LastName VARCHAR(50),
EmailAddress VARCHAR(255),
EmailAddressCheckSum AS CHECKSUM(EmailAddress))
```

现在，我将不再推荐对每个我们所创建的表使用这个技术。我通常只推荐一个这样的技术，当索引的值不符合索引的范围，或表非常的大并且经常进行查找，因此节省的内存是值得在查询前增加额外的 CUP 时间用以哈希结果值。

从而这个技术有几个好处。如果我们检查域名称总数，那么有些字符无法正确统计。同样，检查一个 Unicode 版本的字符串将会得到与同样字符串的非 Unicode 版本不同的结果。

我们可以从下面这三个 SELECT 语句看到:

```
SELECT CHECKSUM(' google.com' ), CHECKSUM(' g-oogle.com' )
SELECT CHECKSUM(' google.com' ), CHECKSUM(N' google.com' )
```

```
SELECT CHECKSUM(N' google.com' ), CHECKSUM(N' g-oogle.com' )
```

我们可以看到在第一个查询中我们获得两个不同的值（分别是 1560309903 和 1560342303）。而对于第二个查询，在 Unicode 和字符串之间我们获得两个不同的值（分别是 1560309903 和 -1136321484）。根据第一个查询，我们可能预期在第三个查询中也会获得两个不同的值，但是结果并不是这样。由于 Unicode 字串“-”似乎并不作为 CHECKSUM 的一部分，因此两个字符串有相同的 CHECKSUM 值（-1136321484）。

这个技术的另外一个版本是最近 Kevin Kline 所讨论的，它使用 SQL Server 2005 的 HASHBYTES 方法来获得字段的哈希值并使用它。在他的博客中，他提出将它用于表分割，其实这个技术也可以用在这里。

```
CREATE TABLE Employee
(EmployeeID INT,
FirstName VARCHAR(50),
LastName VARCHAR(50),
EmailAddress VARCHAR(255),
EmailAddressCheckSum AS HASHBYTES(' SHA1' , EmailAddress)
```

但是，这会得到一个更长的字符串，从而占用索引更多的空间。然而，如果遇到长的 Unicode 字符串，那么这将会是一个更好的选择。

(作者：Denny 译者：曾少宁/陈柳 来源：TT 中国)

SQL Server Index Tuning Wizard 的使用技巧

SQL Server Index Tuning Wizard 将基于我们给定负载量为我们推荐索引方案。但是，记住，当我们设计和检查 Index Tuning Wizard 的推荐方案时，很重要的一点是我们要先理解基础数据库索引方法。适合我们应用环境的最终结果将会根据数据库设计、业务过程、并发度、数据类型等等而有所不同，并且该结果还需要经过充分地测试。我们要敢于根据我们的应用的唯一特性来开发和测试非常规选择，以保证我们所实现的索引在提高一方面性能的同时不会降低另一方面的性能。

下面的一系列技巧将有助于我们更有效地使用 Index Tuning Wizard 来提高 SQL Server 2000 的性能。

1、Index Tuning Wizard 是 Enterprise Manager 中隐藏的重要工具之一。访问该工具，可以打开 Enterprise Manager，找到 Tools 菜单，然后选择 Wizard 选项。当出现了 Select Wizard 界面时，展开 Management 标题，然后双击 Index Tuning Wizard 选项。Index Tuning Wizard 会出现下面的界面：

- Welcome Screen（欢迎屏面）
- Select Server and Database（选择服务器和数据库）
- Specify Workload（指定负载）
- Select Tables to Tune（选择优化表）
  - The analysis will be conducted once the OK button is pressed
  - 当按下 OK 按钮，分析就会进行
- Index Recommendations（索引推荐）
- Schedule Index Update Job（调度索引更新任务）
- Completing the Index Tuning Wizard（完成）

2、SQL Profiler 是 Index Tuning Wizard 成功的关键。运行向导并得到推荐的索引设计对于每个人来说都有很大的好处。但是，我也曾经遇到过向导并不能提供任何索引推荐的情况，这让我非常的沮丧。而正确执行 SQL Profiler 后将会出现完全不同的结果。如果提供给 Index Tuning Wizard 的数据不能反映我们的问题，那么 Index Tuning Wizard 也将无法提供有价值的建议用于索引选择。因此，我们要确保 SQL Profiler 捕捉到我们所感兴趣的性能数据。

3、捕捉一天当中不同时间的 Profiler 会话，并将这些会话结果存储在不同的表中。然后，如果我们将数据转移到一个表中，那么我们可以看到 Index Tuning Wizard 分析的独立的数据集以及单一的数据集的结果。其中的差别可以有助于我们深入理解并决定实现哪些推荐的索引。

4、通过 SQL Profiler 对于单个会话捕捉的数据以一个可控制方式协调一个用户会话与业务应用。然后，使用 Index Tuning Wizard 来单独分析和推荐事务。接着，检查查询计划来决定哪些索引可以使用并验证没有一个查询是通过表扫描实现，而不是使用索引实现。

5、通过选中查询并按 CTRL + I，我们可以直接在 Query Analyzer 中调用 Index Tuning Wizard。然后，按照向导的步骤完成分析。直接在 Query Analyzer 中启动 Index Tuning Wizard 的意义在于，我们可以与其余应用代码独立地分析单个查询。

6、检查 Index Tuning Wizard 所提供的推荐以确定它们有利于我们的应用环境。不能只是盲目地添加索引。将这些索引应用到一个开发环境和测试环境，并测试相同的 SQL Profiler 数据，查看结果是否与预期的一样。

7、不要怕添加索引，但也要重新检查一些基本数据库设计以及编码的最佳实践。有时，使用索引确实可以提升性能。但是，在某些时候，我们仍必须重新检查数据库设计和编码以获取最终解决方案。

(作者：Jeremy Kadlec 译者：曾少宁/陈柳 来源：TT 中国)

处理 SQL Server 2000 索引碎片技巧（一）

当遇到数据库的性能问题时，其中一个最大的性能提升方法可以通过优化索引来实现。索引可以改善数据访问，这样我们就不需要扫描整个表，因为这会消耗大量的 CPU、IO 和内存资源。随着时间的推移，索引可能会产生碎片，从而导致 SQL Server 性能下降、事务时间处理时间变长、阻塞和低吞吐量。

索引碎片的程度取决于表的碎片，其中这可能是由页拆分造成的。当执行插入或更新语句同时一个页的数据超过页面总容量时，就会发生页拆分。数据自动填充到另一个页面是支持数据存储所必需的。当发生页拆分时，SQL Server 将一半的数据保留在原来的页上，而另一半数据则存储到另一个页上。这个操作的开销主要是在时间和并发量上。页拆分最终是由过高的索引填充因数和 pad_index 而引起页面过度压缩所造成的。

消除索引碎片既是艺术也是技术，同时也是在不断变化的应用环境下保证数据库高性能的长期过程。接下来，我们将探讨以下内容：

- 理解数据
- 分析索引
- 理解如何创建索引
- 确定碎裂的索引
- 重建碎裂的索引
- 配置数据库

技巧 1：理解数据

首先需要的是理解所存储的数据以及数据的访问方式。对于新的应用，可以与用户和应用开发人员一起探讨他们的应用流程和数据使用需求——数据访问（SELECT）和数据操作（INSERT、UPDATE 和 DELETE）。对于已有的系统，执行 sp_depends，它将找出与表相

关的数据库对象。如果在前端、中间层或具体代码中嵌入了任何的 T-SQL，那么可以与应用开发人员交谈以了解它们的作用。同时，与用户交谈以理解他们所遇到的应用问题。在任一种情况下，都可以利用工具或脚本来更好地理解业务流程和索引优化的可能，同时要获得用户和开发人员的确认。

(作者：Jeremy Kadlec 译者：曾少宁/陈柳 来源：TT 中国)

处理 SQL Server 2000 索引碎片技巧（二）

技巧 2：分析索引

一旦理解了数据，接着就可以设计或重新设计索引了。在 SQL Server 2000 Books Online 中，有一篇不错的关于索引设计的文章，建议大家可以去阅读一下。首要的一条规则是，索引必须基于 JOIN 条件，以及 WHERE、ORDER BY 和 GROUP BY 子句中使用的字段。阅读该文章以获得其它的推荐。

CRUD 图表（创建、读取、更新、删除）也可以帮助我们理解数据访问所需要的索引实现方式。虽然建立一个 CRUD 图表是一项艰难的任务（很多人甚至都不知道如何着手），但是，为关键的表创建一个 CRUD 图表可能有助于整体索引；因为大多数的系统都有与表子集有关的性能问题，所以关注这些问题将可以最小代价地实现性能的提高。下面是一个执行存储过程\特殊操作（JOIN、WHERE、ORDER BY 等）的表（Customer）的 CRUD 图表示例：

Table Name = Customer						
ID	Column	Object	Create	Read	Update	Delete
1	CustomerID	spCustomerSelectAll	-	INNER JOIN	-	-
-	-	spCustomerUpdate	-	-	-	WHERE
2	Name	spCustomerSelectAll	-	ORDER BY	-	-
-	-	spCustomerDelete	-	-	-	WHERE
-	-	spCustomerLookup	-	WHERE	-	-
3	Address	-	-	-	-	-
4	City	-	-	-	-	-

5	State	spCustomerSelect_ByState	–	WHERE GROUP BY	–	–
6	ZipCode	–	–	–	–	–
7	PhoneNumber	–	–	–	–	–
8	EmailAddress	Ad-Hoc	–	WHERE	–	–
*	spCustomerInsert	None	–	–	–	–

这个图表提供了每一表上的索引的科学理解。从这些数据中，我们需要根据事务的类型来平衡索引需要，这不仅是一个艺术形式。典型地，索引可以提升 SELECT 命令性能，但是它们可能影响 INSERT、UPDATE 或 DELETE 语句性能，因为在这些操作中，SQL Server 必须维护索引，这需要额外的开支。

数据的另外一个重要的方面是索引重建的频率（日、周、月、季度、年以及从不）。索引重建的频率越高，索引方法就越积极。如果系统需要在接下来的三年中 24 小时/7 天地运行以支持一个功能，那么索引的设计一定不同于每周重建索引来支持多个功能的数据库。尽管如此，系统并不是不变的，它们是随着业务需求的改变而改变的。因此对这些进行计划或修正会在将来遇到问题时更主动一些的。

技巧 3：理解如何创建索引

SQL Server 有两种索引：聚簇索引和非聚簇索引。我将使用前面的两个来自 SQL Server 2000 Books Online 的详细信息来探讨这个话题。简单而言，一个表可以没有索引、有一个聚簇索引、一个聚簇索引和一个非聚簇索引，或者只有非聚簇索引。可以为每个表创建一个聚簇索引，然后数据将会根据这个字段来物理地排序。对于没有聚簇索引的表，数据将根据数据项的顺序存储的。

- CREATE INDEX 的两个与索引碎片相关的主要配置是：
- FILL FACTOR——确定叶级（数据）页的填充程度。

PAD_INDEX——确定中间级（从索引到数据页的指针）页填充的程度，典型的情况下，它与 FILL FACTOR 的值是一样的。

如果我们无法确定在特定的间隔（如，日、周、月、季度、半年度、年度等等）中碎片的程度，那么最好把开始的填充因数配置为 80%。监测碎片以便确定基于每个表\聚簇索引的理想配置；它所需要的存储量会与配置为 5%到 10%时有很大的不同。填充因数越低（如，40%到 50%），所需要的存储空间就越大，并且索引需要扫描或查找更多的页以满足查询需要。填充因数越高（90%到 100%），所需要的存储空间就越小，并且需要扫描的页面就越少，但是页拆分的开销会导致性能降低和索引碎片。有什么好建议呢？寻找理想的填充因数来避免页拆分和碎片，同时还不会导致过大数据库存储空间。

（作者：Jeremy Kadlec 译者：曾少宁/陈柳 来源：TT 中国）

处理 SQL Server 2000 索引碎片技巧（三）

技巧 4：确定碎裂的索引

确认索引碎片的主要SQL Server命令是DBCC SHOWCONTIG。下面这个示例代码是用来确定Pubs数据库中的Authors表的索引碎片的：

```
USE Pubs
GO
DBCC SHOWCONTIG ('Authors')
GO
```

确定Pubs数据库中的Authors表的聚簇索引的索引碎片，可以执行下面的命令：

```
USE Pubs
GO
DBCC SHOWCONTIG ('Authors',1)
GO
```

作为一个引用点，1 值显示聚簇索引。2 到 255 值显示一个具体的非聚簇索引。

确定是否重建索引的一条经验法则是看扫描密度是否低于 90%。

要根据技巧 1 来理解我们的数据：如果注意到特定的表总是有较低的扫描密度的，那么可以考虑将填充因数和 pad_index 降低 5%到 10%以减少碎片。

随着数据的增长，会造成数据库增大和事务运行时间的增长，维护窗口每一秒都会有计数。利用我们的备份服务器或将一个最近的生产数据库备份恢复到一个开发/测试服务器，然后执行DBCC命令。这样我们就可以及时地查看数据库碎片情况，从而维护窗口就可以只集中在重建的索引上。一旦我们有了这个数据，我们就可以执行技巧 4 上所列出的命令中的一个来重建碎裂的索引。然后，重新执行DBCC SHOWCONTIG来验证索引碎片是否已经修正。这将可以验证我们的脚本，并且粗略估计生产系统所需要的时间。

技巧 5：重建碎裂的索引

索引维护是一个用来保证索引最佳配置的关键。“Index rebuild options”表概括了用来维护索引的典型方法。

Index rebuild options			
ID	Description	Recommendations	Sample code
1	DROP INDEX CREATE INDEX	执行：当系统没有用户时，当表的索引变化要求删除旧的索引并用新的索引配置替代时，当表的聚簇索引发生变化时，因为所有非聚簇索引都依赖于聚簇索引，所以它们需要重建。	USE PUBS GO DROP INDEX Authors.au_id_ind GO CREATE CLUSTERED INDEX au_id_ind ON authors (au_id) GO

2	DBCC DBREINDEX	执行：当系统没有用户时，当表之间存在引用完整性并且需要维护时，当索引配置修改聚簇索引所需要的填充因数时，它支持一个原子事务，以保证不会丢失任何索引。	USE Pubs GO DBCC DBREINDEX (Authors, '', 70) GO
3	DBCC INDEXDEFRAG	当系统没有用户时执行。用以清除一个索引的碎片。	USE Pubs GO DBCC INDEXDEFRAG (Pubs, Authors, au_id_ind GO

技巧 6：配置数据库

与索引碎片相关的是磁盘级碎片，它是由文件删除和逻辑磁盘文件系统重组所造成的非连续文件系统，它会降低I/O敏感进程的性能。下面的建议可以最小化磁盘级碎片：

- 为数据库准备专用磁盘
- 以连续方式在磁盘上进行数据库写入

- 给大型的数据预分配相对应的数据库大小（如，为每个月增长 1GB的 100GB数据库多准备 10GB空间）
- 不要自动增长和缩小数据库

总结

索引碎片对高性能的数据库是至关重要的。确认有益的索引并不断维护它们将保证整个应用过程中的高性能。祝你好运！

（作者：Jeremy Kadlec 译者：曾少宁/陈柳 来源：TT 中国）

最佳 SQL Server 索引策略

恰当的索引能创建完全不同的性能。对于大多数的数据类型，SQL Server 只支持两种索引类型——聚簇索引和非聚簇索引。同时，SQL Server 也支持全文索引和 XML 索引，但是它们只与特定数据类型相关。

为聚簇索引选择恰当的字段或字段集是至关重要的。这是因为表的数据是根据聚簇索引字段的值而物理地排序的。每个表上只能创建一个聚簇索引。非聚簇索引引用聚簇索引的键（数据值）来确定每个记录的物理位置。

一般推荐在不会频繁修改的、经常会被查询的和有瘦数据类型的字段上创建聚簇索引。在大多数情况下，在标识字段上的聚簇索引是最佳的，因为标识值是最常被查询的——每个记录都有一个唯一的标识值——同时它们从不用更新，而且是使用 SMALLINT、INT 或 BIGINT 数据类型创建的。

然而，通常情况下是不会有一个表从来不被基于它的标识字段查询的。如果真有，我们就需要仔细考虑数据通常是如何检索的，可能是通过指向另外一个表的外键或者一个字符字段。通常情况下，通过在最常用来检索数据的字段或字段集上创建聚簇索引可以提高性能。

有些开发人员倾向于创建组合聚簇索引。它们都包括几个字段，是一个唯一标识每个记录的组合。听起来可能是一个很好的实践方法，因为标识字段没有业务意义，而其它的字段——如入职日期、部门名称和车辆识别码——一定可以被应用用户快速理解。然而，对性能而言，我们必须避免使用组合聚簇索引。

再次，越瘦的索引，SQL Server 可以越快地扫描或查找。对于一个小的数据集，组合索引的性能相对更好些。但是，随着用户数增加，它将会出现更多的问题。

建立恰当的索引可以提高性能，我们也许会觉得工作已经完成了。但是，随着表中数据的添加、修改和删除，各个索引会出现碎片。碎片的程度越高，索引的效率就越低。因此，现在我们需要实现一个将索引中的碎片删除的计划，以便确保索引的有效性。

对于先前的 SQL Server 版本，从大索引（有数百万行的表）中删除碎片往往需要停机。幸运的是，SQL Server 2005 支持在线索引重建。然而，记住，重建索引仍然需要占用系统资源和数据库的 tempdb 空间。可能的话，调度索引维护在活跃用户最少的时间进行。

SQL Server 数据库架构师和管理员可以选择多种方法来使他们的应用在一开始就表现良好。为了保证成功的数据库性能，在设计阶段作出很好的选择非常重要。在这个版本的 SQL Server INSIDER 中，专家 Baya Pavliashvili 探讨了如何作出可以优化性能的数据库设计决定。设计一个数据库包括合适的：

- 数据模型
- 数据类型
- 索引策略
- 代码模块
- 高可用选项

（作者：Baya Pavliashvili 译者：曾少宁/陈柳 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：Denny 译者：曾少宁/陈柳 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：曾少宁/陈柳 来源：TT 中国）

SQL Server 中用于查找索引碎片的存储过程（上）

问题

由于数据修改，SQL Server 表和索引会逐渐出现数据碎片。在大型的 I/O 操作中，在 SQL Server 中用到这些碎片索引和表，可能对应用性能产生不利影响。

如果你是负责十几个 SQL Server 实例和上百个数据库的 DBA 中的一员，那么你将无法控制或者了解各种数据库中的活动类型。在这种情况下，很重要的一点是要定期检查问题碎片表和索引（除非你周期性地整理所有表的碎片）。这个任务经常落在 DBA 的身上，因为开发团队并不知道数据的物理表现。

本文提供一个用于收集所有碎片索引信息到一个中央数据库的代码。

可能的命令

我们可以使用下面两个命令来查找产生碎片的表：

- DBCC SHOWCONTIG, for SQL Server 2000 or SQL Server 2005.
- Select …. From sys.dm_db_index_physical_stats, in SQL Server 2005.

这些命令必须在所有服务器的所有数据库上的索引上执行。下面有几个方法可供选择。比如：

1. Run a DTS/SSIS package.

2. Write a program that loops on the server list and collects the information.

3. Run a SQL Server T-SQL batch job.

4. 运行 DTS/SSIS 包。

5. 编写一个程序遍历服务器列表并收集信息。

6. 运行 SQL Server T-SQL 批处理任务。

为了在任意版本的 SQL Server 上运行而不需要先询问版本号，我决定执行 DBCC SHOWCONTIG 命令。为了简单起见，我仅使用 T-SQL 和链接服务器来创建一个 SQL Server 任务。

注意：当我的所有服务器都更新到 SQL Server 2005 时，为了符合最佳实践方法，我打算使用 sys.dm_db_index_physical_stats 取代 DBCC SHOWCONTIG。

环境

我们在一个中央开发服务器上创建了一个 SQL Server 实例，并为 DBA 创建了一个数据库（DBA_DB），同时将服务器与所有的生产服务器连接。在 DBA_DB 数据库上，我创建了一个表记录所有生产服务器，并用一个名为“IsActive”的字段表示该服务器是否需要检查（这样，我们就可以排除某些已经检查了的服务器）：

```
CREATE TABLE dbo.t_ServersList(
        ServerName   nvarchar(128) NOT NULL,
        IsActive     bit           NOT NULL
)
```

点击此处下载以上脚本

我有一些程序遍历所有“LIVE”服务器（IsActive = 1）并执行一些常规任务。这些任务包括检查在最后 X 天中未备份的数据库、执行周期性能跟踪，等等。现在，我将要添加一个程序来检查碎片索引。我在中央数据库中启用 xp_cmdshell 存储过程。

（作者：Michelle Gutzait 译者：曾少宁/陈柳 来源：TT 中国）

SQL Server 中用于查找索引碎片的存储过程（下）

策略

策略是使用一个简单的存储过程。我从所有服务器和数据库上收集索引碎片信息，并将它们存入我所创建的 DBA_DB 数据库上的常量表中：

```
CREATE TABLE t_FragmentedTables (
        ServerName      sysname   null,
        DatabaseName        sysname   null,
        ObjectName      CHAR(255) null,
        ObjectId            INT       null,
        IndexName       CHAR(255) null,
        IndexId             INT       null,
        Lvl             INT       null,
        CountPages      INT       null,
        CountRows       INT       null,
        MinRecSize      INT       null,
        MaxRecSize      INT       null,
        AvgRecSize      INT       null,
        ForRecCount         INT       null,
        Extents             INT       null,
        ExtentSwitches    INT       null,
        AvgFreeBytes        INT       null,
        AvgPageDensity    INT       null,
        ScanDensity     DECIMAL   null,
        BestCount       INT       null,
        ActualCount     INT       null,
        LogicalFrag     DECIMAL   null,
        ExtentFrag      DECIMAL   null)
Go
```

点击此处下载以上脚本

存储过程是这样的：

点击此处下载完整的存储程序

存储过程结果

碎片数据将被收集到 t_FragmentedTables 表中。这些收集的数据有什么用呢？下面是一些应用例子：

1. Send an email with the fragmented indexes information.
2. Open tickets regarding the fragmentation -- assuming you have a ticketing system and you can interact with it.
3. 发送包含碎片索引信息的邮件
4. 根据碎片创建标签——假设已有一个标签系统并且可以交互。

例如（发送包含碎片表的邮件——消息体中最多可有 8，000 个字符）：

```
declare @msg  varchar(8000),
        @err  int
if exists (select 1 from t_FragmentedTables)
begin
        set @msg = 'Fragmented tables:' + char(10) +
                   '====================' + char(10) + char(10) +
           '(ServerName - DatabaseName - TableName - ' +
           'IndexName - LogicalFragmentation' + char(10) +
           '-------------------------------------------------'

        select @msg = @msg + char(10) + ServerName + ' - ' +
             DatabaseName + ' - ' + ObjectName + ' - ' +
             IndexName + convert(varchar(10),LogicalFrag)
        from t_FragmentedTables

        exec @err=master..xp_sendmail
             @recipients = 'michelle.gutzait@DummyWeb.com',
                   @message = @msg,
                   @subject = 'Fragmented Tables!'
        if @err <> 0
        begin
              print 'ERROR sending mail'
              return
        end
end
else
        print 'NO FRAGMENTED TABLES!'
```


点击此处下载以上脚本

性能

为了说明性能表现，我在当数据库上的活动频率较低时在一个测试环境中运行了我的存储过程：

- Windows 2003 Standard Edition SP1
- 2* Dual Core CPU 3.06 GHz
- 4 GB RAM, maximum of 2 GB configured for SQL Server

- SQL Server 2000 SP3
- Total size of user databases = 46 GB
- Total of 2,170 user tables
- Total of 8,154 indexes
- Total of 5,864 fragmented indexes

执行在大概三个小时之后才成功结束。

总结：

如果我们用下面的对象创建一个环境包含 SQL Server 实例：

- 一个特殊的“DBA”数据库（例如：DBA_DB），它包含：
  - 如本文中所描述的，服务器清单（t_ServerList）表
  - 如本文中所描述的，捕捉碎片数据表
- 为清单表(t_ServerList)每台“活跃的”服务器配置的链接服务器

接着，我们只需要复制和粘贴存储过程代码并执行它。然后，我们就可以根据碎片详细信息分析本地表中的结果，并根据我们的需要来执行警报/报告/发送邮件。我们也可以通过使用 Microsoft Books Online 上所描述的方法之一来自动整理索引碎片。

注意：我例子中的存储过程应该在数据库活动频率低的时候（比如，每个月的周末）候运行。

(作者：Michelle Gutzait 译者：曾少宁/陈柳 来源：TT 中国)