**SQL Server 2014新特性**

[**SQL Server 2014新特性探秘(1)-内存数据库**](http://www.cnblogs.com/CareySon/p/3155753.html)

**简介**

   SQL Server 2014提供了众多激动人心的新功能，但其中我想最让人期待的特性之一就要算内存数据库了。去年我再西雅图参加SQL PASS Summit 2012的开幕式时，微软就宣布了将在下一个SQL Server版本中附带代号为Hekaton的内存数据库引擎。现在随着2014CTP1的到来，我们终于可以一窥其面貌。

**内存数据库**

    在传统的数据库表中，由于磁盘的物理结构限制，表和索引的结构为B-Tree，这就使得该类索引在大并发的OLTP环境中显得非常乏力，虽然有很多办法来解决这类问题，比如说乐观并发控制，应用程序缓存，分布式等。但成本依然会略高。而随着这些年硬件的发展，现在服务器拥有几百G内存并不罕见，此外由于NUMA架构的成熟，也消除了多CPU访问内存的瓶颈问题，因此内存数据库得以出现。

    内存的学名叫做Random Access Memory（RAM），因此如其特性一样，是随机访问的，因此对于内存，对应的数据结构也会是Hash-Index，而并发的隔离方式也对应的变成了MVCC，因此内存数据库可以在同样的硬件资源下，Handle更多的并发和请求，并且不会被锁阻塞，而SQL Server 2014集成了这个强大的功能，并不像Oracle的TimesTen需要额外付费，因此结合SSD AS Buffer Pool特性，所产生的效果将会非常值得期待。

**SQL Server内存数据库的表现形式**

    在SQL Server的Hekaton引擎由两部分组成：内存优化表和本地编译存储过程。虽然Hekaton集成进了关系数据库引擎，但访问他们的方法对于客户端是透明的，这也意味着从客户端应用程序的角度来看，并不会知道Hekaton引擎的存在。如图1所示。

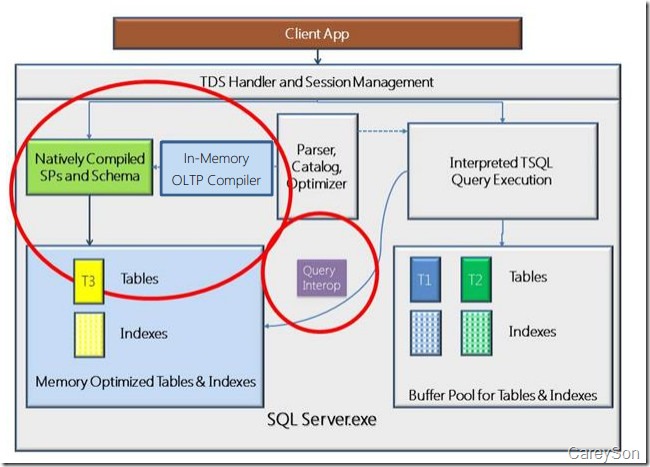
[](http://images.cnitblog.com/blog/35368/201306/25222827-13892506a07c43b1954bfa389f7ba1ee.jpg)

图1.客户端APP不会感知Hekaton引擎的存在

    首先内存优化表完全不会再存在锁的概念（虽然之前的版本有快照隔离这个乐观并发控制的概念，但快照隔离仍然需要在修改数据的时候加锁），此外内存优化表Hash-Index结构使得随机读写的速度大大提高，另外内存优化表可以设置为非持久内存优化表，从而也就没有了日志（适合于ETL中间结果操作，但存在数据丢失的危险）

    下面我们来看创建一个内存优化表：

    首先，内存优化表需要数据库中存在一个特殊的文件组，以供存储内存优化表的CheckPoint文件，与传统的mdf或ldf文件不同的是，该文件组是一个目录而不是一个文件，因为CheckPoint文件只会附加，而不会修改，如图2所示。

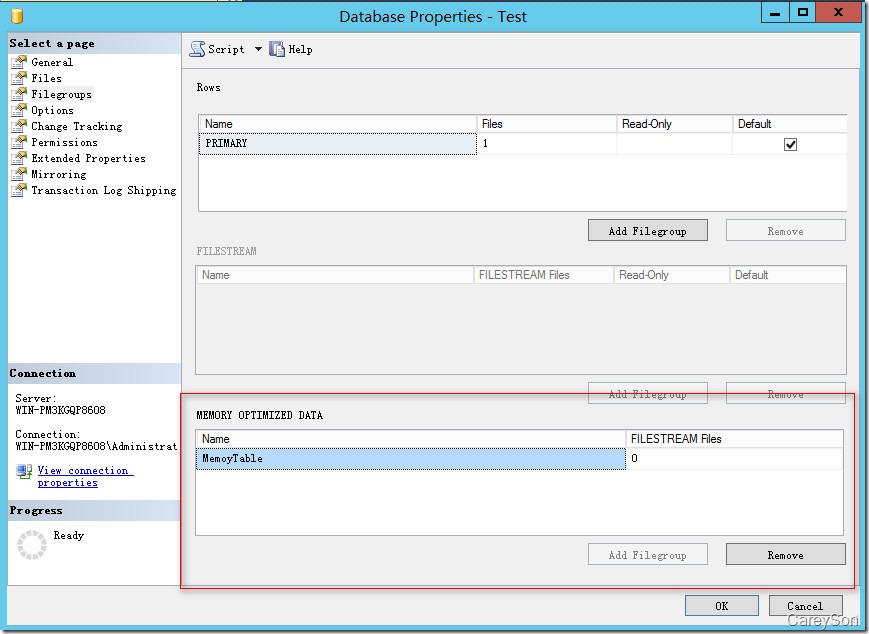
[](http://images.cnitblog.com/blog/35368/201306/25222828-2aa3fad381734f3cb0538b0368dd4fa2.jpg)

图2.内存优化表所需的特殊文件组

我们再来看一下内存优化文件组的样子，如图3所示。

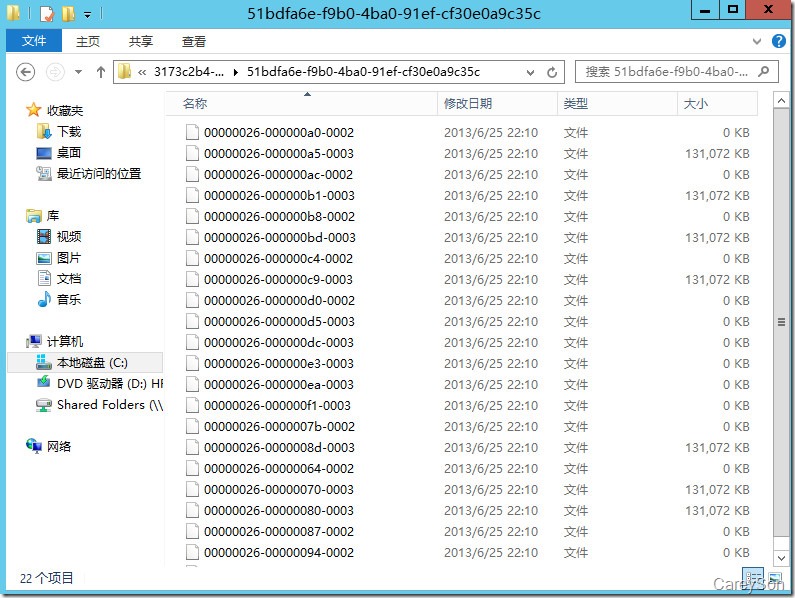
[](http://images.cnitblog.com/blog/35368/201306/25222846-c223ebd563a94cebadee8fe616f1b6bf.jpg)

图3.内存优化文件组

    有了文件组之后，接下来我们创建一个内存优化表，如图4所示。

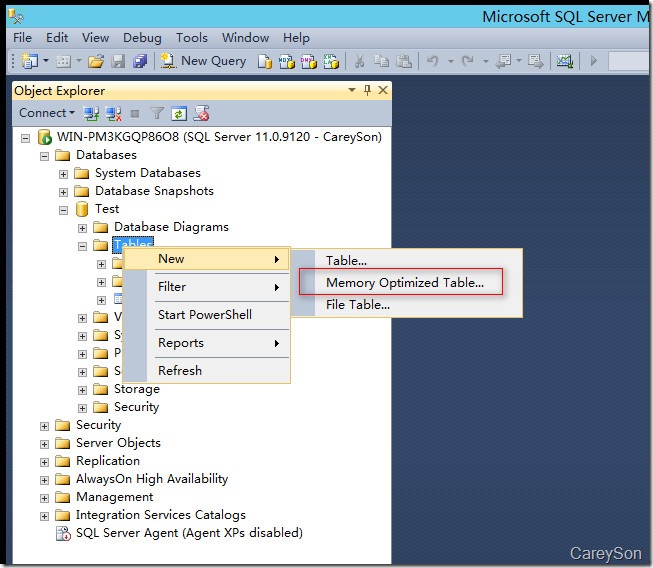
[](http://images.cnitblog.com/blog/35368/201306/25223234-f6fd0ce7c9f34d57afa13bd67a7a51d7.jpg)

图4.创建内存优化表

   目前SSMS还不支持UI界面创建内存优化表，因此只能通过T-SQL来创建内存优化表，如图5所示。

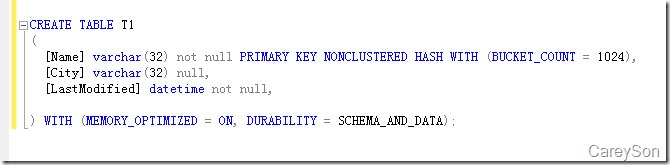
[](http://images.cnitblog.com/blog/35368/201306/25223238-0774e2f8607e4eceb93c3980fd709c70.jpg)

图5.使用代码创建内存优化表

    当表创建好之后，就可以查询数据了，值得注意的是，查询内存优化表需要snapshot隔离等级或者hint，这个隔离等级与快照隔离是不同的，如图6所示。

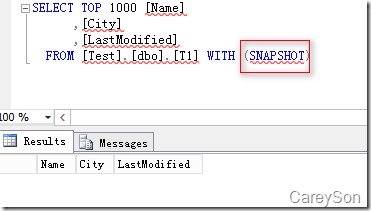
[](http://images.cnitblog.com/blog/35368/201306/25223245-ed5d10efcda94b7f927a6cf1c7561eda.jpg)

图6.查询内存优化表需要加提示

    此外，由创建表的语句可以看出，目前SQL Server 2014内存优化表的Hash Index只支持固定的Bucket大小，不支持动态分配Bucket大小，因此这里需要注意。

**与内存数据库不兼容的特性**

    目前来说，数据库镜像和复制是无法与内存优化表兼容的，但AlwaysOn，日志传送，备份还原是完整支持。

性能测试

   上面扯了一堆理论，大家可能都看郁闷了。下面我来做一个简单的性能测试，来比对使用内存优化表+本地编译存储过程与传统的B-Tree表进行对比，B-Tree表如图7所示，内存优化表+本地编译存储过程如图8所示。

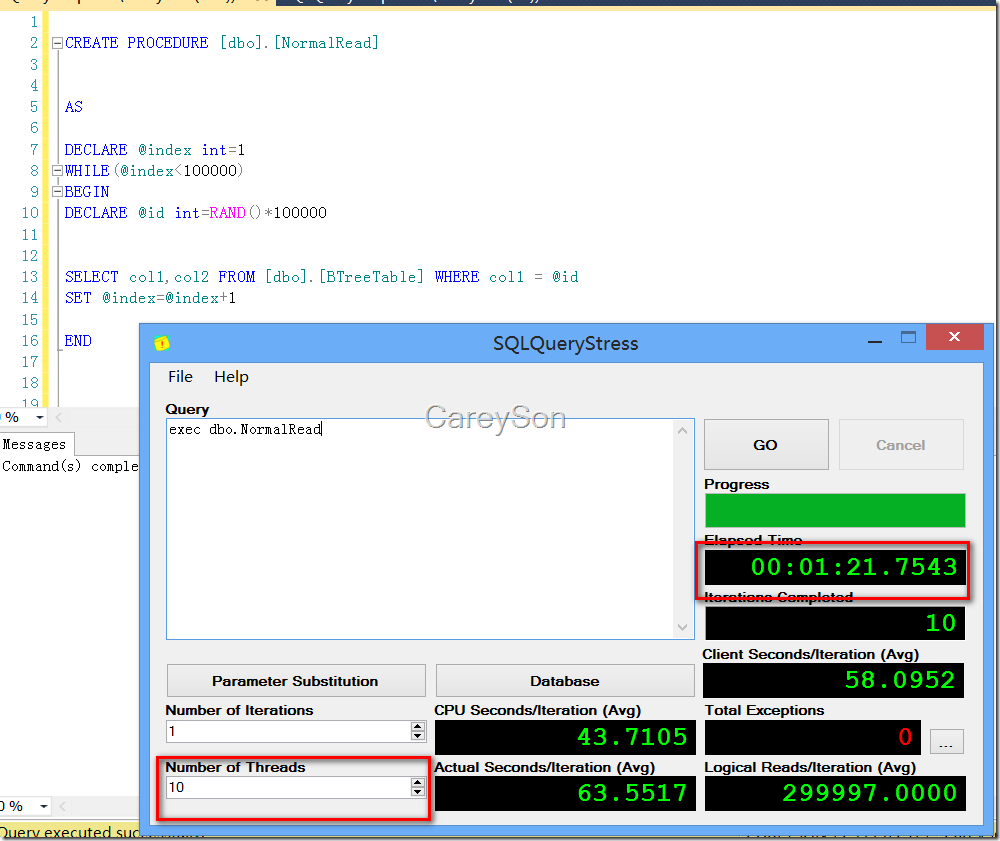
[](http://images.cnitblog.com/blog/35368/201306/25223307-84dd4ea6e46f4e19ba5d33ae22cd93b7.png)

图7.传统的B-Tree表

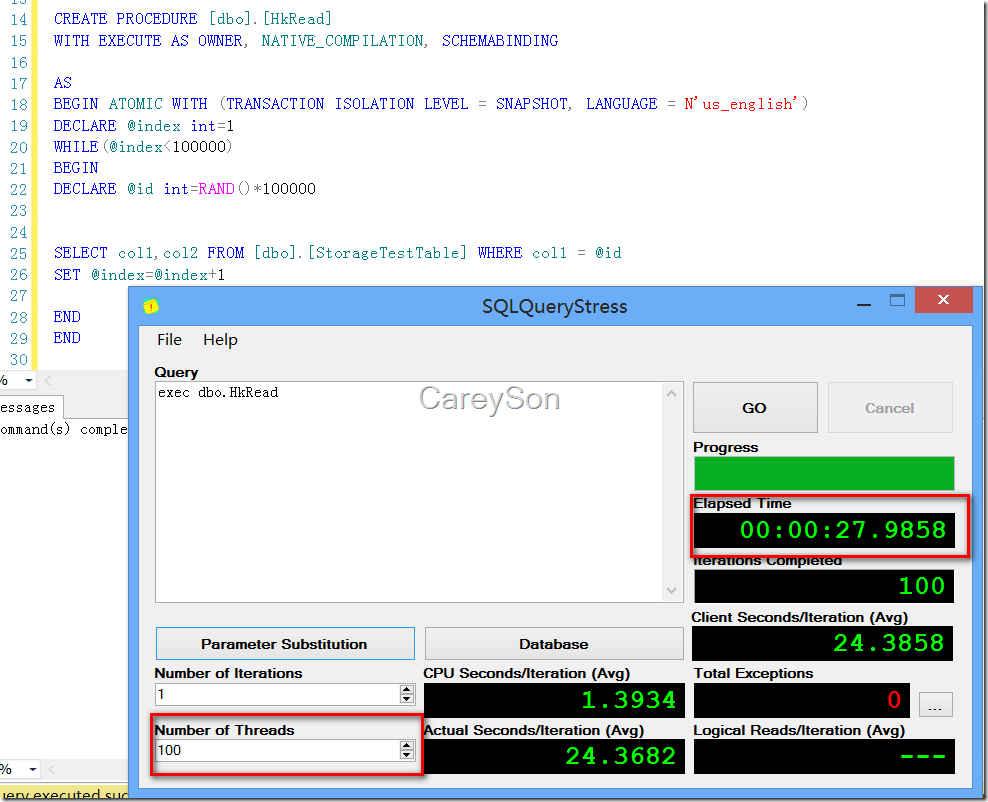
[](http://images.cnitblog.com/blog/35368/201306/25223345-e8a10fff15654cbe85c9f008d1213666.png)

图8.内存优化表+本地编译存储过程

   因此不难看出，内存优化表+本地编译存储过程有接近几十倍的性能提升。

# [SQL Server 2014新特性探秘(2)-SSD Buffer Pool Extension](http://www.cnblogs.com/CareySon/p/3157252.html)

### 简介

    SQL Server 2014中另一个非常好的功能是，可以将SSD虚拟成内存的一部分，来供SQL Server数据页缓冲区使用。通过使用SSD来扩展Buffer-Pool，可以使得大量随机的IOPS由SSD来承载，从而大量减少对于数据页的随机IOPS和PAGE-OUT。

### SSD AS Buffer Pool

    SSD是固态硬盘，不像传统的磁盘有磁头移动的部分，因此随机读写的IOPS远远大于传统的磁盘。将SSD作为Buffer Pool的延伸，就可以以非常低的成本巨量的扩充内存。而传统的模式是内存只能容纳下热点数据的一小部分，从而造成比较大的Page-Out，如图1所示。

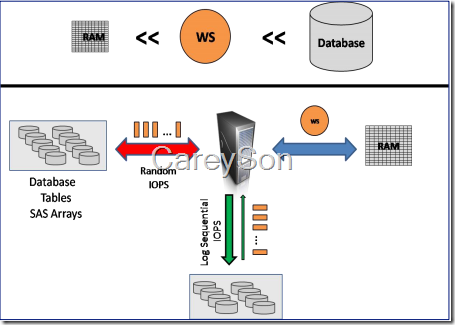
[](http://images.cnitblog.com/blog/35368/201306/26194730-31a114fd6cfb4dab8852844f67e6b7b0.png)

图1.大量随机的IOPS需要由磁盘阵列所承担

    但如果考虑到将SSD加入计算机的存储体系，那么内存可以以非常低的成本扩展到约等于热点数据，不仅仅是提升了性能，还可以减少IO成本，如图2所示。

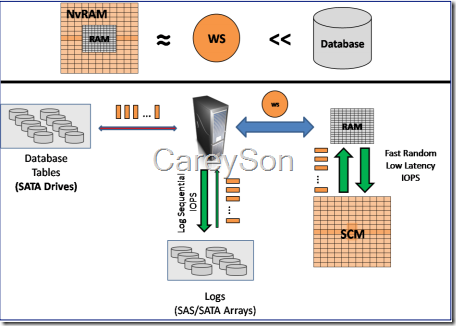
[](http://images.cnitblog.com/blog/35368/201306/26194731-0e5154fb56e34a80a851775061b265f2.png)

图2.扩展后内存几乎能HOLD所有热点数据

    由图1和图2的对比可以看出，扩展后可以使用更便宜的SATA存储。此外，该特性是透明的，无需应用程序端做任何的改变。

    此外，该特性为了避免数据的丢失，仅仅在作为缓冲区的SSD中存储Buffer Pool的Clean Page，即使SSD出现问题，也只需要从辅助存储中Page In页即可。

    最后，该特性对于NUMA进行了特别优化，即使拥有超过8个Socket的系统，CPU也能无障碍的访问内存。

### 启用BUFFER Pool Extension

    在SQL Server 2014总，启用Buffer Pool Extension非常简单，仅仅需要拥有SysAdmin权限后，输入一个T-SQL语句即可，如图3所示。

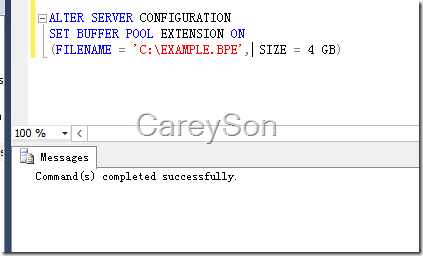
[](http://images.cnitblog.com/blog/35368/201306/26194732-4ad094ae47b94e24acb8690c7c37b18e.png)

    图3.启用Buffer Pool Extension

    对应的，我们可以在物理磁盘中看到这个扩展文件，该文件的性能和Windows的虚拟内存文件非常类似，如图4所示。

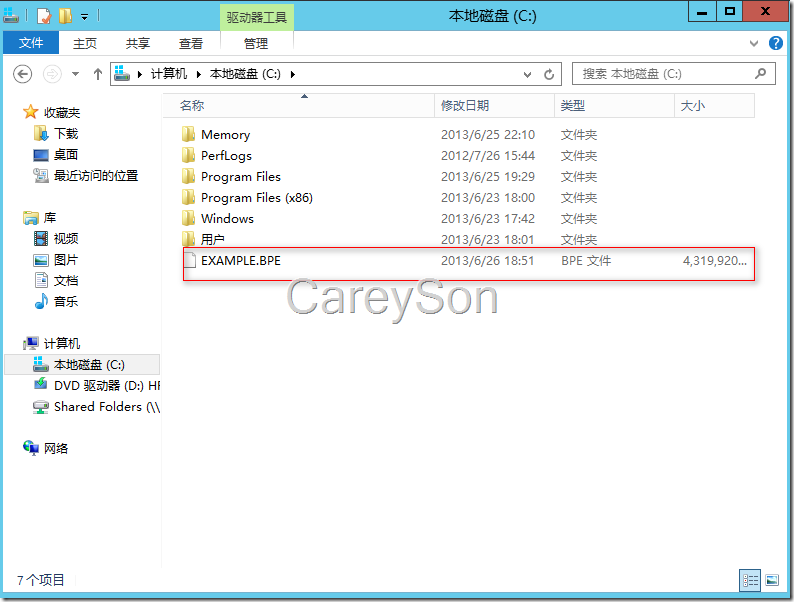
[](http://images.cnitblog.com/blog/35368/201306/26194735-e5e65e251c7c4c5aa22bdf0bf6ddda0b.png)

    图4.对应的Buffer Pool扩展文件

    但这里值得注意的是，我们启用的内存扩展无法小于物理内存或阈值，否则会报错，如图5所示。

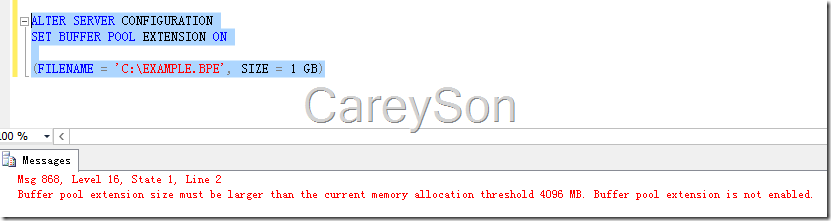
[](http://images.cnitblog.com/blog/35368/201306/26194736-f9f35ec9335c468ca5ebacdffcd9a1c3.png)

    图5.报错信息

    对于该功能，SQL Server引入了一个全新的DMV和在原有的DMV上加了一列，来描述Buffer Pool Extention，如图6所示。

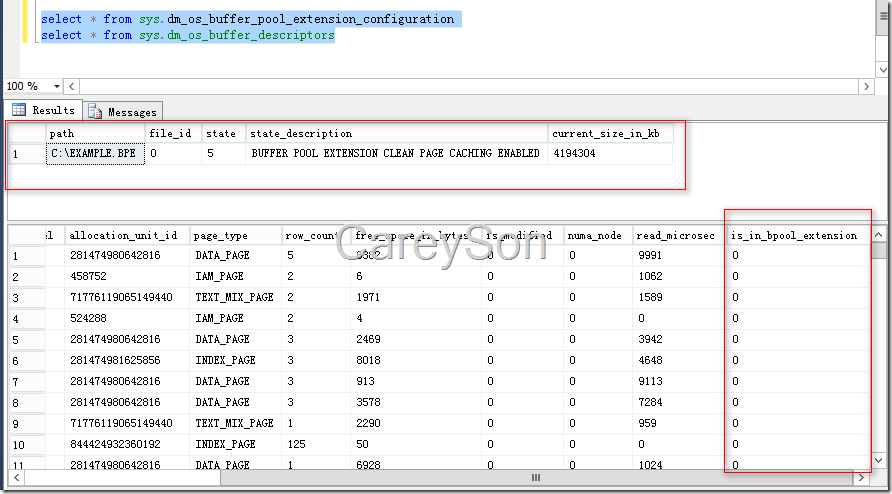
[](http://images.cnitblog.com/blog/35368/201306/26194737-022a065760544c699e3adee7137c684a.png)

    图6.引入的新的DMV和对于原有DMV的更新

     此外，对于该特性的监控，SQL Server还引入了大量与之相关的计数器，如图7所示。

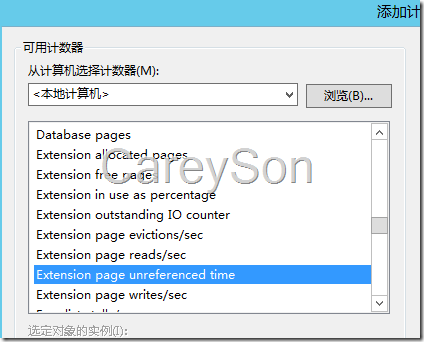
[](http://images.cnitblog.com/blog/35368/201306/26194737-d1cc2beffadf4392bb85787f1ffb6aaf.png)

    图7.相关计数器

### 小结

    SQL Server Buffer Pool Extension给我们提供了以更低成本来满足更高企业级需求的可能，结合内存数据库，未来的可能性将无限延伸。

# [SQL Server 2014新特性探秘(3)-可更新列存储聚集索引](http://www.cnblogs.com/CareySon/p/3530845.html)

### 简介

     列存储索引其实在在SQL Server 2012中就已经存在，但SQL Server 2012中只允许建立非聚集列索引，这意味着列索引是在原有的行存储索引之上的引用了底层的数据，因此会消耗更多的存储空间，但2012中的限制最大的还是一旦将非聚集列存储索引建立在某个表上时，该表将变为只读，这使得即使在数据仓库中使用列索引，每次更新数据都变成非常痛苦的事。SQL Server 2014中的可更新聚集列索引则解决了该问题。

### 可更新聚集列存储索引？

    聚集列存储索引的概念可以类比于传统的行存储，聚集索引既是数据本身，列存储的概念也是同样。将数据按照列存储而不是行存储则提供了诸多好处，

* 首先对于大量聚合、扫描、分组等数据仓库类查询仅仅需要读取选择的列，对于需要Join多个表的星型结构等场景性能提升尤其明显
* 其次是列索引可以更新，并且每个表中只需要一个（这是优点也是缺点，因为无法再建非聚集索引）聚集列索引即可，大大节省了空间
* 列索引由于是按列存储，同一列中数据类型是一样的，因此可以更加容易的实现更高的压缩比率
* 列存储的表会占用更少的存储空间，因此存在更少的IO

### 那么列存储索引有什么弊端呢？

    行存储对于OLTP操作十分适合，因为每个聚集索引键可以标识某一行，该行存储在物理磁盘上也连续，因此可以利用Seek操作完成大量选择性非常高的查询，而列存储索引同一行的每一列并不在物理上联系，并且列存储聚集索引中并没有“主键”的概念，因此并不存在SEEK操作，如果大量OLTP类的查询，性能将会出现问题。

    列存储索引只支持Scan操作，如图1所示。

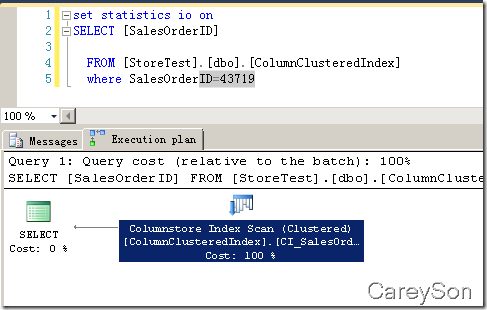
[](http://images.cnitblog.com/blog/35368/201401/231102160224.png)

图1.列存储索引只支持Scan操作

### 那么列索引是如何存储呢？

    列索引存储可以望文生义，就是按列存储。这个过程可以分为3个阶段，首先将一堆行分组，这就是所谓的“行组”，分组完成后，再按列切分，最后将列压缩，如图2所示。

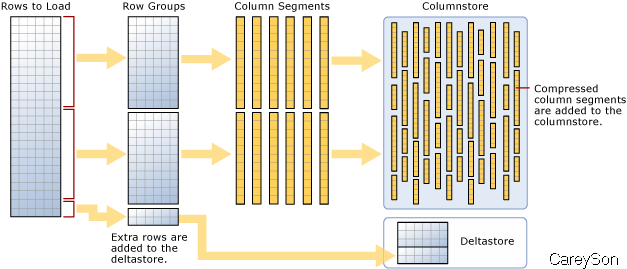
[](http://images.cnitblog.com/blog/35368/201401/231102169763.gif)

图2.列存储的过程

    我们注意到其中有一部分不够分组的，那么就直接让这部分数据以传统行存储的形式老实呆着吧，这就是所谓的Deltastore，等数据增长到可以分组时再进行分组，目前SQL Server 2014认为10W以下的数据都不够分组。

    上述列存储的两部分我们可以通过2014新引入的DMV进行观测，如图3所示。在图3中，我们队目前已经存在31465行的聚集列索引插入了1000行新的数据，则SQL Server认为这部分数据不满10W行，因此以Deltastore的方式存在。

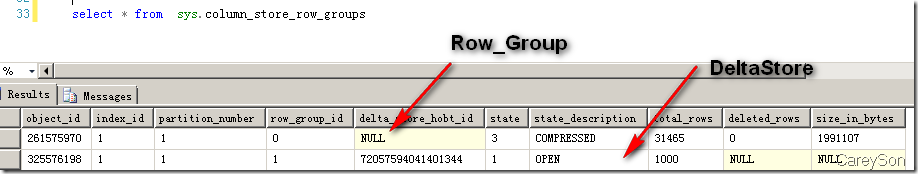
[](http://images.cnitblog.com/blog/35368/201401/231102178192.png)

图3.压缩后的列和Deltastore

     当我们再插入1000数据时，可以观察到DeltaStore中的数据又增加了1000，达到2000，但依然存在DeltaStore中。如图4所示。

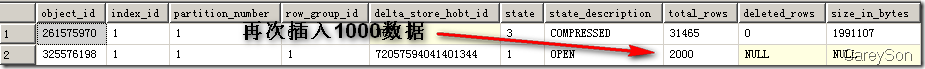
[](http://images.cnitblog.com/blog/35368/201401/231102185546.png)

图4.再次插入的数据依然在DeltaStore中

      那么我插入大量的行进行观测，会发现，大批量的数据依然以DeltaStore的方式存储，如图5。

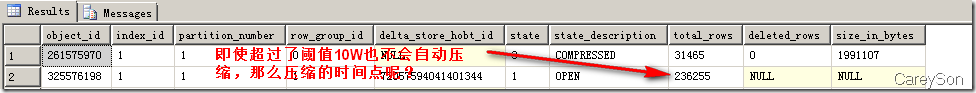
[](http://images.cnitblog.com/blog/35368/201401/231102196165.png)

图5.插入大量数据后也无法将数据压缩

    那么究竟何时会压缩这些数据呢，根据BOL的说法：<http://msdn.microsoft.com/en-us/library/dn223749(v=sql.120).aspx>，会有一个后台的线程定期检测，此外当重建或整理索引时也可以自动归档，如图6所示。

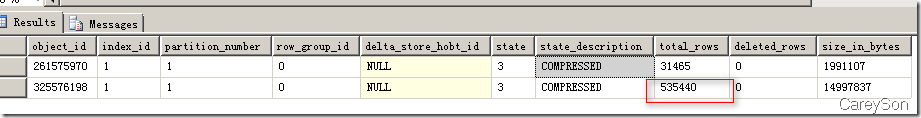
[](http://images.cnitblog.com/blog/35368/201401/231102204445.png)

图6.重建索引后归档列存储索引

### 空间占用比较

    可更新列存储聚集索引的压缩比率是最高的，因为同一列往往是同一类数据，因此这类数据有更好的压缩比。现在我纯粹的从传统聚集索引、页压缩、行压缩、列存储索引所占用的空间进行比较，当然，如果我们把传统表的非聚集索引算上，那么行存储表将会需要更多的空间。我们用3W多条数据进行简单比对，如图7所示。

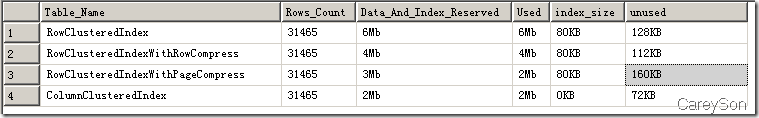
[](http://images.cnitblog.com/blog/35368/201401/231102214133.png)

图7.不同存储占用空间

    图7的示例数据很少，但依然可以看到，列存储比即使没有非聚集索引的行存储，占用空间也几乎少了2/3，提升不可谓不巨大。

### 性能简单比较

    首先，先按照列存储，我们选择所有的列，对于行存储来说需要选择整个表才能把一列数据全部读取出来，但列存储则只需要读取被选择的列，因此如果只选择特定的列的话，列存储性能提升巨大，如图8所示。

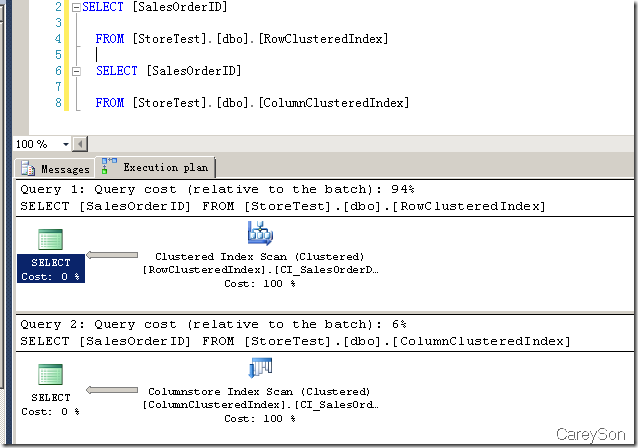
[](http://images.cnitblog.com/blog/35368/201401/231102222418.png)

图8.可更新列存储聚集索引性能提升巨大

    但反之，我们尝试一个典型的OLTP操作，只选择一行的所有列，则会和图8的结果大相庭径了。如图9所示。

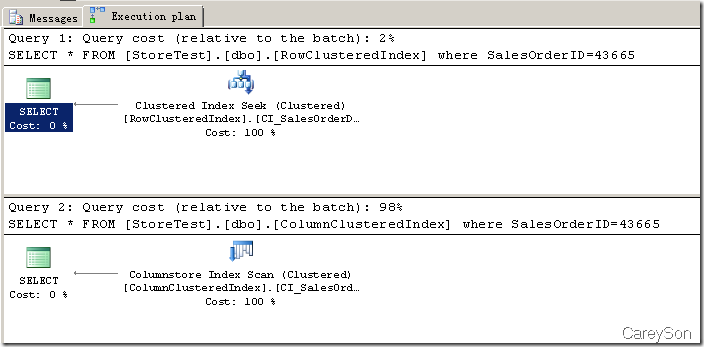
[](http://images.cnitblog.com/blog/35368/201401/231102230070.png)

图9.对于OLTP操作来说，列存储索引非常乏力

### 小结

    本文阐述了SQL Server 2014中可更新列存储索引的原理，概念，适用场景、空间使用情况，并举出两个OLAP和OLTP极端的例子进行性能比对。列存储索引对于数据仓库和类OLAP查询来说是一个巨大的飞跃。