**<http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/>**

**SQL Server Hash Partitioning**

May 31, 2010 [Andrew Hogg](http://sqlfascination.com/author/andrewhogg/) [Leave a comment](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#respond) [Go to comments](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comments)

i

2 Votes

Quantcast

It’s been a while since the last post, primarily due to changing jobs and now spending most of my time on Oracle – although it is always good to see the other side of the coin and see what it has to offer, but I won’t be abandoning SQL Server, that is for certain.

One of the more interesting features to me in Oracle is hash partitioning – the ability to create a partition across a defined number of partitions, and then arbitrarily decide which partition the data will go in based on a hashing function. Why would that be handy? SQL Server partitioning is in effect a range partition, in which you define the dividing points on the number line / alphabet – which suits partitions based on a defined number range or date range, but does not suit partitioning of other types such as a GUID.

The merits of such a partition could be debated, since with a decent index in place the benefits of the partition elimination within the query plan can be limited. Regardless of those merits (and I am pretty sure it is not going to be performant at scale, however could SQL Server implement Hash Partitioning? On a side note, this could be considered semi-dynamic partitioning in that the partition is able to cope with additional data outside of the expected range, due to the hash function.

I’ve seen a few articles try and perform hash partitioning by pre-processing the insert statement, prior to insertion into the database, but what about something a bit more native?

To start with, we need to create a partition function and partition schema to support this endeavour, both are pretty easy to construct.

CREATE PARTITION FUNCTION [myPartitionFunction] (int)

AS RANGE LEFT FOR VALUES (100,200,300,400,500,600,700,800,900)

CREATE PARTITION SCHEME [myPartitionScheme] AS

PARTITION [myPartitionFunction] ALL TO ([FG1])

I’ve set up the partition scheme to assign all of the partitions to FG1, just for convenience, it could easily be set to multiple filegroups, and instead of 9 partitions, this could be constructed with 999 partitions.

There are a variety of hashing algorithms and functions, but given the range covered by the partition function, I have chosen to use a very simple modulo on the converted binary of the unique identifier. The only trick here is that we must create the function with schema binding, otherwise SQL will refuse to use the function later on when we persist the column and partition on it.

CREATE FUNCTION GuidHash (@guid\_value uniqueidentifier) RETURNS int

WITH SCHEMABINDING AS

BEGIN

 RETURN abs(convert(bigint,convert(varbinary,@guid\_value))) % 999

END

That is a pretty simple hashing function, but the point is to demonstrate is can be done, not to implement the best hashing algorithm that will give the most even distribution etc. The next step is to create the table, with the persisted column defined using the GuidHash function. If the function is not schema bound, you get an error thrown at this stage.

CREATE TABLE MyTable(  MyID UniqueIdentifier not null,

SomeField Char(200),

PartitionID as dbo.GuidHash(MyId) PERSISTED

)

ON myPartitionScheme(PartitionID)

The surprise here is that is accepts the table creation definition - since when would you expect a partitioned table’s column to be a computed column?

Time to put an index on the table, given the data is indexed off a unique identifier, it would not be unusual to place a non-clustered index on the table and to use index-alignment, e.g. place it on the same partitioning scheme.

CREATE NONCLUSTERED INDEX [ix\_id] ON [dbo].[MyTable] (

[MyID] ASC,

[PartitionID] ASC

) ON [myPartitionScheme]([PartitionID])

Populate the table with some test data:

DECLARE @guid uniqueidentifier

SET @guid = newid()

INSERT INTO mytable (myid, somefield) VALUES (@guid, 'some text')

go 10000

So what happens when we select a single row from our data, for convienience I looked up a value in the table and grabbed the GUID – comparing the two queries side by side,

SELECT \* FROM mytable WHERE myid = 'D41CA3AC-06D1-4ACC-ABCA-E67A18245596'

SELECT \* FROM mytable WHERE (partitionid = dbo.guidhash ('D41CA3AC-06D1-4ACC-ABCA-E67A18245596')

and myid = 'D41CA3AC-06D1-4ACC-ABCA-E67A18245596')

The comparison is interesting, in percentage terms, it was 85% to 15% for the batch, the IO Statistics reads:

First Query : Scan count 10, Logical Reads 21

Second Query : Scan count 1, Logical Reads 3

So the hashing has clearly made the query faster – but that just means it was faster than the query that didn’t use the partition key which shouldn’t be too surprising – partition elimination vs checking every partition should win, so all it shows is that partition elimination is occurring. So how does it stack up against a normal table, e.g. have we gained anything? To test we need to put 10k rows into the same table, minus the computed column, index it and perform the same kind of select – all easy stuff so I will not write the code here, the results of a select from a normal table?

Normal Table Query : Scan Count 1, Logical Reads 3

And when run side by side, the SSMS window reports a 50% split of work between the two queries within the batch – which is not surprising given the IO costs were listed as the same – so where is the catch? There is no such thing as a free lunch, and the additional cost here is the CPU to generate the PartitionID value for the hashed GUID, but as a technique to partition based on a GUID, it has some merits.

One final thing that I did notice is that under Simple Parameterization, the GuidHash based query does not parameterize, which would start having detrimental effects on the query cache, once the database was placed under forced parameterization, then the query did parameterize appropriately – so you either want forced parameterization on or to use stored procedures – I would vote for the later there.

As a technique is has some merits, but you have to remember to manually include the PartitionID column within each query and run it through the hashing function – which is not ideal, but manageable.

* **Share this:**
* [StumbleUpon](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?share=stumbleupon&nb=1)
* [Digg](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?share=digg&nb=1)
* [Reddit](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?share=reddit&nb=1)
* [Email](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?share=email&nb=1)
* [Facebook](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?share=facebook&nb=1)

**Possibly related posts: (automatically generated)**

* [**Interval Partitioning in SQL Server 2008**](http://www.sqlfascination.com/2010/09/12/interval-partitioning-in-sql-server-2008/)
* [Partition Tables](http://sqlreference.wordpress.com/2010/05/30/partition-tables/)
* [Oracle PLSQL – Table Partition](http://mangaltripathi.wordpress.com/2010/03/24/oracle-table-partition/)

Categories: [SQL Server](http://en.wordpress.com/tag/sql-server/) Tags: [Dynamic Partitioning](http://en.wordpress.com/tag/dynamic-partitioning/), [Query Parameterisation](http://en.wordpress.com/tag/query-parameterisation/), [SQL Server 2005](http://en.wordpress.com/tag/sql-server-2005/), [SQL Server 2008](http://en.wordpress.com/tag/sql-server-2008/), [Table Partitioning](http://en.wordpress.com/tag/table-partitioning/)

[Like](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?like=1&_wpnonce=a7d425906c)

Be the first to like this post.

[Comments (10)](javascript:void(0);) [Trackbacks (0)](javascript:void(0);) [Leave a comment](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#respond) [Trackback](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/trackback/)

1. http://0.gravatar.com/avatar/2a76fade63f2a4ea69d49e829d549e96?s=32&d=identicon&r=G

developer

September 13, 2010 at 11:43 am | [#1](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-76)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=76#respond) | [Quote](javascript:void(0);)

Hi..  
Do u know whether SQL server 2008 supports composite partitioning??  
Thanks in advance..:)

* + http://0.gravatar.com/avatar/8215e290861f1c44a457d26c4f24af70?s=32&d=identicon&r=G

[Andrew Hogg](http://andrewhogg.wordpress.com)

September 13, 2010 at 2:13 pm | [#2](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-77)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=77#respond) | [Quote](javascript:void(0);)

SQL Server 2008 does not natively support composite partitioning, which Oracle does with the partition / sub-partition feature. Oracle supports 9 different composites I believe. It is not equivalent, but you can certainly place a view across multiple partitioned tables, physically creating multiple partitioned tables to represent the sub-partitioning and the view sitting across them all to represent the top level partition. The problem will come in the maintenance and the limit of 1000 partitions that is still in place.

1. http://0.gravatar.com/avatar/2a76fade63f2a4ea69d49e829d549e96?s=32&d=identicon&r=G

developer

September 29, 2010 at 10:14 am | [#3](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-82)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=82#respond) | [Quote](javascript:void(0);)

Hi,  
How does partitioning actually improve query performance..? Is it enough that we include the partitioning column in the WHERE clause or is there something more to be done to improve query performance on a partitioned table?

* + http://0.gravatar.com/avatar/8215e290861f1c44a457d26c4f24af70?s=32&d=identicon&r=G

[Andrew Hogg](http://andrewhogg.wordpress.com)

September 29, 2010 at 12:17 pm | [#4](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-83)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=83#respond) | [Quote](javascript:void(0);)

With the partition field being used within the where clause, partition elimination is at least possible – which can provide a performance benefit. If you fail to use the partition key as part of the query criteria, then all of the partitions will have to be searched to check for matching data. The real key to the performance gain from partitioning is the cheap cost of ageing out older data.  
In terms of improving the performance, partitioning is not a silver bullet, you still have to be mindful of your storage, query plans and indexing strategies.

1. http://0.gravatar.com/avatar/2a76fade63f2a4ea69d49e829d549e96?s=32&d=identicon&r=G

developer

September 30, 2010 at 4:53 am | [#5](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-84)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=84#respond) | [Quote](javascript:void(0);)

Thanks!! And I would like to add one more question. How much does creating an index using the same Partition Scheme and Partition Key, help?

* + http://0.gravatar.com/avatar/8215e290861f1c44a457d26c4f24af70?s=32&d=identicon&r=G

[Andrew Hogg](http://andrewhogg.wordpress.com)

September 30, 2010 at 11:17 am | [#6](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-85)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=85#respond) | [Quote](javascript:void(0);)

If they do not use the same partition scheme (what I would call, partition aligned) then the rolling window effect is not possible, whilst the data could be rolled out, the index would still need a massive deletion of entries to be correct, so it will refuse. To make the index partition aligned the partition key must be somewhere within it, but does not have to be the first field, if anything placing it as the first is likely to create a less efficient index. In most cases I would expect to see the partitioned key field as the last field within the partition aligned index.

1. http://0.gravatar.com/avatar/2a76fade63f2a4ea69d49e829d549e96?s=32&d=identicon&r=G

developer

October 1, 2010 at 12:01 pm | [#7](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-86)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=86#respond) | [Quote](javascript:void(0);)

I’m trying to implement Partition by modulus and right now facing a peculiar thing!! I’m not able to change “Allow nulls” property of any column in a partitioned table without automatically converting the computed partition id column as nullable. And can’t modify this column later!! why is this so..? Right now my tables does not contain any data..

* + http://0.gravatar.com/avatar/8215e290861f1c44a457d26c4f24af70?s=32&d=identicon&r=G

[Andrew Hogg](http://andrewhogg.wordpress.com)

October 1, 2010 at 2:41 pm | [#8](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-88)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=88#respond) | [Quote](javascript:void(0);)

If I am guessing your situation correctly, the hash function is schema bound and attached to the computed column, and you are now attempting to change the schema on a schema bound object – which is not permitted. The point of the schemabound function is that it can rely on the underlying structure of the table will not change.

1. http://0.gravatar.com/avatar/2a76fade63f2a4ea69d49e829d549e96?s=32&d=identicon&r=G

developer

October 2, 2010 at 7:39 am | [#9](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-90)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=90#respond) | [Quote](javascript:void(0);)

So does that mean the underlying structure of a hash-partitioned table cannot be modified later, as it has a schema bound function and an attached computed column?

What I observe is that other columns can be modified at the cost making the computed column nullable .Is there any possible issues that might arise from leaving the computed column as nullable? (It will be only in the schema,in reality there will be data inserted to this column for sure!)  
Hope its not going to affect partition elimination of tenant based horizontally partitioned tables in any way.

* + http://0.gravatar.com/avatar/8215e290861f1c44a457d26c4f24af70?s=32&d=identicon&r=G

[Andrew Hogg](http://andrewhogg.wordpress.com)

October 2, 2010 at 5:35 pm | [#10](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/#comment-91)

[Reply](http://sqlfascination.com/2010/05/31/sql-server-hash-partitioning/?replytocom=91#respond) | [Quote](javascript:void(0);)

I’ll send you an email so you can send me the code to look at, will be easier to diagnose.