Effective Index Partitioning & Compression Strategies

Neil Hambly
Founder | SQLnextSteps Limited
Who Am I?

- SQLnextSteps Limited Neil.Hambly@Hotmail.co.uk
- DBA – 15+ Years, SQL Server 6.5 - 2014
- Held # of DB roles @ many leading organizations
- Regular International PASS Presenter @ SQL Conferences, User-Groups & Webinars
  100+ presentations in last 4 years
- SQL London PASS Chapter Leader, UK Events Organizer
- Regular @ UK & International community events
- Twitter: @Neil_Hambly

Contributing Author
Agenda

❖ Talk about the 2 features – Compression & Partitioning
❖ Using **Data Compression** in SQL Server 2005+.
   ❖ Compression
     ❖ Editions, Data types
     ❖ Vardecimal Compression (2005 SP2)
     ❖ Page | Row Compression (2008+)
     ❖ Unicode Compression
     ❖ Estimate space & resources
   ❖ Partitioning
   ❖ Combining with other features
   ❖ Demo’s
     ❖ Partition-Info Script
     ❖ Finding objects for compression
❖ Q&A + Further links
Data Compression was introduced in SQL 2005 (SP2) via vardecimal storage. This was superseded in SQL Server in 2008 Version (Row & Page level).

Compression feature is only available in Enterprise and Developer SKU’s.

Note: NULL and 0 values across all data types are optimized to take 0 bytes in addition to 4 bits/per Column.

Data Type / benefits with Data Compression

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes – depends on data</td>
<td>No or little benefit (Normally)</td>
</tr>
<tr>
<td>bit / smallint / int / bigint</td>
<td>tinyint / uniqueidentifier</td>
</tr>
<tr>
<td>decimal / numeric</td>
<td></td>
</tr>
<tr>
<td>datetime / datetime2 / datetimeoffset</td>
<td>smalldatetime / date / time</td>
</tr>
<tr>
<td>char / nchar / nvarchar</td>
<td>varchar / text / ntext / image</td>
</tr>
<tr>
<td>binary / timestamp / Rowversion</td>
<td>varbinary</td>
</tr>
<tr>
<td>money / smallmoney</td>
<td></td>
</tr>
<tr>
<td>float / real</td>
<td>sql_variant / FileStream</td>
</tr>
<tr>
<td></td>
<td>cursor / table / xml</td>
</tr>
</tbody>
</table>
The **vardecimal** works the same way as the **varchar** works for efficiently storing alphanumeric data.

**HOW TO TURN ON THE VARDECIMAL OPTION**

**Step 1:**
```
exec sp_db_vardecimal_storage_format '<dbname>', 'ON'
```
This option will enable the vardecimal storage on the specified database.

**Step 2:**
```
exec sp_tableoption '<table>', 'vardecimal storage format', 1
```
This option will turn on the vardecimal storage for the existing table.

**HOW TO ESTIMATE WHETHER IT IS WORTH TURNING ON THIS OPTION**
```
sys.sp_estimated_rowsize_reduction_for_vardecimal '<table>'
```

<table>
<thead>
<tr>
<th>Column precision</th>
<th>Original fixed decimal size (bytes)</th>
<th>Maximum vardecimal data area (bytes)</th>
<th>Overhead to store offset (bytes)</th>
<th>Maximum vardecimal storage used (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>4-6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>7-9</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>10-12</td>
<td>9</td>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>13-15</td>
<td>9</td>
<td>8</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>16-18</td>
<td>9</td>
<td>9</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>19</td>
<td>9</td>
<td>10</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>20-21</td>
<td>13</td>
<td>10</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>22-24</td>
<td>13</td>
<td>11</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>25-27</td>
<td>13</td>
<td>13</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>28</td>
<td>13</td>
<td>14</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>29-30</td>
<td>17</td>
<td>14</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>31-33</td>
<td>17</td>
<td>15</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>34-36</td>
<td>17</td>
<td>16</td>
<td>2</td>
<td>18</td>
</tr>
<tr>
<td>37-38</td>
<td>17</td>
<td>18</td>
<td>2</td>
<td>20</td>
</tr>
</tbody>
</table>
Since 2008 version, we have new Compression algorithms available

Compressing the leaf level of tables and indexes with page compression consists of three operations, these are done in the following order:

1. Row compression
2. Prefix compression
3. Dictionary compression

Note: When using page compression, the non-leaf-level pages of indexes are only row compressed
Unicode Compression

Space savings that can be achieved for different locales (15-50%)

<table>
<thead>
<tr>
<th>Locale</th>
<th>Compression %</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>50%</td>
</tr>
<tr>
<td>German</td>
<td>50%</td>
</tr>
<tr>
<td>Hindi</td>
<td>50%</td>
</tr>
<tr>
<td>Turkish</td>
<td>48%</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>39%</td>
</tr>
<tr>
<td>Japanese</td>
<td>15%</td>
</tr>
</tbody>
</table>

Standard Compression Scheme for Unicode (SCSU) algorithm to compress Unicode values that are stored in row or page compressed objects.

Unicode compression supports the fixed-length nchar(n) and nvarchar(n) data types.

Data values that are stored off row or in nvarchar(max) columns are not compressed.

SCSU can also switch to UTF-16 internally to handle non-alphabetic languages.
WITH CTE_Partitions
( [ObjectID],[Table_Name],[Index_Name],[Partition],[Index_ID],[Index_Type],[Percent_Update],[Percent_Scan],[Compression_Level],[IN_ROW_DATA],[ROW_OVERFLOW_DATA],[LOB_DATA] )
AS
SELECT o.object_id AS [Objectid] ,o.name AS [Table_Name] ,i.name AS [Index_Name] ,ios.partition_number AS [Partition] ,i.index_id AS [Index_ID] ,i.type_desc AS [Index_Type] ,ios.leaf_update_count * 100.0 / (ios.range_scan_count + ios.leaf_insert_count + ios.leaf_delete_count + ios.leaf_update_count + ios.leaf_page_merge_count + ios.singleton_lookup_count) AS [Percent_Update] ,ios.range_scan_count* 100.0 / (ios.range_scan_count + ios.leaf_insert_count + ios.leaf_delete_count + ios.leaf_update_count + ios.leaf_page_merge_count + ios.singleton_lookup_count) AS [Percent_Scan] ,p.data_compression_desc AS [Compression_Level] ,ps.in_row_used_page_count AS [IN_ROW_DATA] ,ps.row_overflow_used_page_count AS [ROW_OVERFLOW_DATA] ,ps.lob_used_page_count AS [LOB_DATA] FROM sys.dm_db_partition_stats ps JOIN sys.partitions p ON ps.partition_id= p.partition_id JOIN sys.indexes i ON p.index_id = i.index_id AND p.object_id = i.object_id JOIN sys.dm_db_index_operational_stats(db_id(),NULL, NULL, NULL) ios ON i.object_id = ios.object_id AND i.index_id = ios.index_id JOIN sys.objects o ON o.object_id= ios.object_id WHERE( ios.range_scan_count + ios.leaf_insert_count + ios.leaf_delete_count + ios.leaf_update_count + ios.leaf_page_merge_count + ios.singleton_lookup_count) != 0 AND objectproperty(i.object_id,'IsUserTable') = 1
)
SELECT * FROM CTE_Partitions
ORDER BY [IN_ROW_DATA] DESC,[Index_Name] ASC, [Percent_Update] ASC
To determine space for each compression version, we can use the system procedure `sp_estimate_data_compression_savings`.

**Example**

```sql
EXEC sp_estimate_data_compression_savings
    'Sales', -- [@schema_name = ] 'schema_name'
    'SalesOrderDetailLoad', -- [@object_name = ] 'object_name'
    NULL, -- [@index_id = ] index_id
    NULL, -- [@partition_number = ] partition_number
    'ROW' -- [@data_compression = ] 'data_compression'
;
```

Tip: you can use `sp_estimate_data_compression_savings` to gauge defragmentation savings, just run with the same value for `@data_compression` as exists already.
Guide for Resource Requirements performing Data Compression

Table shows a summary of workspace, CPU, and I/O requirements for compressing a clustered index as compared to rebuilding the same uncompressed index. Measurements used:

- X = number of pages before compression (or rebuild)
- P = number of pages after compression (P < X)
- Y = number of new or updated pages (by a concurrent application, applies only to the ONLINE case)
- M = size of the mapping index (estimate based on guidelines in the TEMPDB Capacity Planning white paper)
- C = the CPU time taken to rebuild the uncompressed index

Workspace, CPU, and I/O summary for compressing a clustered index

<table>
<thead>
<tr>
<th></th>
<th>Workspace</th>
<th>I/O</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TEMPDB</td>
<td>UserDB</td>
<td></td>
</tr>
<tr>
<td>Rebuild</td>
<td>0</td>
<td>X</td>
<td>~0</td>
</tr>
<tr>
<td>Compress</td>
<td>0</td>
<td>P</td>
<td>~0</td>
</tr>
</tbody>
</table>

OFFLINE with BULK_LOGGED or SIMPLE Recovery Model

<table>
<thead>
<tr>
<th></th>
<th>TEMPDB</th>
<th>UserDB Tran Log</th>
<th>TEMPDB</th>
<th>UserDB</th>
<th>UserDB Tran Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rebuild</td>
<td>0</td>
<td>~0</td>
<td>0</td>
<td>X+2X</td>
<td>~0</td>
</tr>
<tr>
<td>Compress</td>
<td>0</td>
<td>~0</td>
<td>0</td>
<td>X+2P</td>
<td>~0</td>
</tr>
</tbody>
</table>

OFFLINE with FULL Recovery Model

<table>
<thead>
<tr>
<th></th>
<th>TEMPDB</th>
<th>UserDB</th>
<th>UserDB Tran Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rebuild</td>
<td>0</td>
<td>X</td>
<td>~C</td>
</tr>
<tr>
<td>Compress</td>
<td>0</td>
<td>P</td>
<td>1.5C to 5C</td>
</tr>
</tbody>
</table>

ONLINE with FULL Recovery Model

<table>
<thead>
<tr>
<th></th>
<th>TEMPDB</th>
<th>UserDB</th>
<th>UserDB Tran Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rebuild</td>
<td>M+Y</td>
<td>X+Y</td>
<td>2X+Y</td>
</tr>
<tr>
<td>Compress</td>
<td>M+Y</td>
<td>P+Y</td>
<td>3C to 10C</td>
</tr>
</tbody>
</table>
Partitioning....

Each partition can be compressed @ different level if desired

An example Table with partition on Date @ monthly boundaries
A year consists of 12 partitions (EG Jan 2012 – Dec 2012)

```
Alter Table dbo.OurTableName
Rebuild Partition = All
With
  (   
    Data_Compression = Page On Partitions(1 to 8) 
    , Data_Compression = Row  On Partitions(9 to 11) 
    , Data_Compression = None On Partitions(12)
  );
```

With the above the most recent month is NO-compression
3 months before that are compressed with ROW-compression
The 1st 8 months before those have been PAGE-compression

Using a ‘Sliding Window’ strategy, so as Data is Aged.. It has increased compression level it less frequently accessed, so a higher compression CPU can be allowed to save more space
Partitioning Actions....

Splitting a range
When partitions are split by using the ALTER PARTITION statement, both partitions inherit the data compression attribute of the original partition.

Merging a range
When two partitions are merged, the resultant partition inherits the data compression attribute of the destination partition.

Switching partitions
To switch a partition, the data compression property of the partition must match the compression property of the table.
Some other Thoughts....

• **Compression on a per partition level**
  • Rebuild each partition separately {save space}
  • Help keep fragmentation to min level {more granular option}
  • FileGroup backups (Differential FileGroup backups)

• **Determine an effective fill-factors**
  • Based it on the KeySize
  • Use Compression levels to create better page density
  • Based on the activity levels {Non|Row|Page}

• **Consider adding Filtered Multi-column stats**
  • <user created Statistics> for each partition
  • Use the PF to ensure the Filter is just for 1 partition

• TEST...Test... and then Test again
Demos....

- Some think it is now time for our Demo’s
- Here is one Partitioning/Compression strategy I prepared earlier this week
How Compression Affects Other SQL Server Components

**Bulk import and export operations**
When data is exported, even in native format, the data is output in the uncompressed row format. This can cause the size of exported data file to be significantly larger than the source data.

When data is imported, if the target table has been enabled for compression, the data is converted by the storage engine into compressed row format. This can cause increased CPU usage compared to when data is imported into an uncompressed table.

When data is bulk imported into a heap with page compression, the bulk import operation will try to compress the data with page compression when the data is inserted.

**Compression does not affect backup and restore.**
Compression does not affect log shipping.

**Enabling compression can cause query plans to change**
Because the data is stored using a different number of pages and number of rows per page.

**Data compression is supported by SSMS** through the Data Compression Wizard.
Further Links to Resources on Data Compression in SQL Server

**Whitepapers**
Data Compression: Strategy, Capacity Planning and Best Practices (SQL 2008)
Vardecimal Whitepaper (SQL2005)
http://download.microsoft.com/download/2/7/c/27cd7357-2649-4035-84af-e9c47df4329c/vardecimalstorage.docx
SQL Server I/O Reliability Program

**WIKI**

**Blogs**
MS SQL Server Storage Engine
http://blogs.msdn.com/b/sqlserverstorageengine/archive/tags/data+compression/

**KB Articles**
Microsoft SQL Server Database Engine Input/Output Requirements KB 967576

**Confio**
http://www.confio.com/sql-server-performance-resources/primary-key-vs-clustered-index/
LEARN FROM INTERNATIONAL EXPERTS
- Sept 19th: 6 x Precons
- Sept 20th: 6 x Tracks – BI, DEV, DBA, 2014, Cloud

STAY
Hilton Hotel Charlemont – rooms from €159 B&B

REGISTER
http://www.sqlsaturday.com/310/

20th Sept
DUBLIN

PARTY
After party and entertainment during the event
PRE-CONFERENCE DAY
6 FULL-DAY TRAINING SESSIONS

FRIDAY 19th SEPT. 14

Storage and Virtualization for DBA
Denny Cherry (MCM, MVP, VMWare vExpert)

SQL Server 2014 Performance Tuning Day
Klaus Aschenbrenner (MCM)

Building Modern Data Warehouse Architecture
Karen Lopez (MVP)

Real World Database Configuration and Tuning
Kevin Kline (MVP, former PASS President)

SQL Server For Developers
Mladen Prajdić (MVP)

Microsoft Azure Bootcamp
Conor Cunningham (Principal Architect SQL Server Team)
Ewan Fairweather (CAT Team & Azure Product Team)

Register at www.sqlsaturday.com/310/

Location: Hilton Hotel, Charlemont Place, Dublin 2
Cost: €299 Early bird until 31st July (Full Price €399)
Contact & info: sqlsaturday310@sqlsaturday.com