Table Partitioning:
Secret Weapon for Big Data Problems

John Sterrett, Sr. Database Admin Advisor
DELL
About John
How to contact me?

http://johnsterrett.com/go/partition

Twitter: @JohnSterrett
LinkedIn: http://linkedin.com/in/johnsterrett
Agenda

Big Data starting to slow you down? Data growth putting your maintenance tasks in danger of not meeting your SLAs? Wish you could archive old data with minimal impact to your tables during the archive process or that you could eliminate most of the data in your tables when you query them? If so, it’s time you consider implementing table partitioning to help with general performance and reduce your window for completing maintenance tasks.

• Learn if Partitioning can help you manage big data.
• Understand how partitioning works.
• Learn how to maintain table partitioning with an automated sliding window.
Assumptions…

- You are not familiar with Table Partitioning and want to see how it works.
- You have Enterprise Edition
- Highly Transactional Tables with over 50 GB of data
- General Maintenance Tasks are in danger of not meeting your SLA
- Purge and/or Archive process is slowing you down.
The Big Question…

HOW CAN TABLE PARTITIONING MAKE MY LIFE EASIER?
How partitioning helps me?

• Reduce Maintenance Tasks
How partitioning helps me?

- Reduce Maintenance Tasks
- Improves Purging and/or Archiving
How partitioning helps me?

- Reduce Maintenance Tasks
- Improves Purging and/or Archiving
- Improves Performance
Big Question…

HOW DOES TABLE PARTITIONING WORK?
High Level...
High Level…

Partition Table

Partition Scheme

Partition Function
Selecting The Partition Column

Only get one column, use it wisely!
Column should be a highly used filter.
  • Review index usage statistics
  • Review Missing index statistics
  • Review Queries and Talk to developers ;-)
Column must be part of clustered index
Partition Function

```
CREATE PARTITION FUNCTION Demo1_FN (int)
    AS RANGE LEFT FOR VALUES (100, 200, 300);
```

Defines the data type used to distribute data into partitions.
Assigns boundary values to split data between partitions.
Assigns the RANGE for boundary values
Partitions = Boundary values + 1
NULL values go to left most partition
Partition Function

Partition functions are not static. They can change over time with SPLIT and MERGE statements.

Range LEFT is used by default.
Partition Function - Range

\[
\begin{align*}
\text{CREATE PARTITION FUNCTION Demo1_FN (int) as range left for values (100,200,300);} \\
\{\text{min...100}, \{101...200}, \{201...300\}, \{301...max}\} \\
\end{align*}
\]

\[
\begin{align*}
\text{CREATE PARTITION FUNCTION Demo1_FN (int) as range right for values (100,200,300);} \\
\{\text{min...99}, \{100...199}, \{200...299\}, \{300...max}\} \\
\end{align*}
\]
Partition Scheme

```
CREATE PARTITION SCHEME Demo3_Scheme
AS PARTITION Demo3_Function
TO ([FG1], [FG2], [FG3], [FG4],[FG5],[FG6],[FG7]);
```

Assigns a partition function to a partition scheme
Assigns filegroups to partitions
Partitioning

DEMO
Big Question…

HOW DOES TABLE PARTITIONING IMPROVE MY MAINTENANCE TASKS?
Improving Maintenance Tasks

Backup and restore filegroups based on business priorities.

Index Maintenance by partition

New Features in SQL 2014
• Rebuild index online by partition
• Incremental Statistics by partition *(SQL 2014 CTP 2)*
Incremental Statistics

Just added in SQL Server 2014 CTP2

From Books Online:
“When ON, the statistics are recreated as per partition statistics.”

Cannot be used for the following:
- Statistics created with indexes that are not partition-aligned with the base table
- Filtered Indexes
HOW DOES TABLE PARTITIONING IMPROVE PERFORMANCE?
Partition Elimination

Clustered Index Scan (Clustered)

Physical Operation: Clustered Index Scan
Logical Operation: Clustered Index Scan
Actual Execution Mode: Row
Estimated Execution Mode: Row
Storage: RowStore
Actual Number of Rows: 9747
Actual Number of Batches: 0
Estimated I/O Cost: 3.15223
Estimated Operator Cost: 3.35511 (100%)
Estimated Subtree Cost: 3.35511
Estimated CPU Cost: 0.202881
Number of Executions: 1
Estimated Number of Executions: 1
Estimated Number of Rows: 9960.2
Estimated Row Size: 19.8
Actual Rebinds: 0
Actual Rewinds: 0
Partitioned: True
Actual Partition Count: 1
Ordered: True
Node ID: 0

Predicate:
  - (SalesOrderHeaderEnlarged, PT).OrderDate > '2011-01-01 00:00:00.000' AND (AdventureWorks2012).Sales.
  - (SalesOrderHeaderEnlarged, PT).OrderDate <= '2011-12-31 00:00:00.000'

Object:
- [AdventureWorks2012].Sales.
  - (SalesOrderHeaderEnlarged, PT).
  - [PK_SalesOrderHeaderEnlarged, PT, SalesOrderID]
Output List:
- [AdventureWorks2012].Sales.
  - (SalesOrderHeaderEnlarged, PT).CustomerID
Seek Predicates:
- Seek Keys[1]: Prefix: Pnsk1000 = ScalarOperator(6)

Scan count 1, logical reads 230,

(9747 row(s) affected)
Table 'SalesOrderHeaderEnlarged, PT'.

Scan count 5, logical reads 30014,

(9747 row(s) affected)
Skip-Scan: Seek keys

```
<table>
<thead>
<tr>
<th>Physical Operation</th>
<th>Index Seek</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Operation</td>
<td>Index Seek</td>
</tr>
<tr>
<td>Actual Execution Mode</td>
<td>Row</td>
</tr>
<tr>
<td>Estimated Execution Mode</td>
<td>Row</td>
</tr>
<tr>
<td>Storage</td>
<td>RowStore</td>
</tr>
<tr>
<td>Actual Number of Rows</td>
<td>1148</td>
</tr>
<tr>
<td>Actual Number of Batches</td>
<td>0</td>
</tr>
<tr>
<td>Estimated I/O Cost</td>
<td>0.021875</td>
</tr>
<tr>
<td>Estimated Operator Cost</td>
<td>0.0241924 (97%)</td>
</tr>
<tr>
<td>Estimated CPU Cost</td>
<td>0.0023174</td>
</tr>
<tr>
<td>Estimated Subtree Cost</td>
<td>0.0241924</td>
</tr>
<tr>
<td>Number of Executions</td>
<td>1</td>
</tr>
<tr>
<td>Estimated Number of Executions</td>
<td>1</td>
</tr>
<tr>
<td>Estimated Row Size</td>
<td>1107.65</td>
</tr>
<tr>
<td>Actual Relinks</td>
<td>0</td>
</tr>
<tr>
<td>Actual Rewinds</td>
<td>0</td>
</tr>
<tr>
<td>Partitioned</td>
<td>True</td>
</tr>
<tr>
<td>Actual Partition Count</td>
<td>7</td>
</tr>
<tr>
<td>Ordered</td>
<td>True</td>
</tr>
<tr>
<td>Node ID</td>
<td>1</td>
</tr>
</tbody>
</table>

Object:
[AdventureWorks2012].[Sales].
[SalesOrderHeaderEnlarged_PT].
[idx_SalesOrderHeaderEnlarged_PT_CustomerID]

Output List:
[AdventureWorks2012].[Sales].
[SalesOrderHeaderEnlarged_PT].OrderDate

Seek Predicates
Seek Keys[1]: Start: PtnId1000 > = Scalar Operator(1),
               End: PtnId1000 <= Scalar Operator(7), Seek Key[2]:
               Prefix: [AdventureWorks2012].[Sales].
               [SalesOrderHeaderEnlarged_PT].CustomerID = Scalar Operator(11091))
```
Database Compression by Partition

```sql
ALTER INDEX idx_SalesOrderHeaderEnlarged_PT_rowguid
ON Sales.SalesOrderHeaderEnlarged_PT
REBUILD Partition = 6
WITH (DATA_COMPRESSION = PAGE);

ALTER INDEX idx_SalesOrderHeaderEnlarged_PT_rowguid
ON Sales.SalesOrderHeaderEnlarged_PT
REBUILD Partition = 7
WITH (DATA_COMPRESSION = ROW);
```
Partitioning

DEMO
HOW DOES PARTITIONING IMPROVE ARCHIVING AND PURGING?
Real World Example:

<table>
<thead>
<tr>
<th>runDate</th>
<th>Hours</th>
<th>Minutes</th>
<th>Seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-06-18 00:00:00.000</td>
<td>3</td>
<td>29</td>
<td>37</td>
<td>EVENT_LOG Purge Job</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>runDate</th>
<th>Hours</th>
<th>Minutes</th>
<th>Seconds</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-08-20 00:00:00.000</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>EVENT_LOG - SlidingWindow</td>
</tr>
<tr>
<td>2013-08-20 00:00:00.000</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>EVENT_LOG - SlidingWindow</td>
</tr>
<tr>
<td>2013-08-21 00:00:00.000</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>EVENT_LOG - SlidingWindow</td>
</tr>
</tbody>
</table>
Quickest way to move billions of rows

ALTER TABLE [Sales].[SalesOrderHeaderEnlarged_PT]
    SWITCH PARTITION 2 TO [Sales].[SalesOrderHeaderEnlarged_Staging];

Meta data swap is quickest way to move billions of rows assuming you can get a schema lock.
Sliding Window Goals

SPLIT and MERGE with empty partitions
Use SWITCH to do meta-data swaps
Minimize all physical data movement
Range Right - MERGE

```
ALTER PARTITION FUNCTION Demo1_FN()
MERGE RANGE (100)
```

<table>
<thead>
<tr>
<th>PartitionSchemaName</th>
<th>RangeType</th>
<th>PartitionID</th>
<th>BoundaryValue</th>
<th>FileGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo1_Scheme</td>
<td>RIGHT</td>
<td>1</td>
<td>NULL</td>
<td>FG1</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>RIGHT</td>
<td>2</td>
<td>100</td>
<td>FG2</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>RIGHT</td>
<td>3</td>
<td>200</td>
<td>FG3</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>RIGHT</td>
<td>4</td>
<td>300</td>
<td>FG4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PartitionSchemaName</th>
<th>RangeType</th>
<th>PartitionID</th>
<th>BoundaryValue</th>
<th>FileGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo1_Scheme</td>
<td>RIGHT</td>
<td>1</td>
<td>NULL</td>
<td>FG1</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>RIGHT</td>
<td>2</td>
<td>200</td>
<td>FG3</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>RIGHT</td>
<td>3</td>
<td>300</td>
<td>FG4</td>
</tr>
</tbody>
</table>
**Range Left - MERGE**

```sql
ALTER PARTITION FUNCTION Demo1_FN()
MERGE RANGE (200)
```

<table>
<thead>
<tr>
<th>PartitionSchemeName</th>
<th>RangeType</th>
<th>PartitionID</th>
<th>BoundaryValue</th>
<th>FileGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>1</td>
<td>100</td>
<td>FG1</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>2</td>
<td>200</td>
<td>FG2</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>3</td>
<td>300</td>
<td>FG3</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>4</td>
<td>NULL</td>
<td>FG4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PartitionSchemeName</th>
<th>RangeType</th>
<th>PartitionID</th>
<th>BoundaryValue</th>
<th>FileGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>1</td>
<td>100</td>
<td>FG1</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>2</td>
<td>300</td>
<td>FG3</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>3</td>
<td>NULL</td>
<td>FG4</td>
</tr>
</tbody>
</table>
Range Right - SPLIT

```
ALTER PARTITION FUNCTION Demo1_FN()
    SPLIT RANGE (350)
```
Range Left - SPLIT

```
ALTER PARTITION FUNCTION Demo1_FN()
  .SPLIT RANGE (350) |
```

<table>
<thead>
<tr>
<th>PartitionSchemeName</th>
<th>RangeType</th>
<th>PartitionID</th>
<th>BoundaryValue</th>
<th>FileGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>1</td>
<td>100</td>
<td>FG1</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>2</td>
<td>300</td>
<td>FG3</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>3</td>
<td>NULL</td>
<td>FG4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PartitionSchemeName</th>
<th>RangeType</th>
<th>PartitionID</th>
<th>BoundaryValue</th>
<th>FileGroup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>1</td>
<td>100</td>
<td>FG1</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>2</td>
<td>300</td>
<td>FG3</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>3</td>
<td>350</td>
<td>FG2</td>
</tr>
<tr>
<td>Demo1_Scheme</td>
<td>LEFT</td>
<td>4</td>
<td>NULL</td>
<td>FG4</td>
</tr>
</tbody>
</table>
Visual Sliding Window Example
Visual Sliding Window Example

<table>
<thead>
<tr>
<th>Partiton1</th>
<th>Partiton2</th>
<th>Partiton3</th>
<th>Partiton4</th>
<th>Partiton5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(EMPTY)</td>
<td>(DATA)</td>
<td>(DATA)</td>
<td>(DATA)</td>
<td>(DATA)</td>
</tr>
<tr>
<td>FG1</td>
<td>FG2</td>
<td>FG3</td>
<td>FG4</td>
<td>FG5</td>
</tr>
</tbody>
</table>
Visual Sliding Window Example

Partiton 1 (EMPTY)

Partiton 2 (DATA)

Partiton 3 (DATA)

Partiton 4 (DATA)

Partiton 5 (DATA)

FG1

FG2

FG3

FG4

FG5

Staging Table (EMPTY)
Visual Sliding Window Example
Visual Sliding Window Example
Visual Sliding Window Example

Partiton1 (EMPTY)

Partiton2 (DATA)

Partiton3 (DATA)

Partiton4 (DATA)

Staging Table (DATA)

FG1

FG3

FG4

FG5

FG2
Visual Sliding Window Example

Partiton1 (EMPTY)
FG1

Partiton2 (DATA)
FG3

Partiton3 (DATA)
FG4

Partiton4 (DATA)
FG5

NEXT USED

Staging Table (EMPTY)
FG2
Visual Sliding Window Example
Sliding Window Steps...

Create partition swap table (if it doesn’t exist)
1. Insert partition swap meta data
2. Create staging table
3. Meta-data swap (partition 2 with staging table)
4. Merge Partitions #1 and #2
5. Mark next used partition
6. Split to create new partition
7. Update processed partition swap meta data.
Partitioning
DEMO
Questions…

http://johnsterrett.com/go/partition

Twitter: @JohnSterrett
LinkedIn: http://linkedin.com/in/johnsterrett
Thank you for attending this session and the 2013 PASS Summit in Charlotte, NC