# Break it up! Table partitioning

Kirby Richter

SQL Saturday 149, Sept 29, 2012

# Kirby Richter

- Working with SQL Server since version 6.5
- Broad range of experience – Development, BI and Database Administration
- MSCD, MCDBA  (ok it's a little dated!)
- Currently DBA at DELL.
- Amateur astronomer

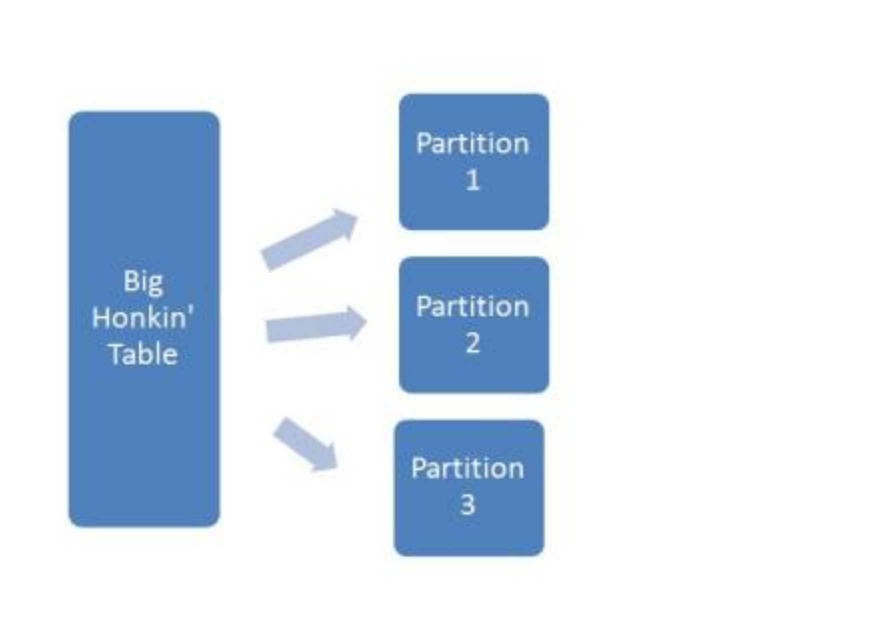
- Email: kirbyrichter@live.com
- Twitter: @sqlkirby

# Session Goals

- Basic Understanding of Table Partitioning in SQL Server
- Understand the pros and cons and why to consider partitioning
- Demonstrate a useful example of how to implement a sliding window approach to table partitioning
- Have fun!

# Overview

- Partitioning is breaking up a table into multiple segments (partitions)
- This is an example of horizontal partitioning
- Partitioning is defined by a single column  - the partition column



# History

- Prior to table partitioning we had (and still have) partitioned views
- Actual table partitioning was introduced in SQL 2005
- This was enhanced in prior versions, changed little in SQL 2012, except…
- In SQL 2008 could only have maximum of 1000 partitions on a single table, with 2012 can now have 15000!
- Several other enhancements, see resources

# Pros and Cons

| Pros | Cons |
|---|---|
| A partitioned tables appears as a normal table | Only available in Enterprise Edition |
| Management by partition, ie can rebuild indexes by partition, compress by partition, etc. | Cannot rebuild a partitioned index with online option |
| Partition Elimination and parallelism (hence potential performance improvement) | More management required by DBA’s |
| Lends well to an archiving, aging of data solution | Complexity |
| Fewer tables | |

# When to Partition

- Current very large tables (> 100 GB or so)
- New table that you foresee growing at enormous rate
- Large tables that need an aging / archiving scheme
- Concurrency (locking) issues
- Index maintenance taking too long
- Best answer as usual: it depends!

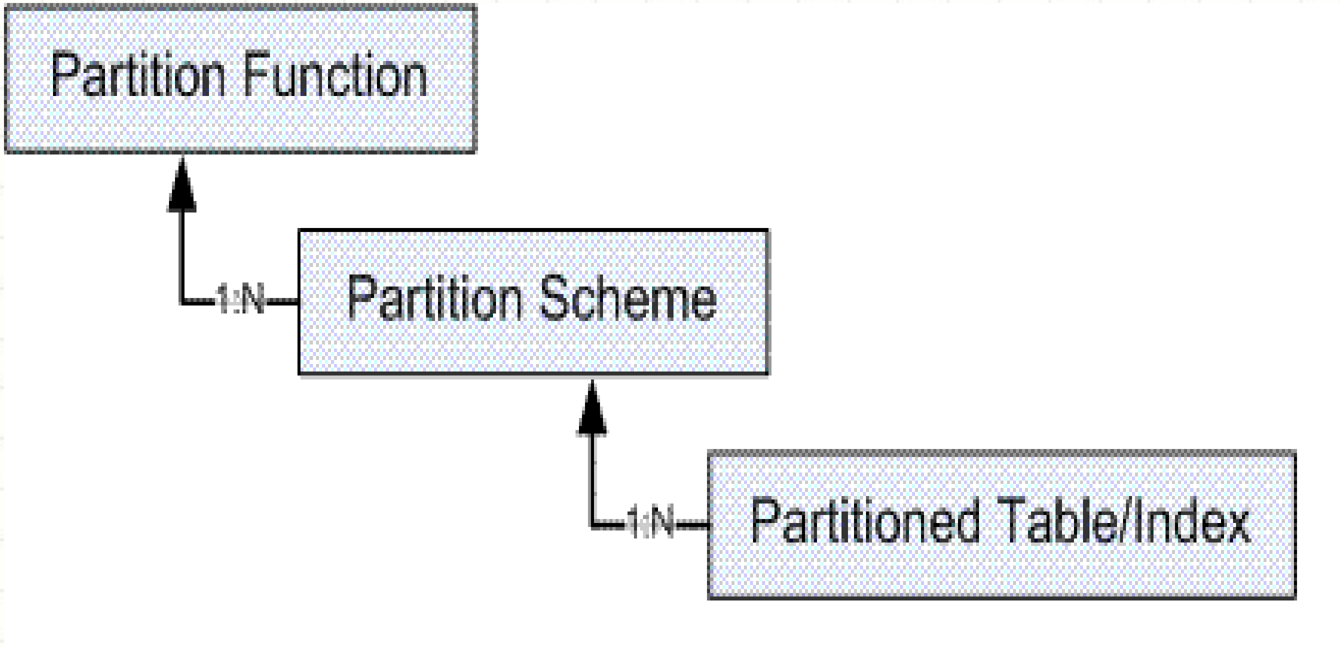
# Implementing Partitioning

- Decide which tables are candidates for partitioning
- Decide how you will partition the data (eg. months, days, customers, quantity) – know how data accessed
- Decide on a single column that lends itself well to partitioning – ie typically a not nullable date or integer type column (but not necessarily) – this is the **partition key**
- **Note: the partition key must be part of the clustered index**
- Decide on the boundaries for the partitions (eg. 201201, 201202, etc)
- Determine disk / filegroup requirements before adding the components
- Note: Storage wizard can be useful for starting / developing scripts

# Demo 1

- Setup tables
- Populate data
- Plan partitioning

# Components of Partitioning



# Partition Function

- Specifies how objects will be partitioned
- Include data type of partition key
- Specifies the boundaries for the partitions and which range to use.

Range Right

- Each boundary value will be the minimum value for its partition

Range Left

- Each boundary value will be the max value for its partition

# Example Range Right and Left

CREATE PARTITION FUNCTION [PF_RR_MyInt] (int)

AS **RANGE RIGHT** FOR VALUES (0,10,20)

- Creates 4 partitions with boundaries as follows
- {min to -1}, {0 to 9}, {10 – 19}, {20 to max}

CREATE PARTITION FUNCTION [PF_RL_MyInt] (int)

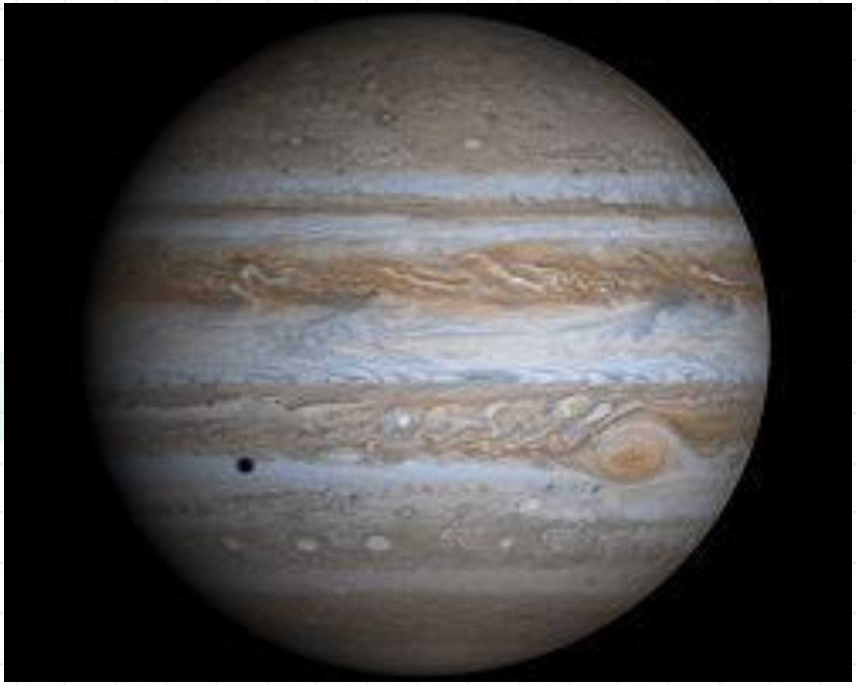
AS **RANGE LEFT** FOR VALUES (0,10,20)

- Creates 4 partitions with boundaries as follows
- {min to 0}, {1 to 10}, {11 – 20}, {21 to max}

# Partitioning Scheme

- Maps the partitions defined by the function to filegroups
- Can use keyword ALL to put all partitions on a single file group  (maybe you use a SAN that uses virtual storage)
- Tables and indexes are created on the partitioning scheme instead of a filegroup

# Demo 2



- Create partition function
- Create partition scheme
- Add tables to the scheme
- Review some DMO’s

# Archiving Data

Overall approach is a sliding window (by date) approach with steps:

1. Mark archive filegroup as read-write
2. Truncate OUT staging table
3. Switch out partition 2 of Sales table into OUT Staging table
4. Merge the last two partitions in Sales
5. Truncate IN Staging table
6. ETL the data from OUT to IN staging tables
7. Add partition to SalesArchive table
8. Mark next used Filegroup for Archive PS
9. Switch data into SalesArchive table from IN Staging table
10. Mark archive FG as read-only

# Adding/Removing Partitions

**SPLIT**

- Adds a new partition by dividing an existing one

**MERGE**

- Remove a partition by merging two neighboring partitions into one.

Considerations
Best to use SPLIT and MERGE on empty partition to avoid data movement

# Moving Data

Use the **SWITCH** option.

- Is a metadata -only operation that allows you to switch data in and out of partitions including to/ from a staging table

## Requirements

There are many, including:

1. Source and destination must have same structure
2. Source and destination must be in same filegroup
3. Clustered indexes must be the same
4. Destination must be empty
5. All indexes on partitioned tables involved must be aligned
6. If source is stand alone table then it must have check constraint matching boundaries of destination partition

# Demo 3

- Archiving of Data

# Performance Considerations

**Index Rebuilds by Partition**

- Can rebuild / reorganize individual partitions
- Can specify different compression schemes for different partitions

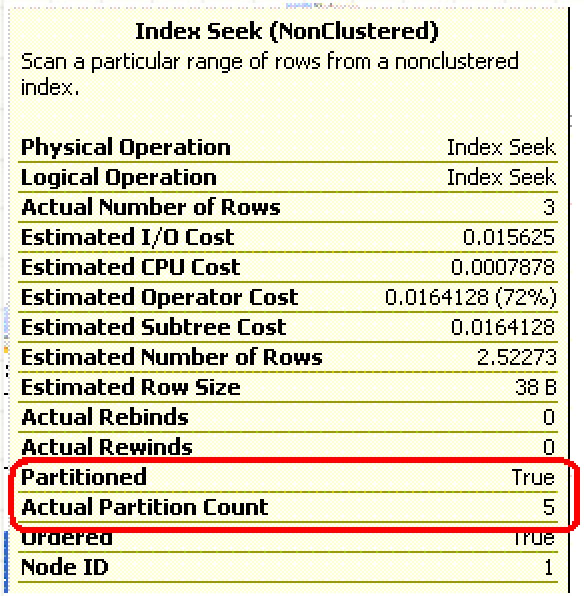
**Index Alignment**

- once you create a partitioned table you should recreate indexes on that table under the partition scheme. This allows the index to participate in several of the performance benefits including partition elimination and parallelism.
- Indexes must be partition aligned to use switch.

**Partition Elimination**

- Allows SQL Server to access only the partitions it needs to return the necessary results. In a large table with many partitions this can be a significant benefit.

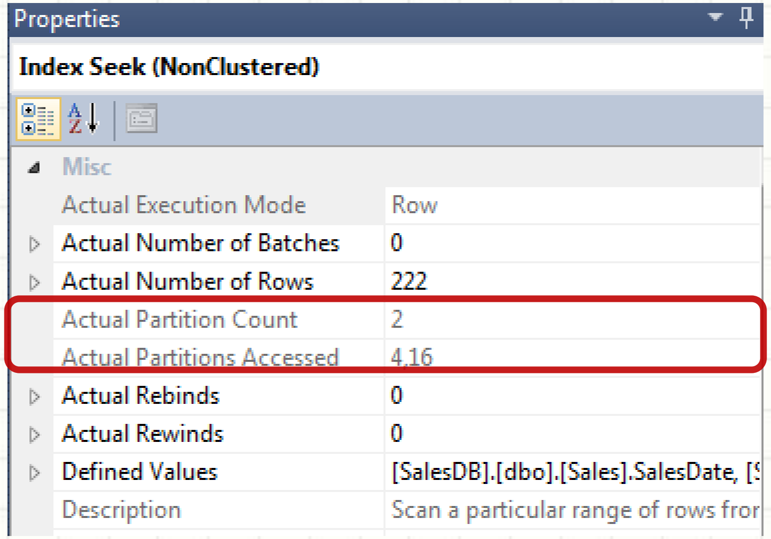
# Execution Plans



**Index Seek (NonClustered)**

Scan a particular range of rows from a nonclustered index.

| | |
|---|---|
| **Physical Operation** | Index Seek |
| **Logical Operation** | Index Seek |
| **Actual Number of Rows** | 3 |
| **Estimated I/O Cost** | 0.015625 |
| **Estimated CPU Cost** | 0.0007878 |
| **Estimated Operator Cost** | 0.0164128 (72%) |
| **Estimated Subtree Cost** | 0.0164128 |
| **Estimated Number of Rows** | 2.52273 |
| **Estimated Row Size** | 38 B |
| **Actual Rebinds** | 0 |
| **Actual Rewinds** | 0 |
| **Partitioned** | True |
| **Actual Partition Count** | 5 |
| **Ordered** | True |
| **Node ID** | 1 |



**Index Seek (NonClustered)**

Misc

| | |
|---|---|
| Actual Execution Mode | Row |
| Actual Number of Batches | 0 |
| Actual Number of Rows | 222 |
| Actual Partition Count | 2 |
| Actual Partitions Accessed | 4,16 |
| Actual Rebinds | 0 |
| Actual Rewinds | 0 |
| Defined Values | [SalesDB].[dbo].[Sales].SalesDate, [S |
| Description | Scan a particular range of rows fror |

```
<RunTimePartitionSummary>
            <PartitionsAccessed PartitionCount="2">
              <PartitionRange Start="4" End="4" />
              <PartitionRange Start="16" End="16" />
            </PartitionsAccessed>
 </RunTimePartitionSummary>
```

# Demo 4

- Rebuild Indexes
- Index Alignment
- Partition Elimination

# Summary

- Table partitioning can be a useful tool when dealing with large tables

- We now (hopefully?) understand the basics of table partitioning

- Demonstrated a useful sliding-window type approach to archiving data

# QUESTIONS

# Resources

- Technet white paper
  http://technet.microsoft.com/en-us/library/dd578580(v=sql.100).aspx
- SQL 2012 Table Partitioning Improvements
  http://channel9.msdn.com/posts/SQL11UPD02-REC-03
- MSDN
  http://msdn.microsoft.com/en-us/library/ms190787.aspx
- Minnesota  SQL Users Group
  http://minnesota.sqlpass.org/