SQL Server 2005 Partitioning

Kimberly L. Tripp
SQLskills.com
Email: Kimberly@SQLskills.com
Blog: http://www.SQLskills.com/Blogs/Kimberly
http://www.SQLskills.com

SQL skills

Speaker – Kimberly L. Tripp

- Independent Consultant/Trainer/Speaker/Writer
- Founder, ***SY*Solutions, Inc.** www.SQLskills.com
  - Email: Kimberly@SQLskills.com
  - *Become a subscriber on SQLskills.com and learn about new resources which can improve your productivity and server performance!*
- Microsoft Regional Director http://msdn.microsoft.com/isv/rd/
- SQL Server MVP http://mvp.support.microsoft.com/
- Author for some SQL Server 2005 Whitepapers on MSDN (links from home page on www.SQLskills.com)
- SQL Server 2005 Launch Content Manager – Data Platform Track Sessions, Demos and Cross-training
- Writer/Editor for SQL Magazine www.sqlmag.com

SQL skills.com
immerse yourself in sql server

Overview

- Motivational slides ☺
- Partitioning Strategies
- Implementing Partitioned Tables
- Why Partitioned Views are still interesting
- Managing the Sliding Window Scenario

Table Partitioning Motivation
Large "Range" Modifications

- Question from a webcast:

  You mention that deletes are not as "big of a deal" since they leave gaps. I had a situation where I had a table with around 50 million rows, about a million rows per day. After about a month and a half, I delete some old ones. I can delete a day's worth of data (1 million rows) in about 2 minutes with a delete statement, if there are no indexes. If I have a clustered index based on the date, it took about 22 minutes. Inserts were instantaneous, about two pages worth of rows every second, and even updates were quicker. Is there anything obvious that would cause this?

- Question wasn't quite complete – one piece of information was misleading
  - 2 minutes with NO indexes
  - 22 minutes with a clustered index based on date

No way! There had to be more!

Indexes, Indexes, Indexes…

- Why the large difference?
- If data ordered by date wouldn't a large range delete be easier than with a heap?
- Where was that large difference between 2 minutes and 22 minutes?
  - In the non-clustered indexes!
- Let's prove it! (*now there goes my weekend!*)
- Blog Entry: Thursday, August 26, 2004

  MSDN Webcast Q&A: Index Defrag Best Practices – Fragmentation, Deletes and the "Sliding Window" Scenario and it's the LAST one!
  http://www.sqlskills.com/blogs/kimberly/PermaLink.aspx?guid=6410cdf0-48de-48b8-8c59-5cb2ba92224b

Key Points

TableName	Avg Time	Min Time	Max Time
ChargeCLDateForDelete	1566	1143	2903
ChargeCLDateForDeleteWithCompPK	2359	1253	8080
ChargeHeap	10392	9603	11476
ChargeCLPKForDelete	11377	9453	18476
ChargeCLReallyBadForDelete	17491	12640	35440
ChargeCLDateForDeleteWithCompPKWNCIndexes	19594	8020	29414
ChargeCLDateForDeleteWNCIndexes	22467	9243	63520
ChargeCLPKForDeleteWNCIndexes	30132	17343	59366
ChargeCLReallyBadForDeleteWNCIndexes	44208	25946	62000
ChargeHeapWNCIndexes	49407	19716	78383

- Load most impacted by non-clustered indexes
- Table structure plays a roll
  (CL date faster than heap)

SQLskills.com
immerse yourself in sql server

Rolling Range (or Sliding Window)
Key Components

- Data Load
  - Single Table
    - Active Table impacted
    - Indexes need to be updated
  - Partitioned Object (PV in 2000/PT in 2005)
    - Table outside of active view manipulated
    - Indexes can be built separately of active tables
- Data Removal
  - Single Table – same problem
    - Active Table impacted
    - Indexes need to be updated
  - Partitioned Object (PV in 2000/PT in 2005)
    - Table can be "dropped" from PO

Data Loading
Single Table Scenario

18 seconds

```
ALTER TABLE ChargeFY04
  ADD CONSTRAINT ChargeFY04ChargeDtCK2
      CHECK (Charge_dt >= '20030701'
            AND Charge_dt < '20041001')
```

80 ms

```
ALTER TABLE ChargeFY04
      DROP CONSTRAINT ChargeFY04ChargeDtCK
```

24.5 minutes

```
BULK INSERT CreditPartitioned.dbo.ChargeFY04
      FROM 'C:\ChargesFY05Q1.csv'
      WITH ( BATCHSIZE = 50000,
  CHECK_CONSTRAINTS,
  DATAFILETYPE = 'char',
  FIELDTERMINATOR = ',', ROWTERMINATOR = '\n',
  KEEPIDENTITY, ORDER (charge_dt, charge_no))
```

Data Loading

New Partition

200 ms

```
CREATE TABLE ChargeFY05Q1
  (…)ON ChargesFY04Staging
```

NOTE: Used a staging area as heap and moved data with the clustered index build.

```
BULK INSERT CreditPartitioned.dbo.ChargeFY04
  FROM 'C:\ChargesFY05Q1.csv'
     WITH ( BATCHSIZE = 50000,
  CHECK_CONSTRAINTS,
  DATAFILETYPE = 'char',
  FIELDTERMINATOR = ',', ROWTERMINATOR = '\n',
  KEEPIDENTITY, ORDER (charge_dt, charge_no))
```

50.7 SECONDS

NOTE: Loaded into a staging file so that the clustered index could be built and then moved the data to a new filegroup.

Data Deletion

Single Table

13.5 MINUTES

```
DELETE ChargeFY04
     WHERE Charge_dt < '20031001'
```

1.75 MINUTES

```
ALTER TABLE ChargeFY04
     ADD CONSTRAINT ChargeFY04ChargeDtCK2
     CHECK (Charge_dt >= '20030701'
          AND Charge_dt < '20041001'),
```

5.6 seconds

```
ALTER TABLE ChargeFY04
     DROP CONSTRAINT ChargeFY04ChargeDtCK
```

Data Deletion
Drop Table

- Immediate
  950 MILLESECONDS!
- Building Indexes
  44.5 SECONDS
  Note: Index build was done after deletion so that the object could be moved into same filegroup as deleted data. Also allows table to be better "clustered" – it minimizes extent scan fragmentation
- Could archive this data and/or backup the filegroup before dropping table
- No active contention on base table during "deletion" as it's immediate and isolated
- Won't cause lock escalation

Total Times

"Single" table process: **40+ minutes**

1. Alter the table's check constraint to support new quarter (drop the old constraint)
2. Load the data using BULK INSERT
3. Delete the first quarter of F04
4. Alter the table's check constraint to show new year range (drop the old constraint)

Partitioned table process: **1 min 36 sec**

1. Create New table - with constraints
2. Load the data using BULK INSERT
3. Build Indexes – CL + 3 NC on FYQ1 Filegroup
4. Change the view to use FY05Q1 and remove FY04Q1
5. Drop the table which represents FY04Q1 (could archived/backup)

Index Rebuilds

```
DBCC DBREINDEX
   ('ChargeFY04', 'ChargeFY04PK')
```

8 minutes, 54 seconds

Table 'ChargeFY04'. Scan count 1, logical reads 33178, physical reads 68, read-ahead reads 14386.

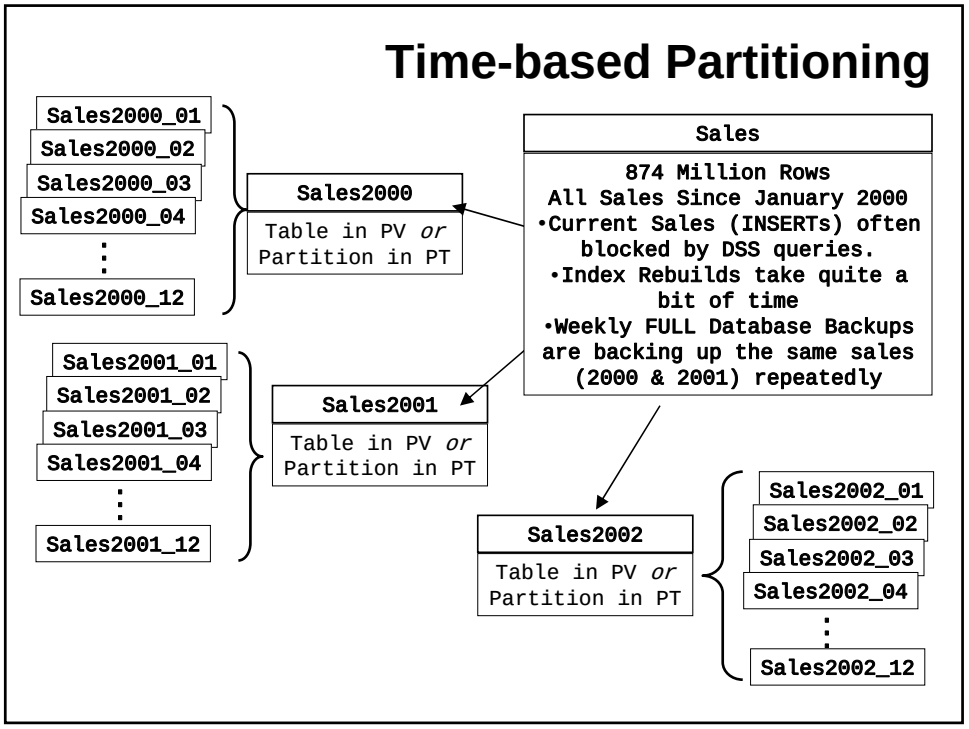
```
DBCC DBREINDEX
   ('ChargeFY04Q4', 'ChargeFY04Q4PK')
```

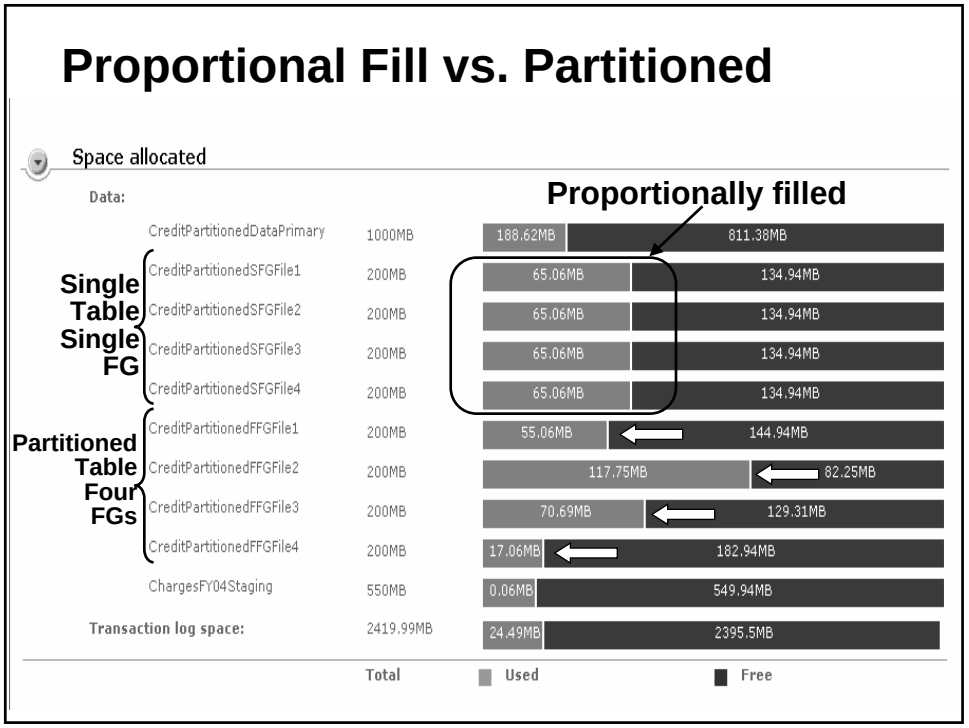
44 seconds

Table 'ChargeFY04Q4'. Scan count 1, logical reads 2156, physical reads 0, read-ahead reads 2163.

Horizontal Range Partitioning

- Any release – separate tables, separate views, you design appropriate access path – functionally motivated, complex implementation
- SQL Server 7.0 and higher
  Proportional fill
- SQL Server 7.0 – Partitioned Views
  Queryable partitions
- SQL Server 2000 – Updateable Partitioned Views
  Updateable partitions
- SQL Server 2005 – Partitioned Tables/Indexes
  More manageable partitioned BASE structures – only one table to manage





Key Differences

- Single Filegroup is easier to create/administer
- CAN perform file/filegroup backups however, no guarantee of where data lives so all files/filegroups must be backed up more frequently vs. frequently backing up ONLY the active partition
- If a file (within a filegroup) becomes damaged the ENTIRE filegroup must be taken OFFLINE
- Cannot manipulate data except at the table level, no concept of data separation or partitions
- Partitioned Table is ORDERS OF MAGNITUDE faster on Rolling Range/Sliding Window operations

Partitioned Views

Still motivation for creating in SQL Server 2005

- Feature added in SQL Server 7.0
- Separately named/created tables
- Manual placement on different filegroups
- Checked/Verified Constraints
- UNION ALL Views
- Query Optimizer removes irrelevant tables from query plan (partition elimination)
- Inserts/Updates/Deletes need to be directed to correct base table

Range Partitioned Tables

- Step 1: Create Filegroups
- Step 2: Create Files in Filegroups
- Step 3: Create Partition Function
  - This is a completely new concept!
- Step 4: Create Partition Scheme
  - This is a completely new concept!
- Step 5: Create Table(s) on Scheme (similar to FG)
- Step 6: Verify Data using system table (optional)
- Step 7: Add data to tables – SQL Server redirects data and queries to appropriate partition

Demo from updated scripts created for MSDN Whitepaper

Step 1: Create Filegroups

- When a file is added to a database its filegroup must be specified; it cannot later be changed
- Filegroups should be logically named; however, when data is going to "roll" through a fixed number of filegroups you won't want to name the filegroup based on the date.
- For simplicity, number your filegroups

```
ALTER DATABASE SalesDB
  ADD FILEGROUP [FG1]
```

Step 2: Create Filegroups

- Add files to filegroups – make sure to specify:
  - Initial Size: reasonable pre-allocation space
  - Max Size: not till 0 bytes
  - Growth Rate: fixed, reasonable
- Again, for simplicity, number your files based on their filegroups

```
ALTER DATABASE SalesDB
ADD FILE
  (NAME = N'SalesDBFG1File1',
  FILENAME = N'C:\SalesDBFG1File1.ndf',
  SIZE = 1MB,
  MAXSIZE = 100MB,
  FILEGROWTH = 5MB)
TO FILEGROUP FG1
```

Step 3: Partition Function

- Can ONLY include ONE column of a table
- Always includes the entire domain of possible values from negative infinity (-∞) to infinity (∞)
- Should be combined with constraints to limit the domain of data values
- Should think in terms of where the data will "roll"
- Make sure that data does NOT move when rolling forward

PF Boundary Conditions

- Always define n-1 boundary points for n partitions
- You will always have one partition (at the extreme right for a LEFT partition function or at the extreme left for a RIGHT partition function) that will not have a boundary point explicitly defined
- To define where or not this undefined boundary point is on the left or right you will define your boundaries

Partition Function

Left or Right?

- No performance difference between left or right
- Values can use functions to calculate boundary point but boundary point will always be stored as a literal value based on calculation of the function at time PF created
- Only the boundary point is stored – not the expression used to calculate it
- Can see boundary points stored within the catalog view (sys.partition_range_values)

Partition Function
Left or Right?

- Right
  - Defines the LOWER boundary of the SECOND/RIGHT (of the first two) partitions
  - Creates a partitioned structure where the first partition is empty (first boundary condition is not defined as the lower limit is negative infinity to less than the boundary condition)
- Imagine the boundary point '20040101' if defined with RIGHT then
  - 1st partition is all data < '20040101'
  - 2nd partition is all data >= '20040101'
- Use a “beginning point” for RIGHT-based partition

Create the Partition Function
Using RIGHT

- Orders and Order Details will include the OrderDate column (*yes, duplicated data needed in the Order Details table*)
- Orders range from October 2002 through September 2004

```
CREATE PARTITION FUNCTION
  TwoYearDateRangePFN(datetime)
AS
RANGE RIGHT FOR VALUES
  ('20021001',   -- Oct 2002
   '20021101',   -- Nov 2002
   '20021201',   -- Dec 2002
  ...
   '20040901')   -- Sep 2004
```

Step 4: Partition Scheme

- Maps the partitions (number defined by the function) to the filegroups (and therefore files) with the database
- Because both the extreme left and extreme right boundary cases are included, a partition function with *n* boundary conditions actually creates *n+1* partitions
- This first (in RIGHT) partition will remain empty as data slides/rolls forward

Step 4: Partition Scheme

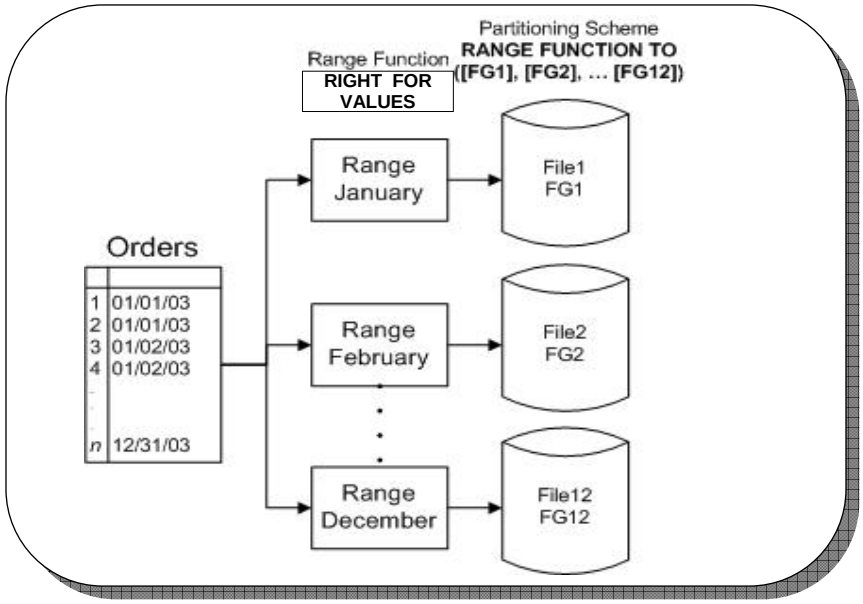
- No special filegroup is needed for this empty partition (data should *never* reside in it). This empty partition is to help “roll” data forward without causing record relocation…
- Constraints should be used to restrict the table's data – but are not required.
- Explicitly name a filegroup for each of the 24 "active" partitions and then use PRIMARY for the empty partition
- Empty partition is:
  - First partition in RIGHT scenario
  - Last partition in LEFT scenario

Create the Partition Scheme

Since we hold 2 years worth of Order/Order Details data in 24 partitions we will create a partition scheme to handle 25 partitions where the FIRST one will remain empty (because of this being a RIGHT partition function):

```
CREATE PARTITION SCHEME [TwoYearDateRangePScheme]
AS PARTITION TwoYearDateRangePFN TO
(  [PRIMARY],
  [FG1], [FG2], [FG3], [FG4], [FG5],
  [FG6], [FG7], [FG8], [FG9], [FG10],
  [FG11],[FG12],[FG13],[FG14],[FG15],
  [FG16],[FG17],[FG18],[FG19],[FG20],
  [FG21],[FG22],[FG23],[FG24])
GO
```

What does it look like?



Verify partition data/location

```
SELECT $partition.PFN(o.OrderDate)
            AS [Partition Number]
      , min(o.OrderDate)
            AS [Min Order Date]
      , max(o.OrderDate)
            AS [Max Order Date]
      , count(*)
            AS [Rows In Partition]
FROM dbo.Orders AS o
GROUP BY $partition.PFN(o.OrderDate)
ORDER BY [Partition Number]
```

PVs v. PTs wrt to Indexes

Differences and Benefits

Partitioned View

- Each table can have separate indexes – allows for better control of indexing strategies in DSS v. OLTP
- Having lots of indexes and needing to confirm that each table of a large PV has the same indexes is at a higher cost in terms of optimization
- Indexes can be controlled and maintained with any frequency desired
- Much harder to manage – especially in terms of constraints

Partitioned Table

- Each partition has to have the SAME indexes – cannot index partitions differently if partitions have different access patterns
- Each partition then has the same structure – allowing the optimizer faster processing at optimization
- Indexes can be analyzed at the partition level and even rebuilt at the partition level; however, ONLINE rebuild is only supported at the table level
- Easier to manage and better availability options…

PVs v. PTs wrt to Availability

Differences and Benefits

Partitioned View	Partitioned Table
• If a partition (i.e. a table) of a partitioned view becomes unavailable, the view will no longer continue to be accessible – for queries or updates	• If a partition to a partitioned table becomes unavailable, the partitioned table STAYS accessible and can continue to be queried and updated…for the partitions that are still available

Best Combination – BOTH!

- Create a Partitioned Table for the read-only portion of the table
- Create a standalone (or even a partitioned table) for the read-write portion of the table
- Create a partitioned view that unifies this data together – virtually, for queries
  - If you want to update through the view – must follow rules for Updateable Partitioned Views
  - However, can direct modifications to read-write table (if only one or a partitioned table) and then ONLY use the View for Queries

SQLskills.com
immerse yourself in sql server

The Process of Data Loading

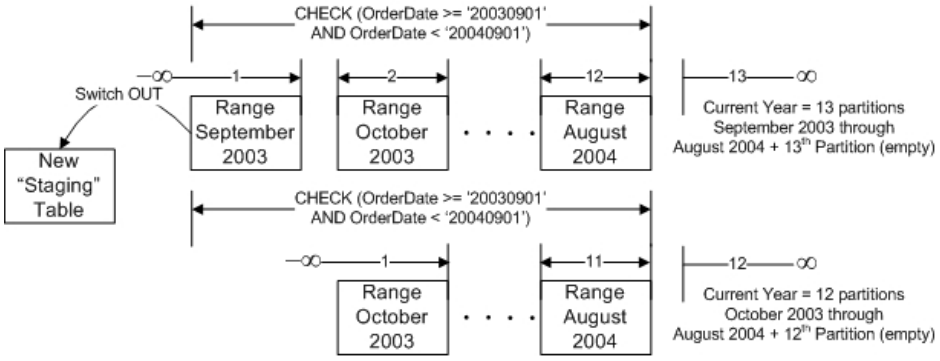
- LOAD: Getting the data in – fast!
  - Partitioned Table Requirements
  - Staging Area/Loading in a Heap
  - Database Recovery Model
  - Bulking Loading with Parallel streams
- CLEANSE: Transforming/Verifying data
  - During load
  - After load
- STRUCTURE: Move to final location w/Index

Optimizing the Rolling Range

- Old data must be removed
- New Data must come in
- Simple metadata changes = FAST
- Metadata ONLY changes require:
  - New table on the same filegroup where the partition resides or is going to reside
  - Structure it IDENTICALLY to that of the partitioned table – including indexes
  - SWITCH the partition's data IN/OUT with the data of the new table

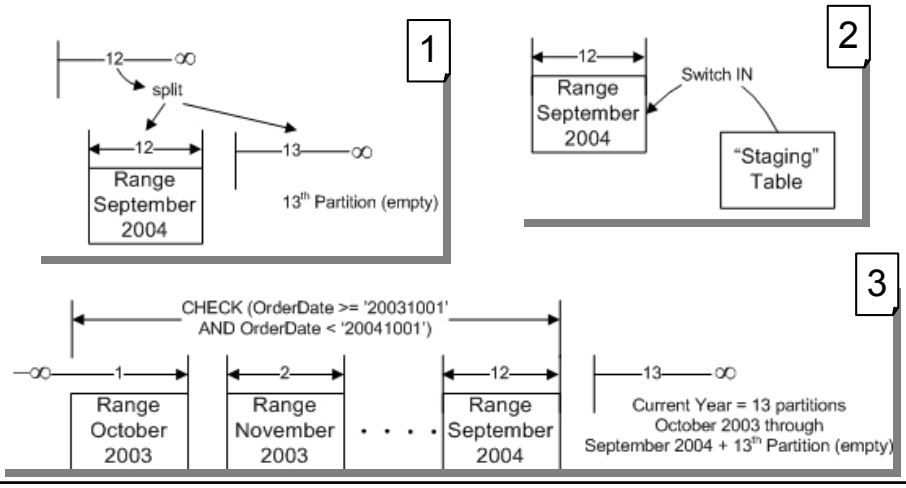
The Rolling Range – Switch

- Create new table (same structure/indexes)
- Switch OUT the old data

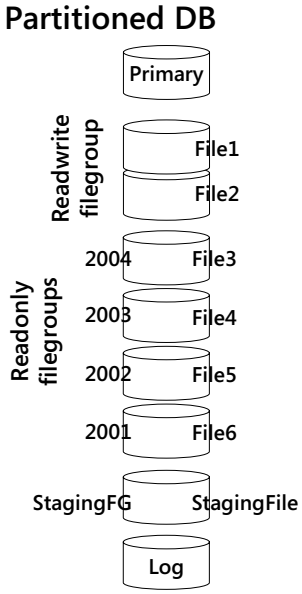


The Rolling Range – Switch

- Create new heap, load data, build indexes, switch IN the current data



Staging Area/Loading in a Heap



Database *consists of…*
Filegroups *consist of…*
Files *consist of…*
Objects *are placed on a filegroup*

Switch in as a partition! (3)

Move to Final FG (2)

⇦ **Create CL Index on FG**

Remember – the CL Index is the data

Data Load (1)

⇦ **Create Heap on Staging FG**

Recovery Model

- If database is critical in recovery:
  - FULL Recovery Model – for systems that have 24x7x365 access and do not want possible work loss exposure
  - BULK_LOGGED Recovery Model – for systems that can have a minimized amount of work loss exposure
- If database is data warehouse being completely rebuilt:
  - SIMPLE Recovery Model – backup database OR backup READ_WRITE_FILEGROUPS after completion

SQLskills.com
immerse yourself in sql server

Session Summary

- Consider Functionally-based Horizontal Partitioning to manage the Sliding Window Scenario
  - Get a feel for the right design
  - Design appropriately
  - Test your sliding window performance!
- Check out the whitepapers, demo scripts and even the MSDN Webcast Series

MSDN Webcast Series

http://www.microsoft.com/events/series/msdnsqlserver2005.mspx

- Session 1: Interaction between data and log
- Session 2: Recovery Models
- Session 3: Table optimization strategies
- Session 4: Optimization through indexes
- Session 5: Optimization through maintenance
- Session 6: Isolation, locking and blocking
- Session 7: Optimizing procedural code
- Session 8: Partitioning…
- Session 9: Profiling for the unknown problems
- Session 10: Common Roadblocks, *A Series Wrapup*

Resources

Whitepapers written by Kimberly L. Tripp

- "SQL Server 2005 Snapshot Isolation"
  On MSDN and www.SQLskills.com
- "SQL Server 2005 Partitioned Tables"
  On MSDN and www.SQLskills.com
- Blogged this new one with a preview on SQLskills: The Database Administrator's Guide to the SQL Server Database Engine .NET Common Language Runtime Environment
  www.sqlskills.com/blogs/kimberly/PermaLink.aspx?guid=385c0e05-497f-4c4c-b24a-155106a08959
- Another one in the works: SQL Server 2005 Management Tools

SQL skills

Thank you!

Please take a moment to fill out your evaluation.

Kimberly L. Tripp

Consultant . Trainer . Writer . Speaker

email: **Kimberly@SQLskills.com**

Make sure to register for special offers and other helpful information and resources!

www.SQLskills.com

SQL skills.com

immerse yourself in sql server