# Internals of Database Snapshot

Suhas De 31 Jul 2011 1:09 AM 3

In this post, we will explore the internals of Database Snapshot. Most of this information is already available in the Internet, in the posts authored by the SQL Server Development Team and the CSS Escalation Team; however, through this post we will attempt to consolidate all of these information for the benefit of everybody.

All credit for this post goes to my colleague, Mohammad Sufian, without whose help, this post may have not been possible.

**So, what are Database Snapshots?** According to the Books OnLine, "*A database snapshot provides a read-only, static view of a source database as it existed at snapshot creation, minus any uncommitted transactions. Uncommitted transactions are rolled back in a newly created database snapshot because the Database Engine runs recovery after the snapshot has been created (transactions in the database are not affected).*" This means that the database snapshot is a transactionally consistent image of the database as of the snapshot creation time.

**What can I use Database Snapshots for?** Again, according to Books OnLine, some uses of Database Snapshots might be:

- Maintaining historical data for report generation.
- Using snapshots on Mirrored databases can help in offloading reporting.
- Safeguarding data against administrative and user errors.
- Managing a test database.

**What can't I do in Database Snapshots?** The following lists a few things that are not allowed or will not work in Database Snapshots:

- Database snapshots are specific to a SQL Server instance.
- Database snapshots can be created only if the source database is ONLINE or if it is the mirrored database.
- Database snapshots cannot be backed up or restored.
- Database snapshots cannot be detached or attached.
- Database snapshots are read-only so they cannot be renamed. Settings cannot be modified as well.
- Database snapshots inherit security settings, filegroup state from the source database and these settings cannot be modified on the database snapshot.
- Database snapshot file specifications cannot be modified.
- Database snapshot is marked suspect if there is no disk space when a modified page is being copied from the source database.
- The point-in-time of a database snapshot only approximate in nature and can be determined from the create_date column of sys.databases catalog view entry for the database snapshot.
- Database snapshots created on two source databases that have say views pointing from one database to another will retain their original definition in the snapshots also. Any queries against those views in the database snapshots will refer to the source databases if they are online. If the source databases are mirrored to a different server and database snapshots are created against those then the queries will fail since the source databases will be in recovery mode.
- Full-text indexing is not supported on database snapshots.
- Database snapshots of system databases are not allowed.

**How can we create a Snapshot and what happens when we create a snapshot?** To create a snapshot of a database we can use the `CREATE DATABASE <specifications> AS SNAPSHOT OF <DatabaseName>` command. Below is an example of the command that I used to create a Snapshot of the AdventureWorks database:

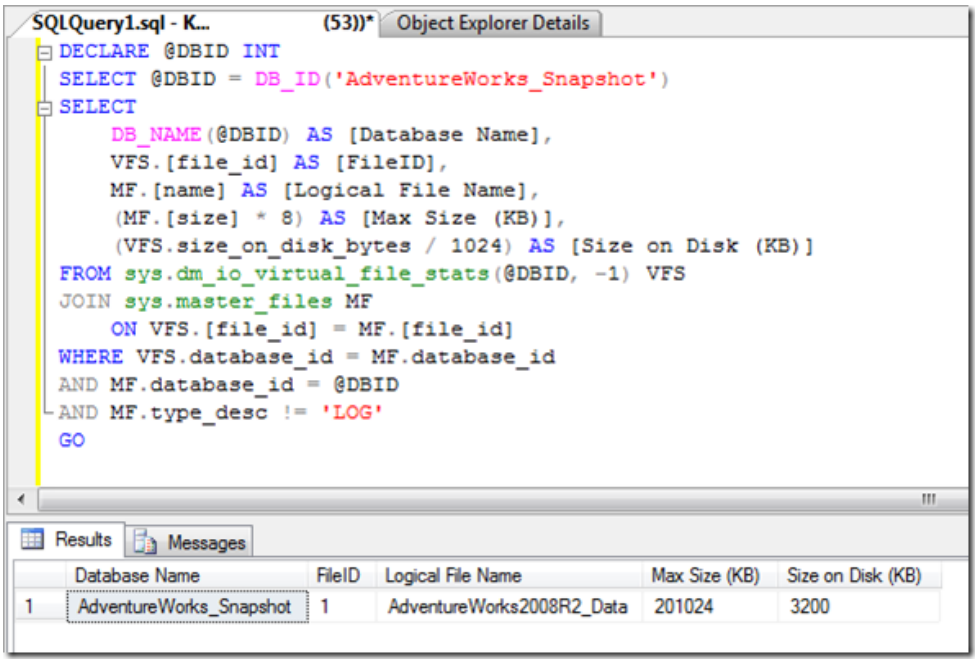
```
CREATE DATABASE AdventureWorks_Snapshot
ON
(
    NAME = AdventureWorks2008R2_Data,
    FILENAME = 'D:\AdventureWorks_Snapshot.ss'
) AS SNAPSHOT OF AdventureWorks
GO
```

When we use this command to create a database snapshot, the snapshot is implemented as a **sparse file**. By definition, *Sparse files are a feature of the NTFS file system. Initially, a sparse file contains no user data, and disk space for user data is not allocated to it. When first created, a sparse file takes up very little disk space. As data is written to the sparse file, NTFS allocates disk space gradually and the file grows in size. Potentially, a sparse file can grow very large.*

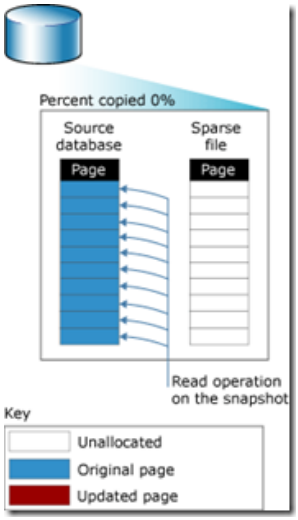
> **Remember:** To be able to understand all of the below discussion, we need a database that is not touched by user transactions. Hence, I am using the AdventureWorks database on my local machine.

Don't believe it? Let's see… The query below will give us the Max Size and the Size on Disk for the Snapshot Database:

```
DECLARE @DBID INT
SELECT @DBID = DB_ID('AdventureWorks_Snapshot')
SELECT
    DB_NAME(@DBID) AS [Database Name],
    VFS.[file_id] AS [FileID],
    MF.[name] AS [Logical File Name],
    (MF.[size] * 8) AS [Max Size (KB)],
    (VFS.size_on_disk_bytes / 1024) AS [Size on Disk (KB)]
FROM sys.dm_io_virtual_file_stats(@DBID, -1) VFS
JOIN sys.master_files MF
    ON VFS.[file_id] = MF.[file_id]
WHERE VFS.database_id = MF.database_id
AND MF.database_id = @DBID
AND MF.type_desc != 'LOG'
GO
```




Till now, the ***AdventureWorks_Snapshot.ss*** sparse file that we created contains no user data – just the necessary metadata for the database. The Max Size of the Snapshot is equal to the Size of the database when the snapshot was created, and the Size on Disk is the actual space used by the snapshot. Hence, 3200 KB or just a little above 3 MB is allocated, and this is sufficient for maintaining the metadata of the database. So, now, the obvious question is, how will data be returned if we query the snapshot? This is clearly documented in Books OnLine:




Since we have no user data in the snapshot at present and the data pages in the Source Database has not changed, the Read operation on the Snapshot reads the original page from the source database to retrieve the data from that page. The metadata in the Snapshot has the information about whether a page has been copied from the source; and based on this information, reads are redirected.

This is, in fact, a very good design, as otherwise, we would have to copy huge sets of data (GBs or even TBs of data) to the Snapshot file. The copy operation would have taken huge time to complete, and would have exerted a huge load on the disk.

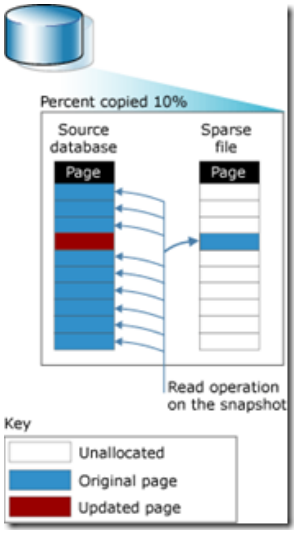
**Myth 1: The sparse file contains all zero-d out pages.**
Contrary to common belief, the sparse file does not contain zero-d out pages just yet. No extents are allocated, and the sparse file contains no data pages.

Now, when a page is modified on the source database, the pre-modified copy of the page is first copied to the database snapshot, and then the original page is modified. This operation is irrespective of whether the transaction that modified the page commits or rolls back. Here are the steps that are followed when a page is modified or dirtied:

- The Page is prepared to be dirtied.
- An EX Latch is acquired on the page.
- Replica writes are completed – this means that a replica of the page is created in the snapshot sparse file.

- Changes are now made to the page.
- The EX Latch on the page is now released.

This operation is known as the **Copy On Write (COW)** operation on database snapshots.




Once a page has been copied to the Snapshot, the metadata in the Snapshot is changed to reflect that the original state of the page is now available in the snapshot.

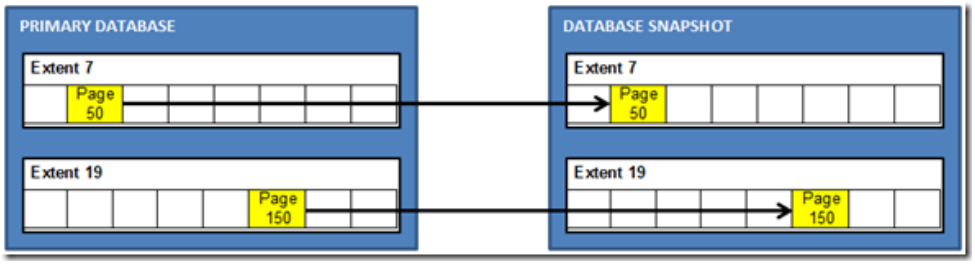
Now, when a query attempts to read this page from the Snapshot (by querying the Snapshot Database), the page is returned directly from the Snapshot. For all other pages that have not been copied to the Snapshot, the source database will still be read to fetch the data.

**Note:** Certain operations on the data, such as a Truncate Table, do not modify the data pages; rather, the allocations related to the table are removed from the metadata. So, when you run Truncate Table on a database, and cause a huge table (say 400 GB) to be emptied, the pages (data and index) belonging to the table are not copied over. When this same table or any other object attempts to re-use these allocations and over-write the pages – that's when these pages are actually copied to the snapshot.

**Myth 2: Modified pages are placed adjacent to each other in the snapshot to reduce the sparse file used size.**
Contrary to common belief, modified pages are not placed adjacent to each other. Let us assume, Pages 50 and 150 are being modified. Let us also assume Page 50 belongs to Extent 7 and Page 150 belongs to Extent 19. So, Page 50 is the 2nd Page of Extent 7 and Page 150 is the 6th page of Extent 19.

When Page 50 is dirtied, a complete allocation of Extent 7 will be created in the sparse file at storage offset ((7 – 1) * 8 + 1) * 8192 bytes, and the contents of Page 50 will be copied to the 2nd Page in this extent. All other pages will be zero-d out. So, essentially, offset of the Page 50 will be 50 * 8192 bytes. When Page 51 is dirtied, a new extent will not be allocated; Extent 7 already has the place-holder for the Page 51, and hence, when this page is dirtied on the source, it will be copied in the data offset 51 * 8192 bytes.

Similarly, when Page 150 is modified, allocation of Extent 19 will be created at offset ((19 – 1) * 8 + 1) * 8192 = 145 * 8192 bytes. Page 150 will be the 6th Page in this extent and hence will be at offset 150 * 8192 bytes.. So, as you can see, these pages will be spread out in the sparse file at its own offset (depending on the number of the page). Below is a diagrammatic view of the same:



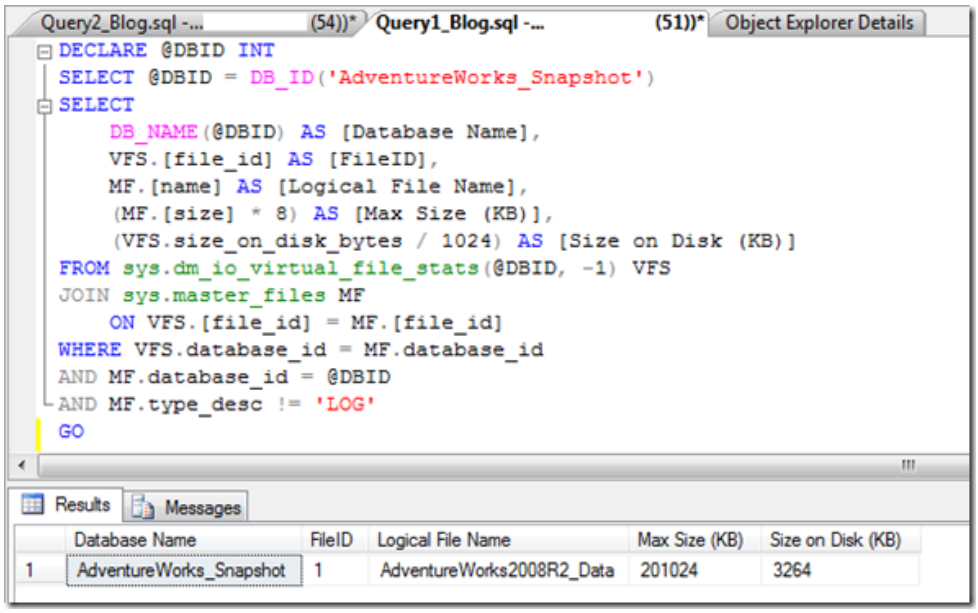

Let us modify a few rows and see what happens to the snapshot. I will update the Sales.CreditCard table. Using undocumented commands, I know that:

- CreditCardID = 615 is the first row of page 1:2550.
- CreditCardID = 719 is the first row of page 1:2551.
- CreditCardID = 2663 is the first row of page 1:2570.

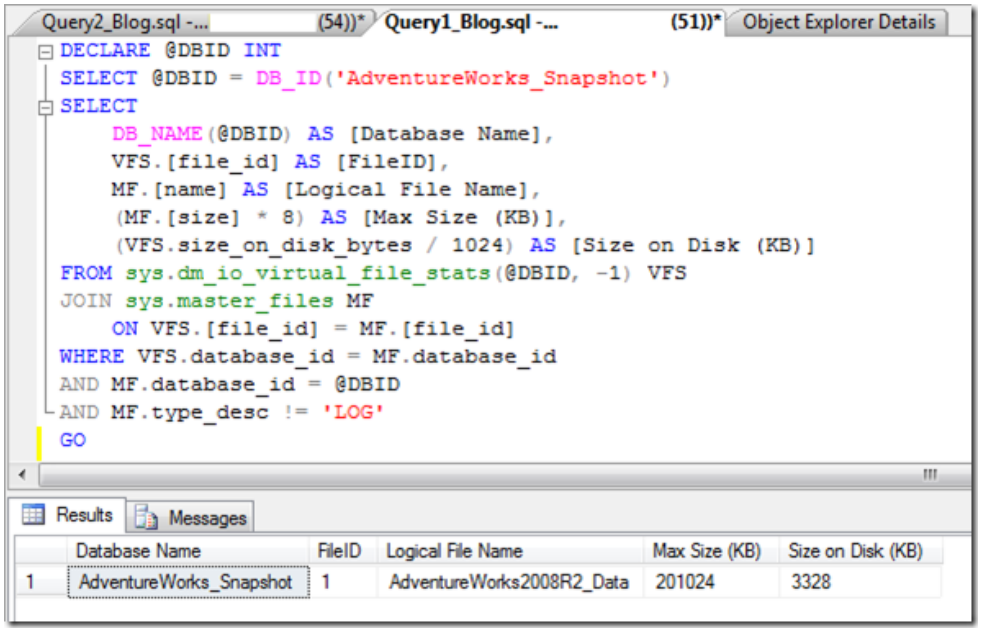
I will modify Page 2550 by updating the row with CreditCardID = 615. I used the following command:

```
UPDATE Sales.CreditCard SET ExpMonth = 8 WHERE CreditCardID = 615
```

After the update, if we check the size of the snapshot, we will see that the Size on Disk of the Snapshot file has increased to 3264 KB, which means that 1 extent has been allocated to the snapshot file.




Now, lets modify Page 2551, and we will see that the Size on Disk of the Snapshot file will not change, as both of these pages belong to the same extent. However, if we modify Page 2570, which belongs to another extent, we will see that the Size on Disk of the snapshot will further increase by 64 KB (3264 KB + 64 KB = 3328 KB):




All of the experiments above clearly demonstrate the **Copy On Write** mechanism as we had discussed above.

> **Remember:** Once a page has been copied over to the Snapshot, further modifications to the page will not cause it to be copied over.

This means that for a Transactional Database, each page that is modified after the Snapshot is created, will cause the page to be copied over to the Snapshot. This also means that you will see a performance degradation when a page is modified for the first time after the Snapshot is created.

If you have multiple Snapshots of the same database, each page, when dirtied, will have to be copied to each Snapshot if it does not already exists in it. To explain this further, let us assume that you have created 3 Snapshots at the following times:

- Snapshot 1 is created at 1:00 AM.
- Snapshot 2 is created at 3:00 AM.
- Snapshot 3 is created at 5:00 AM.

If Page 2500 is modified, for the first time, at 6:00 AM after say, 12:30 AM, this page will have to be copied over to all 3 Snapshots before it can be modified. On the other hand, if Page 2890 is modified at 1:30 AM, and then at 5:30 AM, the modification at 1:30 AM will cause it to be copied over to Snapshot 1; and the modification at 5:30 AM will cause it to be copied over to Snapshot 2 and Snapshot 3 (it will not be copied to Snapshot 1 as the page already exists in Snapshot 1).

**Summary: If you plan to create one or more Snapshots of your Production Database, please make sure you consider the performance degradation caused by the Snapshots.**

In the next post, we will investigate how data is retrieved from Database Snapshots and its effect on the Buffer Pool.

**Additional Reading:**

- Database Snapshots on Books OnLine.
- Misconceptions around database snapshots and transaction rollbacks.
- Database snapshots - when things go wrong.
- How It Works: SQL Server 2005 Database Snapshots (Replica).
- How It Works: Shapshot Database (Replica) Dirty Page Copy Behavior (NewPage).
- How It Works: SQL Server Sparse Files (DBCC and Snapshot Databases) Revisited.



## Comments

Varun_SQL 16 Aug 2011 5:32 AM

A nice detailed blog on SNAPSHOTS. This gives lot of idea of internals.

I'm taking time to read everybit of it...

Varun

LearnerSql 14 Dec 2011 3:20 AM

Nice article.

**Randy M.** 30 Sep 2013 4:43 PM

Gentlemen, a very nice piece of work. I shall print this one for my sql note book.