**Who Truncated That Table & What Do I Do Now?**

**http://www.sqlservercentral.com/articles/Truncate/109387/**

**By**[**Sujeet Singh**](http://www.sqlservercentral.com/Authors/Articles/Sujeet_Singh/666779/)**, 2014/06/05**

Someone comes to us, just to tell that he\she do not see any data in that \*Important Table\*, and it seems like the table has been truncated! The table in question had some data in it and all that data has gone now.

Perhaps all of us have seen this, at some point in our DBA career. Here are few questions that quickly go through our mind after this situation:

1. Who truncated the table & at what time?
2. How can we recover the maximum possible data?

Well, that’s what we are going to discuss in this article & we will try to find the answer for these questions.

**Who Truncated the Table & When?**

We can get this information by using these undocumented functions: fn\_dblog () and fn\_dump\_dblog (). All the credit belongs to Mr. Paul Randal who taught us about these extremely useful functions. He has written excellent article on how to use these functions to restore the database Point-In-Time. You can read that article at this URL: <http://www.sqlskills.com/blogs/paul/using-fn_dblog-fn_dump_dblog-and-restoring-with-stopbeforemark-to-an-lsn/>

The function fn\_dblog () works only when transaction log entries have not been purged from the transaction log file (either by CHECKPOINT in Simple recovery model or by the T-Log backup in Full recovery model).

In case the transaction log entries has been purged from the log file because a transaction log backup has been taken after the truncate occurred, we can still use the function fn\_dump\_dblog() to read the log backup file to get our information.

**\*\*\*As both these functions fn\_dblog () & fn\_dump\_dblog () are undocumented, use them at your own risk. \*\*\***

Let’s check it with an example. First, we need to setup our test environment.

--================================================================================================== Section 1: --====================================================================================

USE [master]

GO

IF DB\_ID('FnDBLogTest') IS NOT NULL

BEGIN

DROP DATABASE FnDBLogTest;

END

GO

CREATE DATABASE FnDBLogTest;

GO

----============== Setting the recovery model for database as FULL

ALTER DATABASE FnDBLogTest SET RECOVERY FULL;

GO

USE FnDBLogTest;

GO

IF OBJECT\_ID('Example', 'U') IS NOT NULL

BEGIN

DROP TABLE Example;

END

GO

----============== Creating a test table with some data for our demo

CREATE TABLE Example

(

TheID INT IDENTITY(1, 1)

, TheData VARCHAR(1)

)

GO

INSERT INTO Example

VALUES ( 'A' ),

( 'B' ),

( 'C' ),

( 'D' )

GO

----============== Taking a Full database backup to ensure that database is not

----============== in pseudo Simple recovery model

BACKUP DATABASE FnDBLogTest

TO DISK = 'D:\SQLTestLab\FnDBLogTest.BAK' WITH STATS=10, INIT

GO

----============== Truncating the test table

TRUNCATE TABLE Example

What we have done above is

* We have created a new database named FnDBLogTest.
* Inside this database we have created a test table with name ExampleTable with some data in it.
* We took a Full database backup for database FnDBLogTest to make sure that it is indeed using Full Recovery model.
* We truncated the table.

Now, to find the user name who executed the Truncate Table command & the Time when he\she executed it, we first need to extract the [Transaction Id] for that particular transaction, from the Transaction Log (.ldf) file. Here is the syntax to do it:

--============== Retrieving the last TransactionID for Example

DECLARE @ObjectName SYSNAME

DECLARE @TransactionID NVARCHAR(500)

SET @ObjectName = 'dbo.Example'

-- Your schema qualified table name here

SET @TransactionID = ( SELECT TOP 1 [Transaction ID]

FROM fn\_dblog(NULL, NULL)

WHERE AllocUnitName = @ObjectName

ORDER BY [Transaction ID] DESC

)

PRINT @TransactionID

Once we have got the [Transaction ID] we are ready to fetch User Name & Time when the table was truncated.

--============== Retrieving the UserName & Time when the table was truncated,

--============== based on the TransactionID

SELECT [Transaction Name]

, SUSER\_SNAME([Transaction SID]) AS UserName

, [Begin Time]

, Operation

, [Transaction ID]

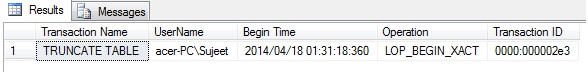
FROM fn\_dblog(NULL, NULL)

WHERE [Transaction ID] = '0000:000002e3' -- @TransactionID

AND [Transaction Name] LIKE 'TRUNCATE%'

AND Operation = 'LOP\_BEGIN\_XACT'

Executing above query should produce a result something like this:



http://www.sqlservercentral.com/Resources/Images/zoom.gif[Zoom in](javascript:;)  |  [Open in new window](javascript:;)

Now we know, who truncated the table & at what time :-).

By combining above queries, a final script to do this, where you do not need to manually enter the [Transaction ID] is given below:

SET NOCOUNT ON;

DECLARE @ObjectName SYSNAME

DECLARE @TransactionID NVARCHAR(500)

SET @ObjectName = 'dbo.Example'

 -- Your schema qualified table name here

--============== Retrieving the last TransactionID for the table

SET @TransactionID = ( SELECT TOP 1 [Transaction ID]

                        FROM fn\_dblog(NULL, NULL)

                        WHERE AllocUnitName = @ObjectName

                        ORDER BY [Transaction ID] DESC

                     )

PRINT 'Transaction Id : ' + @TransactionID

--============== Retrieving the UserName & Time when the table was truncated, based on the TransactionID

SELECT @ObjectName AS ObjectName

    ,   [Transaction Name]

    ,   SUSER\_SNAME([Transaction SID]) AS UserName

    ,   [Begin Time]

    ,   Operation

    ,   [Transaction ID]

    FROM fn\_dblog(NULL, NULL)

    WHERE [Transaction ID] = @TransactionID -- Transaction ID

        AND [Transaction Name] LIKE 'TRUNCATE%'

        AND Operation = 'LOP\_BEGIN\_XACT'

Moving ahead to our next requirement.

**How can we recover the maximum possible data?**

The answer is, we can recover maximum possible data by restoring our database to a Point-in-Time. As the name suggests, a Point-in-Time restore allows us to restore/recover a database up to a specific date & time. It means all the changes made to the database after that specified time will not be restored.

It helps us immensely in situations like the one described above where someone accidently truncates a table OR in the worst cases, where complete database gets crashed due to a faulty hardware etc.

**Prerequisites for Point-In-Time Restore**

There are few prerequisites to be able to restore a database Point-In-Time. These prerequisites are given below:

* The database should be in either Full OR Bulk-Logged recovery model.
* You should have at least one Full database backup available (to start the restore sequence) & all the T-Log backups taken after this Full database backup. You can use differential backups in order to reduce the number of T-Log backups to restore.
* If the database is using Bulk-Logged recovery model, then T-Log backups that contain any Bulk-Logged operation (such as ALTER INDEX etc.) can’t be used for Point-In-Time recovery. These backups will always be restored to end of the T-Log backup.

Let’s examine how Point-In-Time recovery works with an example. First we need to create a test database that has a table with some data.

USE [master]

GO

CREATE DATABASE PITRTest ON (NAME='PITRTest', FILENAME = 'D:\SQLTestLab\PITRTest.mdf')

LOG ON (NAME='PITRTest\_log', FILENAME = 'D:\SQLTestLab\PITRTest\_log.ldf')

GO

USE PITRTest

GO

----============== Setting the recovery model for database as FULL

ALTER DATABASE PITRTest SET RECOVERY FULL

GO

IF OBJECT\_ID('Example') IS NOT NULL

BEGIN

DROP TABLE Example

END

GO

----============== Creating a test table with some data for our demo

CREATE TABLE Example

(

ID INT IDENTITY(1, 1)

, Data VARCHAR(1)

)

GO

INSERT INTO Example

( Data )

VALUES ( 'A' ),

( 'B' ),

( 'C' ),

( 'D' ),

( 'E' )

This will create a database named PITRTest.  Further, it will create a table named Example with some data in it, inside the database PITRTest. Next, we take a Full backup for our PITRTest database.

----============== Taking a Full database backup

BACKUP DATABASE PITRTest TO DISK = 'D:\SQLTestLab\PITRTest\_FULL\_1.BAK'

WITH STATS=1, INIT

 After the Full backup is done, we insert few more rows into our test table and take the first T-Log backup.

INSERT INTO Example

VALUES ( 'F' ),

( 'G' ),

( 'H' ),

( 'I' ),

( 'J' )

GO

BACKUP LOG PITRTest TO DISK = 'D:\SQLTestLab\PITRTest\_Log\_1.trn'

WITH STATS=1, INIT

GO

We insert few more rows into the table and then truncate the table.

INSERT INTO Example VALUES ('K'), ('L'), ('M'), ('N'), ('O'), ('P'), ('Q'), ('R'), ('S'), ('T')

GO

--=============== Printing the time before performing the truncate

SELECT GETDATE () AS TimeBeforeTruncate ---> 2014-04-18 02:20:37.277

GO

--=============== Truncating the table

TRUNCATE TABLE Example

GO

Let's take another T-Log backup after the truncate.

BACKUP LOG PITRTest TO DISK = 'D:\SQLTestLab\PITRTest\_Log\_2.trn' WITH STATS=1, INIT

We have our test environment ready now. It is time to perform the restores & check whether we will be able to recover all the data.

**Recovering the Data**

As you might have noticed, we are printing the time, right before truncating the table. We will restore our database up to that point to get all the data back.

However, in the real world, you wouldn’t get this lucky, as the person who truncates the table will clearly not inform you before doing that so you can record the time. In fact you can call yourself lucky if he\she informs you even after the Truncate ;-)

In these cases where we are not aware about the time when the truncate took place, what can we do? Here are our options:

**Restore the Database in STANDBY Mode & Crawl Through the T-Log Backups:**

If we have some idea about by what time the data was not available in the table, we can restore our database in STANDBY mode, 2-4 minutes prior to that time. Restoring the database in STANDBY mode serves 2 purposes.

1. It allows further logs to be restored on it.
2. It keeps the restored database in Read-Only mode; therefore we can check what data is available in the table after every successive restore.

If the data is available after first restore, we can progressively (second-by-second\ minute-by-minute) restore our database using STOPAT option & simultaneously we can query the table to check what data is available after every restore.

During the restore process, there will be a moment when you will have the data at this second and the next second you will find the table empty. When that happens, you got the point where you had all the data present in the table before it vanished away. Now you can restart the restore sequence and stop at just that point.

We will start the restore sequence with restoring the full backup.

RESTORE DATABASE PITRTest\_2 FROM DISK = 'D:\SQLTestLab\PITRTest\_FULL\_1.BAK'

WITH STATS=1,STANDBY='D:\SQLTestLab\PITRTestStandBy.tuf',

MOVE 'PITRTest' TO 'D:\SQLTestLab\PITRTest\_2.mdf',

MOVE 'PITRTest\_log' TO 'D:\SQLTestLab\PITRTest\_2\_log.ldf'

Since we are resting the database in Standby mode; we can check what data is available after every restore using a simple SELECT statement.

SELECT \*

    FROM PITRTest\_2..Example --> 5 Rows

Next, we are going to restore the first T-Log backup.

RESTORE LOG PITRTest\_2 FROM DISK = 'D:\SQLTestLab\PITRTest\_Log\_1.trn'

WITH STATS=1,STANDBY='D:\SQLTestLab\PITRTestStandBy.tuf'

GO

SELECT \* FROM PITRTest\_2..Example -- 10 Rows

Here, we are going to restore the final T-Log backup with STOPAT option to restore it up to a specific time that we recorded earlier before executing the truncate.

--=============== Stopping the restore at a point just before the truncate

RESTORE LOG PITRTest\_2 FROM DISK = 'D:\SQLTestLab\PITRTest\_Log\_2.trn'

WITH STATS=1,STANDBY='D:\SQLTestLab\PITRTestStandBy.tuf',

STOPAT='2014-04-18 02:20:37.277'

GO

SELECT \* FROM PITRTest\_2..Example -- All 20 Rows

If we continue the restore beyond this point we will see that truncate operation will be rolled forward & the table will be empty.

--=============== Continuing the restore beyond the point where truncate was executed

RESTORE LOG PITRTest\_2 FROM DISK = 'D:\SQLTestLab\PITRTest\_Log\_2.trn'

WITH STATS=1,STANDBY='D:\SQLTestLab\PITRTestStandBy.tuf',

STOPAT='2014-04-18 02:20:38.000'

GO

SELECT \* FROM PITRTest\_2..Example -- 0 Rows

-- (Because we restored the log beyond the point where table was truncated.)

Therefore, as we can see, we can restore a T-Log backup Point-In-Time & get all the data back up to that point. Also, we can say that restoring this way i.e. crawling through the log backup file & keep adjusting the STOPAT time will sure be a tedious task to do.

**Use fn\_dblog ()\fn\_dump\_dblog () Functions to Know Where to Stop:**

Remember, in the begining of this article, we retrieved the User Name who truncated the table as well as we got the TIME when truncate operation started with the help of an undocumented function fn\_dblog ().

You can use these functions if you find yourself comfortable with them to get the exact time when truncate started & stop your restore sequence right before that point using STOPAT option as described in above example.

**Busting the Myth:**

In the end, the restore example above proves that TRUNCATE is a fully logged operation, contrary to the popular myth that it is a non-logged operation & it can be rolled forward by the restore of a T-Log backup.

That's all for now. Thanks for reading, friends.