**3 Solutions For Auditing Long SQL Server Extracts or Loads**

By: [Tim Smith](https://www.mssqltips.com/sqlserverauthor/92/tim-smith/)

FROM: <https://www.mssqltips.com/sqlservertip/4949/3-solutions-for-auditing-long-sql-server-extracts-or-loads/>

**Problem**

We've been considering running audits (data validation and verification) for our ETL flow during different points of the process. We face constraints when we may be able to run these validations at multiple points since the length of time often requires an overnight or multi-hour run. We're afraid that if we run these too late, the reports for users may not be available in time, especially if we need to make adjustments due to a potential issue. What are some steps to possibly resolving this problem?

**Solution**

In this tip, I will use the term audit to refer to the ETL context of validating data or verifying data before, after or during a load. Similar to financial audits, these validations can may be independent of the flow, or completed during the flow. Also, even when using tools, you may want an additional check for verification purposes to avoid an error with a tool potentially resulting in a bad extract, load or transform.

1 - Use audit-based extracts or loads

Let's compare an example of this using the T-SQL based bulk insert. The below code will bulk insert the file *20170103.csv* into a table starting at the 2nd row of the file. If the file has one million rows after the second row, it will bulk insert all million rows. If we wanted to know the status of this insert as it was happening, we won't know until it completes. In other words, for this specific insert, we don't know what's happening until it's done.

BULK INSERT OurTable

FROM 'E:\ETL\20170103.csv'

WITH (

FIELDTERMINATOR = ','

,ROWTERMINATOR = '0x0a'

,FIRSTROW=2

,ERRORFILE='D:\Logs\20170103\_log.log'

)

GO

Let's contrast this with an audit-based insert using the same functionality - T-SQL's bulk insert:

BULK INSERT OurTable1

FROM 'E:\ETL\20170103.csv'

WITH (

FIELDTERMINATOR = ','

,ROWTERMINATOR = '0x0a'

,FIRSTROW=2

,LASTROW=10000

,ERRORFILE='D:\Logs\20170103\_1\_log.log'

)

GO

In this bulk insert, we specified a last row for the insert at row 10000 and included the batch number in the error file (**20170103\_1\_log**) and used a horizontally-scaled table matching the error log (OurTable1). By using this smaller batch, we can add a confirmation of the status after the batch is complete, such as stating the batch number during iteration (less expensive) or running an aggregate after the load (more expensive). In the former audit, the first batch would be up to row 10000, the second batch would be up to 20000, etc. In addition, if we experienced an issue, the log file would state the batch number and we would know *up to what batch* the insert succeeded, along with what table could potentially have fewer records and why. Using the same T-SQL, PowerShell could perform this loop to iterate based on the file's line length:

[int]$end = 1000000

[int]$start = 2

[int]$i = 1

while ($start -lt $end)

{

$tsql = "

CREATE TABLE OurTable\_$i (ColumnExampleOne VARCHAR(25), Price DECIMAL(13,4), ColumnExampleThree VARCHAR(25))

BULK INSERT OurTable\_$i

FROM 'E:\ETL\20170103.csv'

WITH (

FIELDTERMINATOR = ','

,ROWTERMINATOR = '0x0a'

,FIRSTROW=$start

,LASTROW=" + ($start + 10000) + "

,ERRORFILE='D:\Logs\20170103\_" + $i + "\_log.log

)

"

### Bulk insert T-SQL

Invoke-Sqlcmd -Query $tsql

### Audit T-SQL example - saving the date and time of the number of passes:

Invoke-Sqlcmd -Query "INSERT INTO bulkinsertAudit VALUES ($i,GETDATE())"

$start = $start + 10000

$i++;

}

This demonstrates one way to think about extracts and loads which take significant time and limit post audits; the process of loading or extracting data is validating the data as it flows - in this case, each iteration is of an amount of records. When a possible problem with a data range occurs, we have the logs scaled for verifying what values may have been problematic and in what range; we can either rollback by preventing that range of data further in the ETL process, we may discover a few erroneous records that were excluded, or we may rollback the entire data set if too many problems occur (this last step is rare, but does occur if enough data points throw errors). In the second example, this may be removing one of the many tables involved, whereas in the first example, this would be more complex if no identity field for filtering exists. We might validate data other ways as well using this approach, such as auditing the average of the price of each table (using 3 tables as an example here):

SELECT 'OurTable\_1' AS DataSet

, AVG(Price) AS AvgPrice

FROM OurTable\_1

UNION ALL

SELECT 'OurTable\_2' AS DataSet

, AVG(Price) AS AvgPrice

FROM OurTable\_2

UNION ALL

SELECT 'OurTable\_3' AS DataSet

, AVG(Price) AS AvgPrice

FROM OurTable\_3

With a data set like gold, which fluctuates within a strict range over time except significant uncertainty, a report showing this information may be a check we want to validate, or we could add a validation on top of it (see further in the tip for an example of this). If using SSIS or C#, the same logic applies - we simply have updates during the extracts and loads with the status and whether these succeeded.

If we've collected enough historic metrics on our data sets, we may be able to run audits before a load. Using a similar example to [this tip](https://www.mssqltips.com/sqlservertip/3236/prevalidating-data-with-powershell-before-importing-to-sql-server/), we use StreamReader to quickly collect some metrics on a file's data - in this case, the length of the lines and the number of lines:

Function Get-FileDetails {

Param(

[Parameter(Mandatory=$true)][string]$file

)

Process

{

$readfile = New-Object System.IO.StreamReader($file)

[int]$readfilecount = 1

while (($line = $readfile.ReadLine()) -ne $null)

{

if ($readfilecount -eq 1)

{

Write-Host "Header skipped - this may not apply."

}

else

{

$save = "INSERT INTO FileMetaDetails VALUES (" + $line.Length + " ,$readfilecount)"

Invoke-Sqlcmd -Query $save

}

$readfilecount++

}

}

}

Get-FileDetails -file "E:\ETL\20170103.csv"

We should consider that with some data sets, this would not be helpful - for instance, consider a data set with a possible long description versus a data set with natural gas pricing information and date. With natural gas information, the price and date per line will generally fall in a strict range, such as a length of 10 data characters and 8 price characters (it may be more or less relative to the data), while a description could be up to 8000 or more characters. We should first evaluate whether we can pre-validate our data set under strict data requirements, or whether we should audit the data after importing them. If you need to keep potentially bad data, the above tip shows one way to write out invalid lines.

2 - Scale data extracts and loads horizontally for increased extract or load speed

In some ETL environments, we will have the option to horizontally scale data by a key relative to what's commonly queried. Consider a financial example with 90% searches involving the current tax year while the remaining 10% involve the rest of the tax years. Scaling the data by tax year across multiple servers, databases, or tables may increase extract or load speeds, as new loads involving the current tax year (or past years) would load faster since the source is smaller. A smaller data set also allows us to use extract and load techniques of dropping all indexes prior to importing data and adding them after the data are saved to the table or tables.

In addition to scaling horizontally, also consider using INSERT over UPDATE when possible with some imports. I've seen multiple ETL processes that imported an entirely new data set solely for the purpose of updating *some records* in an original set of data. Let's compare these examples using a list of the process involved with numbers:

Process 1

1. Bulk insert 20,000,000 rows of data
2. Update rows of data from new data to original data set - 1,000,000 total rows
3. Rebuild all indexes on original data set
4. Drop new data set

Process 2

1. Drop original table and create the table again (same schema)
2. Bulk insert 20,000,000 rows of data to the created table
3. Create indexes

If the new data set has all the data in addition to some updated values, and the size of the data model requires little or no time to create, test dropping the table, creating a new table with the same schema, and inserting all of the data. One example I've seen with this is commercial hedging and commodity pricing, where an ETL process simply imports new pricing information with a few updated values without the data model involving complexity. In many of these cases, the second process involves much less time.

3 - Use strict business rules that send alerts or restrict data outside of ranges after a load

If you live in an area where only 20 million distinct houses exist, and you import data showing that 100 million distinct houses exist in the area after one month, the growth of five multiples in one month seems suspicious when you compare it against a past growth rate of about 2-5% a year. With every data set, a business may choose to define rules that require human oversight outside of automation, though human oversight will slow down the data flow if human approval is required to continue. In the below example from [this tip](https://www.mssqltips.com/sqlservertip/4145/using-powershell-for-sql-server-replication-tracking/), a business rule defines that a row mismatch between a source and destination server equal to the outlier based on historical information would be something worth reporting as problematic:

;WITH CatchIssues AS(

SELECT

t.SourceServer

, t.SourceObject

, t.DestinationServer

, t.DestinationObject

, AVG(t.CountDifference) AverageDiff

, STDEV(t.CountDifference) StandardDeviationDiff

, ABS(AVG(t.CountDifference)+(3\*STDEV(t.CountDifference))) Outlier

FROM tbTransRepMismatchHistory t

GROUP BY t.SourceServer, t.SourceObject, t.DestinationServer, t.DestinationObject

)

SELECT

tt.SavedRowCnt

FROM CatchIssues t

INNER JOIN tbCurrentCount tt ON

t.SourceServer = tt.SourceServer

t.SourceObject = tt.SourceObject

t.DestinationServer = tt.DestinationServer

t.DestinationObject = tt.DestinationObject

WHERE tt.SavedRowCnt >= t.Outlier

In some organizations where data are used by multiple applications or tools use the same data sets, or underlying data sets, you may consider master data management and define strict rules for these data on their import point and their flow points. For instance, if you are sharing these data across 5 different servers for 6 applications, the rules for data flow should limit how the data flow, like data can only be inserted, but never removed, or data can be inserted or updated, but never removed. A popular example of master data management is customer data and if a company does business with one customer, it may have requirements to never remove that customer for record keeping purposes; this rule would be defined in data transmission in that a delete would never be performed on any of the destination tables (or source table). In most cases, the metadata of these tables should be identical, such as a record count of customers should return the same for the source as the destination (metadata like the size of a table could differ if compression is used on one table and not the other table):

SELECT COUNT\_BIG(\*)

FROM [SourceLs].[SourceDb].[SourceSs].[SourceTb]

SELECT COUNT\_BIG(\*)

FROM [DestinationLs].[DestinationDb].[DestinationSc].[DestinationTb]

**Next Steps**

* When consider auditing data, think about the past problems that have arisen and what audits are required.
* If auditing during an extract or load, how comprehensive should this be? Should you run aggregates, or would simple counts work?
* For considerations of scale, what are the most common queries from users? What could be a key to shard or partition data by?
* For notification alerts, you can [send emails in tabular form](https://www.mssqltips.com/sqlservertip/2347/send-email-in-a-tabular-format-using-sql-server-database-mail/), [review a report](https://www.mssqltips.com/sqlservertutorial/222/sql-server-reporting-services-ssrs-tutorial/) on a scheduled basis, or even send [emails through PowerShell](https://www.mssqltips.com/sqlservertip/3151/use-powershell-to-send-email-notification-for-sql-server-cluster-failover/).

Last Update: 2017-06-27