**Using T-SQL to Verify Tables Row Counts in Transactional Replication**

*This article will help you to use T-SQL to verify that the Subscribers have the same number of rows of replicated data as the Publisher in transactional replication*

*Written by: Thanh Ngay Nguyen*

*Written on: 7/22/2010*

Although the Microsoft SQL Server Replication Monitor provides detailed information on the status and performance of publications and subscriptions, it still lacks the ability to view details deep down at the article level, e.g. how fast each table is syncing up and if there is any issue, then how far behind replication is. The script that I am about to discuss, will help to answer these questions. My goal for this script is pretty simple: obtaining the rowcount for each replicated table, then verifying that the Subscriber(s) have the same number of rows of data as the Publisher. For documentation purposes, this script also provides a complete picture of your current transactional replication setup.

As part of my DBA daily/hourly routine, I run this script to check how replication is performing. If I notice for a particular table that there is a substantial difference in the rowcount between the Publisher and Subscriber(s), I may want to run the script again and again. If the difference in the rowcounts is getting smaller, then I tell myself that the article will soon catch up. But if the difference keeps getting bigger and bigger every time I run, then "Houston, we might have a problem". Another practical use for this script is to check on the replication of data imports. Let's say you just ran a script five minutes ago to insert a few millions rows of data into dozens of tables on the OLTP server. Now you are asked if those changes have been completely replicated to the OLAP server so business users can pull reports. This script will help you determine how far along the replication is for these data changes.

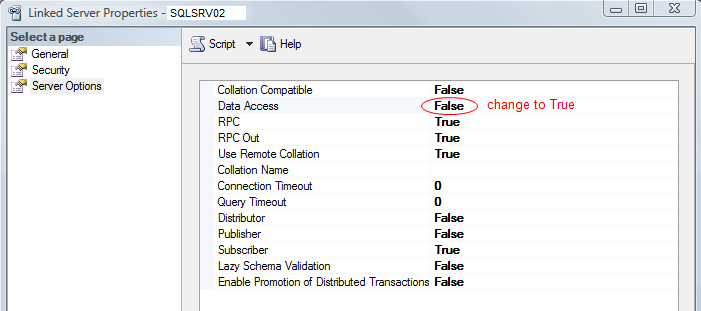
**Audience:** I assume my audience has at least some basic understanding of transactional replication

**Testing:** This script was tested thoroughly in the transactional replication environment that has one Publisher with 10 user databases, which are all replicated to four Subscribers by 30 publications using pull subscriptions. A publication is delivered to multiple Subscribers. The total replicated data is around 1.5 TB. The Publisher also acts as its own Distributor.

This script also works when user objects are replicated to a database which resides on the Publisher server. You might find this a bit odd: Why does one want to replicate data to the same server as its source? Well, for instance, there might be a replicated database on the OLAP server to support web-based application. Later this database was moved to the OLTP server. To save developers' time from modifying code, the database was kept intact and only the connection string at the application end was changed. That is the reason one of the Subscribers and the Publisher ended up on the same SQL instance. Please note, this script does not work when the Distributor and Publisher are on different SQL instances. I leave this scenario to the experts out there to expand the script's capability.

**Where to run this script from:** This script must be run at the SQL instance where both Publisher and Distributor reside.

**Set up linked servers:** When you first setup replication on the Publisher, SQL Server created linked servers to all the Subscribers. These linked servers by default were not configured for "Data Access". In order for this script to work, you need to go to each linked server Properties, under "Server Options", change "Data Access" from False to True



Enough for the talk, let's get right to it, shall we?

To achieve my goal, I divide the script into five simple steps. First, I need to know which objects are currently being replicated. Second, I would like to know the rowcount for the replicated tables at the Publisher side. Third, the rowcount for these replicated tables at the Subscriber(s) side. Fourth, I calculate the difference of rowcount for these replicated tables between Publisher and Subscriber(s) sides. Fifth, I display the final results. I will describe each of these steps in more detail.

**STEP 1: Gather information about current transactional replication**

In this step, I create a temp table *#tempTransReplication* to store detailed information about each article. One article occupies one row in the table. I then populate this *#tempTransReplication* table with details that are queried from the following system tables:

distribution.dbo.MSpublications

distribution.dbo.MSsubscriptions

distribution.dbo.MSarticles

master.sys.servers

Table MSpublications tells me which databases on the OLTP are being replicated and for each replicated database what its publications are. Table MSsubscriptions provides information, such as, which server an article is replicated to (i.e. Subscriber server), and to which database (i.e. Subscriber database). For each article, I can obtain from table MSarticles the following information: article name, schema name, which database it is replicated from, and which publication this article belongs to. In my query, I join table sys.servers twice to retrieve server names of Publisher and its Subscriber(s).

Code Listing for Step 1:

IF OBJECT\_ID('tempdb..#tempTransReplication') IS NOT NULL

DROP TABLE #tempTransReplication

CREATE TABLE #tempTransReplication

(

publisher\_id INT,

publisher\_srv VARCHAR(255),

publisher\_db VARCHAR(255),

publication VARCHAR(255),

subscriber\_id INT,

subscriber\_srv VARCHAR(255),

subscriber\_db VARCHAR(255),

object\_type VARCHAR(255),

source\_owner VARCHAR(255),

source\_object VARCHAR(255),

destination\_owner VARCHAR(255),

destination\_object VARCHAR(255),

rowcount\_publisher INT,

rowcount\_subscriber INT,

rowcount\_diff INT

)

INSERT INTO #tempTransReplication

SELECT s.publisher\_id,

ss2.data\_source,

a.publisher\_db,

p.publication,

s.subscriber\_id,

ss.data\_source,

s.subscriber\_db,

NULL,

a.source\_owner,

a.source\_object,

ISNULL(a.destination\_owner, a.source\_owner), -- if NULL, schema name remains same at subscriber side

a.destination\_object,

NULL,

NULL,

NULL

FROM distribution.dbo.MSarticles AS a

INNER JOIN distribution.dbo.MSsubscriptions AS s

ON a.publication\_id = s.publication\_id

AND a.article\_id = s.article\_id

INNER JOIN [master].sys.servers AS ss

ON s.subscriber\_id = ss.server\_id

INNER JOIN distribution.dbo.MSpublications AS p

ON s.publication\_id = p.publication\_id

LEFT OUTER JOIN [master].sys.servers AS ss2

ON p.publisher\_id = ss2.server\_id

WHERE s.subscriber\_db <> 'virtual'

The reason I filter on s.subscriber\_db <> 'virtual' in the WHERE clause is because obviously I do not want them. If I do not filter them out, they will cause duplicates in my query result. I only want one record per article.

Why are these 'virtual' entries there? Well, when you first set up transactional replication, SQL Server by default turned on both "allow\_anonymous" and "immediate\_sync" flags. These "virtual" entries were created in the distribution.dbo.MSsubscriptions table which have subscriber\_id equal to -1 and -2. SQL Server added them to signify that a publication can have anonymous subscriptions. These "virtual" entries are also used to help snapshot agent decide when and what to clean up. To improve performance in transactional replication, it is recommended to turn these two flags off. To verify if you have these two flags turned on by SQL Server as default, run the query below at the Distributor server instance:

SELECT publisher\_db,

publication,

allow\_anonymous,

immediate\_sync

FROM distribution.dbo.MSpublications

For more information about the two flags "allow\_anonymous" and "immediate\_sync", see this nice article [How to Massively Optimise Transactional Replication](http://www.replicationanswers.com/TransactionalOptimisation.asp) written by Paul Ibison

At this point, if you run query *Select \* from #tempTransReplication*, you will notice that four columns (object\_type, rowcount\_publisher, rowcount\_subscriber and rowcount\_diff) are filled with NULL. In later steps 2 and 3, we will figure out the values of these columns and store them in separated temp tables. At step 4, we will update table #tempTransReplication with these values. At step 5, we will query this table #tempTransReplication for our final results.

**STEP 2: Gather rowcount at Publisher side**

In this step, I create another temp table #tempPublishedArticles to store information about each article at the Publisher side. I am particularly interested in obtaining the rowcount for each replicated table. To do this, I use two system tables from each replicated database:

dbo.sysindexes

sys.objects

To retrieve the rowcount for all replicated tables from each replicated database, I need to dynamically query both *dbo.sysindexes* and *sys.objects* from each database. For this reason, I use dynamic SQL. The dynamic SQL allows me to hop from one database to another at runtime.

Here is the logic inside the cursor: For each replicated database, I pull information about its objects, such as, schema name, object name, object type, rowcount. While all replicated objects are pulled, only the rowcount from tables can be obtained.

Code Listing for Step 2:

IF OBJECT\_ID('tempdb..#tempPublishedArticles') IS NOT NULL

DROP TABLE #tempPublishedArticles

CREATE TABLE #tempPublishedArticles

(

publisher\_db VARCHAR(255),

source\_owner VARCHAR(255),

source\_object VARCHAR(255),

object\_type VARCHAR(255),

rowcount\_publisher INT

)

DECLARE @pub\_db VARCHAR(255),

@strSQL\_P VARCHAR(4000)

DECLARE db\_cursor\_p CURSOR

FOR SELECT DISTINCT

publisher\_db

FROM distribution.dbo.MSpublications

OPEN db\_cursor\_p

FETCH NEXT FROM db\_cursor\_p INTO @pub\_db

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @strSQL\_P = 'SELECT ' + '''' + @pub\_db + ''''

+ ' AS publisher\_db, s.name AS source\_owner, o.name AS source\_object, o.Type\_Desc AS object\_type, i.rowcnt AS rowcount\_publisher

FROM ' + @pub\_db + '.sys.objects AS o

INNER JOIN ' + @pub\_db + '.sys.schemas AS s

on o.schema\_id = s.schema\_id

LEFT OUTER JOIN ' + @pub\_db + '.dbo.sysindexes AS i

on o.object\_id = i.id

WHERE ' + '''' + @pub\_db + ''''

+ ' + ' + '''' + '.' + '''' + ' + s.name'

+ ' + ' + '''' + '.' + '''' + ' + o.name'

+ ' IN (SELECT publisher\_db + ' + '''' + '.' + ''''

+ ' + source\_owner + ' + '''' + '.' + ''''

+ ' + source\_object FROM #tempTransReplication)

AND ISNULL(i.indid, 0) IN (0, 1)

ORDER BY i.rowcnt DESC'

-- heap (indid=0); clustered index (indix=1)

INSERT INTO #tempPublishedArticles

EXEC ( @strSQL\_P

)

FETCH NEXT FROM db\_cursor\_p INTO @pub\_db

END

CLOSE db\_cursor\_p

DEALLOCATE db\_cursor\_p

**STEP 3: Gather rowcount at Subscriber(s) side**

In this step, I create another temp table *#tempSubscribedArticles* and use same strategy as in step 2 to obtain the rowcount for each replicated table on the Subscriber(s) side. The only difference in this step is the script accesses the remote servers via linked servers that we have set up earlier.

Code Listing for Step 3:

IF OBJECT\_ID('tempdb..#tempSubscribedArticles') IS NOT NULL

DROP TABLE #tempSubscribedArticles

CREATE TABLE #tempSubscribedArticles

(

subscriber\_srv VARCHAR(255),

subscriber\_db VARCHAR(255),

destination\_owner VARCHAR(255),

destination\_object VARCHAR(255),

object\_type VARCHAR(255),

rowcount\_subscriber INT

)

DECLARE @sub\_srv VARCHAR(255),

@sub\_db VARCHAR(255),

@strSQL\_S VARCHAR(4000)

DECLARE db\_cursor\_s CURSOR

FOR SELECT DISTINCT

subscriber\_srv,

subscriber\_db

FROM #tempTransReplication

OPEN db\_cursor\_s

FETCH NEXT FROM db\_cursor\_s INTO @sub\_srv, @sub\_db

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @strSQL\_S = 'SELECT ' + '''' + @sub\_srv + ''''

+ ' AS subscriber\_srv, ' + '''' + @sub\_db + ''''

+ ' AS subscriber\_db, '

+ 's.name AS destination\_owner, o.name AS destination\_object, o.Type\_Desc AS object\_type, i.rowcnt AS rowcount\_subscriber

FROM ' + @sub\_srv + '.' + @sub\_db + '.sys.objects AS o

INNER JOIN ' + @sub\_srv + '.' + @sub\_db

+ '.sys.schemas AS s on o.schema\_id = s.schema\_id

LEFT OUTER JOIN ' + @sub\_srv + '.' + @sub\_db

+ '.dbo.sysindexes AS i on o.object\_id = i.id

WHERE ' + '''' + @sub\_srv + '.' + @sub\_db + ''''

+ ' + ' + '''' + '.' + ''''

+ ' + s.name' + ' + ' + '''' + '.' + '''' + ' + o.name'

+ ' IN (SELECT subscriber\_srv + ' + '''' + '.' + ''''

+ ' + subscriber\_db + ' + '''' + '.' + ''''

+ ' + destination\_owner + ' + '''' + '.' + ''''

+ ' + destination\_object FROM #tempTransReplication)

AND ISNULL(i.indid, 0) IN (0, 1)

ORDER BY i.rowcnt DESC'

-- heap (indid=0); clustered index (indix=1)

INSERT INTO #tempSubscribedArticles

EXEC ( @strSQL\_S

)

FETCH NEXT FROM db\_cursor\_s INTO @sub\_srv, @sub\_db

END

CLOSE db\_cursor\_s

DEALLOCATE db\_cursor\_s

**STEP 4: Update table #tempTransReplication with rowcount**

Let's recap what we have done so far. In step 1, I gathered all the information about each replicated article, but I did not yet know about its rowcount at Publisher and Subscriber(s). For each replicated table, step 2 did provide me the rowcount at Publisher side, and step 3 also gave me the rowcount at Subscriber(s) side. In this step 4, I now can go ahead update columns *rowcount\_publisher* and *rowcount\_subscriber* from table *#tempTransReplication*. Column *object\_type* can also be updated with values that were obtained from step 2. I also calculate the *rowcount\_diff* by substracting *rowcount\_subscriber* from *rowcount\_publisher*.

Code Listing for Step 4:

UPDATE t

SET rowcount\_publisher = p.rowcount\_publisher,

object\_type = p.object\_type

FROM #tempTransReplication AS t

INNER JOIN #tempPublishedArticles AS p ON t.publisher\_db = p.publisher\_db

AND t.source\_owner = p.source\_owner

AND t.source\_object = p.source\_object

UPDATE t

SET rowcount\_subscriber = s.rowcount\_subscriber

FROM #tempTransReplication AS t

INNER JOIN #tempSubscribedArticles AS s ON t.subscriber\_srv = s.subscriber\_srv

AND t.subscriber\_db = s.subscriber\_db

AND t.destination\_owner = s.destination\_owner

AND t.destination\_object = s.destination\_object

UPDATE #tempTransReplication

SET rowcount\_diff = ABS(rowcount\_publisher - rowcount\_subscriber)

**STEP 5: Display final results**

Since I am only interested in the replicated tables that fall behind, therefore I filter the *object\_type* to be 'USER\_TABLE' and the *rowcount\_diff* to be greater than 0. The first result displays the sum of *rowcount\_diff* by database. Second result shows the sum of *rowcount\_diff* by publication. The last result displays the sum of *rowcount\_diff* by table.

If all three results happen to return nothing, that means there are no difference in rowcount between Publisher and Subscriber(s). This also means your transactional replication is totally synced up in term of rowcount.

Code Listing for Step 5:

-- rowcount result by replicated database

SELECT publisher\_srv,

publisher\_db,

subscriber\_srv,

subscriber\_db,

sum(rowcount\_diff) AS rowcount\_diff

FROM #tempTransReplication

WHERE object\_type = 'USER\_TABLE' -- tables only

GROUP BY publisher\_srv,

publisher\_db,

subscriber\_srv,

subscriber\_db

HAVING sum(rowcount\_diff) > 0 -- only show those databases which fall behind

ORDER BY rowcount\_diff DESC

-- rowcount result by publication

SELECT publisher\_srv,

publisher\_db,

publication,

subscriber\_srv,

subscriber\_db,

sum(rowcount\_diff) AS rowcount\_diff

FROM #tempTransReplication

WHERE object\_type = 'USER\_TABLE' -- tables only

GROUP BY publisher\_srv,

publisher\_db,

publication,

subscriber\_srv,

subscriber\_db

HAVING sum(rowcount\_diff) > 0 -- only show those publications which fall behind

ORDER BY rowcount\_diff DESC

-- rowcount result by table

SELECT publisher\_srv,

publisher\_db,

subscriber\_srv,

subscriber\_db,

publication,

object\_type,

( source\_owner + '.' + source\_object ) AS source\_objectname,

( destination\_owner + '.' + destination\_object ) AS destination\_objectname,

rowcount\_diff AS rowcount\_diff

FROM #tempTransReplication

WHERE object\_type = 'USER\_TABLE' -- tables only

AND rowcount\_diff > 0 -- only show those tables which fall behind

ORDER BY rowcount\_diff DESC

**SUMMARY:**

I have shown how to use T-SQL to access system tables from both master and distribution databases to obtain information about your current transaction replication. I have also demonstrated how to use system tables from each replicated database to obtain the rowcount for each replicated table, and then verify results between Publisher and Subscriber(s) to make sure these replicated tables are indeed synced-up. As mentioned in the beginning of this article, this script also provides a complete view of your current transactional replication setup. The query below will give you just that!

-- information about current transactional replication setup

SELECT publisher\_srv,

publisher\_db,

subscriber\_srv,

subscriber\_db,

publication,

object\_type,

( source\_owner + '.' + source\_object ) AS source\_objectname,

( destination\_owner + '.' + destination\_object ) AS destination\_objectname,

rowcount\_publisher

FROM #tempTransReplication

ORDER BY publisher\_srv,

publisher\_db,

subscriber\_srv,

subscriber\_db,

publication,

object\_type,

source\_objectname,

rowcount\_publisher