

SQL, Auxiliary table of numbers

Asked 16 years, 4 months ago Modified 2 years, 6 months ago

Viewed 27k times



67



For certain types of sql queries, an auxiliary table of numbers can be very useful. It may be created as a table with as many rows as you need for a particular task or as a user defined function that returns the number of rows required in each query.



What is the optimal way to create such a function?



sql

sql-server

Share

Improve this question

Follow

edited Jun 25, 2013 at 20:04



Taryn

247k ● 57 ● 370 ● 408

asked Aug 14, 2008 at 9:01



VZCZC

9,380 ● 5 ● 55 ● 62

1 Could you explain why you'd do this rather than use a table pre-filled with numbers? – [jammus](#) Aug 14, 2008 at 9:39

12 To fill such a table for example. – [vzczc](#) May 27, 2009 at 10:57

- 5 Not all DBA's and/or 3rd party apps will allow the addition of a permanant table. – [JeffO](#) Feb 25, 2013 at 20:54

Vote for a built-in virtual numbers table feature that does not waste memory and IO at

<https://feedback.azure.com/forums/908035-sql-server/suggestions/32890519-add-a-built-in-table-of-numbers> – [Louis Somers](#) Apr 9, 2019 at 11:56

- 1 @LouisSomers - [it is coming](#) – [Martin Smith](#) Mar 10, 2022 at 15:48

8 Answers

Sorted by:

Highest score (default)



117



Heh... sorry I'm so late responding to an old post. And, yeah, I had to respond because the most popular answer (at the time, the Recursive CTE answer with the link to 14 different methods) on this thread is, ummm... performance challenged at best.



First, the article with the 14 different solutions is fine for seeing the different methods of creating a Numbers/Tally table on the fly but as pointed out in the article and in the cited thread, there's a very important quote...

"suggestions regarding efficiency and performance are often subjective. Regardless of how a query is being used, the physical implementation determines the efficiency of a query. Therefore, rather than relying on biased

guidelines, it is imperative that you test the query and determine which one performs better."

Ironically, the article itself contains many subjective statements and "biased guidelines" such as "*a recursive CTE can generate a number listing **pretty efficiently***" and "*This is **an efficient method** of using WHILE loop from a newsgroup posting by Itzik Ben-Gen*" (which I'm sure he posted just for comparison purposes). C'mon folks... Just mentioning Itzik's good name may lead some poor slob into actually using that horrible method. The author should practice what (s)he preaches and should do a little performance testing before making such ridiculously incorrect statements especially in the face of any scalability.

With the thought of actually doing some testing before making any subjective claims about what any code does or what someone "likes", here's some code you can do your own testing with. Setup profiler for the SPID you're running the test from and check it out for yourself... just do a "Search'n'Replace" of the number 1000000 for your "favorite" number and see...

```
--===== Test for 1000000 rows =====
GO
--===== Traditional RECURSIVE CTE method
WITH Tally (N) AS
(
    SELECT 1 UNION ALL
    SELECT 1 + N FROM Tally WHERE N < 1000000
)
SELECT N
```

```

        INTO #Tally1
        FROM Tally
        OPTION (MAXRECURSION 0);
GO
----- Traditional WHILE LOOP method
CREATE TABLE #Tally2 (N INT);
        SET NOCOUNT ON;
DECLARE @Index INT;
        SET @Index = 1;
        WHILE @Index <= 1000000
        BEGIN
                INSERT #Tally2 (N)
                VALUES (@Index);
                SET @Index = @Index + 1;
        END;
GO
----- Traditional CROSS JOIN table method
SELECT TOP (1000000)
        ROW_NUMBER() OVER (ORDER BY (SELECT 1)) AS N
        INTO #Tally3
        FROM Master.sys.All_Columns ac1
        CROSS JOIN Master.sys.ALL_Columns ac2;
GO
----- Itzik's CROSS JOINED CTE method
        WITH E00(N) AS (SELECT 1 UNION ALL SELECT 1),
        E02(N) AS (SELECT 1 FROM E00 a, E00 b),
        E04(N) AS (SELECT 1 FROM E02 a, E02 b),
        E08(N) AS (SELECT 1 FROM E04 a, E04 b),
        E16(N) AS (SELECT 1 FROM E08 a, E08 b),
        E32(N) AS (SELECT 1 FROM E16 a, E16 b),
        cteTally(N) AS (SELECT ROW_NUMBER() OVER (ORDER BY
SELECT N
        INTO #Tally4
        FROM cteTally
        WHERE N <= 1000000;
GO
----- Housekeeping
        DROP TABLE #Tally1, #Tally2, #Tally3, #Tally4;
GO

```

While we're at it, here's the numbers I get from SQL Profiler for the values of 100, 1000, 10000, 100000, and

1000000...

SPID	TextData	Dur(ms)
51	--===== Test for 100 rows =====	8
51	--===== Traditional RECURSIVE CTE method	16
51	--===== Traditional WHILE LOOP method CR	73
51	--===== Traditional CROSS JOIN table met	11
51	--===== Itzik's CROSS JOINED CTE method	6
51	--===== Housekeeping DROP TABLE #Tally	35
51	--===== Test for 1000 rows =====	0
51	--===== Traditional RECURSIVE CTE method	47
51	--===== Traditional WHILE LOOP method CR	80
51	--===== Traditional CROSS JOIN table met	5
51	--===== Itzik's CROSS JOINED CTE method	2
51	--===== Housekeeping DROP TABLE #Tally	6
51	--===== Test for 10000 rows =====	0
51	--===== Traditional RECURSIVE CTE method	434
51	--===== Traditional WHILE LOOP method CR	671
51	--===== Traditional CROSS JOIN table met	25
51	--===== Itzik's CROSS JOINED CTE method	24
51	--===== Housekeeping DROP TABLE #Tally	7
51	--===== Test for 100000 rows =====	0
51	--===== Traditional RECURSIVE CTE method	4143
51	--===== Traditional WHILE LOOP method CR	5820
51	--===== Traditional CROSS JOIN table met	160
51	--===== Itzik's CROSS JOINED CTE method	153
51	--===== Housekeeping DROP TABLE #Tally	10
51	--===== Test for 1000000 rows =====	0
51	--===== Traditional RECURSIVE CTE method	41349
51	--===== Traditional WHILE LOOP method CR	59138
51	--===== Traditional CROSS JOIN table met	1224
51	--===== Itzik's CROSS JOINED CTE method	1448
51	--===== Housekeeping DROP TABLE #Tally	8

As you can see, **the Recursive CTE method is the second worst only to the While Loop for Duration and**

CPU and has 8 times the memory pressure in the form of logical reads than the While Loop. It's RBAR on steroids and should be avoided, at all cost, for any single row calculations just as a While Loop should be avoided. **There are places where recursion is quite valuable but this ISN'T one of them.**

As a side bar, Mr. Denny is absolutely spot on... a correctly sized permanent Numbers or Tally table is the way to go for most things. What does correctly sized mean? Well, most people use a Tally table to generate dates or to do splits on VARCHAR(8000). If you create an 11,000 row Tally table with the correct clustered index on "N", you'll have enough rows to create more than 30 years worth of dates (I work with mortgages a fair bit so 30 years is a key number for me) and certainly enough to handle a VARCHAR(8000) split. Why is "right sizing" so important? If the Tally table is used a lot, it easily fits in cache which makes it blazingly fast without much pressure on memory at all.

Last but not least, every one knows that if you create a permanent Tally table, it doesn't much matter which method you use to build it because 1) it's only going to be made once and 2) if it's something like an 11,000 row table, all of the methods are going to run "good enough". **So why all the indignation on my part about which method to use???**

The answer is that some poor guy/gal who doesn't know any better and just needs to get his or her job done might

see something like the Recursive CTE method and decide to use it for something much larger and much more frequently used than building a permanent Tally table and I'm trying to **protect those people, the servers their code runs on, and the company that owns the data on those servers**. Yeah... it's that big a deal. It should be for everyone else, as well. Teach the right way to do things instead of "good enough". Do some testing before posting or using something from a post or book... the life you save may, in fact, be your own especially if you think a recursive CTE is the way to go for something like this. ;-)

Thanks for listening...

Share Improve this answer

edited Apr 18, 2010 at 17:53

Follow

answered Apr 18, 2010 at 17:44



Jeff Moden

3,485 ● 2 ● 28 ● 26

2 I really really wish more people have your sense of social responsibility. Have said that and apart one'd need once to populate a Numbers table for all kind of stuff, if need for some reason, [it seems SELECT INTO w/ IDENTITY is faster than CTE](#). – Andre Figueiredo Jul 16, 2017 at 3:09

Thank you for the very kind feedback, Andre. – Jeff Moden Jul 23, 2017 at 0:31

You don't know that CTEs are dreadfully slow, that they are rendered using temp tables ? And that this requirement can

be fulfilled without CTE ? Of course, without that knowledge, CTEs are marvellous. – [PerformanceDBA](#) Aug 23, 2022 at 4:36 ✎

- 1 Actually, I do know that they're dreadfully slow. I wrote an article about it a long time ago. Please feel free to read it. [sqlservercentral.com/articles/...](#) Now, that was back in the day of less modern machines and, while newer machines have gotten faster, rCTE still use 8X more reads than a WHILE loop and a WHILE loop in a transaction will still beat it for speed. Like I said, "dreadfully" slow and hardly ever "marvelous". Learn the right way to do such things.
– [Jeff Moden](#) Aug 24, 2022 at 0:19 ✎
-



12



The most optimal function would be to use a table instead of a function. Using a function causes extra CPU load to create the values for the data being returned, especially if the values being returned cover a very large range.

Share Improve this answer

answered Sep 2, 2008 at 9:48

Follow



[mrdenny](#)

5,078 ● 2 ● 22 ● 31


- 1 I think it depends then on your situation. Between the two best performing options, you can trade between IO and CPU costs, depending on what is more expensive for you.
– [Robert Cutajar](#) Jun 18, 2013 at 10:19
-

IO will almost always be cheaper than CPU, especially as this table would be small and probably already in budferpool.
– [mrdenny](#) Jun 18, 2013 at 11:53

- 1 @mrdenny I/O is always way more expensive and slower than CPU. SSDs have changed this somewhat in recent years, but in most production architectures those SSDs have

a network link between them and the CPUs. The only databases I see that are truly CPU bound are running untuned ORM-only apps or heavy machine learning.

– [rmaalayter](#) Nov 21, 2017 at 4:25 

@rmaalayter except if the table is used often enough for us to care, it will almost certainly be in memory, and memory is cheaper to upgrade and usually doesn't impact licensing the way adding CPU cores can. SQL Server Enterprise edition is going to be in the ball park of a 5 digit number PER CORE, i.e. adding cores will probably cost you more in licensing alone than the entire cost out the door of throwing more ram in the server. – [Dogs](#) Feb 14, 2019 at 16:31 



[This article](#) gives 14 different possible solutions with discussion of each. The important point is that:

5



suggestions regarding efficiency and performance are often subjective. Regardless of how a query is being used, the physical implementation determines the efficiency of a query. Therefore, rather than relying on biased guidelines, it is imperative that you test the query and determine which one performs better.



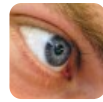
I personally liked:

```
WITH Nbrs ( n ) AS (
    SELECT 1 UNION ALL
    SELECT 1 + n FROM Nbrs WHERE n < 500 )
SELECT n FROM Nbrs
OPTION ( MAXRECURSION 500 )
```

Share Improve this answer

edited Apr 6, 2020 at 12:04

Follow



Vadzim

26.1k ● 14 ● 150 ● 159

answered Sep 25, 2009 at 19:50



Chadwick

12.7k ● 7 ● 51 ● 66

-
- 3 Proven wrong by the accepted answer? It is not 'optimal', though it looks handsome. – [Robert Cutajar](#) Jun 18, 2013 at 10:17
-



This view is super fast and contains all positive `int` values.

4



```
CREATE VIEW dbo.Numbers
WITH SCHEMABINDING
AS
```

```
    WITH Int1(z) AS (SELECT 0 UNION ALL SELECT 0)
    , Int2(z) AS (SELECT 0 FROM Int1 a CROSS JOIN Int1)
    , Int4(z) AS (SELECT 0 FROM Int2 a CROSS JOIN Int2)
    , Int8(z) AS (SELECT 0 FROM Int4 a CROSS JOIN Int4)
    , Int16(z) AS (SELECT 0 FROM Int8 a CROSS JOIN Int8)
    , Int32(z) AS (SELECT TOP 2147483647 0 FROM Int16)
    SELECT ROW_NUMBER() OVER (ORDER BY z) AS n
FROM Int32
```

```
GO
```

Share Improve this answer

answered Jul 4, 2011 at 12:24

Follow



Anthony Faull

17.9k ● 5 ● 56 ● 75

-
- 1 `0` is often useful. And I would probably convert the final column to `int`. Also you should know that basically the

method is included in the accepted answer (without `0` or conversion to `int` either) by the name of *Itzik's CROSS JOINED CTE method*. – [Andriy M](#) Jul 4, 2011 at 21:02

Any particular reason to add `WITH SCHEMABINDING` in the view? – [ca9163d9](#) Feb 27, 2012 at 22:12

Adding 'WITH SCHEMABINDING' can make queries faster. It helps the optimizer know that no data is accessed. (See blogs.msdn.com/b/sqlprogrammability/archive/2006/05/12/...) – [Anthony Faull](#) Feb 28, 2012 at 8:07

I wonder if @AnthonyFaull can back this up with some measurements. – [Robert Cutajar](#) Jun 18, 2013 at 10:22



From SQL Server 2022 [you will be able to do](#)

3

```
SELECT Value
FROM GENERATE_SERIES(START = 1, STOP = 100, STEP=1)
```



In the public preview of SQL Server 2022 (CTP2.0) there are some very promising elements and other less so. Hopefully the negative aspects can be addressed before the actual release.

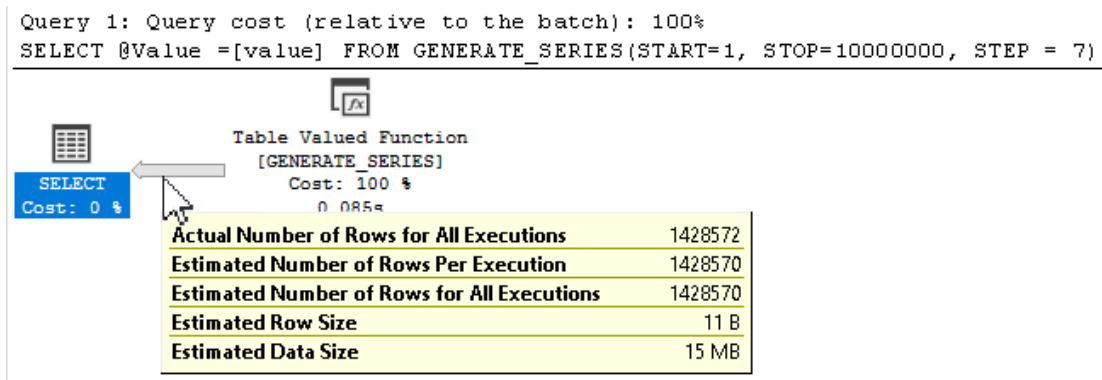
✅ **Execution time for number generation** The below generates 10,000,000 numbers in 700 ms in my test VM (the assigning to a variable removes any overhead from sending results to the client)

```
DECLARE @Value INT
```

```
SELECT @Value =[value]
FROM GENERATE_SERIES(START=1, STOP=10000000)
```

✓ Cardinality estimates

It is simple to calculate how many numbers will be returned from the operator and SQL Server takes advantage of this as shown below.



✗ Unnecessary Halloween Protection

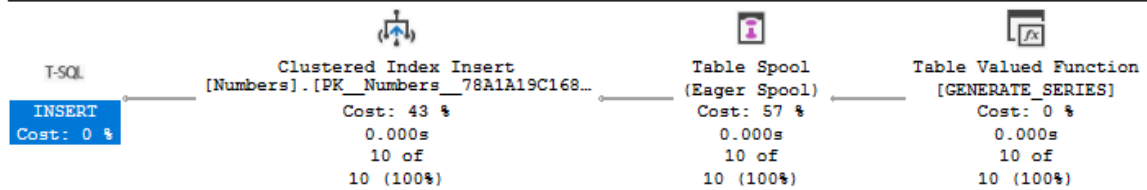
The plan for the below insert has a completely unnecessary spool - presumably as SQL Server does not currently have logic to determine the source of the rows is not potentially the destination.

```
CREATE TABLE dbo.NumberHeap(Number INT);

INSERT INTO dbo.Numbers
SELECT [value]
FROM GENERATE_SERIES(START=1, STOP=10);
```

When inserting into a table with a clustered index on Number the spool may be replaced by a sort instead (that also provides the phase separation)

```
INSERT INTO dbo.Numbers SELECT [value] FROM GENERATE_SERIES(START=1, STOP=10)
```

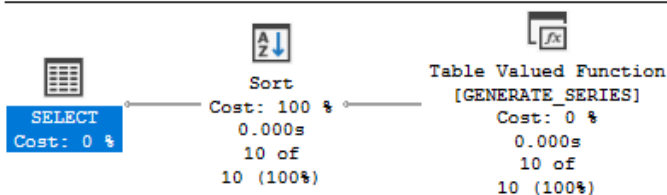


✗ Unnecessary sorts

The below will return the rows in order anyway but SQL Server apparently does not yet have the properties set to guarantee this and take advantage of it in the execution plan.

```
SELECT [value]
FROM GENERATE_SERIES(START=1, STOP=10)
ORDER BY [value]
```

```
SELECT [value] FROM GENERATE_SERIES(START=1, STOP=10) ORDER BY [value]
```



RE: This last point [Aaron Bertrand indicates](#) that this is not a box currently ticked but that this may be forthcoming.

Share Improve this answer

edited May 25, 2022 at 21:01

Follow

answered Mar 10, 2022 at 15:47



Martin Smith

452k ● 94 ● 767 ● 870



1



Using `SQL Server 2016+` to generate numbers table you could use `OPENJSON` :

```
-- range from 0 to @max - 1
DECLARE @max INT = 40000;

SELECT rn = CAST([key] AS INT)
FROM OPENJSON(CONCAT('[1', REPLICATE(CAST(',',1' AS VARCHAR
```

[LiveDemo](#)

Idea taken from [How can we use OPENJSON to generate series of numbers?](#)

Share Improve this answer

edited May 2, 2016 at 18:10

Follow

answered May 2, 2016 at 16:29



[Lukasz Szozda](#)

175k ● 25 ● 267 ● 310

1 Nice. I guess, one could have used XML similarly to this if `position()` had been fully supported in SQL Server's XQuery. – [Andriy M](#) May 2, 2016 at 18:06

1 Sorry for the late comment but that code uses 11.4 times more CPU and infinitely more logical reads (2,000,023) than Itik's cascading CTE method. – [Jeff Moden](#) Sep 5, 2018 at 18:01

edit: see Conrad's comment below.



0



Jeff Moden's answer is great ... but I find on Postgres that the Itzik method fails unless you remove the E32 row.

Slightly faster on postgres (40ms vs 100ms) is another method I found on [here](#) adapted for postgres:

```
WITH
  E00 (N) AS (
    SELECT 1 UNION ALL SELECT 1 UNION ALL SELECT 1
    ALL SELECT 1 UNION ALL
    SELECT 1 UNION ALL SELECT 1 UNION ALL SELECT 1
    ALL SELECT 1 ),
  E01 (N) AS (SELECT a.N FROM E00 a CROSS JOIN E00 b
  E02 (N) AS (SELECT a.N FROM E01 a CROSS JOIN E01 b
  E03 (N) AS (SELECT a.N FROM E02 a CROSS JOIN E02 b
    LIMIT 11000 -- end record 11,000 good for 30
  ), -- max is 100,000,000, starts slowing e.g. 1 mi
secs, 3 mill 4 secs
  Tally (N) as (SELECT row_number() OVER (ORDER BY a

SELECT N
FROM Tally
```

As I am moving from SQL Server to Postgres world, may have missed a better way to do tally tables on that platform ... INTEGER()? SEQUENCE()?

Share Improve this answer

edited Jun 24, 2013 at 14:55

Follow

answered Jan 20, 2011 at 9:28



Ruskin

6,151 ● 4 ● 48 ● 65

2 *may have missed a better way to do tally tables on [postgres]*
Yeah you did [generate_series](#) – [Conrad Frix](#) Jan 24, 2013 at 21:07 ✎

@Conrad Frix , Apologies for the very late question (more than 5 years late) but have you done any performance testing to compare that great built in tool with other methods?
– [Jeff Moden](#) Sep 5, 2018 at 18:02 ✎

@JeffModen Sorry no, but it's easy to test. Take Ruskin's query and compare it to call to generate series. – [Conrad Frix](#) Sep 5, 2018 at 20:06

@Conrad Frix , since you made the claim of performance and you have access to both environments (which I don't) and you do also claim it's easy to test, I was hoping you'd take the time to test it. ;-) – [Jeff Moden](#) Sep 6, 2018 at 22:39

1 @Conrad Frix, Heh... you already have it setup and you can't take 5 minutes to test your own claim of performance. NP. Moving on, – [Jeff Moden](#) Sep 8, 2018 at 0:31



0

Still much later, I'd like to contribute a slightly different 'traditional' CTE (does not touch base tables to get the volume of rows):



```
--===== Hans CROSS JOINED CTE method
WITH Numbers_CTE (Digit)
AS
(SELECT 0 UNION ALL SELECT 1 UNION ALL SELECT 2 UNION
SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6 UNION A
SELECT 8 UNION ALL SELECT 9)
SELECT HundredThousand.Digit * 100000 + TenThousand.Di
Thousand.Digit * 1000 + Hundred.Digit * 100 + Ten.Digi
Number
INTO #Tally5
FROM Numbers_CTE AS One CROSS JOIN Numbers_CTE AS Ten
```


Hundred [CROSS JOIN](#) Numbers_CTE [AS](#) Thousand [CROSS JOIN](#)
TenThousand [CROSS JOIN](#) Numbers_CTE [AS](#) HundredThousand

This CTE performs more READs then Itzik's CTE but less then the Traditional CTE. **However, it consistently performs less WRITES then the other queries.** As you know, Writes are consistently quite much more expensive then Reads.

The duration depends heavily on the number of cores (MAXDOP) but, on my 8core, performs consistently quicker (less duration in ms) then the other queries.

I am using:

```
Microsoft SQL Server 2012 - 11.0.5058.0 (X64)  
May 14 2014 18:34:29  
Copyright (c) Microsoft Corporation  
Enterprise Edition (64-bit) on Windows NT 6.3 <X64> (B
```

on Windows Server 2012 R2, 32 GB, Xeon X3450
@2.67Ghz, 4 cores HT enabled.

[Share](#) [Improve this answer](#)

answered Oct 22, 2014 at 10:06

[Follow](#)



[HansLindgren](#)

369 ● 2 ● 9
