When should I use a table variable vs temporary table in sql server?

Asked 12 years, 4 months ago Modified 5 years, 5 months ago

Viewed 259k times

345

I'm learning more details in table variable. It says that temp tables are always on disk, and table variables are in memory, that is to say, the performance of table variable is better than temp table because table variable uses less IO operations than temp table.

But sometimes, if there are too many records in a table variable that can not be contained in memory, the table variable will be put on disk like the temp table.

But I don't know what the "too many records" is. 100,000 records? or 1000,000 records? How can I know if a table variable I'm using is in memory or is on disk? Is there any function or tool in SQL Server 2005 to measure the scale of the table variable or letting me know when the table variable is put on disk from memory?

sql-server temp-tables table-variable

Share

Improve this question

Follow

edited Jan 21, 2016 at 10:15

Luke Girvin

13.4k ● 10 ● 67 ● 85

asked Aug 8, 2012 at 4:46

yman

3,515 ● 3 ● 16 ● 9

8 A table variable is almost always in `tempDB` - that "in memory" is a myth. Also: table variables will always be regarded by the query optimizer to hold exactly one row - if you have a lot more, this can lead to seriously bad execution plans. – marc_s Aug 8, 2012 at 4:50

1 You may find this helpful stackoverflow.com/questions/27894/… – Igor Borisenko Aug 8, 2012 at 5:27

2 @marc_s - You can drop the "almost" in that statement. It is always in `tempdb` (but may also be entirely in memory) – Martin Smith Dec 8, 2012 at 13:34

3 With SQL 2014 you can now create a table variable in memory – paparazzo May 13, 2014 at 15:14

6 Answers

Sorted by: Highest score (default)

401

Your question shows you have succumbed to some of the common misconceptions surrounding table variables and temporary tables.

I have written quite an extensive answer on the DBA site looking at the differences between the two object types. This also addresses your question about disk vs memory (I didn't see any significant difference in behaviour between the two).

Regarding the question in the title though as to when to use a table variable vs a local temporary table you don't always have a choice. In functions, for example, it is only possible to use a table variable and if you need to write to the table in a child scope then only a `#temp` table will do (table-valued parameters allow readonly access).

Where you do have a choice some suggestions are below (though the most reliable method is to simply test both with your specific workload).

1. If you need an index that cannot be created on a table variable then you will of course need a `#temporary` table. The details of this are version dependant however. For SQL Server 2012 and below the only indexes that could be created on table variables were those implicitly created through a `UNIQUE` or `PRIMARY KEY` constraint. SQL Server 2014 introduced inline index syntax for a subset of the options available in `CREATE INDEX`. This has been extended since to allow filtered index conditions. Indexes with `INCLUDE`-d columns or columnstore indexes are still not possible to create on table variables however.
2. If you will be repeatedly adding and deleting large numbers of rows from the table then use a `#temporary` table. That supports `TRUNCATE` (which is more efficient than `DELETE` for large tables) and additionally subsequent inserts following a `TRUNCATE` can have better performance than those following a `DELETE` as illustrated here.

3. If you will be deleting or updating a large number of rows then the temp table may well perform much better than a table variable - if it is able to use rowset sharing (see "Effects of rowset sharing" below for an example).
4. If the optimal plan using the table will vary dependent on data then use a `#temporary` table. That supports creation of statistics which allows the plan to be dynamically recompiled according to the data (though for cached temporary tables in stored procedures the recompilation behaviour needs to be understood separately).
5. If the optimal plan for the query using the table is unlikely to ever change then you may consider a table variable to skip the overhead of statistics creation and recompiles (would possibly require hints to fix the plan you want).
6. If the source for the data inserted to the table is from a potentially expensive `SELECT` statement then consider that using a table variable will block the possibility of this using a parallel plan.
7. If you need the data in the table to survive a rollback of an outer user transaction then use a table variable. A possible use case for this might be logging the progress of different steps in a long SQL batch.
8. When using a `#temp` table within a user transaction locks can be held longer than for table variables (potentially until the end of transaction vs end of

statement dependent on the type of lock and isolation level) and also it can prevent truncation of the `tempdb` transaction log until the user transaction ends. So this might favour the use of table variables.

9. Within stored routines, both table variables and temporary tables can be cached. The metadata maintenance for cached table variables is less than that for `#temporary` tables. Bob Ward points out in his `tempdb` presentation that this can cause additional contention on system tables under conditions of high concurrency. Additionally, when dealing with small quantities of data this can make a measurable difference to performance.

Effects of rowset sharing

```
DECLARE @T TABLE(id INT PRIMARY KEY, Flag BIT);

CREATE TABLE #T (id INT PRIMARY KEY, Flag BIT);

INSERT INTO @T
output inserted.* into #T
SELECT TOP 1000000 ROW_NUMBER() OVER (ORDER BY @@SPID)
FROM master..spt_values v1, master..spt_values v2

SET STATISTICS TIME ON

/*CPU time = 7016 ms,  elapsed time = 7860 ms.*/
UPDATE @T SET Flag=1;

/*CPU time = 6234 ms,  elapsed time = 7236 ms.*/
DELETE FROM @T

/* CPU time = 828 ms,  elapsed time = 1120 ms.*/
UPDATE #T SET Flag=1;

/*CPU time = 672 ms,  elapsed time = 980 ms.*/
```

```
DELETE FROM #T

DROP TABLE #T
```

Share Improve this answer Follow

edited Jul 2, 2019 at 10:13

answered Dec 8, 2012 at 13:20

Martin Smith

452k ● 94 ● 767 ● 870

2 Hi, Mister Martin Smith. In mi case I just want to store a set of Ids values to use them in others query inside the Store procedure. So what do you recommend me? – Jeancarlo Fontalvo Nov 9, 2016 at 22:53

@JeancarloFontalvo - a table variable with a primary key on `id` and use of `OPTION (RECOMPILE)` would probably be fine for that - but test both. – Martin Smith Feb 18, 2018 at 15:33

is the metadata contention same for both temp table and table variable? – Aqeel Ashiq Apr 6, 2018 at 6:12

@Syed. Generally less for TV. Locks can be released earlier if inside a user transaction . Also see the Bob Ward link. – Martin Smith Apr 6, 2018 at 6:50

86

Use a **table variable** if for a very small quantity of data (thousands of bytes)

Use a **temporary table** for a lot of data

Another way to think about it: if you think you might benefit from an index, automated statistics, or any SQL optimizer goodness, then your data set is probably too large for a table variable.

In my example, I just wanted to put about 20 rows into a format and modify them as a group, before using them to UPDATE / INSERT a permanent table. So a table variable is perfect.

But I am also running SQL to back-fill thousands of rows at a time, and I can definitely say that the temporary tables perform *much* better than table variables.

This is not unlike how CTE's are a concern for a similar size reason - if the data in the CTE is very small, I find a CTE performs as good as or better than what the optimizer comes up with, but if it is quite large then it hurts you bad.

My understanding is mostly based on http://www.developerfusion.com/article/84397/table-variables-v-temporary-tables-in-sql-server/, which has a lot more detail.

edited Jul 6, 2016 at 16:21

Mina Gabriel

25k ● 26 ● 99 ● 128

answered Jan 22, 2013 at 18:06

Abacus

2,099 ● 1 ● 18 ● 23

The takeaway is table variable is fine for small dataset, but use temp table for bigger dataset. I have a query with thousands of rows. By switching from table variable to temp table, the query time gets down from 40s to just 5s with everything else being equal. – liang Jan 10, 2019 at 3:06

46

Microsoft says here

> Table variables does not have distribution statistics, they will not trigger recompiles. Therefore, in many cases, the optimizer will build a query plan on the assumption that the table variable has no rows. For this reason, you should be cautious about using a table variable if you expect a larger number of rows (greater than 100). Temp tables may be a better solution in this case.

Share Improve this answer

Follow

answered Feb 26, 2014 at 19:56

Paul Sturm

2,128 ● 2 ● 18 ● 23

15

I totally agree with Abacus (sorry - don't have enough points to comment).

Also, keep in mind it doesn't necessarily come down to *how many* records you have, but the *size* of your records.

For instance, have you considered the performance difference between 1,000 records with 50 columns each vs 100,000 records with only 5 columns each?

Lastly, maybe you're querying/storing more data than you need? Here's a good read on SQL optimization strategies. Limit the amount of data you're pulling, especially if you're not using it all (some SQL programmers do get lazy and just select everything even though they only use a tiny subset). Don't forget the SQL query analyzer may also become your best friend.

Share Improve this answer Follow

answered Jul 7, 2014 at 3:24

user3810900

4

Variable table is available only to the current session, for example, if you need to `EXEC` another stored procedure within the current one you will have to pass the table as `Table Valued Parameter` and of course this will affect the performance, with **temporary tables** you can do this with only passing the temporary table name

To test a Temporary table:

- Open management studio query editor
- Create a temporary table
- Open another query editor window
- Select from this table "Available"

To test a Variable table:

- Open management studio query editor
- Create a Variable table
- Open another query editor window
- Select from this table "Not Available"

something else I have experienced is: If your schema doesn't have `GRANT` privilege to create tables then use variable tables.

Share Improve this answer

Follow

edited Nov 15, 2018 at 12:06

Martijn Pieters

1.1m ● 319 ● 4.2k ● 3.4k

answered Jul 6, 2016 at 16:20

Mina Gabriel

25k ● 26 ● 99 ● 128

3

writing data in tables declared `declare @tb` and after joining with other tables, I realized that the response time compared to temporary tables `tempdb .. # tb` is much higher.

When I join them with *@tb* the time is much longer to return the result, unlike *#tm*, the return is almost instantaneous.

I did tests with a 10,000 rows join and join with 5 other tables

Share Improve this answer

Follow

edited Jul 14, 2017 at 15:13

Sam Oneła

8,295 ● 8 ● 37 ● 60

answered Jul 14, 2017 at 14:38

César Augusto

724 ● 1 ● 8 ● 9

1 Could you post the test you ran to get these figures?
– Dan Def Jul 17, 2017 at 7:44