SQL SERVER Performance Tuning Notes

http://msbiskills.com/

Which one is better | Temp Table or Table Variable?

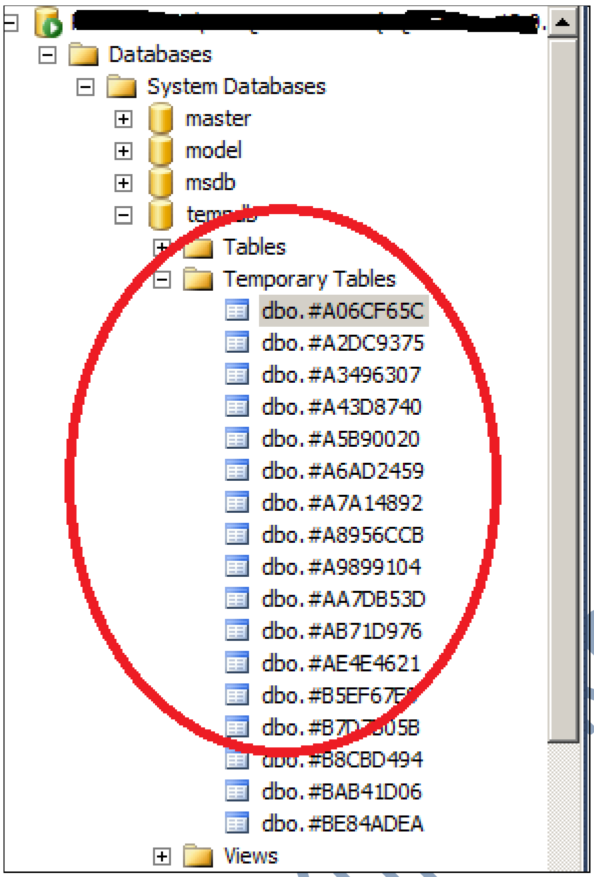
Whether to use temp tables or table variables is an old interesting question since they were first introduced. In this post we will touch-base the differences between the table variable and Temp Tables.

Let's check out the differences below then we will see which is better for performance.

Detail	Temp Tables (#)	Table Variables(@)
Storage Location	Stored in TempDB	Stored in TempDB
Indexes	Yes. Temp table allows both clustered and non-clustered indexes	1 PK/UK Only
Truncate Table	Yes	No
Alter Table	Yes	No
SELECT INTO	Yes	No
Types	Yes - Local and Global	No
Transaction participation	Yes	No
Log file writing	Yes	Yes
Insert with Exec	Yes	Yes
Statistics	Yes	No(1 Row always)
Parallelism participation	Yes	No
Recompiles	Yes	No

Temp tables are like normal SQL tables that are defined and stored in TempDB. The only difference between Temp table and a physical table is that temp table doesn't allow foreign keys.

You can view the temp tables currently defined via SSMS by going to TempDB and Expanding Temp Tables.



```
[redacted]
  Databases
    System Databases
      master
      model
      msdb
      tempdb
        Tables
        Temporary Tables
          dbo.#A06CF65C
          dbo.#A2DC9375
          dbo.#A3496307
          dbo.#A43D8740
          dbo.#A5B90020
          dbo.#A6AD2459
          dbo.#A7A14892
          dbo.#A8956CCB
          dbo.#A9899104
          dbo.#AA7DB53D
          dbo.#AB71D976
          dbo.#AE4E4621
          dbo.#B5EF67E[illegible]
          dbo.#B7D[illegible]05B
          dbo.#B8CBD494
          dbo.#BAB41D06
          dbo.#BE84ADEA
        Views
```

Performance Considerations

As per my experience temp tables are better than table variables. The problem with the table variables is that query optimizer will generate bad/unpredictable query plans as they don't have statistics on them.

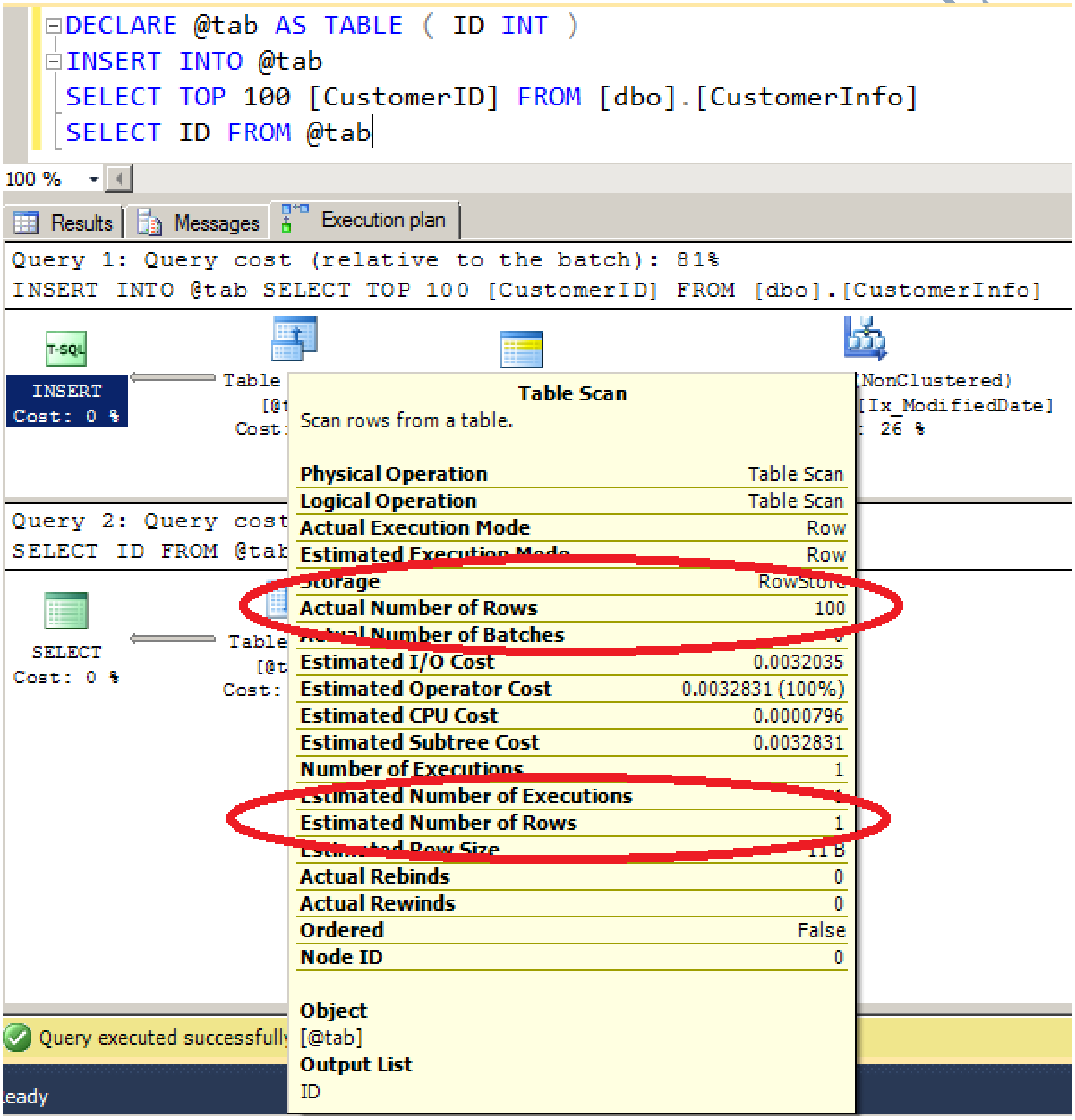
If you check the estimated number of rows then it will always be 1.

```
DECLARE @tab AS TABLE ( ID INT )

INSERT INTO @tab
SELECT TOP 100 [CustomerID] FROM [dbo].[CustomerInfo]

SELECT ID FROM @tab

Output
-----------
```



```
DECLARE @tab AS TABLE ( ID INT )
INSERT INTO @tab
SELECT TOP 100 [CustomerID] FROM [dbo].[CustomerInfo]
SELECT ID FROM @tab
```

100 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 81%
INSERT INTO @tab SELECT TOP 100 [CustomerID] FROM [dbo].[CustomerInfo]
```

```
Query 2: Query cost
SELECT ID FROM @tab
```

Table Scan

Scan rows from a table.

Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	100
Actual Number of Batches	0
Estimated I/O Cost	0.0032035
Estimated Operator Cost	0.0032831 (100%)
Estimated CPU Cost	0.0000796
Estimated Subtree Cost	0.0032831
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	1
Estimated Row Size	11 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	0

Object
[@tab]
Output List
ID

Query executed successfully

eady

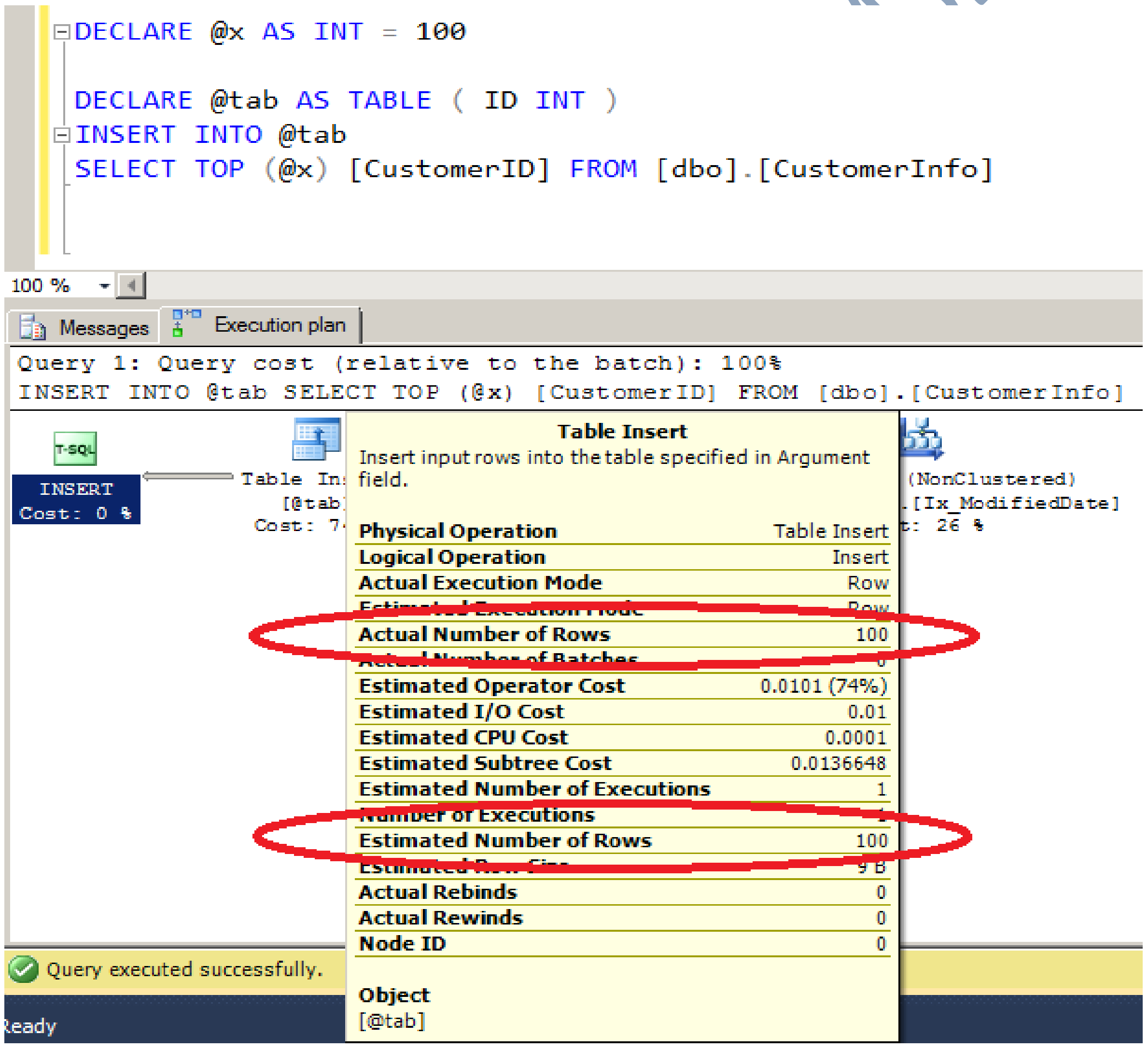
As a general rule of thumb across many SQL SERVER communities is that if you have 1000 rows or less then go for table variable otherwise go for temp table.

Well we can influence the query optimizer. One of the ways is given below-

```
DECLARE @x AS INT = 100

DECLARE @tab AS TABLE ( ID INT )
INSERT INTO @tab
SELECT TOP (@x) [CustomerID] FROM [dbo].[CustomerInfo]

Output
-----------
```



```
DECLARE @x AS INT = 100

DECLARE @tab AS TABLE ( ID INT )
INSERT INTO @tab
SELECT TOP (@x) [CustomerID] FROM [dbo].[CustomerInfo]
```

```
Query 1: Query cost (relative to the batch): 100%
INSERT INTO @tab SELECT TOP (@x) [CustomerID] FROM [dbo].[CustomerInfo]
```

Table Insert

Insert input rows into the table specified in Argument field.

Physical Operation	Table Insert
Logical Operation	Insert
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows	100
Actual Number of Batches	0
Estimated Operator Cost	0.0101 (74%)
Estimated I/O Cost	0.01
Estimated CPU Cost	0.0001
Estimated Subtree Cost	0.0136648
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	100
Estimated Row Size	9 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	0

Object
[@tab]

Table variables do not qualifying for parallelism that's why they are better suited for small amounts of data.

Table Variables can write to the disk if threshold goes over a certain number of records.

Table variables do write to the log file and they can have non clustered indexes if associated with a 'NONCLUSTERED UNIQUE' or 'NONCLUSTERED PRIMARY KEY' constraint