# Table Variable

## Overview:

Table Variable Is a special data type that can be used to store a result set for processing at a later time. T**able** variables provide the following benefits for small-scale queries that have query plans that do not change and when recompilation concerns are dominant:

**Syntax:**

Declare @csustomer table (CustomerID nchar(5) NOT NULL)

### Benefits:

1. A **table** variable behaves like a local variable. It has a well-defined scope. This is the function, stored procedure, or batch that it is declared in. Within its scope, a **table** variable can be used like a regular table. It may be applied anywhere a table or table expression is used in SELECT, INSERT, UPDATE, and DELETE statements.
2. **Table** variables are automatically cleaned up at the end of the function, stored procedure, or batch in which they are defined. T**able** variables used in stored procedures cause fewer recompilations of the stored procedures than when temporary tables are used when there are no cost-based choices that affect performance.
3. Transactions involving **table** variables last only for the duration of an update on the **table** variable. Therefore, **table** variables require less locking and logging resources.

### Limitations and restrictions

1. **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. Alternatively, for queries that join the table variable with other tables, use the RECOMPILE hint, which will cause the optimizer to use the correct cardinality for the table variable.
2. **Table** variables are not supported in the SQL Server optimizer's cost-based reasoning model. Therefore, they should not be used when cost-based choices are required to achieve an efficient query plan. Temporary tables are preferred when cost-based choices are required. This typically includes queries with joins, parallelism decisions, and index selection choices.
3. Queries that modify **table** variables do not generate parallel query execution plans. Performance can be affected when very large **table** variables, or **table** variables in complex queries, are modified. In these situations, consider using temporary tables instead.
4. Queries that read **table** variables without modifying them can still be parallelized.
5. Indexes cannot be created explicitly on **table** variables, and no statistics are kept on **table** variables. In some cases, performance may improve by using temporary tables instead, which support indexes and statistics.
6. CHECK constraints, DEFAULT values and computed columns in the table type declaration cannot call user-defined functions.
7. Assignment operation between **table** variables is not supported.
8. Because **table** variables have limited scope and are not part of the persistent database, they are not affected by transaction rollbacks.   
   Table variables cannot be altered after creation.