**Auto Generate Create Table Script Based on SQL Server Query**

By: [Daniel Farina](https://www.mssqltips.com/sqlserverauthor/111/daniel-farina/)

FROM: <https://www.mssqltips.com/sqlservertip/5699/auto-generate-create-table-script-based-on-sql-server-query/>

**Problem**

You have to create a SQL Server table to store the results from a query. At first you think about looking at each column data type to create the table, but realize it will be a tedious task. In this tip we look at a function you can use to generate a create table script that has the correct data types for each column based on the source columns.

**Solution**

We as SQL Server database professionals write queries each day and some circumstances we need the query to store the data in a table so we can do further analysis or we need to get the data from the query from different sources into one common table. In such case, we can’t use a SELECT INTO statement to create the destination table. This is especially important in case we are working inside a stored procedure that uses a temporary table that needs to be populated with a SELECT statement twice.

For the purposes of this tip, I will use the query from my previous tip [Create SQL Server Disk Space Report for All Servers](https://www.mssqltips.com/sqlservertip/5040/create-sql-server-disk-space-report-for-all-servers/) as an example.

T-SQL Script to Generate a Table based on a Query

During the course of this tip we will go through the process of creating a scalar function that takes a SELECT statement as a parameter and returns the CREATE TABLE script for the query, so you can pass it as a parameter to an EXEC or sp\_executesql statement.

In order to achieve this, we will be using the Dynamic Management Function sys.dm\_exec\_describe\_first\_result\_set.

The DMF sys.dm\_exec\_describe\_first\_result\_set was introduced in SQL Server 2012. This function has the following parameters: @tsql, @params and @browse\_information\_mode. The table below has the descriptions of the parameters as found at the following Microsoft help [link](https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-describe-first-result-set-transact-sql).

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| tsql | The Transact SQL script or batch. |
| params | It provides a declaration string for parameters for the Transact-SQL batch, which is similar to sp\_executesql. Parameters may be nvarchar(n) or nvarchar(max). |
| browse\_information\_mode | Specifies if additional key columns and source table information are returned. If set to 1, each query is analyzed as if it includes a FOR BROWSE option on the query. Additional key columns and source table information are returned.   * If set to 0, no information is returned. * If set to 1, each query is analyzed as if it includes a FOR BROWSE option on the query. This will return base table names as the source column information. * If set to 2, each query is analyzed as if it would be used in preparing or executing a cursor. This will return view names as source column information. |

Let’s run a SELECT statement using this function and using the query from this tip [Create SQL Server Disk Space Report for All Servers](https://www.mssqltips.com/sqlservertip/5040/create-sql-server-disk-space-report-for-all-servers/).  Note, I had to make all single quotes into double single quotes since the query is passed as a parameter.

USE AdventureWorks2012

GO

SELECT \* FROM sys.dm\_exec\_describe\_first\_result\_set ('

SELECT @@SERVERNAME [Server] ,

DB\_NAME() [Database] ,

MF.name [File Name] ,

MF.type\_desc [Type] ,

MF.physical\_name [Path] ,

CAST(CAST(MF.size / 128.0 AS DECIMAL(15, 2)) AS VARCHAR(50)) + '' MB'' [File Size] ,

CAST(CONVERT(DECIMAL(10, 2), MF.size / 128.0 - ( ( size / 128.0 ) - CAST(FILEPROPERTY(MF.name, ''SPACEUSED'') AS INT) / 128.0 )) AS VARCHAR(50)) + '' MB'' [File Used Space] ,

CAST(CONVERT(DECIMAL(10, 2), MF.size / 128.0 - CAST(FILEPROPERTY(MF.name, ''SPACEUSED'') AS INT) / 128.0) AS VARCHAR(50)) + '' MB'' [File Free Space] ,

CAST(CONVERT(DECIMAL(10, 2), ( ( MF.size / 128.0 - CAST(FILEPROPERTY(MF.name, ''SPACEUSED'') AS INT) / 128.0 ) / ( MF.size / 128.0 ) ) \* 100) AS VARCHAR(50)) + ''%'' [% Free File Space] ,

IIF(MF.growth = 0, ''N/A'',

CASE WHEN MF.is\_percent\_growth = 1 THEN CAST(MF.growth AS VARCHAR(50)) + ''%''

ELSE CAST(MF.growth / 128 AS VARCHAR(50)) + '' MB''

END) [Autogrowth] ,

VS.volume\_mount\_point ,

CAST(CAST(VS.total\_bytes / 1024. / 1024 / 1024 AS DECIMAL(20, 2)) AS VARCHAR(50)) + '' GB'' [Total Volume Size] ,

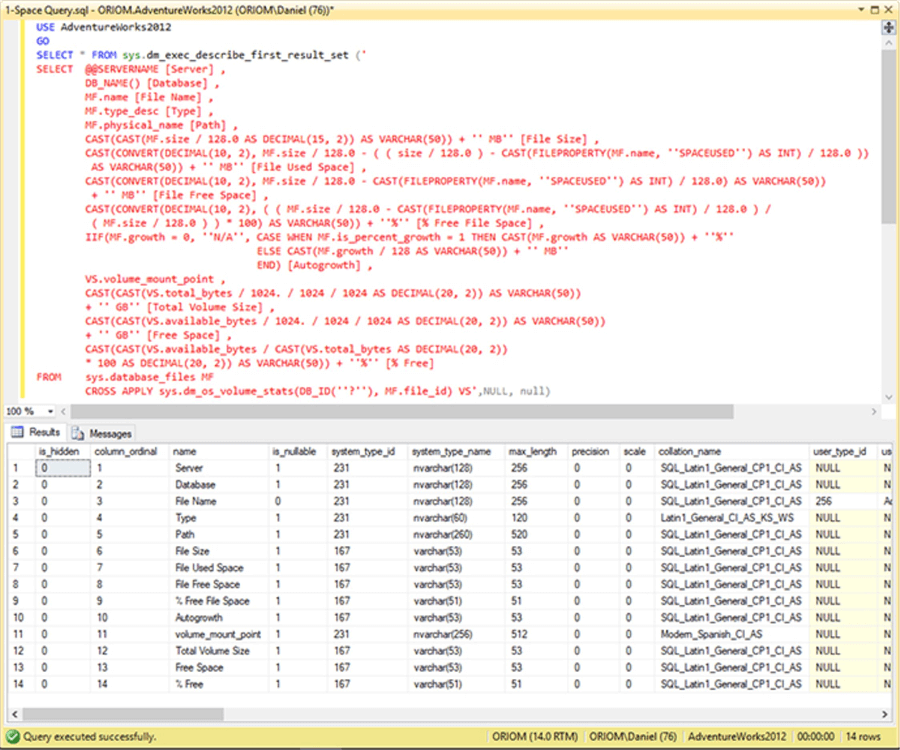
CAST(CAST(VS.available\_bytes / 1024. / 1024 / 1024 AS DECIMAL(20, 2)) AS VARCHAR(50)) + '' GB'' [Free Space] ,

CAST(CAST(VS.available\_bytes / CAST(VS.total\_bytes AS DECIMAL(20, 2)) \* 100 AS DECIMAL(20, 2)) AS VARCHAR(50)) + ''%'' [% Free]

FROM sys.database\_files MF

CROSS APPLY sys.dm\_os\_volume\_stats(DB\_ID(''?''), MF.file\_id) VS',NULL, null)

As you can see in the next screen capture, the output of this function is a table that contains a detailed description of each column of the query that we provided as the parameter.



In the next table you will see a description of each output column. You can see the full table at this [link](https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-describe-first-result-set-transact-sql).

For our purposes we will only use the following columns: name, is\_nullable, system\_type\_name, collation\_name and is\_hidden to filter because we don’t want columns than don’t appear in the result set.

|  |  |
| --- | --- |
| **Column** | **Description** |
| **is\_hidden** | Specifies that the column is an extra column added for browsing and informational purposes that does not actually appear in the result set. |
| column\_ordinal | Contains the ordinal position of the column in the result set. Position of the first column will be specified as 1. |
| **name** | Contains the name of the column if a name can be determined. If not, will contain NULL. |
| **is\_nullable** | Contains the following values: Value 1 if column allows NULLs. Value 0 if the column does not allow NULLs. Value 1 if it cannot be determined that the column allows NULLs. |
| system\_type\_id | Contains the system\_type\_id of the column data type as specified in sys.types. For CLR types, even though the system\_type\_name column will return NULL, this column will return the value 240. |
| **system\_type\_name** | Contains the name and arguments (such as length, precision, scale), specified for the data type of the column. If data type is a user-defined alias type, the underlying system type is specified here. If data type is a CLR user-defined type, NULL is returned in this column. |
| max\_length | Maximum length (in bytes) of the column. -1 = Column data type is varchar(max), nvarchar(max), varbinary(max), or xml. For text columns, the max\_length value will be 16 or the value set by sp\_tableoption 'text in row'. |
| precision | Precision of the column if numeric-based. Otherwise returns 0. |
| scale | Scale of column if numeric-based. Otherwise returns 0. |
| **collation\_name** | Name of the collation of the column if character-based. Otherwise returns NULL. |
| user\_type\_id | For CLR and alias types, contains the user\_type\_id of the data type of the column as specified in sys.types. Otherwise is NULL. |
| user\_type\_database | For CLR and alias types, contains the name of the database in which the type is defined. Otherwise is NULL. |
| user\_type\_schema | For CLR and alias types, contains the name of the schema in which the type is defined. Otherwise is NULL. |
| user\_type\_name | For CLR and alias types, contains the name of the type. Otherwise is NULL. |
| assembly\_qualified\_type\_name | For CLR types, returns the name of the assembly and class defining the type. Otherwise is NULL. |
| xml\_collection\_id | Contains the xml\_collection\_id of the data type of the column as specified in sys.columns. This column returns NULL if the type returned is not associated with an XML schema collection. |
| xml\_collection\_database | Contains the database in which the XML schema collection associated with this type is defined. This column returns NULL if the type returned is not associated with an XML schema collection. |
| xml\_collection\_schema | Contains the schema in which the XML schema collection associated with this type is defined. This column returns NULL if the type returned is not associated with an XML schema collection. |
| xml\_collection\_name | Contains the name of the XML schema collection associated with this type. This column returns NULL if the type returned is not associated with an XML schema collection. |
| is\_xml\_document | Returns 1 if the returned data type is XML and that type is guaranteed to be a complete XML document (including a root node), as opposed to an XML fragment). Otherwise returns 0. |
| is\_case\_sensitive | Returns 1 if the column is of a case-sensitive string type. Returns 0 if it is not. |
| is\_fixed\_length\_clr\_type | Returns 1 if the column is of a fixed-length CLR type. Returns 0 if it is not. |
| source\_server | Name of the originating server (if it originates from a remote server). The name is given as it appears in sys.servers. Returns NULL if the column originates on the local server or if it cannot be determined which server it originates on. Is only populated if browsing information is requested. |
| source\_database | Name of the originating database returned by the column in this result. Returns NULL if the database cannot be determined. Is only populated if browsing information is requested. |
| source\_schema | Name of the originating schema returned by the column in this result. Returns NULL if the schema cannot be determined. Is only populated if browsing information is requested. |
| source\_table | Name of the originating table returned by the column in this result. Returns NULL if the table cannot be determined. Is only populated if browsing information is requested. |
| source\_column | Name of the originating column returned by the result column. Returns NULL if the column cannot be determined. Is only populated if browsing information is requested. |
| is\_identity\_column | Returns 1 if the column is an identity column and 0 if not. Returns NULL if it cannot be determined that the column is an identity column. |
| is\_part\_of\_unique\_key | Returns 1 if the column is part of a unique index (including unique and primary constraints) and 0 if it is not. Returns NULL if it cannot be determined that the column is part of a unique index. Is only populated if browsing information is requested. |
| is\_updateable | Returns 1 if the column is updateable and 0 if not. Returns NULL if it cannot be determined that the column is updateable. |
| is\_computed\_column | Returns 1 if the column is a computed column and 0 if not. Returns NULL if it cannot be determined if the column is a computed column. |
| is\_sparse\_column\_set | Returns 1 if the column is a sparse column and 0 if not. Returns NULL if it cannot be determined that the column is a part of a sparse column set. |
| ordinal\_in\_order\_by\_list | The position of this column is in ORDER BY list. Returns NULL if the column does not appear in the ORDER BY list, or if the ORDER BY list cannot be uniquely determined. |
| order\_by\_list\_length | The length of the ORDER BY list. NULL is returned if there is no ORDER BY list or if the ORDER BY list cannot be uniquely determined. Note that this value will be the same for all rows returned by sp\_describe\_first\_result\_set. |
| order\_by\_is\_descending | If the ordinal\_in\_order\_by\_list is not NULL, the order\_by\_is\_descending column reports the direction of the ORDER BY clause for this column. Otherwise it reports NULL. |
| error\_number | Contains the error number returned by the function. If no error occurred, the column will contain NULL. |
| error\_severity | Contains the severity returned by the function. If no error occurred, the column will contain NULL. |
| error\_state | Contains the state message. returned by the function. If no error occurred, the column will contain NULL. |
| error\_message | Contains the message returned by the function. If no error occurred, the column will contain NULL. |
| error\_type | Contains an integer representing the error being returned. Maps to error\_type\_desc. See the list under remarks. |
| error\_type\_desc | Contains a short uppercase string representing the error being returned. Maps to error\_type. See the list under remarks. |

SQL Server Function to Genereate Create Table Script

Here is the code of the function.

CREATE FUNCTION fn\_Table\_Structure (@InputSQL AS NVARCHAR(4000), @TableName AS NVARCHAR(128) = NULL)

RETURNS NVARCHAR(4000)

AS

BEGIN

DECLARE @SQL AS NVARCHAR(4000)

DECLARE @name NVARCHAR(128)

DECLARE @is\_nullable BIT

DECLARE @system\_type\_name NVARCHAR(128)

DECLARE @collation\_name NVARCHAR(128)

DECLARE @NewLine NVARCHAR(2) = CHAR(13) + CHAR(10) -- CRLF

DECLARE CUR\_Table CURSOR LOCAL FAST\_FORWARD

FOR

SELECT name ,

is\_nullable ,

system\_type\_name ,

collation\_name

FROM sys.dm\_exec\_describe\_first\_result\_set(@InputSQL, NULL, NULL)

WHERE is\_hidden = 0

ORDER BY column\_ordinal ASC

OPEN CUR\_Table

FETCH NEXT FROM CUR\_Table INTO @name, @is\_nullable, @system\_type\_name,

@collation\_name

SET @SQL = 'CREATE TABLE [' + ISNULL(@TableName, 'TableName') + '] ('

+ @NewLine

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @SQL += @NewLine + '[' + @name + ']' + ' ' + @system\_type\_name

+ CASE WHEN @collation\_name IS NOT NULL

THEN ' COLLATE ' + @collation\_name + ' '

ELSE ''

END + CASE WHEN @is\_nullable = 0 THEN ' NOT NULL '

ELSE ''

END + ','

FETCH NEXT FROM CUR\_Table INTO @name, @is\_nullable, @system\_type\_name,

@collation\_name

END

SET @SQL = LEFT(@SQL, LEN(@SQL) - 1) + @NewLine + ')'

CLOSE CUR\_Table

DEALLOCATE CUR\_Table

RETURN @SQL

end

The way this function works by declaring a cursor for each row returned by the sys.dm\_exec\_describe\_first\_result\_set Dynamic Management Function.

Parameters

This function receives the following parameters:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| InputSQL | The Transact SQL script or batch that will be used to create the table structure. |
| TableName | The name of the table you want to generate from the create table script. |

The function returns the create table statement based on the query passed as the parameter. It includes the definition of nullable columns as well as the collation for the string columns.

Here is an example of its use. Note, since the query is passed as a parameter, I had to make each single quote into two single quotes for the original query.

DECLARE @sql AS NVARCHAR(4000)

SELECT @sql = dbo.fn\_Table\_Structure('

SELECT @@SERVERNAME [Server] ,

DB\_NAME() [Database] ,

MF.name [File Name] ,

MF.type\_desc [Type] ,

MF.physical\_name [Path] ,

CAST(CAST(MF.size / 128.0 AS DECIMAL(15, 2)) AS VARCHAR(50)) + '' MB'' [File Size] ,

CAST(CONVERT(DECIMAL(10, 2), MF.size / 128.0 - ( ( size / 128.0 ) - CAST(FILEPROPERTY(MF.name, ''SPACEUSED'') AS INT) / 128.0 )) AS VARCHAR(50)) + '' MB'' [File Used Space] ,

CAST(CONVERT(DECIMAL(10, 2), MF.size / 128.0 - CAST(FILEPROPERTY(MF.name, ''SPACEUSED'') AS INT) / 128.0) AS VARCHAR(50)) + '' MB'' [File Free Space] ,

CAST(CONVERT(DECIMAL(10, 2), ( ( MF.size / 128.0 - CAST(FILEPROPERTY(MF.name, ''SPACEUSED'') AS INT) / 128.0 ) / ( MF.size / 128.0 ) ) \* 100) AS VARCHAR(50)) + ''%'' [% Free File Space] ,

IIF(MF.growth = 0, ''N/A'',

CASE WHEN MF.is\_percent\_growth = 1 THEN CAST(MF.growth AS VARCHAR(50)) + ''%''

ELSE CAST(MF.growth / 128 AS VARCHAR(50)) + '' MB''

END) [Autogrowth] ,

VS.volume\_mount\_point ,

CAST(CAST(VS.total\_bytes / 1024. / 1024 / 1024 AS DECIMAL(20, 2)) AS VARCHAR(50)) + '' GB'' [Total Volume Size] ,

CAST(CAST(VS.available\_bytes / 1024. / 1024 / 1024 AS DECIMAL(20, 2)) AS VARCHAR(50)) + '' GB'' [Free Space] ,

CAST(CAST(VS.available\_bytes / CAST(VS.total\_bytes AS DECIMAL(20, 2)) \* 100 AS DECIMAL(20, 2)) AS VARCHAR(50)) + ''%'' [% Free]

FROM sys.database\_files MF

CROSS APPLY sys.dm\_os\_volume\_stats(DB\_ID(), MF.file\_id) VS

', 'Test' )

PRINT @sql

EXEC (@sql)

On the next image you can see the execution of the function. This will display the results as well as create the table.

**Next Steps**

* If you don’t understand why I included the collation on the CREATE TABLE script, take a look at this tip [Create SQL Server temporary tables with the correct collation](https://www.mssqltips.com/sqlservertip/2440/create-sql-server-temporary-tables-with-the-correct-collation/).
* Take a look at [SQL Server Dynamic Management Views and Functions Tips Category](https://www.mssqltips.com/sql-server-tip-category/31/dynamic-management-views-and-functions/).
* Are you new to cursors? Take a look at this tip for an introduction: [SQL Server Cursor Example](https://www.mssqltips.com/sqlservertip/1599/sql-server-cursor-example/).
* Take a look at [SQL Server Functions - User Defined UDF Tips Category](https://www.mssqltips.com/sql-server-tip-category/160/functions---user-defined-udf/) for more ideas.

Last Update: 2018-10-05