







# SELECT - INTO Clause (Transact-SQL)

Article • 05/23/2023

**Applies to:**  SQL Server  Azure SQL Database  Azure SQL Managed Instance  Azure Synapse Analytics  Analytics Platform System (PDW)  Warehouse in Microsoft Fabric

SELECT...INTO creates a new table in the default filegroup and inserts the resulting rows from the query into it. To view the complete SELECT syntax, see [SELECT \(Transact-SQL\)](#).

 [Transact-SQL syntax conventions](#)

## Syntax

syntaxsql

```
[ INTO new_table ]  
[ ON filegroup ]
```

### Note

To view Transact-SQL syntax for SQL Server 2014 (12.x) and earlier versions, see [Previous versions documentation](#).

## Arguments

### *new\_table*

Specifies the name of a new table to be created, based on the columns in the select list and the rows chosen from the data source.

The format of *new\_table* is determined by evaluating the expressions in the select list. The columns in *new\_table* are created in the order specified by the select list. Each column in *new\_table* has the same name, data type, nullability, and value as the corresponding expression in the select list. The IDENTITY property of a column is transferred except under the conditions defined in "Working with Identity Columns" in the Remarks section.

To create the table in another database on the same instance of SQL Server, specify *new\_table* as a fully qualified name in the form *database.schema.table\_name*.

You cannot create *new\_table* on a remote server; however, you can populate *new\_table* from a remote data source. To create *new\_table* from a remote source table, specify the source table using a four-part name in the form *linked\_server.catalog.schema.object* in the FROM clause of the SELECT statement. Alternatively, you can use the [OPENQUERY](#) function or the [OPENDATASOURCE](#) function in the FROM clause to specify the remote data source.

#### *filegroup*

Specifies the name of the filegroup in which new table will be created. The filegroup specified should exist on the database else the SQL Server engine throws an error.

**Applies to:** SQL Server 2016 (13.x) SP2 and later.

## Data Types

The FILESTREAM attribute does not transfer to the new table. FILESTREAM BLOBs are copied and stored in the new table as **varbinary(max)** BLOBs. Without the FILESTREAM attribute, the **varbinary(max)** data type has a limitation of 2 GB. If a FILESTREAM BLOB exceeds this value, error 7119 is raised and the statement is stopped.

When an existing identity column is selected into a new table, the new column inherits the IDENTITY property, unless one of the following conditions is true:

- The SELECT statement contains a join.
- Multiple SELECT statements are joined by using UNION.
- The identity column is listed more than one time in the select list.
- The identity column is part of an expression.
- The identity column is from a remote data source.

If any one of these conditions is true, the column is created NOT NULL instead of inheriting the IDENTITY property. If an identity column is required in the new table but such a column is not available, or you want a seed or increment value that is different than the source identity column, define the column in the select list using the IDENTITY function. See "Creating an identity column using the IDENTITY function" in the Examples section below.

## Remarks

The `SELECT...INTO` statement operates in two parts - the new table is created, and then rows are inserted. This means that if the inserts fail, they will all be rolled back, but the new (empty) table will remain. If you need the entire operation to succeed or fail as a whole, use an [explicit transaction](#).

Warehouse in Microsoft Fabric doesn't support filegroups. References and examples in this article to filegroups don't apply to Warehouse in Microsoft Fabric.

## Limitations and Restrictions

You cannot specify a table variable or table-valued parameter as the new table.

You cannot use `SELECT...INTO` to create a partitioned table, even when the source table is partitioned. `SELECT...INTO` does not use the partition scheme of the source table; instead, the new table is created in the default filegroup. To insert rows into a partitioned table, you must first create the partitioned table and then use the `INSERT INTO...SELECT...FROM` statement.

Indexes, constraints, and triggers defined in the source table are not transferred to the new table, nor can they be specified in the `SELECT...INTO` statement. If these objects are required, you can create them after executing the `SELECT...INTO` statement.

Specifying an `ORDER BY` clause does not guarantee the rows are inserted in the specified order.

When a sparse column is included in the select list, the sparse column property does not transfer to the column in the new table. If this property is required in the new table, alter the column definition after executing the `SELECT...INTO` statement to include this property.

When a computed column is included in the select list, the corresponding column in the new table is not a computed column. The values in the new column are the values that were computed at the time `SELECT...INTO` was executed.

## Logging Behavior

The amount of logging for `SELECT...INTO` depends on the recovery model in effect for the database. Under the simple recovery model or bulk-logged recovery model, bulk operations are minimally logged. With minimal logging, using the `SELECT...INTO` statement can be more efficient than creating a table and then populating the table with an `INSERT` statement. For more information, see [The Transaction Log \(SQL Server\)](#).

`SELECT...INTO` statements that contain user-defined functions (UDFs) are fully logged operations. If the user-defined functions that are used in the `SELECT...INTO` statement don't perform any data access operations, you can specify the `SCHEMABINDING` clause for the user-defined functions, which will set the derived `UserDataAccess` property for those user-defined functions to 0. After this change, `SELECT...INTO` statements will be minimally logged. If the `SELECT...INTO` statement still references at least one user-defined function that has this property set to 1, the operation is fully logged.

## Permissions

Requires `CREATE TABLE` permission in the destination database.

## Examples

### A. Creating a table by specifying columns from multiple sources

The following example creates the table `dbo.EmployeeAddresses` in the AdventureWorks2022 database by selecting seven columns from various employee-related and address-related tables.

SQL

```
SELECT c.FirstName, c.LastName, e.JobTitle, a.AddressLine1, a.City,
       sp.Name AS [State/Province], a.PostalCode
INTO dbo.EmployeeAddresses
FROM Person.Person AS c
     JOIN HumanResources.Employee AS e
       ON e.BusinessEntityID = c.BusinessEntityID
     JOIN Person.BusinessEntityAddress AS bea
       ON e.BusinessEntityID = bea.BusinessEntityID
     JOIN Person.Address AS a
       ON bea.AddressID = a.AddressID
     JOIN Person.StateProvince AS sp
       ON sp.StateProvinceID = a.StateProvinceID;
```

G0

### B. Inserting rows using minimal logging

The following example creates the table `dbo.NewProducts` and inserts rows from the `Production.Product` table. The example assumes that the recovery model of the AdventureWorks2022 database is set to FULL. To ensure minimal logging is used, the

recovery model of the AdventureWorks2022 database is set to BULK\_LOGGED before rows are inserted and reset to FULL after the SELECT...INTO statement. This process ensures that the SELECT...INTO statement uses minimal space in the transaction log and performs efficiently.

SQL

```
ALTER DATABASE AdventureWorks2022 SET RECOVERY BULK_LOGGED;
GO

SELECT * INTO dbo.NewProducts
FROM Production.Product
WHERE ListPrice > $25
AND ListPrice < $100;
GO
ALTER DATABASE AdventureWorks2022 SET RECOVERY FULL;
GO
```

## C. Creating an identity column using the IDENTITY function

The following example uses the IDENTITY function to create an identity column in the new table `Person.USAddress` in the AdventureWorks2022 database. This is required because the SELECT statement that defines the table contains a join, which causes the IDENTITY property to not transfer to the new table. Notice that the seed and increment values specified in the IDENTITY function are different from those of the `AddressID` column in the source table `Person.Address`.

SQL

```
-- Determine the IDENTITY status of the source column AddressID.
SELECT OBJECT_NAME(object_id) AS TableName, name AS column_name,
       is_identity, seed_value, increment_value
FROM sys.identity_columns
WHERE name = 'AddressID';

-- Create a new table with columns from the existing table
Person.Address.
-- A new IDENTITY column is created by using the IDENTITY function.
SELECT IDENTITY (int, 100, 5) AS AddressID,
       a.AddressLine1, a.City, b.Name AS State, a.PostalCode
INTO Person.USAddress
FROM Person.Address AS a
INNER JOIN Person.StateProvince AS b
ON a.StateProvinceID = b.StateProvinceID
WHERE b.CountryRegionCode = N'US';
```

```

-- Verify the IDENTITY status of the AddressID columns in both ta-
bles.
SELECT OBJECT_NAME(object_id) AS TableName, name AS column_name,
       is_identity, seed_value, increment_value
FROM sys.identity_columns
WHERE name = 'AddressID';

```

## D. Creating a table by specifying columns from a remote data source

The following example demonstrates three methods of creating a new table on the local server from a remote data source. The example begins by creating a link to the remote data source. The linked server name, `MyLinkServer`, is then specified in the FROM clause of the first SELECT...INTO statement and in the OPENQUERY function of the second SELECT...INTO statement. The third SELECT...INTO statement uses the OPENDATASOURCE function, which specifies the remote data source directly instead of using the linked server name.

**Applies to:** SQL Server 2008 (10.0.x) and later.

SQL

```

USE master;
GO
-- Create a link to the remote data source.
-- Specify a valid server name for @datasrc as 'server_name'
-- or 'server_name\instance_name'.
EXEC sp_addlinkedserver @server = N'MyLinkServer',
    @srvproduct = N' ',
    @provider = N'SQLNCLI',
    @datasrc = N'server_name',
    @catalog = N'AdventureWorks2022';
GO

USE AdventureWorks2022;
GO
-- Specify the remote data source in the FROM clause using a four-
part name
-- in the form linked_server.catalog.schema.object.
SELECT DepartmentID, Name, GroupName, ModifiedDate
INTO dbo.Departments
FROM MyLinkServer.AdventureWorks2022.HumanResources.Department
GO
-- Use the OPENQUERY function to access the remote data source.
SELECT DepartmentID, Name, GroupName, ModifiedDate
INTO dbo.DepartmentsUsingOpenQuery
FROM OPENQUERY(MyLinkServer, 'SELECT *
                             FROM AdventureWorks2022.HumanResources.Department');
GO

```

```
-- Use the OPENDATASOURCE function to specify the remote data source.
-- Specify a valid server name for Data Source using the format
-- server_name or server_name\instance_name.
SELECT DepartmentID, Name, GroupName, ModifiedDate
INTO dbo.DepartmentsUsingOpenDataSource
FROM OPENDATASOURCE('SQLNCLI',
    'Data Source=server_name;Integrated Security=SSPI')
    .AdventureWorks2022.HumanResources.Department;
GO
```

## E. Import from an external table created with PolyBase

Import data from Hadoop or Azure Storage into SQL Server for persistent storage. Use `SELECT INTO` to import data referenced by an external table for persistent storage in SQL Server. Create a relational table on-the-fly and then create a column-store index on top of the table in a second step.

**Applies to:** SQL Server.

SQL

```
-- Import data for car drivers into SQL Server to do more in-depth
analysis.
SELECT DISTINCT
    Insured_Customers.FirstName, Insured_Customers.LastName,
    Insured_Customers.YearlyIncome,
    Insured_Customers.MaritalStatus
INTO Fast_Customers from Insured_Customers INNER JOIN
(
    SELECT * FROM CarSensor_Data where Speed > 35
) AS SensorD
ON Insured_Customers.CustomerKey = SensorD.CustomerKey
ORDER BY YearlyIncome;
```

## F. Copying the data from one table to another and create the new table on a specified filegroup

The following example demonstrates creating a new table as a copy of another table and loading it into a specified filegroup different from the default filegroup of the user.

**Applies to:** SQL Server 2016 (13.x) SP2 and later.

SQL

```
ALTER DATABASE [AdventureWorksDW2022] ADD FILEGROUP FG2;
ALTER DATABASE [AdventureWorksDW2022]
```

```
ADD FILE
(
NAME='FG2_Data',
FILENAME = '/var/opt/mssql/data/AdventureWorksDW2022_Data1.mdf'
)
TO FILEGROUP FG2;
GO
SELECT * INTO [dbo].[FactResellerSalesXL] ON FG2 FROM [dbo].[FactResellerSales];
```

## See Also

[SELECT \(Transact-SQL\)](#)

[SELECT Examples \(Transact-SQL\)](#)

[INSERT \(Transact-SQL\)](#)

[IDENTITY \(Function\) \(Transact-SQL\)](#)

---

## Feedback

Was this page helpful?

[Provide product feedback](#) [↗](#) | [Get help at Microsoft Q&A](#)