INSERT Examples (Transact-SQL)

This topic provides examples of using the Transact-SQL INSERT statement in SQL Server 2008 R2. The INSERT examples are grouped by the following categories.

Category	**Featured syntax elements**
Basic syntax	INSERT • table value constructor
Handling column values	IDENTITY • NEWID • default values • user-defined types
Inserting data from other tables	INSERT...SELECT • INSERT...EXECUTE • WITH common table expression • TOP
Specifying target objects other than standard tables	Views • table variables
Inserting rows into a remote table	Linked server • OPENQUERY rowset function • OPENDATASOURCE rowset function
Bulk importing data from tables or data files	INSERT...SELECT • OPENROWSET function
Overriding the default behavior of the query optimizer by using hints	Table hints
Capturing the results of the INSERT statement	OUTPUT clause

Basic Syntax

Examples in this section demonstrate the basic functionality of the INSERT statement using the minimum required syntax.

A. Inserting a single row of data

The following example inserts one row into the table `Production.UnitMeasure`. The columns in this table are `UnitMeasureCode`, `Name`, and `ModifiedDate`. Because values for all columns are supplied and are listed in the same order as the columns in the table, the column names do not have to be specified in the column list.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
INSERT INTO Production.UnitMeasure
VALUES (N'FT', N'Feet', '20080414');
GO
```

B. Inserting multiple rows of data

The following example uses the table value constructor to insert three rows into the `Production.UnitMeasure` table in a single INSERT statement. Because values for all columns are supplied and are listed in the same order as the columns in the table, the column names do not have to be specified in the column list.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
INSERT INTO Production.UnitMeasure
VALUES (N'FT2', N'Square Feet ', '20080923'), (N'Y', N'Yards', '20080923'), (N'Y3', N'Cubic Yards', '20080923');
GO
```

C. Inserting data that is not in the same order as the table columns

The following example uses a column list to explicitly specify the values that are inserted into each column. The column order in the `Production.UnitMeasure` table is `UnitMeasureCode`, `Name`, `ModifiedDate`; however, the columns are not listed in that order in *column_list*.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
INSERT INTO Production.UnitMeasure (Name, UnitMeasureCode,
    ModifiedDate)
VALUES (N'Square Yards', N'Y2', GETDATE());
GO
```

Handling Column Values

Examples in this section demonstrate methods of inserting values into columns that are defined with an IDENTITY property, DEFAULT value, or are defined with data types such as **uniqueidentifer** or user-defined type columns.

A. Inserting data into a table with columns that have default values

The following example shows inserting rows into a table with columns that automatically generate a value or have a default value. `Column_1` is a computed column that automatically generates a value by concatenating a string with the value inserted into `column_2`. `Column_2` is defined with a default constraint. If a value is not specified for this column, the default value is used. `Column_3` is defined with the **rowversion** data type, which automatically generates a unique, incrementing binary number. `Column_4` does not automatically generate a value. When a value for this column is not specified, NULL is inserted. The INSERT statements insert rows that contain values for some of the columns but not all. In the last INSERT statement, no columns are specified and only the default values are inserted by using the DEFAULT VALUES clause.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('dbo.T1', 'U') IS NOT NULL
    DROP TABLE dbo.T1;
GO
CREATE TABLE dbo.T1
(
    column_1 AS 'Computed column ' + column_2,
    column_2 varchar(30)
        CONSTRAINT default_name DEFAULT ('my column default'),
    column_3 rowversion,
    column_4 varchar(40) NULL
);
GO
INSERT INTO dbo.T1 (column_4)
    VALUES ('Explicit value');
INSERT INTO dbo.T1 (column_2, column_4)
    VALUES ('Explicit value', 'Explicit value');
INSERT INTO dbo.T1 (column_2)
    VALUES ('Explicit value');
INSERT INTO T1 DEFAULT VALUES;
GO
SELECT column_1, column_2, column_3, column_4
FROM dbo.T1;
GO
```

B. Inserting data into a table that has an identity column

The following example shows different methods of inserting data into an identity column. The first two INSERT statements allow identity values to be generated for the new rows. The third INSERT statement overrides the IDENTITY property for the column with the SET IDENTITY_INSERT statement and inserts an explicit value into the identity column.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('dbo.T1', 'U') IS NOT NULL
    DROP TABLE dbo.T1;
GO
CREATE TABLE dbo.T1 ( column_1 int IDENTITY, column_2 VARCHAR(30));
GO
INSERT T1 VALUES ('Row #1');
INSERT T1 (column_2) VALUES ('Row #2');
GO
SET IDENTITY_INSERT T1 ON;
GO
INSERT INTO T1 (column_1,column_2)
    VALUES (-99, 'Explicit identity value');
GO
SELECT column_1, column_2
FROM T1;
GO
```

C. Inserting data into a uniqueidentifier column by using NEWID()

The following example uses the NEWID() function to obtain a GUID for `column_2`. Unlike for identity columns, the Database Engine does not automatically generate values for columns with the uniqueidentifier data type, as shown by the second `INSERT` statement.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
```

```
IF OBJECT_ID ('dbo.T1', 'U') IS NOT NULL
    DROP TABLE dbo.T1;
GO
CREATE TABLE dbo.T1
(
    column_1 int IDENTITY,
    column_2 uniqueidentifier,
);
GO
INSERT INTO dbo.T1 (column_2)
    VALUES (NEWID());
INSERT INTO T1 DEFAULT VALUES;
GO
SELECT column_1, column_2
FROM dbo.T1;
GO
```

D. Inserting data into user-defined type columns

The following Transact-SQL statements insert three rows into the `PointValue` column of the `Points` table. This column uses a CLR user-defined type (UDT). The `Point` data type consists of X and Y integer values that are exposed as properties of the UDT. You must use either the CAST or CONVERT function to cast the comma-delimited X and Y values to the `Point` type. The first two statements use the CONVERT function to convert a string value to the `Point` type, and the third statement uses the CAST function. For more information, see Manipulating UDT Data.

```
INSERT INTO dbo.Points (PointValue) VALUES (CONVERT(Point, '3,4'));
INSERT INTO dbo.Points (PointValue) VALUES (CONVERT(Point, '1,5'));
INSERT INTO dbo.Points (PointValue) VALUES (CAST ('1,99' AS Point));
```

Inserting Data from Other Tables

Examples in this section demonstrate methods of inserting rows from one table into another table.

A. Using the SELECT and EXECUTE options to insert data from other tables

The following example shows how to insert data from one table into another table by using INSERT...SELECT or INSERT...EXECUTE. Each is based on a multi-table SELECT statement that includes an expression and a literal value in the column list.

The first INSERT statement uses a SELECT statement to derive the data from the source tables (`Employee`, `SalesPerson`, and `Person`) and store the result set in the `EmployeeSales` table. The second INSERT statement uses the EXECUTE clause to call a stored procedure that contains the SELECT statement, and the third INSERT uses the EXECUTE clause to reference the SELECT statement as a literal string.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('dbo.EmployeeSales', 'U') IS NOT NULL
    DROP TABLE dbo.EmployeeSales;
GO
IF OBJECT_ID ('dbo.uspGetEmployeeSales', 'P') IS NOT NULL
    DROP PROCEDURE uspGetEmployeeSales;
GO
CREATE TABLE dbo.EmployeeSales
( DataSource   varchar(20) NOT NULL,
  BusinessEntityID   varchar(11) NOT NULL,
  LastName     varchar(40) NOT NULL,
  SalesDollars money NOT NULL
);
GO
CREATE PROCEDURE dbo.uspGetEmployeeSales
AS
    SET NOCOUNT ON;
    SELECT 'PROCEDURE', sp.BusinessEntityID, c.LastName,
        sp.SalesYTD
    FROM Sales.SalesPerson AS sp
    INNER JOIN Person.Person AS c
        ON sp.BusinessEntityID = c.BusinessEntityID
    WHERE sp.BusinessEntityID LIKE '2%'
    ORDER BY sp.BusinessEntityID, c.LastName;
GO
--INSERT...SELECT example
INSERT INTO dbo.EmployeeSales
    SELECT 'SELECT', sp.BusinessEntityID, c.LastName, sp.SalesYTD
    FROM Sales.SalesPerson AS sp
    INNER JOIN Person.Person AS c
        ON sp.BusinessEntityID = c.BusinessEntityID
    WHERE sp.BusinessEntityID LIKE '2%'
    ORDER BY sp.BusinessEntityID, c.LastName;
GO
--INSERT...EXECUTE procedure example
INSERT INTO dbo.EmployeeSales
EXECUTE dbo.uspGetEmployeeSales;
GO
--INSERT...EXECUTE('string') example
INSERT INTO dbo.EmployeeSales
EXECUTE
('
SELECT ''EXEC STRING'', sp.BusinessEntityID, c.LastName,
    sp.SalesYTD
    FROM Sales.SalesPerson AS sp
    INNER JOIN Person.Person AS c
        ON sp.BusinessEntityID = c.BusinessEntityID
    WHERE sp.BusinessEntityID LIKE ''2%''
    ORDER BY sp.BusinessEntityID, c.LastName
');
GO
--Show results.
SELECT DataSource,BusinessEntityID,LastName,SalesDollars
FROM dbo.EmployeeSales;
GO
```

B. Using WITH common table expression to define the data inserted

The following example creates the `NewEmployee` table. A common table expression (`EmployeeTemp`) defines the rows from one or more tables to be inserted into the `NewEmployee` table. The INSERT statement references the columns in the common table expression.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID (N'HumanResources.NewEmployee', N'U') IS NOT NULL
    DROP TABLE HumanResources.NewEmployee;
GO
CREATE TABLE HumanResources.NewEmployee
(
    EmployeeID int NOT NULL,
    LastName nvarchar(50) NOT NULL,
    FirstName nvarchar(50) NOT NULL,
    PhoneNumber Phone NULL,
    AddressLine1 nvarchar(60) NOT NULL,
    City nvarchar(30) NOT NULL,
    State nchar(3) NOT NULL,
    PostalCode nvarchar(15) NOT NULL,
    CurrentFlag Flag
);
GO
WITH EmployeeTemp (EmpID, LastName, FirstName, Phone,
                   Address, City, StateProvince,
                   PostalCode, CurrentFlag)
AS (SELECT
       e.BusinessEntityID, c.LastName, c.FirstName, pp.PhoneNumber,
       a.AddressLine1, a.City, sp.StateProvinceCode,
       a.PostalCode, e.CurrentFlag
    FROM HumanResources.Employee e
        INNER JOIN Person.BusinessEntityAddress AS bea
        ON e.BusinessEntityID = bea.BusinessEntityID
        INNER JOIN Person.Address AS a
        ON bea.AddressID = a.AddressID
        INNER JOIN Person.PersonPhone AS pp
        ON e.BusinessEntityID = pp.BusinessEntityID
        INNER JOIN Person.StateProvince AS sp
        ON a.StateProvinceID = sp.StateProvinceID
        INNER JOIN Person.Person as c
        ON e.BusinessEntityID = c.BusinessEntityID
    )
INSERT INTO HumanResources.NewEmployee
    SELECT EmpID, LastName, FirstName, Phone,
           Address, City, StateProvince, PostalCode, CurrentFlag
    FROM EmployeeTemp;
GO
```

C. Using TOP to limit the data inserted from the source table

The following example uses the TOP clause to limit the number of rows inserted into the `NewEmployee` table from the `Employee` table. The example inserts address data for the first random set of 10 employees from the `Employee` table into it. The SELECT statement is then executed to verify the contents of the `NewEmployee` table.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID (N'HumanResources.NewEmployee', N'U') IS NOT NULL
    DROP TABLE HumanResources.NewEmployee;
GO
CREATE TABLE HumanResources.NewEmployee
(
    BusinessEntityID int NOT NULL,
    LastName nvarchar(50) NOT NULL,
    FirstName nvarchar(50) NOT NULL,
    PhoneNumber Phone NULL,
    AddressLine1 nvarchar(60) NOT NULL,
    City nvarchar(30) NOT NULL,
    State nchar(3) NOT NULL,
    PostalCode nvarchar(15) NOT NULL,
    CurrentFlag Flag
);
GO
-- Insert 10 random rows into the table NewEmployee.
INSERT TOP (10) INTO HumanResources.NewEmployee
    SELECT
       e.BusinessEntityID, c.LastName, c.FirstName, pp.PhoneNumber,
       a.AddressLine1, a.City, sp.StateProvinceCode,
       a.PostalCode, e.CurrentFlag
    FROM HumanResources.Employee e
        INNER JOIN Person.BusinessEntityAddress AS bea
        ON e.BusinessEntityID = bea.BusinessEntityID
        INNER JOIN Person.Address AS a
        ON bea.AddressID = a.AddressID
        INNER JOIN Person.PersonPhone AS pp
        ON e.BusinessEntityID = pp.BusinessEntityID
        INNER JOIN Person.StateProvince AS sp
        ON a.StateProvinceID = sp.StateProvinceID
        INNER JOIN Person.Person as c
        ON e.BusinessEntityID = c.BusinessEntityID;
GO
SELECT  BusinessEntityID, LastName, FirstName, PhoneNumber,
        AddressLine1, City, State, PostalCode, CurrentFlag
FROM HumanResources.NewEmployee;
GO
```

Specifying Target Objects Other Than Standard Tables

Examples in this section demonstrate how to insert rows by specifying a view or table variable.

A. Inserting data by specifying a view

The following example specifies a view name as the target object; however, the new row is inserted in the underlying base table. The order of the values in the `INSERT` statement must match the column order of the view. For more information, see Modifying Data Through a View.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
IF OBJECT_ID ('dbo.T1', 'U') IS NOT NULL
    DROP TABLE dbo.T1;
GO
IF OBJECT_ID ('dbo.V1', 'V') IS NOT NULL
    DROP VIEW dbo.V1;
GO
CREATE TABLE T1 ( column_1 int, column_2 varchar(30));
GO
CREATE VIEW V1 AS
SELECT column_2, column_1
FROM T1;
GO
INSERT INTO V1
    VALUES ('Row 1',1);
GO
SELECT column_1, column_2
FROM T1;
GO
SELECT column_1, column_2
FROM V1;
GO
```

B. Inserting data into a table variable

The following example specifies a table variable as the target object.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
-- Create the table variable.
DECLARE @MyTableVar table(
    LocationID int NOT NULL,
    CostRate smallmoney NOT NULL,
    NewCostRate AS CostRate * 1.5,
    ModifiedDate datetime);

-- Insert values into the table variable.
INSERT INTO @MyTableVar (LocationID, CostRate, ModifiedDate)
    SELECT LocationID, CostRate, GETDATE() FROM Production.Location
    WHERE CostRate > 0;

-- View the table variable result set.
SELECT * FROM @MyTableVar;
GO
```

Inserting Rows into a Remote Table

Examples in this section demonstrate how to insert rows into a remote target table by using a linked server or a rowset function to reference the remote table.

A. Inserting data into a remote table by using a linked server

The following example inserts rows into a remote table. The example begins by creating a link to the remote data source by using sp_addlinkedserver. The linked server name, `MyLinkServer`, is then specified as part of the four-part object name in the form *server.catalog.schema.object*.

Transact-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'AdventureWorks2008R2';
GO
```

Transact-SQL

```
USE AdventureWorks2008R2;
GO
-- Specify the remote data source in the FROM clause using a four-part name
-- in the form linked_server.catalog.schema.object.

INSERT INTO MyLinkServer.AdventureWorks2008R2.HumanResources.Department (Name, GroupName)
VALUES (N'Public Relations', N'Executive General and Administration');
GO
```

B. Inserting data into a remote table by using the OPENQUERY function

The following example inserts a row into a remote table by specifying the OPENQUERY rowset function. The linked server name created in the previous example is used in this example.

Transact-SQL

```
-- Use the OPENQUERY function to access the remote data source.

INSERT OPENQUERY (MyLinkServer, 'SELECT Name, GroupName FROM AdventureWorks2008R2.HumanResources.Department')
VALUES ('Environmental Impact', 'Engineering');
GO
```

C. Inserting data into a remote table by using the OPENDATASOURCE function

The following example inserts a row into a remote table by specifying the OPENDATASOURCE rowset function. Specify a valid server name for the data source by using the format *server_name* or *server_name\instance_name*.

Transact-SQL

```
-- 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.

INSERT INTO OPENDATASOURCE('SQLNCLI',
    'Data Source= <server_name>; Integrated Security=SSPI')
    .AdventureWorks2008R2.HumanResources.Department (Name, GroupName)
    VALUES (N'Standards and Methods', 'Quality Assurance');
GO
```

Bulk Importing Data from Tables or Data Files

Examples in this section demonstrate two methods to bulk import (bulk load) data into a table by using the INSERT statement.

A. Inserting data into a heap with minimal logging

The following example creates a a new table (a heap) and inserts data from another table into it using minimal logging. The example assumes that the recovery model of the AdventureWorks2008R2 database is set to FULL. To ensure minimal logging is used, the recovery model of the AdventureWorks2008R2 database is set to BULK_LOGGED before rows are inserted and reset to FULL after the INSERT INTO...SELECT statement. In addition, the TABLOCK hint is specified for the target table `Sales.SalesHistory`. This ensures that the statement uses minimal space in the transaction log and performs efficiently.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
-- Create the target heap.
CREATE TABLE Sales.SalesHistory(
    SalesOrderID int NOT NULL,
    SalesOrderDetailID int NOT NULL,
    CarrierTrackingNumber nvarchar(25) NULL,
    OrderQty smallint NOT NULL,
    ProductID int NOT NULL,
    SpecialOfferID int NOT NULL,
    UnitPrice money NOT NULL,
    UnitPriceDiscount money NOT NULL,
    LineTotal money NOT NULL,
    rowguid uniqueidentifier ROWGUIDCOL  NOT NULL,
    ModifiedDate datetime NOT NULL );
GO
-- Temporarily set the recovery model to BULK_LOGGED.
ALTER DATABASE AdventureWorks2008R2
SET RECOVERY BULK_LOGGED;
GO
-- Transfer data from Sales.SalesOrderDetail to Sales.SalesHistory
INSERT INTO Sales.SalesHistory WITH (TABLOCK)
    (SalesOrderID,
     SalesOrderDetailID,
     CarrierTrackingNumber,
     OrderQty,
     ProductID,
     SpecialOfferID,
     UnitPrice,
     UnitPriceDiscount,
     LineTotal,
     rowguid,
     ModifiedDate)
SELECT * FROM Sales.SalesOrderDetail;
GO
-- Reset the recovery model.
ALTER DATABASE AdventureWorks2008R2
SET RECOVERY FULL;
GO
```

B. Using the OPENROWSET function with BULK to bulk import data into a table

The following example inserts rows from a data file into a table by specifying the OPENROWSET function. The IGNORE_TRIGGERS table hint is specified for performance optimization. For more examples, see Importing Bulk Data by Using BULK INSERT or OPENROWSET(BULK...).

Transact-SQL

```
-- Use the OPENROWSET function to specify the data source and specifies the IGNORE_TRIGGERS table hint.
INSERT INTO HumanResources.Department WITH (IGNORE_TRIGGERS) (Name, GroupName)
SELECT b.Name, b.GroupName
FROM OPENROWSET (
    BULK 'C:\SQLFiles\DepartmentData.txt',
    FORMATFILE = 'C:\SQLFiles\BulkloadFormatFile.xml',
    ROWS_PER_BATCH = 15000)AS b ;
GO
```

Overriding the Default Behavior of the Query Optimizer by Using Hints

Examples in this section demonstrate how to use table hints to temporarily override the default behavior of the query optimizer when processing the INSERT statement.

Caution

Because the SQL Server query optimizer typically selects the best execution plan for a query, we recommend that hints be used only as a last resort by experienced developers and database administrators.

A. Using the TABLOCK hint to specify a locking method

The following example specifies that an exclusive (X) lock is taken on the table `Production.Location` and is held until the end of the INSERT statement.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
INSERT INTO Production.Location WITH (XLOCK)
(Name, CostRate, Availability)
VALUES ( N'Final Inventory', 15.00, 80.00);
GO
```

Capturing the Results of the INSERT Statement

Examples in this section demonstrate how to use the OUTPUT Clause to return information from, or expressions based on, each row affected by an INSERT statement. These results can be returned to the processing application for use in such things as confirmation messages, archiving, and other such application requirements.

A Using OUTPUT with an INSERT statement

The following example inserts a row into the `ScrapReason` table and uses the OUTPUT clause to return the results of the statement to the `@MyTableVar` table variable. Because the `ScrapReasonID` column in the `ScrapReason` table is defined with an IDENTITY property, a value is not specified in the INSERT statement for that column. However, note that the value generated by the Database Engine for that column is returned in the OUTPUT clause in the `INSERTED.ScrapReasonID` column.

Transact-SQL

```
USE AdventureWorks2008R2;
GO
DECLARE @MyTableVar table( NewScrapReasonID smallint,
                           Name varchar(50),
                           ModifiedDate datetime);
INSERT Production.ScrapReason
    OUTPUT INSERTED.ScrapReasonID, INSERTED.Name, INSERTED.ModifiedDate
        INTO @MyTableVar
VALUES (N'Operator error', GETDATE());

--Display the result set of the table variable.
SELECT NewScrapReasonID, Name, ModifiedDate FROM @MyTableVar;
--Display the result set of the table.
SELECT ScrapReasonID, Name, ModifiedDate
FROM Production.ScrapReason;
GO
```

B. Using OUTPUT with identity and computed columns

The following example creates the `EmployeeSales` table and then inserts several rows into it using an INSERT statement with a SELECT statement to retrieve data from source tables. The `EmployeeSales` table contains an identity column (`EmployeeID`) and a computed column (`ProjectedSales`). Because these values are generated by the Database Engine during the insert operation, neither of these columns can be defined in `@MyTableVar`.

Transact-SQL

```
USE AdventureWorks2008R2 ;
GO
IF OBJECT_ID ('dbo.EmployeeSales', 'U') IS NOT NULL
    DROP TABLE dbo.EmployeeSales;
GO
CREATE TABLE dbo.EmployeeSales
( EmployeeID   int IDENTITY (1,5)NOT NULL,
  LastName     nvarchar(20) NOT NULL,
  FirstName    nvarchar(20) NOT NULL,
  CurrentSales money NOT NULL,
  ProjectedSales AS CurrentSales * 1.10
);
GO
DECLARE @MyTableVar table(
  LastName     nvarchar(20) NOT NULL,
  FirstName    nvarchar(20) NOT NULL,
  CurrentSales money NOT NULL
  );

INSERT INTO dbo.EmployeeSales (LastName, FirstName, CurrentSales)
  OUTPUT INSERTED.LastName,
         INSERTED.FirstName,
         INSERTED.CurrentSales
  INTO @MyTableVar
    SELECT c.LastName, c.FirstName, sp.SalesYTD
    FROM Sales.SalesPerson AS sp
    INNER JOIN Person.Person AS c
        ON sp.BusinessEntityID = c.BusinessEntityID
    WHERE sp.BusinessEntityID LIKE '2%'
    ORDER BY c.LastName, c.FirstName;

SELECT LastName, FirstName, CurrentSales
FROM @MyTableVar;
GO
SELECT EmployeeID, LastName, FirstName, CurrentSales, ProjectedSales
FROM dbo.EmployeeSales;
GO
```

C. Inserting data returned from an OUTPUT clause

The following example captures data returned from the OUTPUT clause of a MERGE statement, and inserts that data into another table. The MERGE statement updates the `Quantity` column of the `ProductInventory` table daily, based on orders that are processed in the `SalesOrderDetail` table. It also deletes rows for products whose inventories drop to 0. The example captures the rows that are deleted and inserts them into another table, `ZeroInventory`, which tracks products with no inventory.

Transact-SQL

```sql
USE AdventureWorks2008R2;
GO
IF OBJECT_ID(N'Production.ZeroInventory', N'U') IS NOT NULL
    DROP TABLE Production.ZeroInventory;
GO
--Create ZeroInventory table.
CREATE TABLE Production.ZeroInventory (DeletedProductID int, RemovedOnDate DateTime);
GO

INSERT INTO Production.ZeroInventory (DeletedProductID, RemovedOnDate)
SELECT ProductID, GETDATE()
FROM
(   MERGE Production.ProductInventory AS pi
    USING (SELECT ProductID, SUM(OrderQty) FROM Sales.SalesOrderDetail AS sod
           JOIN Sales.SalesOrderHeader AS soh
           ON sod.SalesOrderID = soh.SalesOrderID
           AND soh.OrderDate = '20070401'
           GROUP BY ProductID) AS src (ProductID, OrderQty)
    ON (pi.ProductID = src.ProductID)
    WHEN MATCHED AND pi.Quantity - src.OrderQty <= 0
        THEN DELETE
    WHEN MATCHED
        THEN UPDATE SET pi.Quantity = pi.Quantity - src.OrderQty
    OUTPUT $action, deleted.ProductID) AS Changes (Action, ProductID)
WHERE Action = 'DELETE';
IF @@ROWCOUNT = 0
PRINT 'Warning: No rows were inserted';
GO
SELECT DeletedProductID, RemovedOnDate FROM Production.ZeroInventory;
```

See Also

Reference
INSERT (Transact-SQL)
IDENTITY (Property) (Transact-SQL)

Community Additions

Too many keystrokes for my tired ol' fingers...

You could use UNION ALL for a simple insert of multiple rows:

```
INSERT INTO [EmployeeTable] (EmployeeID,Login,Title)
SELECT '001','BGreenberg','serf'
UNION ALL
SELECT '002','SWood','serf'
[...]
UNION ALL
SELECT '082','JLane','manager'
```

ScrappyLaptop
12/16/2014

OUTPUT for IDENTITY and COMPUTED columns

"The EmployeeSales table contains an identity column (EmployeeID) and a computed column (ProjectedSales). Because these values are generated by the Database Engine during the insert operation, neither of these columns can be defined in @MyTableVar."

This is incorrect, adding INSERTED.EmployeeID to the OUTPUT and adding this to the table variable allows selection of the IDENTITY. This works the same for computed columns for as ProjectedSales in this example.

M Outlaw
10/15/2011