

SQL Server Integration Services (SSIS) has transformations, which are key components to the Data Flow, that transform the data to a desired format as data moves from one step to another step. Transformation in SSIS is all done in-memory; after adding a transformation the data is altered and passed down the path in the Data Flow.

In SSIS, transformations are available in two main categories--Synchronous and Asynchronous. During ETL design it's recommended to use all Synchronous transformation components.

Synchronous are components like the Conditional Split or Derived Column Transformation where rows flow into memory buffers in the transformation and the same buffers come out. No rows are held and characteristically these transformations perform very quickly with marginal impact to Data Flow.

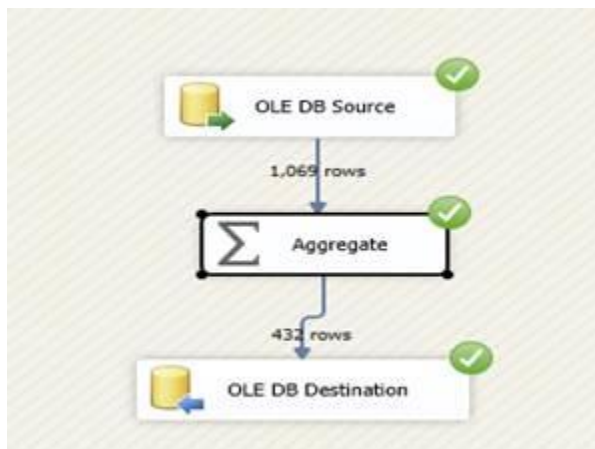
Asynchronous transformation has two types, fully blocking and partial blocking. Partial blocking transformation is that transformation which creates new memory buffers for the output of the transformation than what come into the transformation, like Union All Transformation; fully blocking transformations also require a new memory buffer similar to partial blocking. Asynchronous transformations additionally cause a full block of the data like Sort and Aggregate transformations.

Let's take a quick deep dive with some important SSIS transformations under both categories, Synchronous and Asynchronous, which can be useful to manage and transform data during ETL execution.

Aggregate

An Asynchronous full blocking transformation, Aggregate transformation allows to aggregate data from Data Flow to apply certain T-SQL functions that are done in a GROUP BY statement.

Data Flow task design for Aggregate:

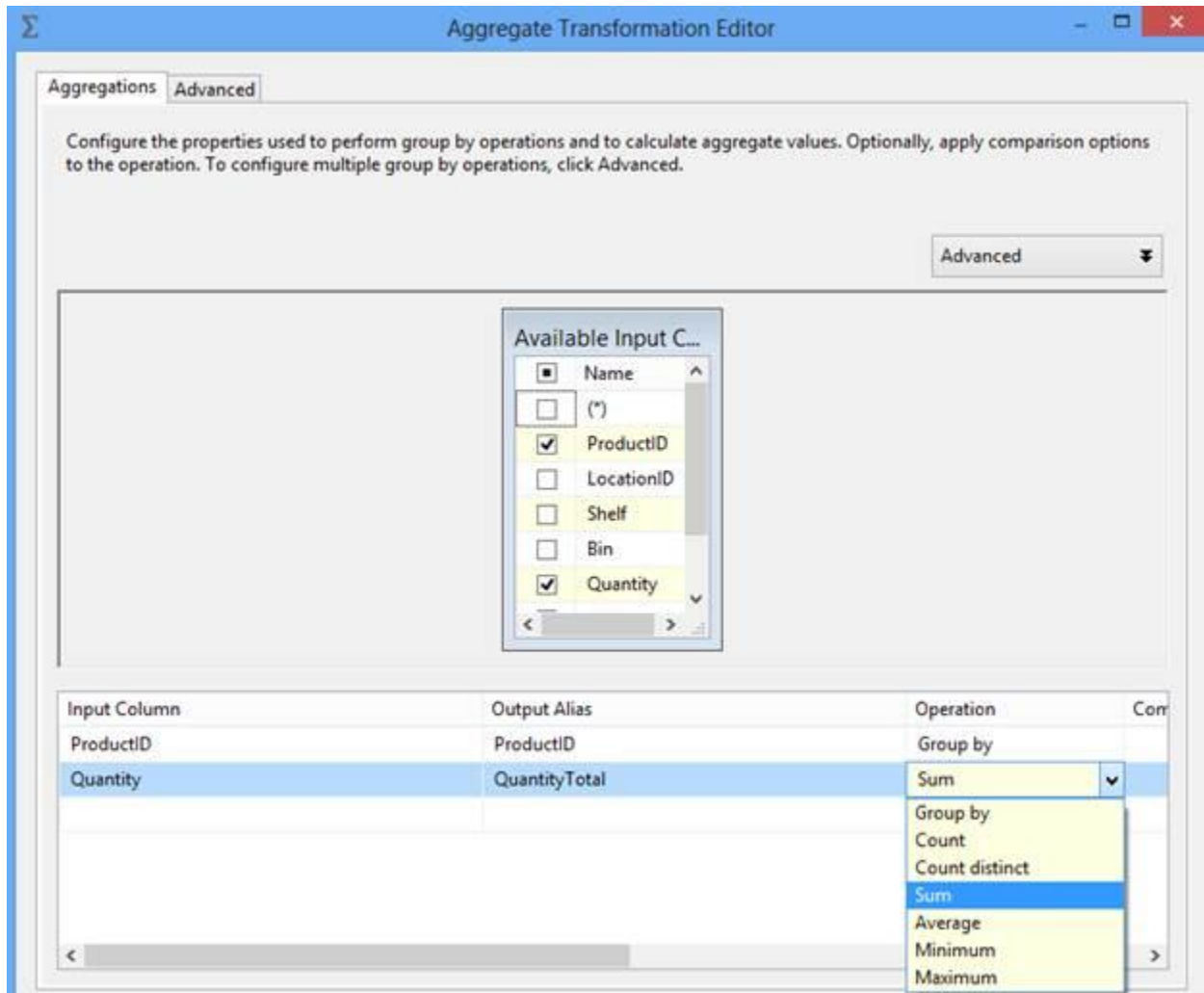


Aggregate Data Flow Task Design

Related Articles

- [Top 10 Methods to Improve ETL Performance Using SSIS](#)
- [Best Practices: ETL Development for Data Warehouse Projects](#)

Settings in Aggregate Transformation with all aggregate options:



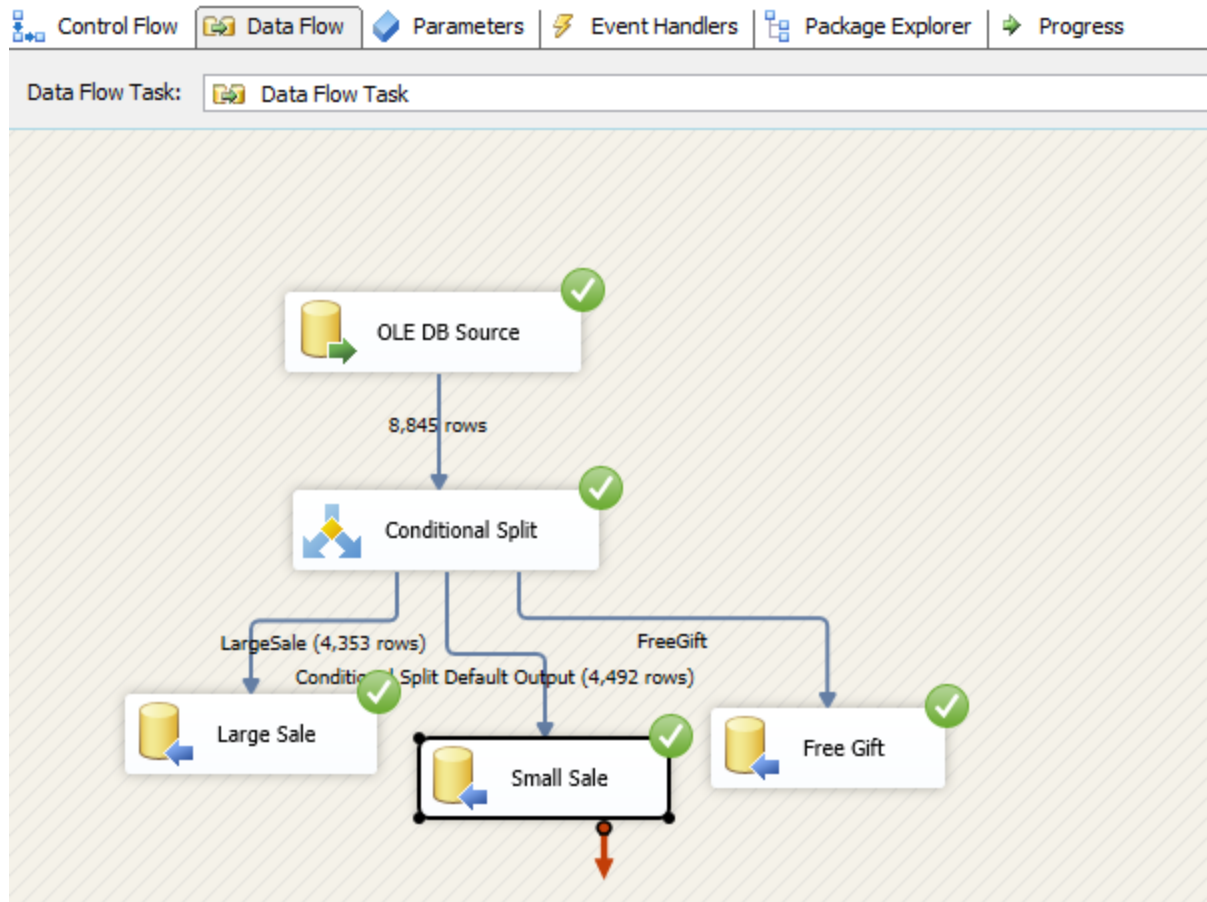
Aggregate Transformation Editor

In the above example we have applied SUM aggregation but Aggregation transformation provides other options to aggregate data like Count, Count distinct, Average, Minimum and Maximum.

Conditional Split

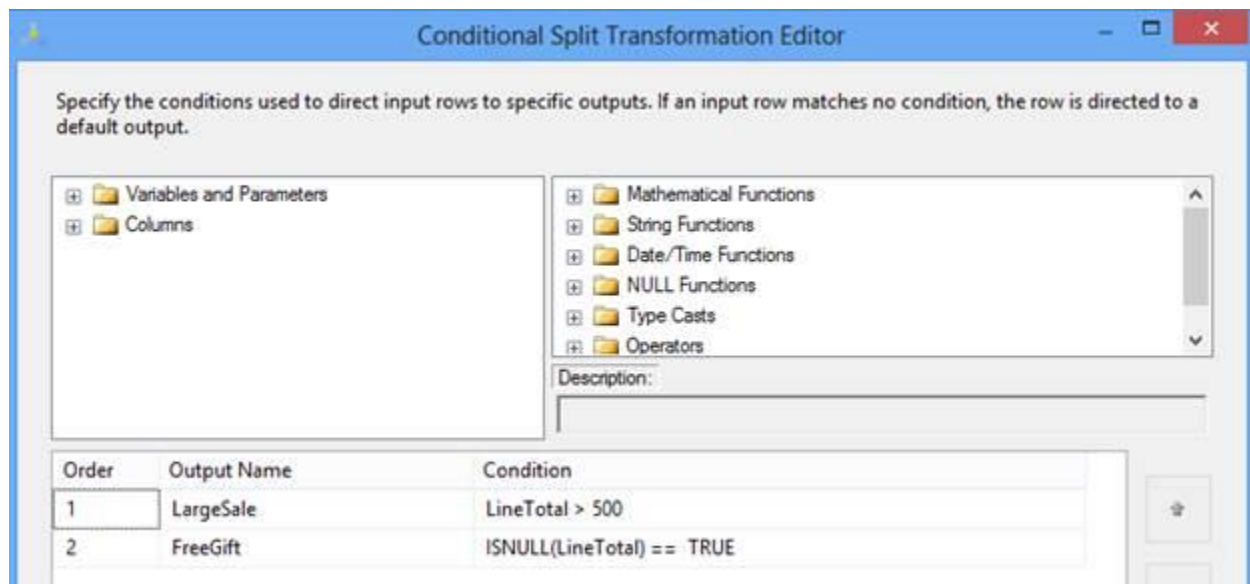
Synchronous transformation, allows you to send the data from a single data path to various outputs or paths based on conditions that use the SSIS expressions.

Data flow task design for Conditional Split:



Conditional Split Data Flow Task Design

Conditional Split transformation settings:



Conditional Split Transformation Editor

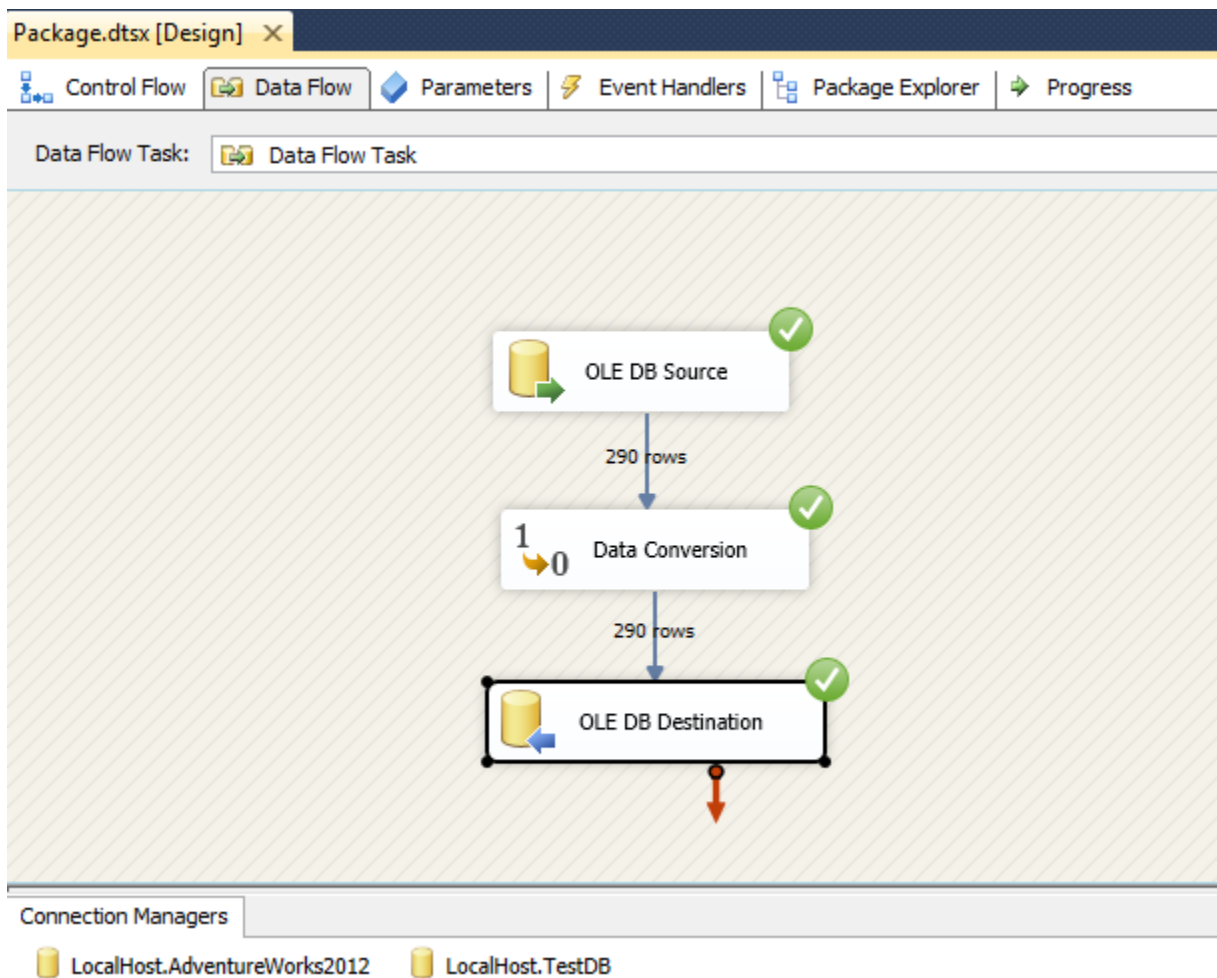
In the above example, we are splitting input records based on total order cost. If cost is more than 500, the record will be considered as part of a large sale. If LineTotal is NULL, we are assuming it's a free gift and no cost is associated with it. The rest we can consider part of small sale, in current implementation it is the default output of Conditional Split transformation.

After execution of DFT the data will move in three different destinations as per ETL design.

Data Conversion

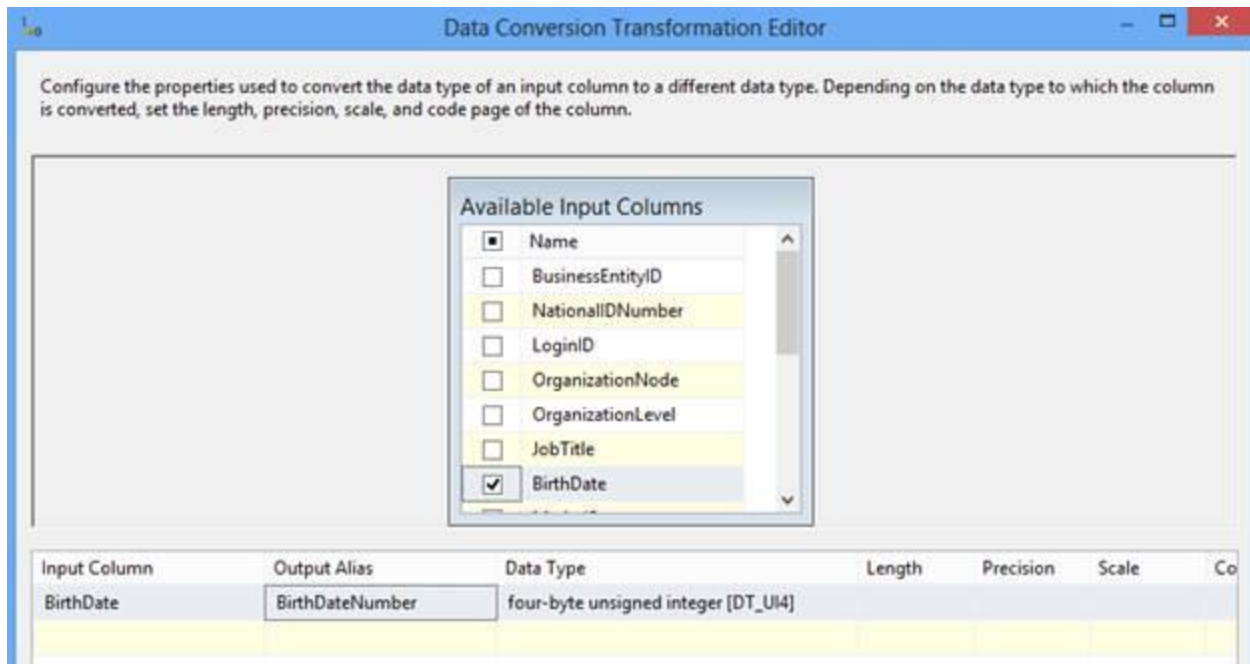
Synchronous transformation is used for data conversion. It is a similar function to the Convert or Cast functions in T-SQL. It is a very useful transformation if we are pulling same data from multiple sources.

Data flow task design for Data conversion:



Data Conversion Data Flow Task Design

Data Conversion transformation settings:



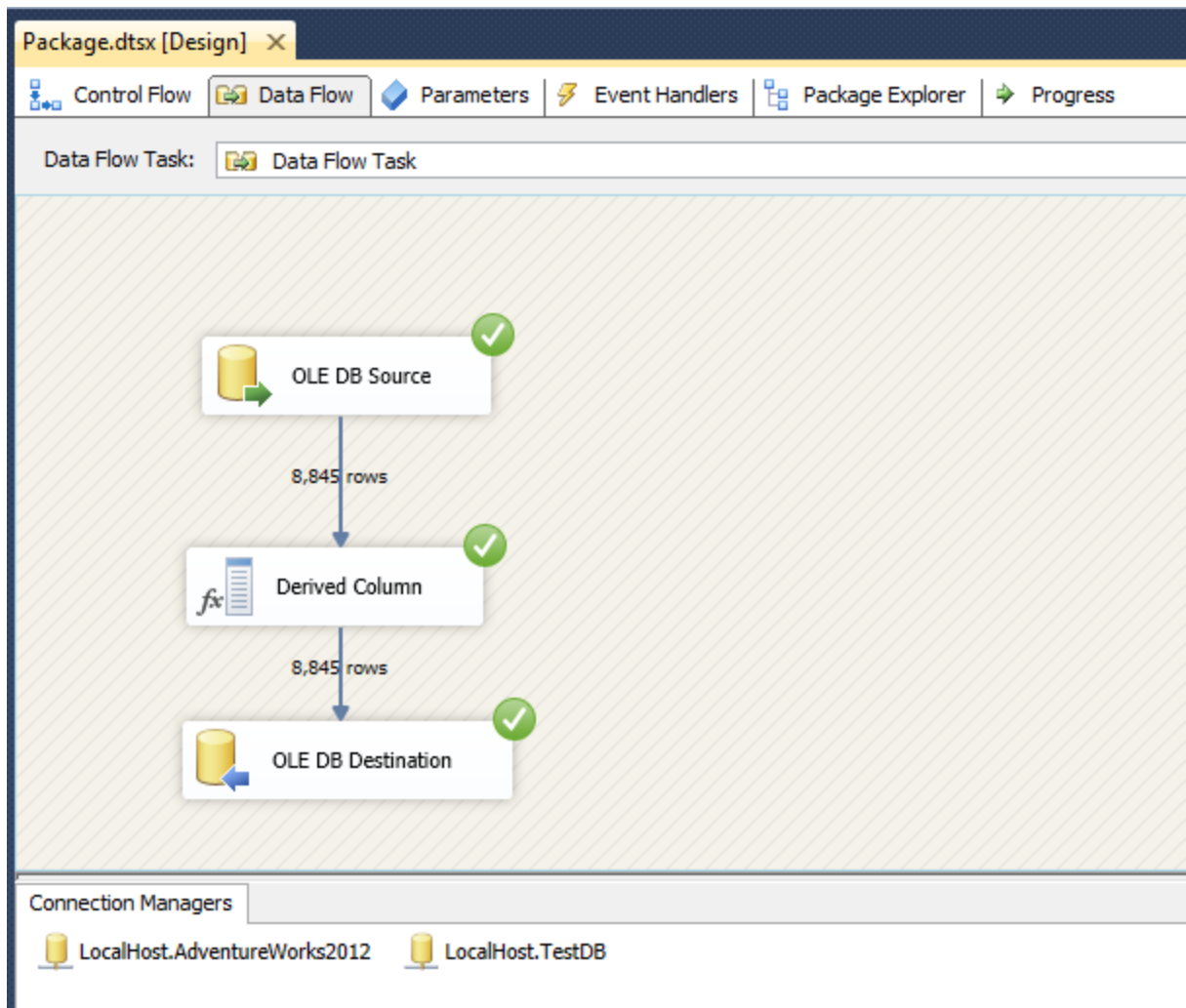
Data Conversion Transformation Editor

In this example we converted the BirthDate column of the datetime data type in another column BirthDateNumber of Integer data type.

Derived Column

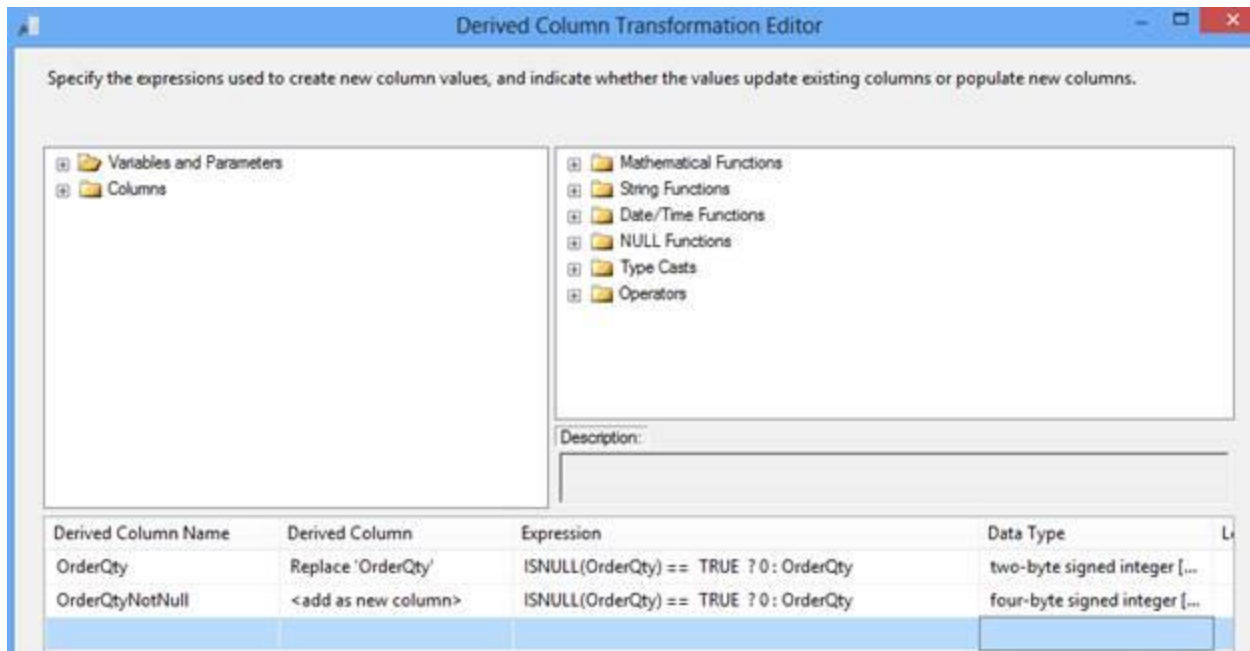
Synchronous transformation, this transformation creates a new column that is derived from the output of another column. This transformation provides you two options; either you can create a new column as a derived column or replace the existing column with a new derived column.

Data flow task design for Derived column:



Derived Column Data Flow Task Design

Derived column transformation settings:



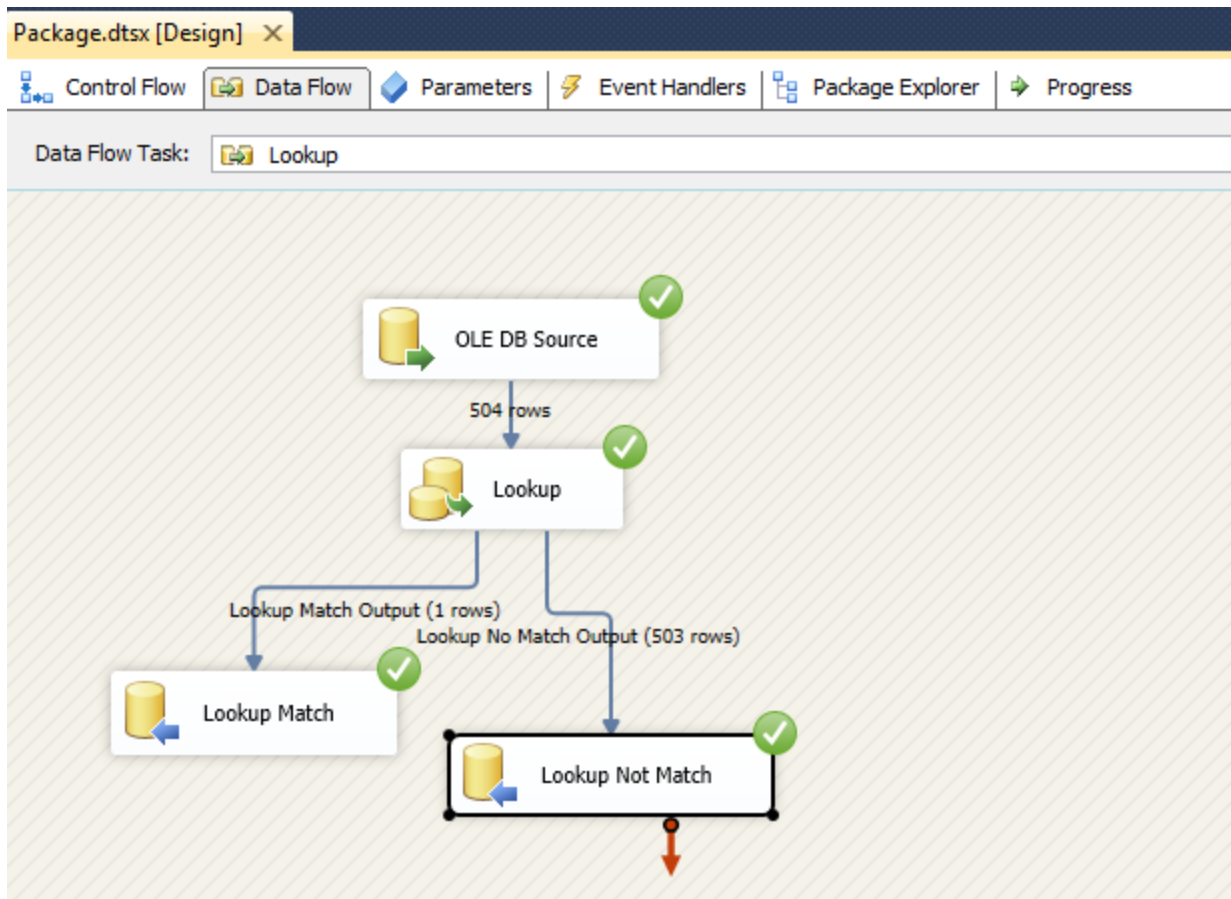
Derived Column Transformation Editor

In this example, in the first row, check the if OrderQty value is NULL then update with 0 and in the second row apply the same operation as in the first row; the only difference is it will create one new column OrderQtyNotNull in the output. So, with the help of Derived Column transformation you can either update an existing column value or introduce a new column in the output.

Lookup

Synchronous transformation, allows you to perform an equi-join between values in the transformation input and values in the reference dataset similar to T-SQL. This transformation is used to join two datasets at a time. To join more than two datasets we need to put multiple Lookup transformations, similar to a T-SQL join condition.

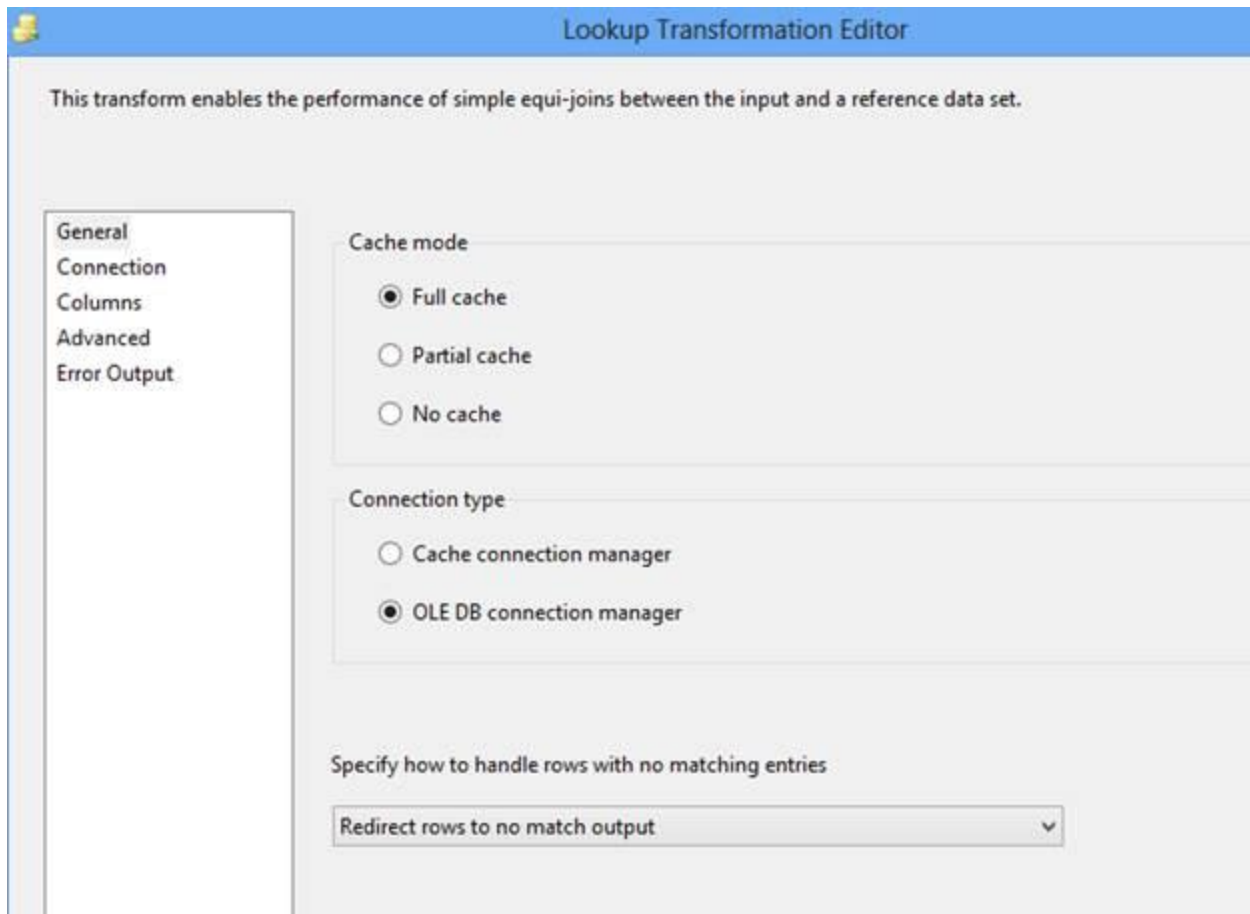
Data Flow task design for Lookup:



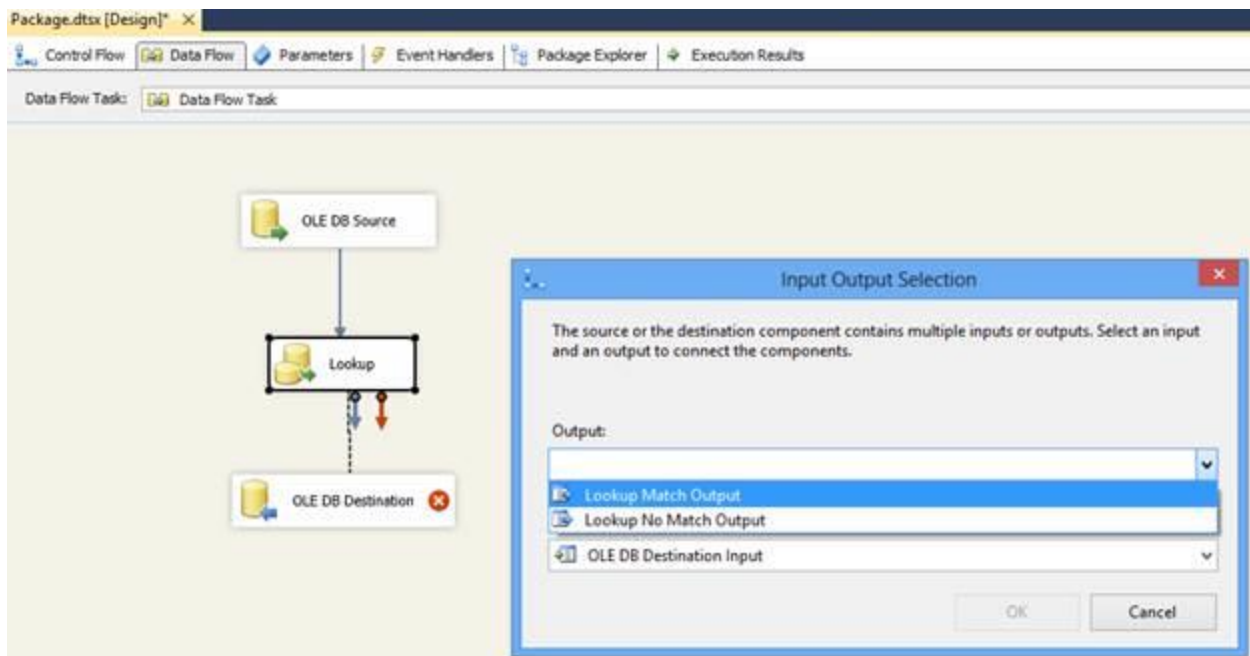
Lookup Data Flow Task Design

Lookup transformation settings:

If there is no matching entry in the reference dataset, no join occurs. By default, the Lookup transformation treats rows without matching entries as errors. However, it can configure the Lookup transformation to redirect such rows to a no match output as shown in the images below:

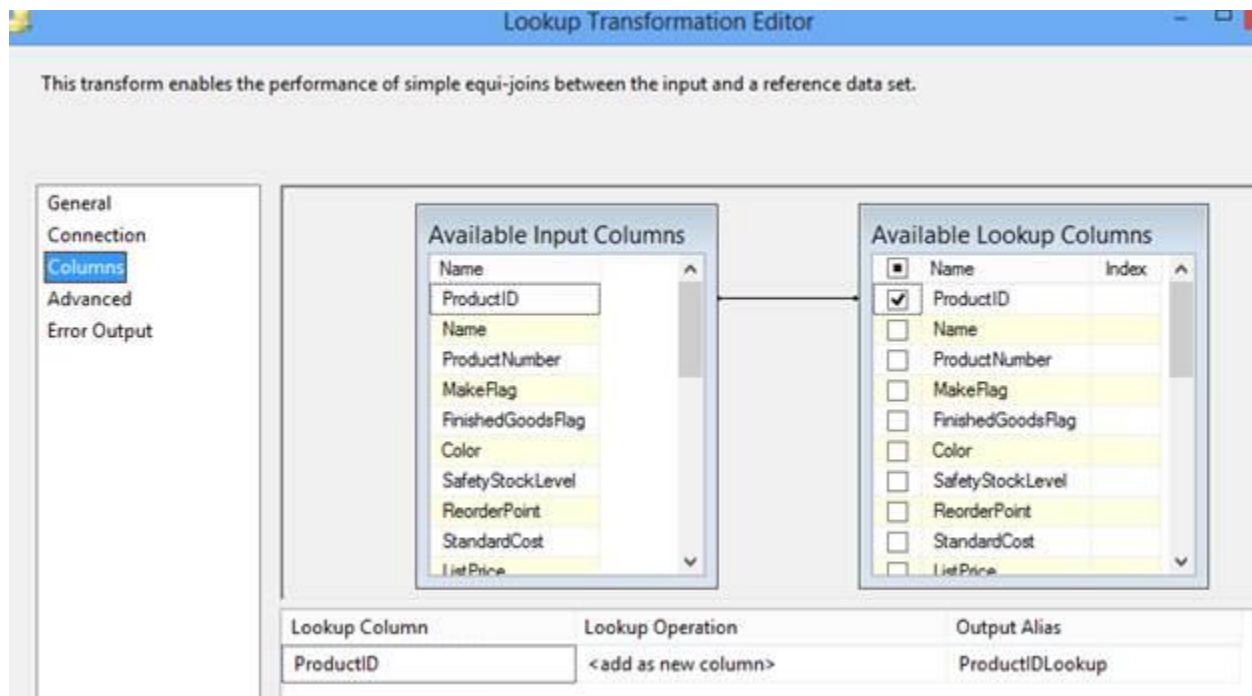


Lookup Transformation Editor



Input Output Selection

The join can be a composite join, which means that multiple columns can be used in the join in the transformation input to columns in the reference dataset; for simplification we used only one column. Refer to the below image:



Lookup Transformation Editor

In above image, you can observe in the Lookup Operation that we specified “<add as new column>”; its mean values from the reference dataset are added as a new column to the transformation output. For example, the Lookup transformation can extract the ProductID details from a table using a value from an input column, and then add the ProductIDLookup to the transformation output. The values from the reference table can replace column values or can be added to new columns.

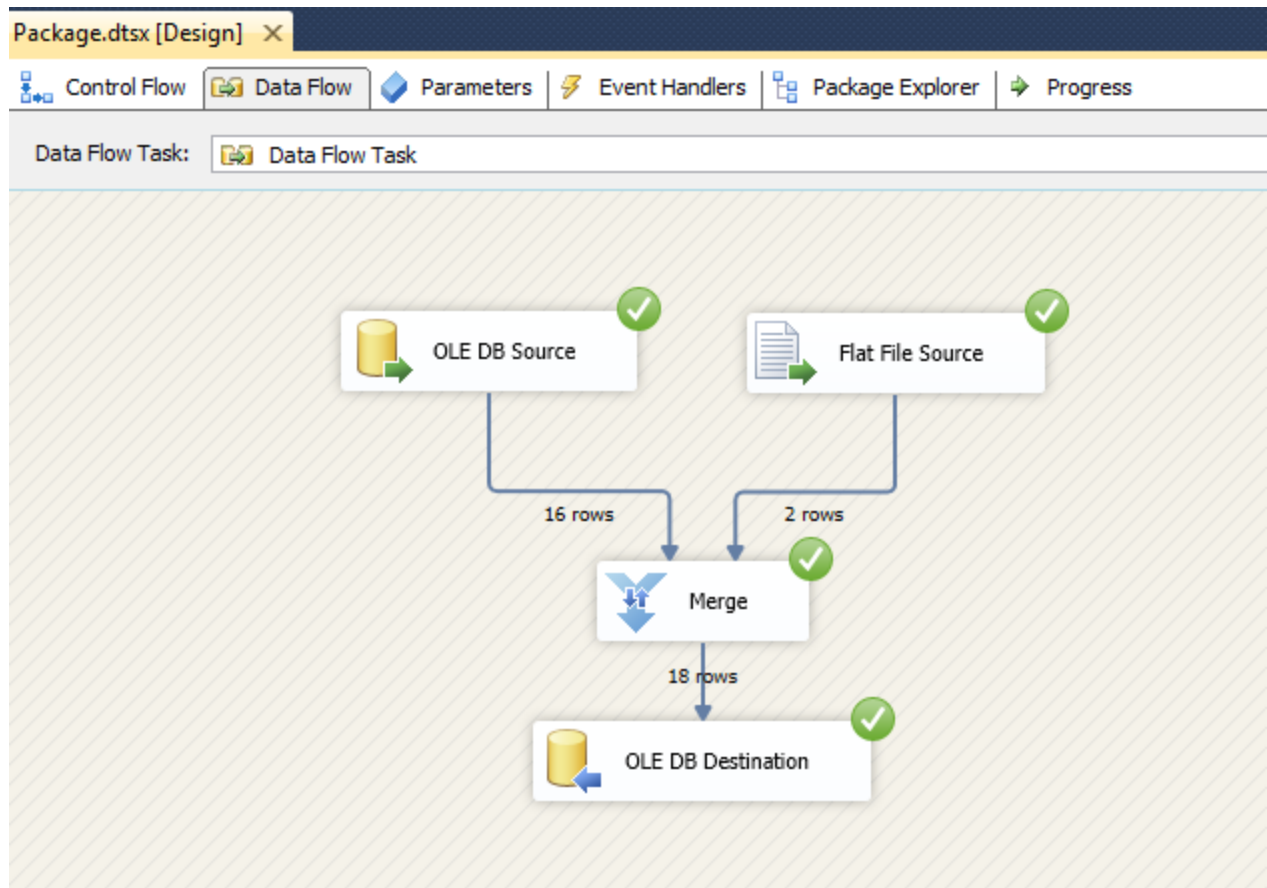
Lookup transformations provides several modes of operations, Full cache, Partial cache or No cache, that allows a trade-off between performance and resource usage.

You can refer to [MSDN](#) to learn more interesting facts about Lookup transformation.

Merge

An Asynchronous partial blocking transformation merges two sorted data sets into a single dataset. This transformation is very useful when during ETL its needs to merge data from two different data sources. Merge transformation can’t merge a column that has a numeric data type with a column that has a character data type.

Data Flow task design for Merge:



Merge Data Flow Task Design

Merge transformation settings:

The 'Merge Transformation Editor' window displays the configuration for merging two sorted inputs. It shows a table with columns for 'Output Column Name', 'Merge Input 1', and 'Merge Input 2'. The table lists four columns: 'DepartmentID (Sort key: 1)', 'Name', 'GroupName', and 'ModifiedDate'. The 'Name' column is highlighted in yellow.

Output Column Name	Merge Input 1	Merge Input 2
DepartmentID (Sort key: 1)	DepartmentID (Sort key: 1)	DepartmentID (Sort key: 1)
Name	Name	Name
GroupName	GroupName	GroupName
ModifiedDate	ModifiedDate	ModifiedDate

Merge Transformation Editor

In the above example, we are merging data from two sources; OLEDB and Flat File. The Merge transformation automatically maps columns that have the same metadata. You can then manually map other columns that have compatible data types.

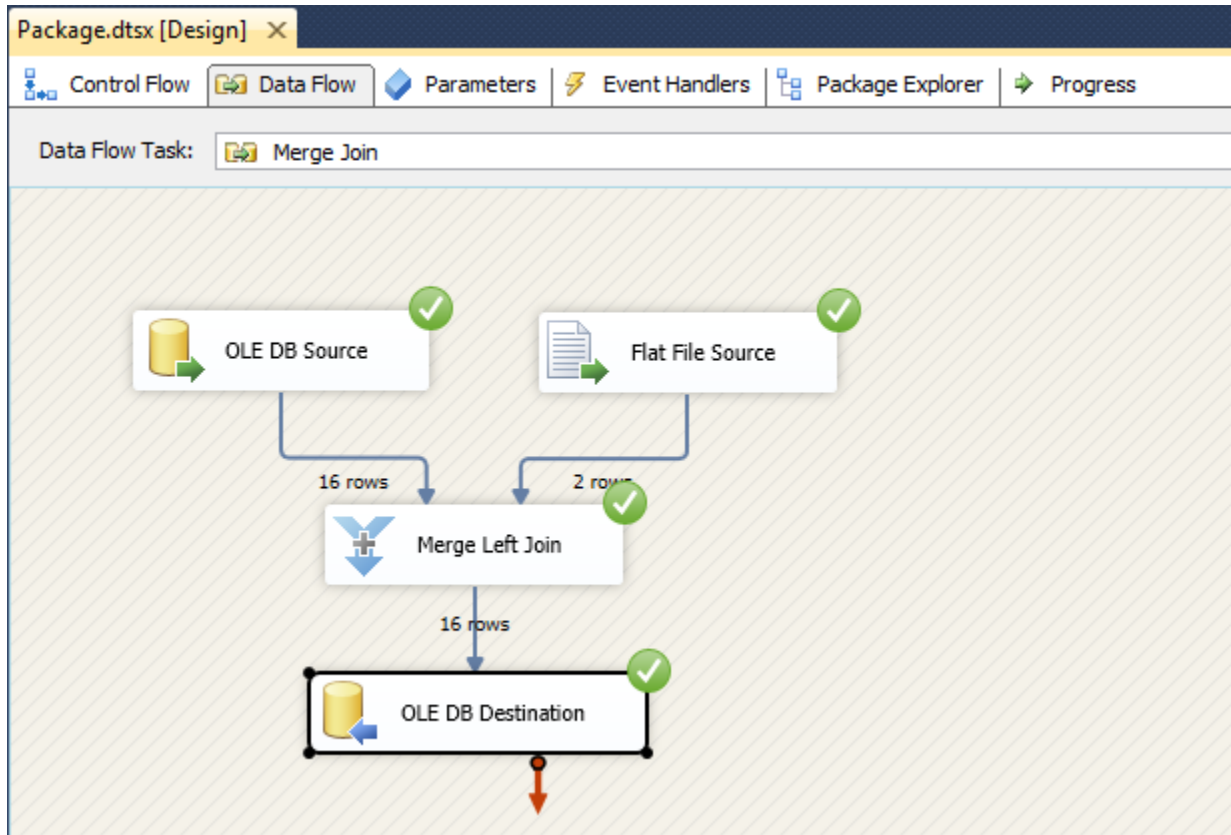
This transformation has two inputs and one output. It does not support an error output.

Merge Join

An Asynchronous partial blocking transformation, allows joining data from two sorted datasets using a FULL, LEFT, or INNER join.

It also has two inputs and one output and like Merge transformation, does not support an error output.

Data Flow task design for Merge Join:



Merge Join Data Flow Task Design

Merge Join transformation settings:

Merge Join Transformation Editor

Configure the properties used to join two sources of sorted data. Select the join type and then specify the columns to be used as the join key. Join keys must be used in the order specified by the sort-key position of the column.

Join type: Left outer join Swap Inputs

OLE DB Source

<input checked="" type="checkbox"/>	Name	Order	Join Key
<input checked="" type="checkbox"/>	DepartmentID	1	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/>	Name	0	<input type="checkbox"/>
<input checked="" type="checkbox"/>	GroupName	0	<input type="checkbox"/>
<input checked="" type="checkbox"/>	ModifiedDate	0	<input type="checkbox"/>

Flat File Source

<input type="checkbox"/>	Name	Order	Join Key
<input type="checkbox"/>	DepartmentID	1	<input checked="" type="checkbox"/>
<input type="checkbox"/>	Name	0	<input type="checkbox"/>
<input type="checkbox"/>	GroupName	0	<input type="checkbox"/>
<input type="checkbox"/>	ModifiedDate	0	<input type="checkbox"/>

Input	Input Column	Output Alias
OLE DB Source	DepartmentID	DepartmentID
OLE DB Source	Name	Name
OLE DB Source	GroupName	GroupName
OLE DB Source	ModifiedDate	ModifiedDate

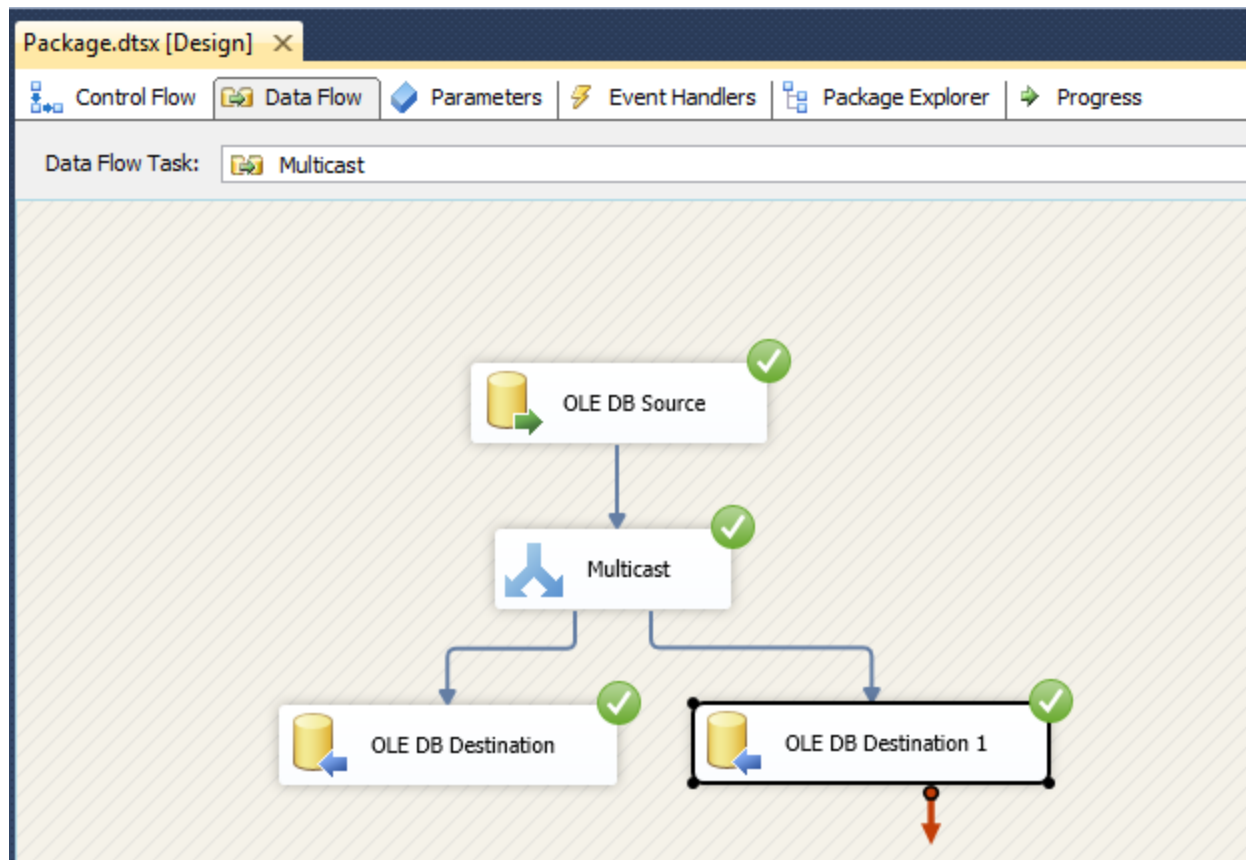
Merge Join Transformation Editor

In above example, we merged data from two different sources; OLEDB and Flat File, applying a Left outer join on DepartmentID.

Multicast

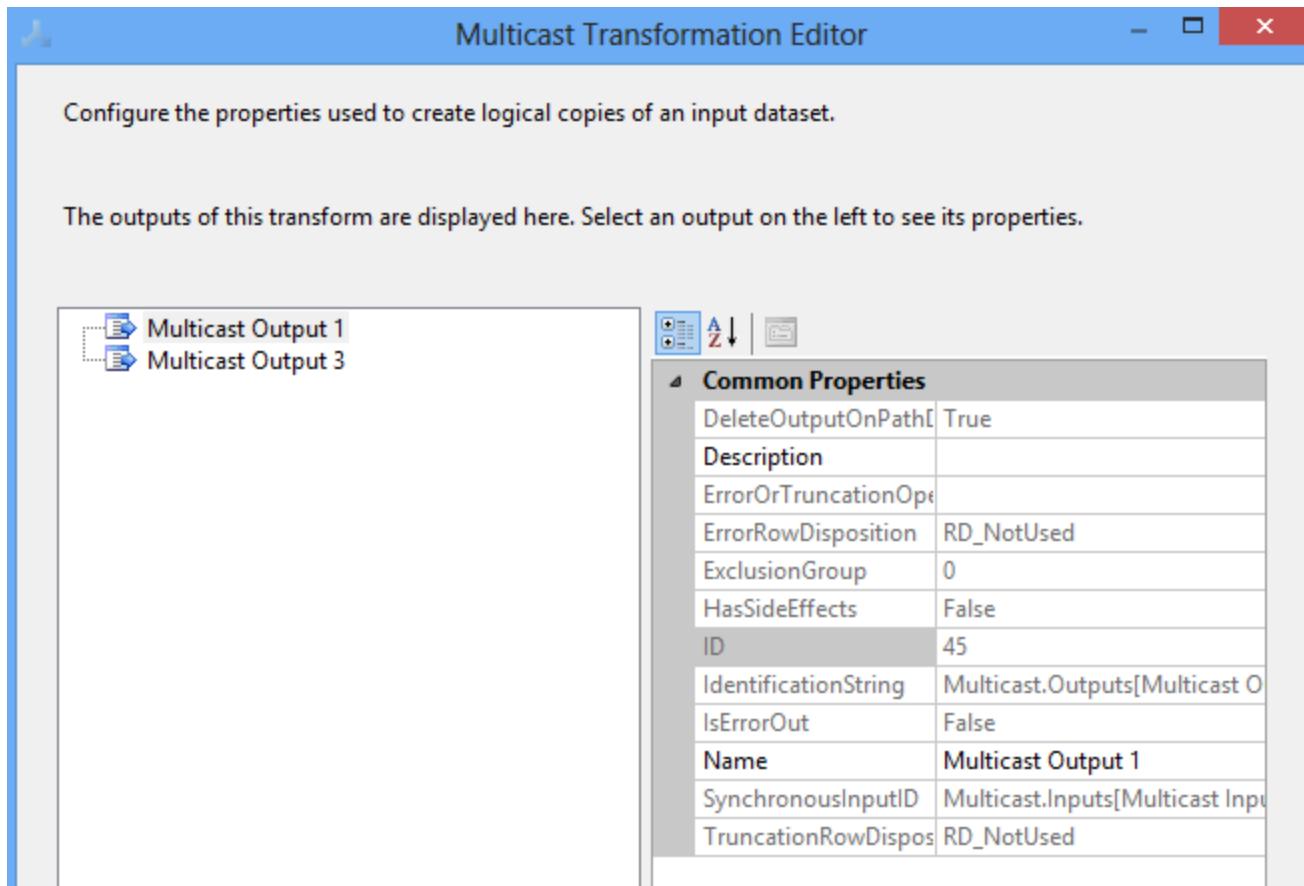
Synchronous transformation allows you to distribute its input to one or more outputs. This transformation is similar to the Conditional Split transformation. Both transformations direct an input to multiple outputs. The difference between the two is that the Multicast transformation directs every row to every output, and the Conditional Split directs a row to a single output.

Data Flow task design for Multicast:



Multicast Data Flow Task Design

Multicast transformation settings:



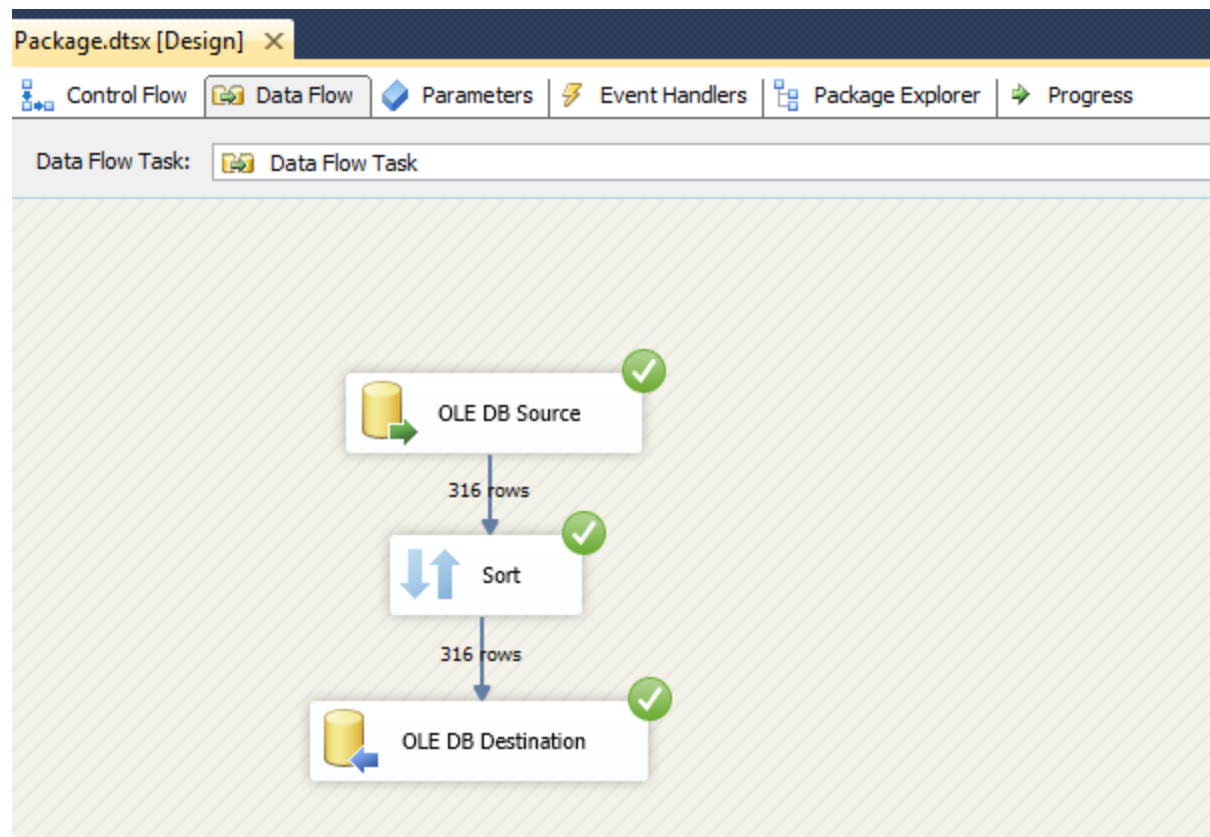
Multicast Transformation Editor

In the above example, we are distributing log data to two different destinations.

Sort

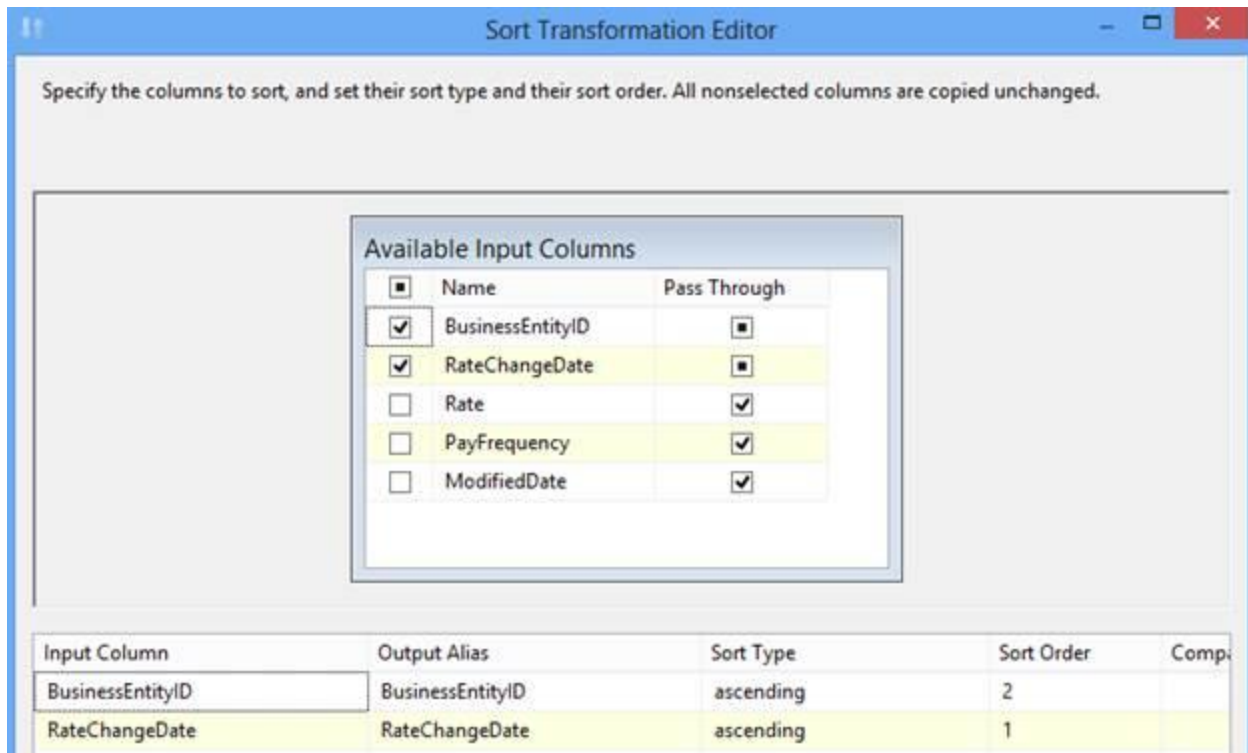
An Asynchronous full blocking transformation allows sort or arrange input data in ascending or descending order and copies the sorted data to the transformation output. You can apply multiple sorts to an input; the column with the lowest number is sorted first, the sort column with the second lowest number is sorted next.

Data Flow task design for Sort:



Sort Data Flow Task Design

Sort transformation settings:



Sort Transformation Editor

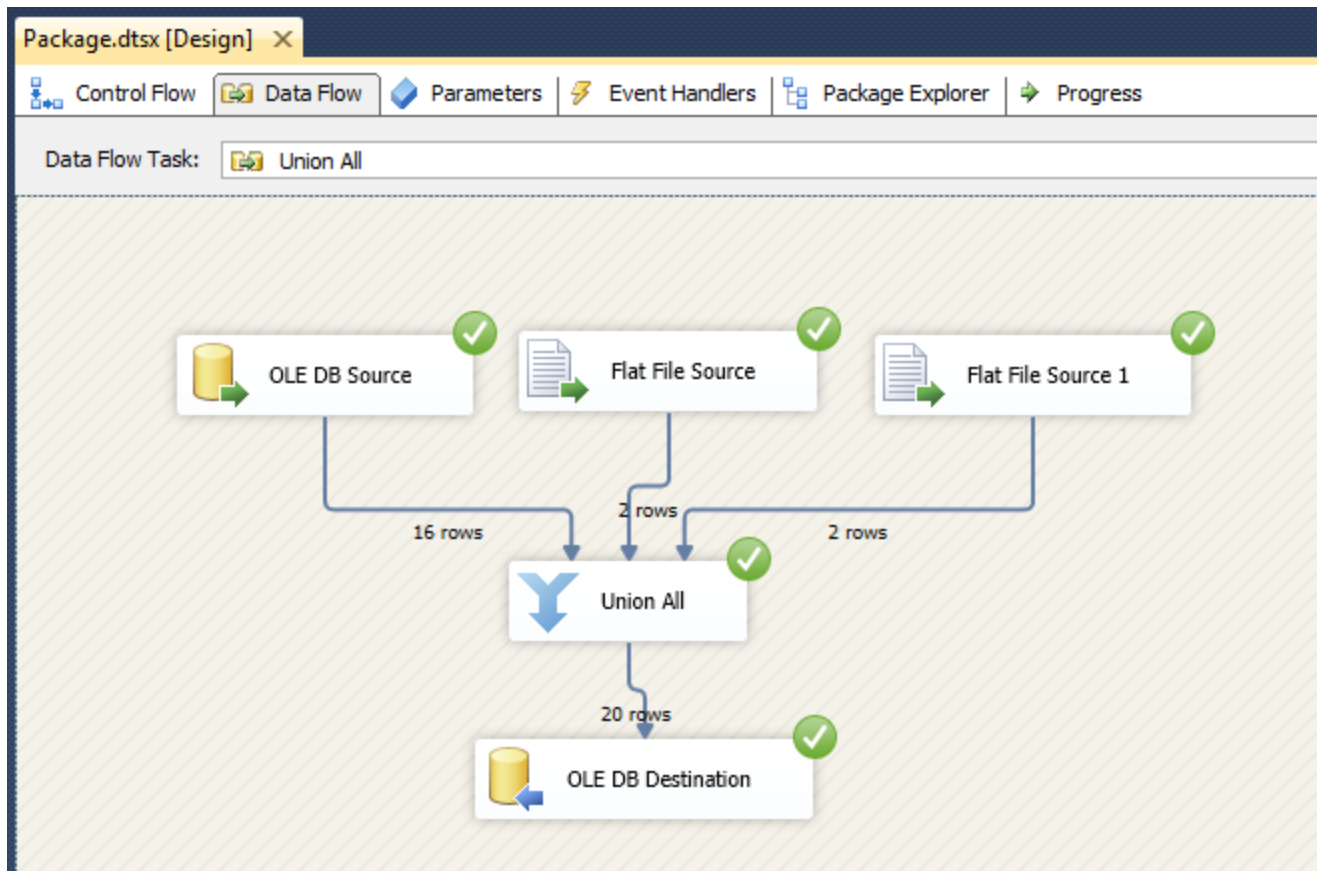
In above example, we arranged input data in ascending order of RateChangeDate first and BusinessEntityID column second.

Sort transformation has one input and one output. It does not support error outputs.

Union All

An Asynchronous partial blocking transformation, allows you to combine multiple (more than two) input and produce one output. Its add inputs to transformation output one after the other and doesn't sort the data.

Data Flow task design of Union All:



Union All Data Flow Task Design

Union All transformation settings:

The screenshot shows the 'Union All Transformation Editor' window. The title bar says 'Union All Transformation Editor'. The main text says 'Configure the properties used to merge multiple inputs into one output by creating mappings between columns.' Below this is a table with the following columns: 'Output Column Name', 'Union All Input 1', 'Union All Input 2', and 'Union All Input 3'. The rows are as follows:

Output Column Name	Union All Input 1	Union All Input 2	Union All Input 3
DepartmentID	DepartmentID	DepartmentID	DepartmentID
Name	Name	Name	Name
GroupName	GroupName	GroupName	GroupName
ModifiedDate	ModifiedDate	ModifiedDate	ModifiedDate

Union All Transformation Editor

In above example, we used three sources as input and combine all using the Union All transformation before inserting into the destination. Here, we took two different type of sources; OLEDB and Flat File.

Below is the list of transformations under both categories, which will help you to design ETL and data warehouse system.

Synchronous Transformations	Asynchronous Transformations	
	Partial blocking	Fully blocking
Audit	Data Mining Query	Aggregate
Character Map	Merge	Fuzzy Grouping
Conditional Split	Merge Join	Fuzzy Lookup
Copy Column	Pivot	Row Sampling
Data Conversion	Term Lookup	Sort
Derived Column	Union All	Term Extraction
Export Column	Unpivot	
Import Column		
Lookup		
Multicast		
OLE DB Command		
Percent Sampling		
Row Count		
Script Component		
Slowly Changing Dimension		

Export Column Transformation

The name itself explains Export Column Transformation task is useful for exporting .

The Export Column transformation reads data in a data flow and inserts the data into a file. For example, if the data flow contains product information, such as a picture of each product, you could use the Export Column transformation to save the images to files.

We can configure the Export Column transformation in the following different ways:

- Specify the data columns and the columns that contain the path of files to which to write the data.
- Specify whether the data-insertion operation appends or truncates existing files.
- Specify whether a byte-order mark (BOM) is written to the file. Export Column

Transformation task exports Binary Data – which means things like Images, Documents and other media – which have been stored in a relational database. Also Export Column Transformation task exports them out to the file system.

The main use for this would be for extracting items stored in the database, or for placing them as files as you move them from point to point in or between data flows.



Fuzzy Grouping Transformation

The Fuzzy Grouping transformation performs data cleaning tasks by identifying rows of data that are likely to be duplicates and selecting a canonical row of data to use in standardizing the data.

The Fuzzy Grouping transformation Finds close or exact matches between multiple rows in the data source and also adds columns to the output including the values and similarity scores.



Fuzzy Lookup Transformation

The Fuzzy Lookup transformation differs from the Lookup transformation in its use of fuzzy matching.

A Fuzzy Lookup transformation frequently follows a Lookup transformation in a package data flow. First, the Lookup transformation tries to find an exact match. If it fails, the Fuzzy Lookup transformation provides close matches from the reference table.

The Lookup transformation uses an equi-join to locate matching records in the reference table. It returns either an exact match or nothing from the reference table. In contrast, the Fuzzy Lookup transformation uses fuzzy matching to return one or more close matches from the reference table.

How it works : A Fuzzy Lookup transformation needs access to a reference data source that contains the values that are used to clean and extend the input data. The reference data source must be a table in a SQL Server 2000 or later database. The match between the value in an input column and the value in the reference table can be an exact match or a fuzzy match. However, the transformation requires at least one column match to be configured for fuzzy matching. If you want to use only exact matching, use the Lookup transformation instead.

This transformation has one input and one output.



Import Column Transformation

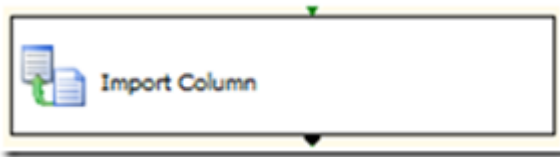
This task is an extension of Export Column transformation

The Import Column transformation reads data from files and adds the data to columns in a data flow. Using this transformation, a package can add text and images stored in separate files to a data flow.

Lets also try this explanation : Import column reads data from the file name given in the column of the input dataset. It reads the file content from the given file and add the data along with the data flow as a column data for each row.

The Import and Export Wizard protects you from the complexity of SSIS while allowing you to move data between any of these data sources:

- SQL Server databases
- Flat files
- Microsoft Access databases
- Microsoft Excel worksheets



- Other OLE DB providers

OLE DB Command Transformation

OLE DB Command Transformation is most used Transformation task in SSIS task as it directly communicate with SQL Server via SQL Query

The OLE DB Command transformation runs an SQL statement for each row in a data flow

We can can configure the OLE DB Command Transformation in the following ways

- Provide the SQL statement that the transformation runs for each row
- Specify the number of seconds before the SQL statement times out.

- Specify the default code page.

The OLE DB Command transformation in the data flow engine provides for performing data-related operations.

We can log the calls that the OLE DB Command transformation makes to external data providers. we can use this logging capability to troubleshoot the connections and commands to external data sources that the OLE DB Command transformation performs.

To log the calls that the OLE DB Command transformation makes to external data providers, enable package logging and select the Diagnostic event at the package level.



Percentage Sampling Transformation

The Percentage Sampling transformation is especially useful for data mining. By using this transformation, you can randomly divide a data set into two data sets: one for training the data mining model, and one for testing the model.

The Percentage Sampling transformation creates a sample data set by selecting a percentage of the transformation input rows. The sample data set is a random selection of rows from the transformation input, to make the resultant sample representative of the input.

Use the Percentage Sampling Transformation Editor dialog box to split part of an input into a sample using a specified percentage of rows. This transformation divides the input into two separate outputs.

Percentage Sampling Transformation When you need to give out data to call centers for telesales activities, we are generally asked to create a sample set from a data segmentation. Sometimes the requirement is defined as a percentage.

The Percentage Sampling transformation uses an algorithm to select at random the number of rows according to the specified percentage.



Pivot Transformation

The Pivot transformation makes a normalized data set into a less normalized but more compact version by pivoting the input data on a column value.

- Pivot transformation is being used to transpose rows into columns just like Excel Transpose
- Pivot transformation T-SQL also, we have PIVOT command available which is more reliable and faster than SSIS
- Pivot in SSIS is not recommended generally

The Pivot transformation does much exactly what that name implies. This pivots data along an x-axis which is determined by values in a column. Another column of value are used along the y-axis to determine the columns to include.

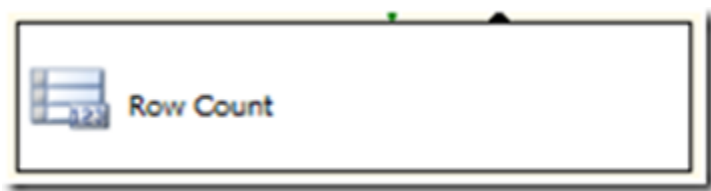
Since this columns on the y-axis are determined by the data in the source there is some configuring in the Pivot transformation that needs to occur to properly pivot data.



Row Count Transformation

The Row Count transformation counts rows as they pass through a data flow and stores the final count in a variable.

Also this transformation stores the row count value in the variable only after the last row has passed through the transformation. Therefore, the value of the variable is not updated in time to use the updated value in the data flow that contains the Row Count transformation. You can use the updated variable in a separate data flow.



Row Sampling Transformation

The Row Sampling transformation is useful during package development for creating a small but representative dataset. You can test package execution and data transformation with richly representative data, but more quickly because a random sample is used instead of the full dataset. Because the sample dataset used by the test package is always the same size, using the sample subset also makes it easier to identify performance problems in the package.

Row Sampling Transformation The Row Sampling transformation works quite similar to the Percentage Sampling transformation to sample a data set.

The Row Sampling transformation outputs an exact number of rows as specified in the transformation.

This random selection of a precise number of rows is sometimes very useful. An example of such a scenario can be a gift allocation to the random selection of people. Suppose you're running a campaign to introduce your new product to different segments of your customers and prospects by sending them an e-mail every week.

The Row Sampling transformation is used to obtain a randomly selected subset of an input dataset. You can specify the exact size of the output sample, and specify a seed for the random number generator

The Row Count transformation counts rows as they pass through a data flow and stores the final count in a variable.

A SQL Server Integration Services package can use row counts to update the variables used in scripts, expressions, and property expressions.



Script Component Transformation

Script Component is also one most important and highly used Task.

There is lots of Difference between Script Task and Script Component

Script Task – Used in Control flow Tab

Script Component – Used in Data Flow Tab

The Script task provides code to perform functions that are not available in the built-in tasks and transformations that SQL Server Integration Services provides. The Script task can also combine functions in one script instead of using multiple tasks and transformations. You use the Script task for work that must be done once in a package (or once per enumerated object), instead than once per data row.

We can use the Script task for the following purposes:

- Access data by using other technologies that are not supported by built-in connection types. For example, a script can use Active Directory Service Interfaces (ADSI) to access and extract user names from Active Directory.

- Create a package-specific performance counter. For example, a script can create a performance counter that is updated while a complex or poorly performing task runs.
- Identify whether specified files are empty or how many rows they contain, and then based on that information affect the control flow in a package. For example, if a file contains zero rows, the value of a variable set to 0, and a precedence constraint that evaluates the value prevents a File System task from copying the file.



Slowly Changing Dimension Transformation

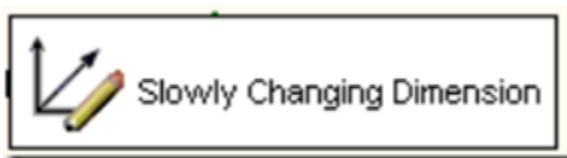
The Slowly Changing Dimension transformation coordinates the updating and inserting of records in data warehouse dimension tables

The Slowly Changing Dimension task help us to insert and update Dimensions in in Data warehouse , Wizard's for same is most simple among all other task

Slowly changing dimensions reflect the natural tendency to modify dimension member values over time

The Slowly Changing Dimension transformation provides the following functionality for managing slowly changing dimensions:

- Matching incoming rows with rows in the lookup table to identify new and existing rows.
- Identifying incoming rows that contain changes when changes are not permitted.
- Identifying inferred member records that require updating.
- Identifying incoming rows that contain historical changes that require insertion of new records and the updating of expired records.
- Detecting incoming rows that contain changes that require the updating of existing records, including expired ones.



Term Extraction Transformation

Term Extraction transformation uses various defined data mining algorithms to to extract nouns and OR or phrases that are passed through it and then give a score to each term or phrase based on the frequency of its occurrence

The Term Extraction transformation extracts terms from text in a transformation input column, and then writes the terms to a transformation output column. The transformation works only with English text and it uses its own English dictionary and linguistic information about English

The Text Extraction transformation uses internal algorithms and statistical models to generate its results. You may have to run the Term Extraction transformation several times and examine the results to configure the transformation to generate the type of results that works for your text mining solution.

The Term Extraction transformation has one regular input, one output, and one error output.



Term Lookup Transformation

Term Lookup transformation performs a lookup, it extracts words from the text in an input column using the same method as the Term Extraction transformation:

- Text is broken into sentences.
- Sentences are broken into words.
- Words are normalized.

Use the Term Lookup tab of the Term Lookup Transformation Editor dialog box to map an input column to a lookup column in a reference table and to provide an alias for each output column.

Using the Term Lookup transformation, we can count the number of times a text term occurs in the input data row and create custom word lists and word frequency statistics.

This transformation reads the terms from a lookup table to look for matches in an input column and then, by default, adds two columns named Term and Frequency to the output containing the term and the count for the term.



Union All Transformation

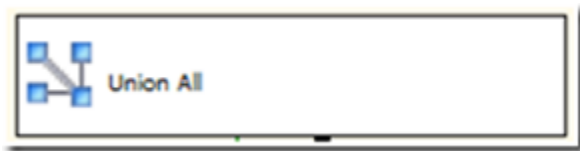
The Union All transformation combines multiple inputs into one output.

The transformation inputs are added to the transformation output one after the other; no reordering of rows occurs. If the package requires a sorted output, you should use the Merge transformation instead of the Union All transformation.

The first input that we need to connect to the Union All transformation is the input from which the transformation creates the transformation output. The columns in the inputs we can subsequently connect to the transformation are mapped to the columns in the transformation output.

As we have already seen The Merge transformation is similar to the Union All transformations. Use the Union All transformation instead of the Merge transformation in the following situations:

- The transformation inputs are not sorted
- The combined output does not need to be sorted.
- The transformation has more than two inputs.



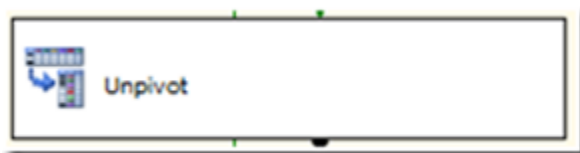
Unpivot Transformation

The Unpivot transformation makes an un normalized dataset into a more normalized version by expanding values from multiple columns in a single record into multiple records with the same values in a single column.

the SSIS Unpivot transformation does the the opposite of a Pivot transformation

Unpivot transformation will convert data in columns into rows. Data that as previously been pivoted is difficult to manipulate further- so this is where the unpivot transformation.

User interface also is also very easy as compare to actual scripting of pivot and unpivot in T-SQL



Sort Transformation

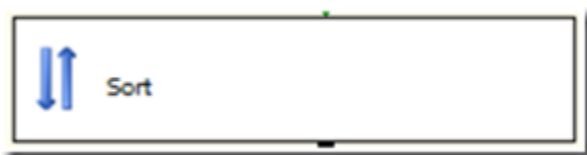
The Sort transformation sorts input data in ascending or descending order and copies the sorted data to the transformation output

Sort Transformation can be used to sort incoming data stream. In addition, the Sort Transformation distinct option can be used to remove duplicate values from the input.

Sort Transformation is a blocking transformation meaning that the input records are accumulated until the end of input. Blocking transformations affect the performance of overall dataflow because subsequent steps cannot execute until all the records have been received and processed by the blocking transformation.

Sort Transformation uses storage on the server for temporary data during sorting. The server must have enough capacity to store the entire data set and index.

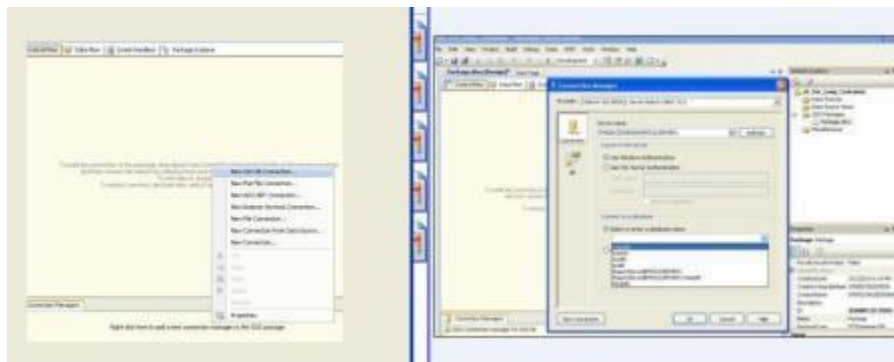
The Sort transformation includes a set of comparison options to define how the transformation handles the string data in a column. For more information.



Containers :

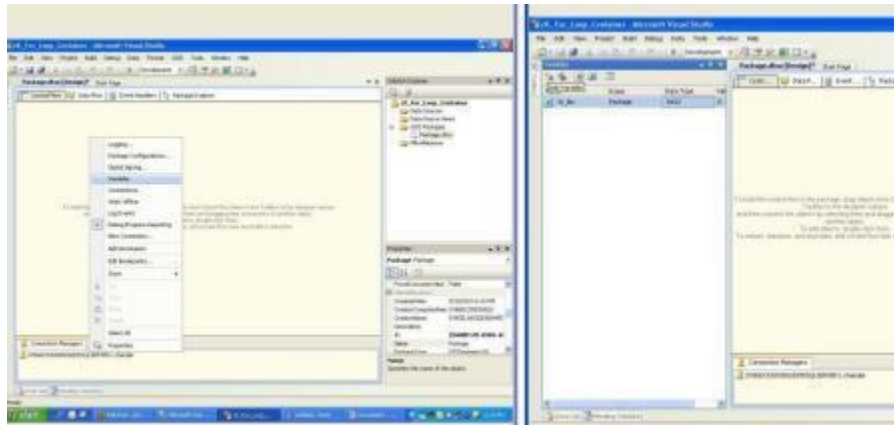
For loop containers

STEP 3. Add a **New OLEDB Connection** to the Connection Manager. Click New button and configure OLEDB Connection Manager pointing to your **working database** (as in this example, I am using **chander** as my database) and Click OK.



STEP 4. Open the **cK_For_Loop_Container.dtsx package**, Create a new **Variable**, and name it (say Sr_No).

- To open the Variables window, Right click in the design pane and select Variables or click on the Variables icon on the top right of your Package designer screen. Once the window is open, click the Add Variable button. Accept defaults for the Variable i.e *Data Type* – int32, *Scope* – Package level and a *value* to be 1.



STEP 5. Go to **For Loop Container** in the toolbox and **Drag** it to the **Control Flow** and Double click it to open the editor. It will have the following fields:-

(1). *InitExpression* :- This expression is **Optional** and evaluated once at the beginning only. It is used to **initialize a variable** that will be used in the For Loop Container.

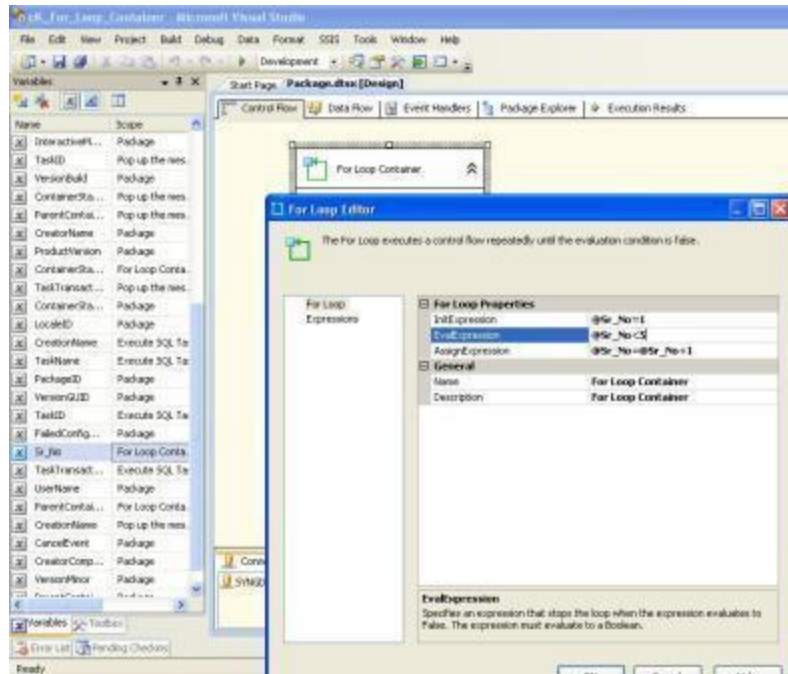
- Set the InitExpression option to – **@Sr_No= 1**. This will initialize the loop by setting the Counter variable to 1.

(2). *EvalExpression* :- This expression is **Required** and also evaluated before any work is performed inside the container. This is the expression that determines if the **loop continues or terminates**. If the expression entered evaluates to TRUE, the loop executes again. If it evaluates to FALSE, the loop ends.

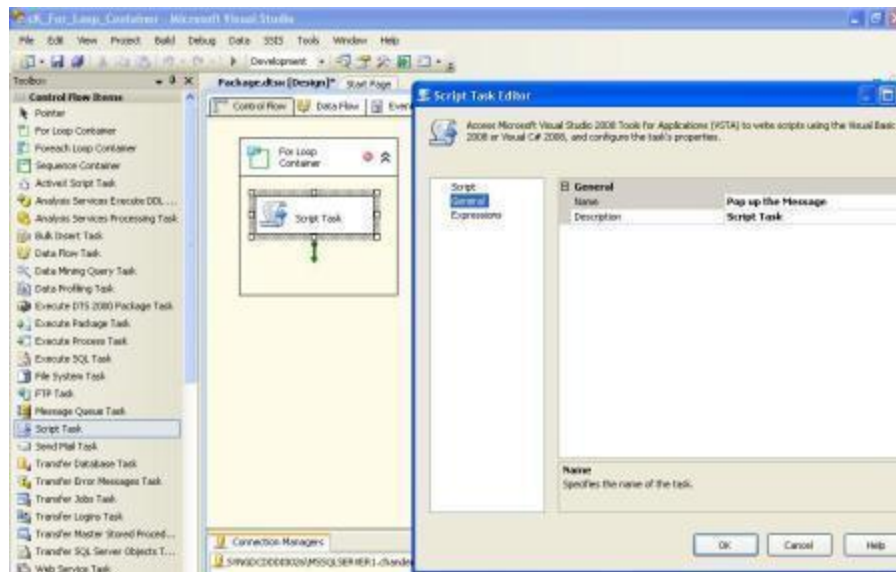
- Set the EvalExpression option to – **@Sr_No <=5**.

(3). *AssignExpression* :- This is the last expression used in For Loop and is **optional**. It changes the value of the variable used in the EvalExpression.

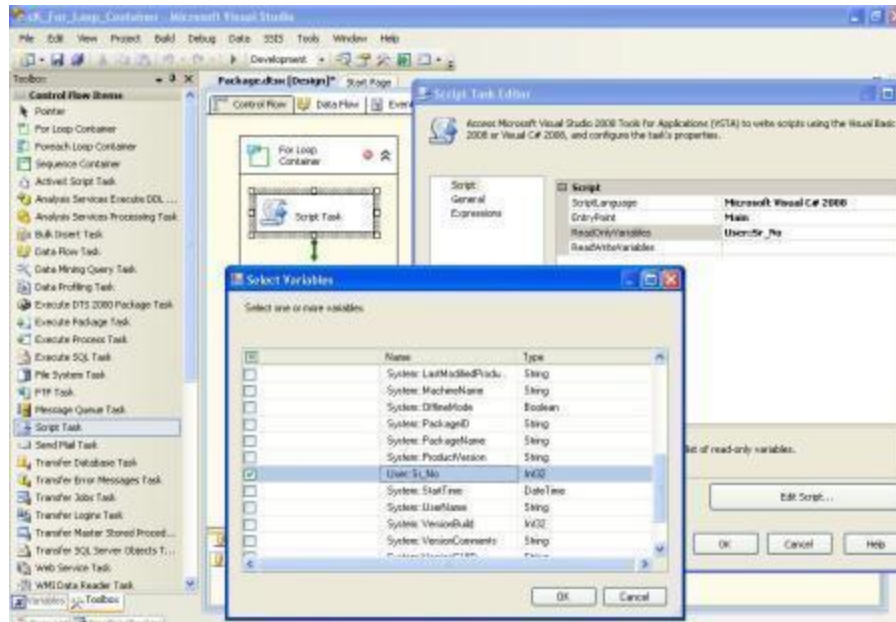
- Set it to – **@Sr_No = @Sr_No + 1** for the AssignExpression and Click OK.



STEP 6. Drag a **Script Task** into the **For Loop Container** and double-click the script task to edit it. In the **General** tab, name the task *Pop up the message*.



STEP 7. In the **Script** tab, set the Read Only Variables to Sr_No. Finally, click **Edit Script** to open the Visual Studio designer. By typing Sr_No for that option, you're going to pass in the variable parameter to be used by the Script Task.



STEP 8. When you click **Edit Script**, the Visual Studio design environment will open. Replace the Main () subroutine with the following code. This code will read the variable and pop up a message box that displays the value of the variable:-

- For C#**

```
Public void Main ()
{
    System.Windows.Forms.Message
    Dts.TaskResult = (int)ScriptRes
}
```

```
1  Public void Main ()
2  {
3  System.Windows.Forms.MessageBox.Show("Sr_No = " + Dts.Variables["User::Sr_No"].Value.ToString());
4  Dts.TaskResult = (int)ScriptResults.Success;
5  }
```

- For VB**

```
Public Sub Main ()
    MessageBox.Show (Dts.Variab
    Dts.TaskResult = ScriptResults.
End Sub
```



```

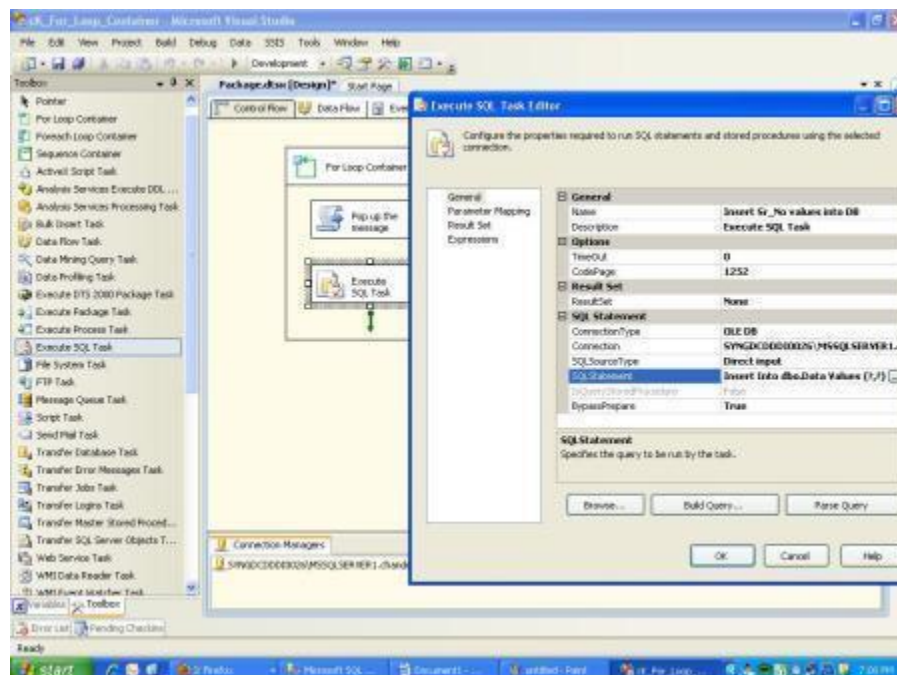
1 Public Sub Main ()
2     MessageBox.Show(Dts.Variables("Sr_No").Value.ToString())
3     Dts.TaskResult = ScriptResults.Success
4 End Sub

```

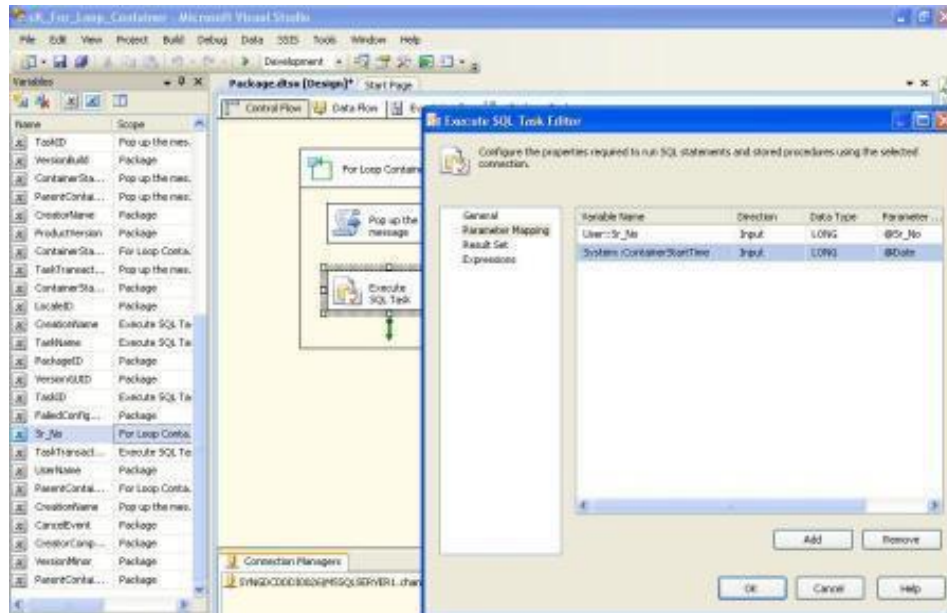
Save and exit the Visual Studio design environment, then Click **OK** to exit the **Script Task**.

STEP 9. Drag an **ExecuteSQLtask** into the **For Loop** container and double click the task to **edit** it.

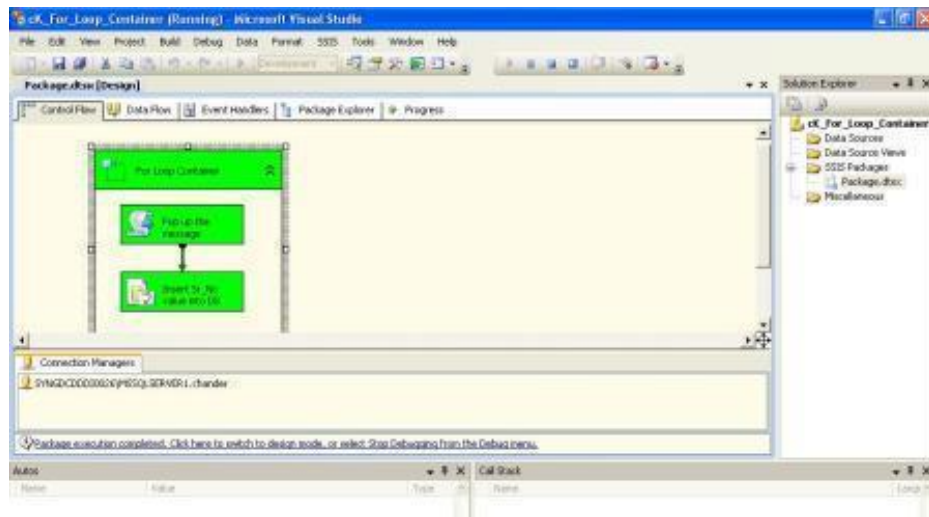
- Set the **Name** as Insert Sr_No values into DB.
- Set the **Description** as Execute SQL Task.
- Set **Connection Type** as OLEDB and set **Connection** to the connection created in STEP 3.
- Set **SQLSourceType** as Direct Input and in the **SQLStatement**, put the following insert statement :- *Insert Into dbo.Data Values (?,?)*



STEP 10. Inside the **Execute SQL Task** editor, Go to **Parameter mapping** tab and create two parameters as shown in the figure.

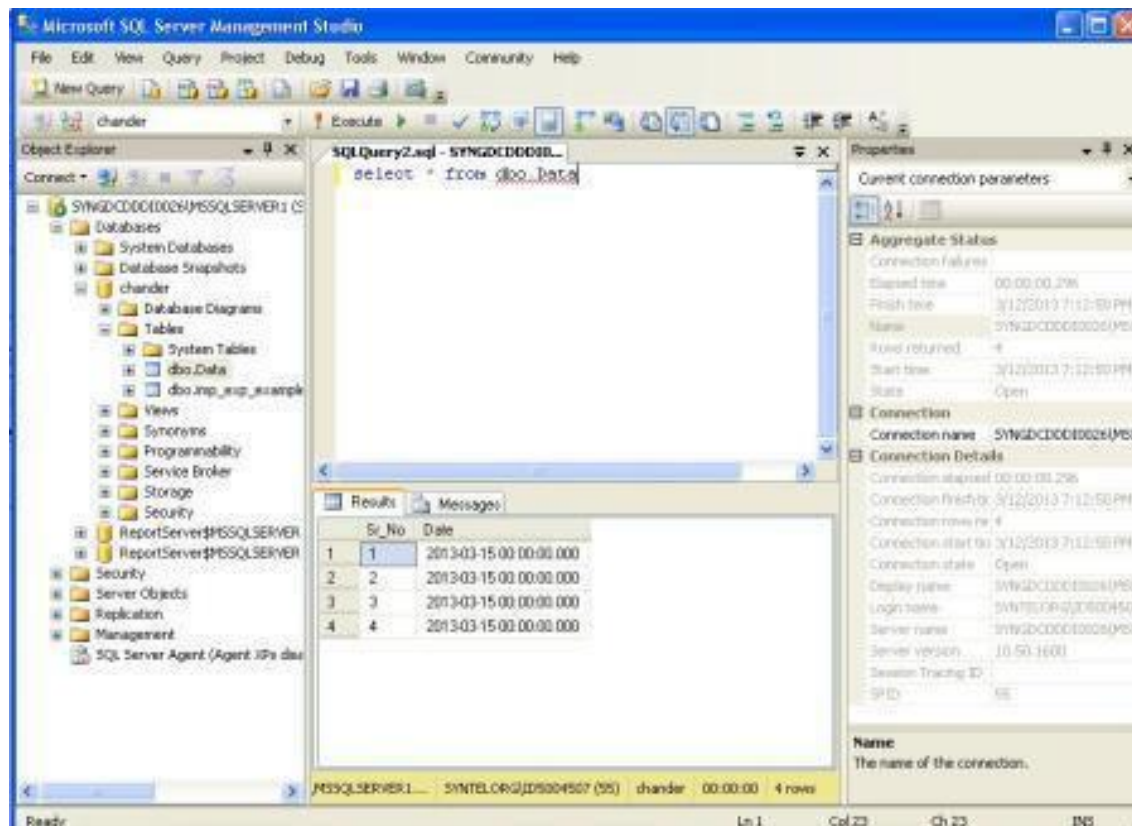


STEP 11. Connect the **success precedence constraint** from Script task to the **ExecuteSQLtask**.



STEP 12. Execute the **package**, you should see **message box popup** with each new value of **Sr_No** variable. That is, you should see **five pop-up boxes** one at a time, starting at iteration 1 and proceeding through iteration 4. Only one pop-up will appear at any given point. After the loop is complete, the **For Loop Container**, the **ScriptTask** and the **ExecuteSQL task** will all be **green**. Check the rows in the **Data table**.

Select * from dbo.Data



This completes the **implementation of For Loop Container**. I hope you find this post interesting. Your comments are invited and I will be happy to solve your queries

Description – Earlier in SQL Server 2000, Data Transformation Service (DTS) was used to loop over **data files** of a given type present in a directory and to **import** them into the **destination** (In simple words, moving or copying files based on the file name from one directory to another) was very hard to perform. But with the release of **SQL Server 2005**, this has been solved by adding **Foreach Loop Container**. So, now you would be having an idea what this container performs and we will learn how it performs that.

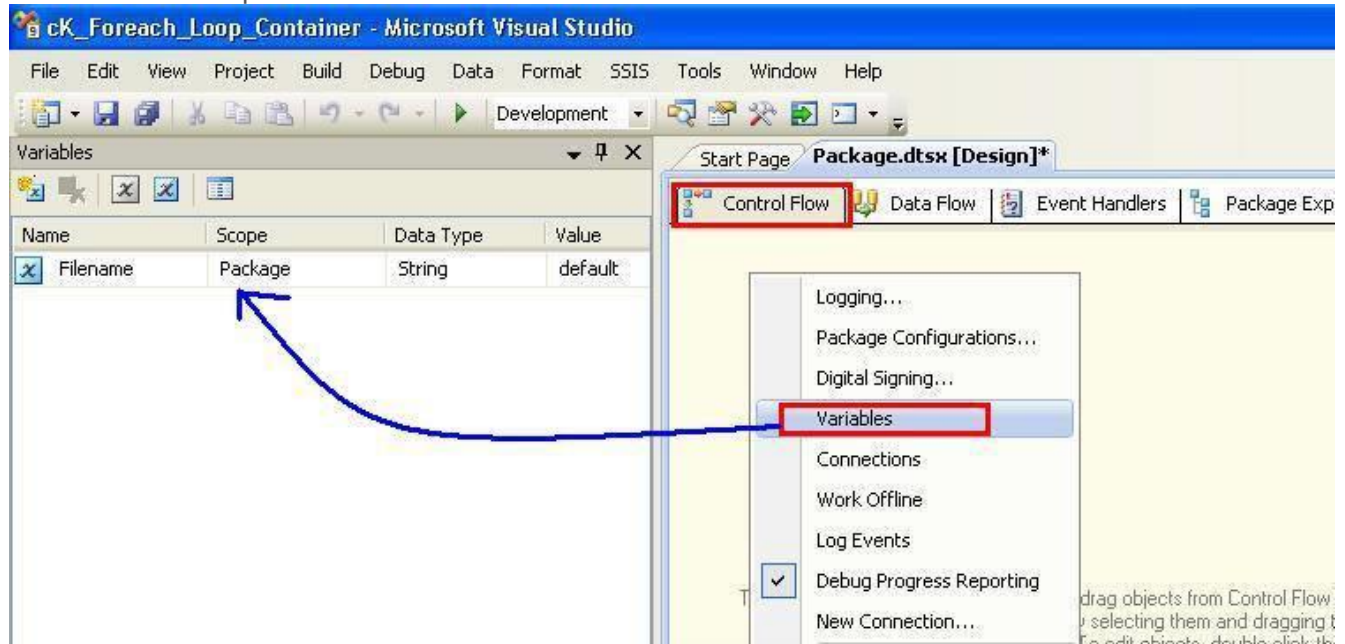
Example Scenario :- We will create **one Source folder** containing some data files (say text files) in it. Then we will create a **package** and with the help of **Foreach Loop Container**, these files will be **copied** to another folder which will act as our **Destination**.

STEPS TO FOLLOW :-

Step 1. Create a **Source** folder (I have created a folder named **Chander**) containing some text files and Create an empty **Destination** Folder where you want to **copy** these files.

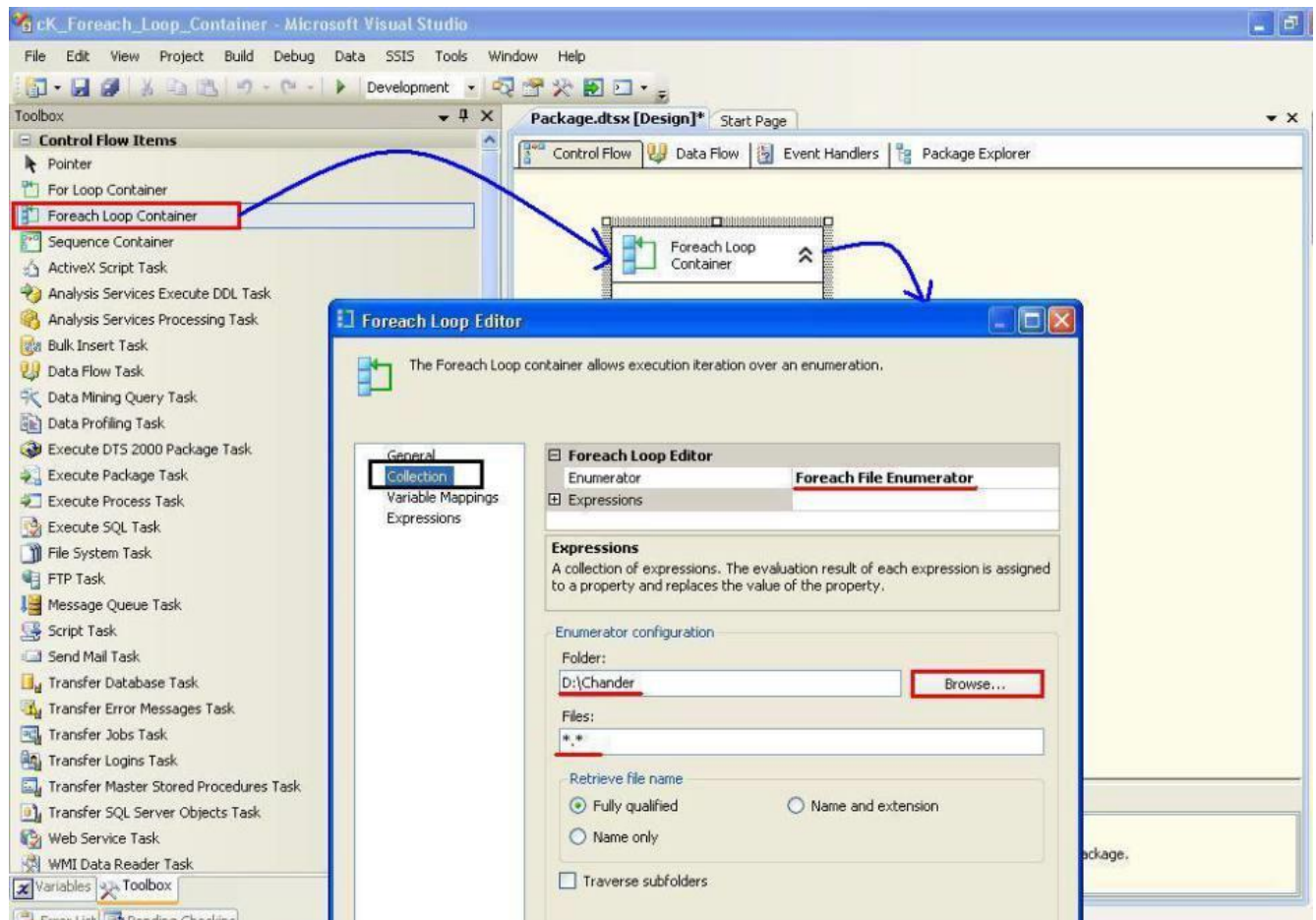
Step 2. Create a **package** using Business Intelligence Development Studio (**BIDS**).

Step 3. Create a **variable** with the name *Filename* with *Scope*-Package, *Data type*-String and *Value*-Default. This variable will hold the name of the **file** that SSIS is working on during each **iteration** of the loop.

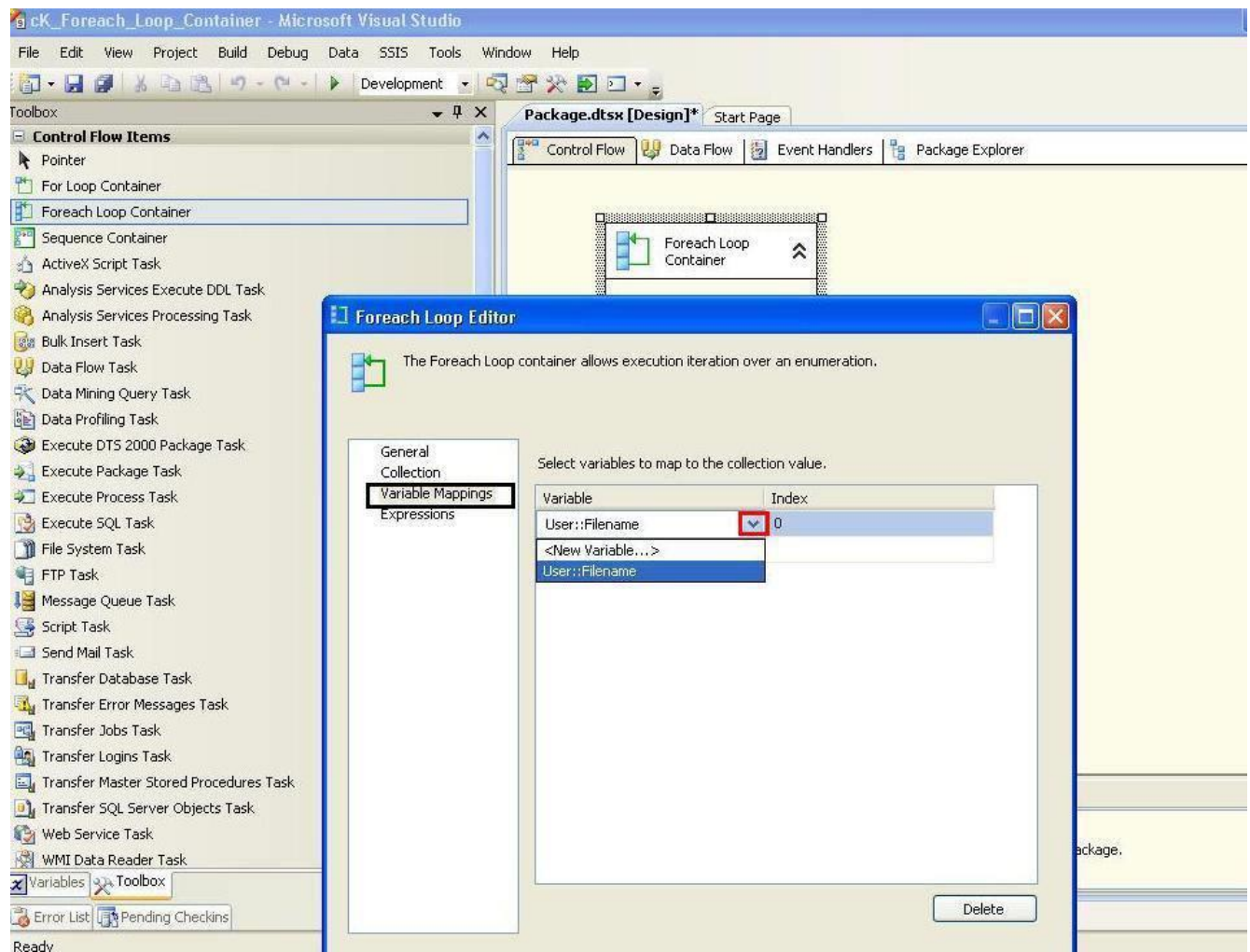


Step 4. Next, drag a **Foreach Loop Container** onto the Control Flow and double click on the container to **configure** it. Go to **Collection tab**, and specify the following fields as:-

- Select the option **Foreach File Enumerator** under Foreach Loop Editor.
- In the **Enumerator configuration** group, Go to *folder* field and click Browse and set the folder property to the **source folder** that has data files in it.
- Set the **Files** property to default i.e. < *.* >.

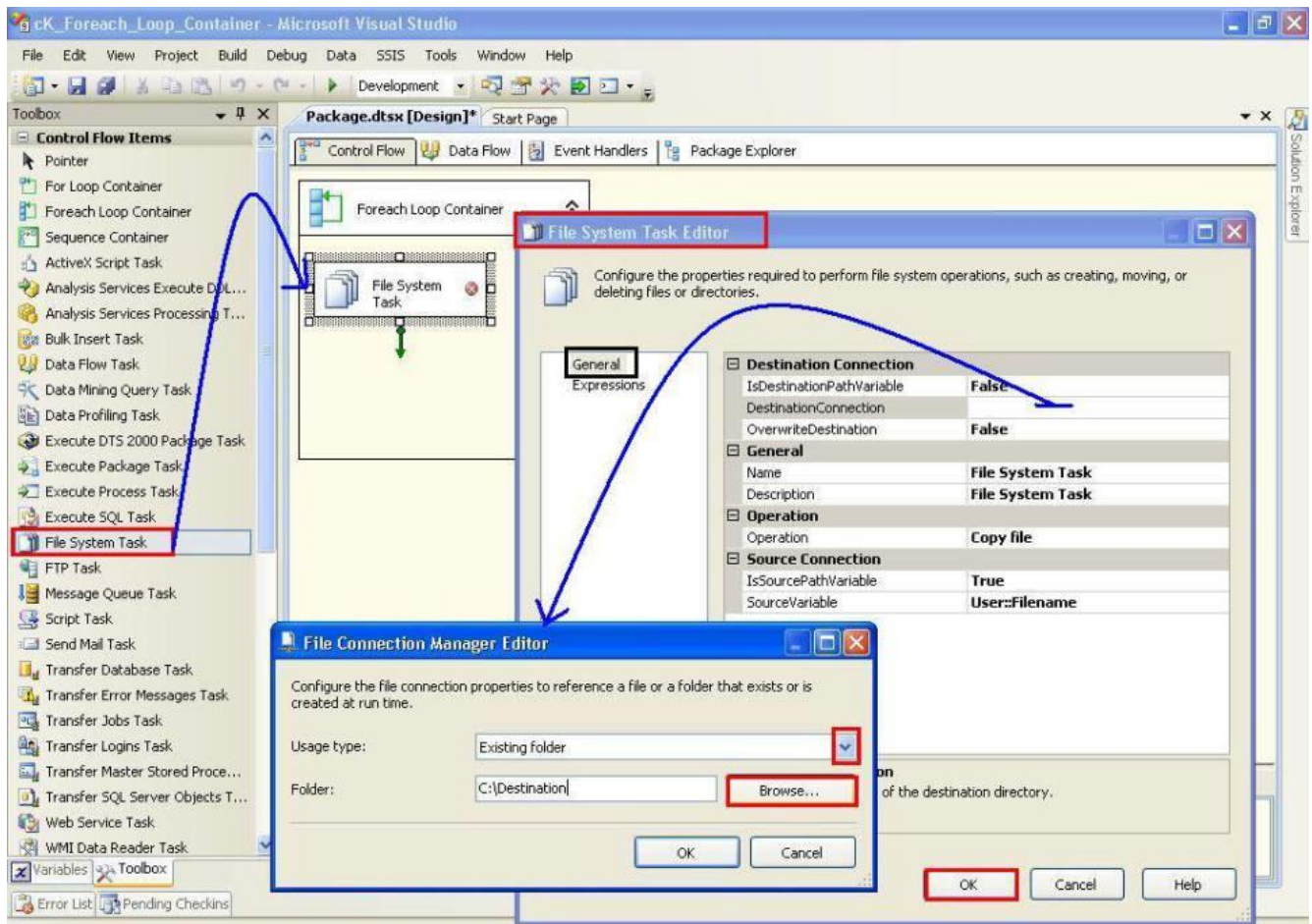


Step 5. Go to **Variable mapping tab**, select the variable (in this example it's **Filename**) you created earlier from the Variable dropdown box, and then accept the *default* of 0 for the **index**. Click OK to save the settings and return to the **Control Flow** tab in the Package Designer.



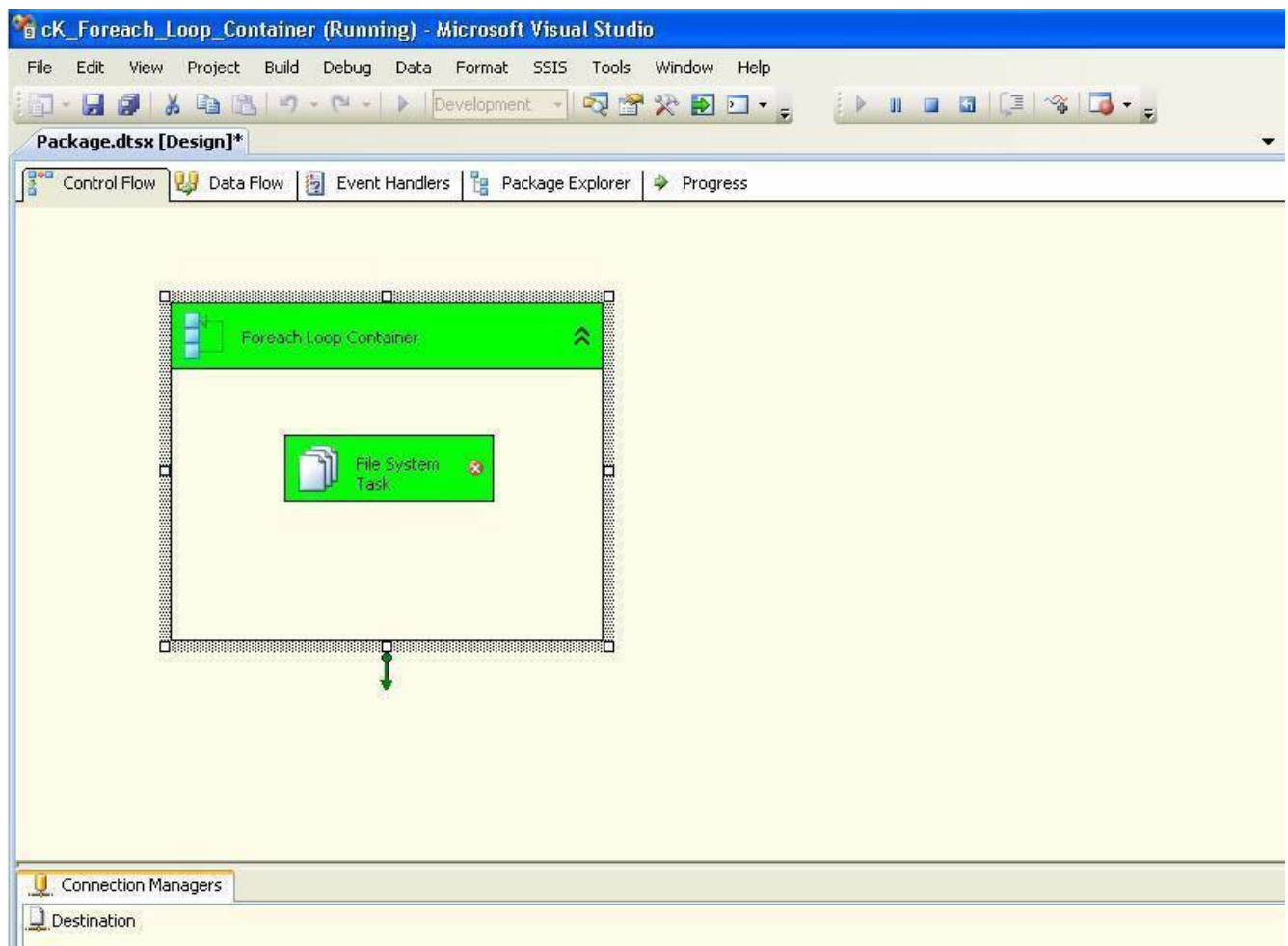
Step 7. Drag a **File System Task** into the container's box. Double-click the new task to **configure** it in the editor that pops up as:-

- Set the *operation* field to **Copy file**.
- Select **<New Connection>** for the Destination Connection property. When the Connection Manager dialog opens, select **Existing Folder** (as we have created **Destination** folder already) and Browse the folder
- Set the *IsSourcePathVariable* property field to **True**.
- Set the *Source Variable* field to **User::FileName**.



Step 8. We are now ready to strike the **execute** button to see how the **files** are transferred from **one directory to another directory**. During the execution of our **package**, we will see each file being **picked up** from the **Source folder** and getting **copied** to the **Destination folder**.

NOTE :- _If you had set the **Overwrite Destination property** to True in the File System Task, the file would be overwritten if there was a conflict of duplicate filenames.



With these we finish our post on Implementing **Foreach Loop Container** in SSIS

Sequence Container in SSIS

Posted in [SSIS](#) By [Chander Sharma](#) On March 13, 2013

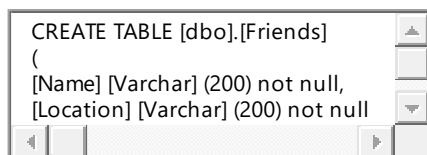
Description – **Sequence container** group related tasks in a package to show what the complex package is doing in a clear and simple way. The task of Sequence container is to have multiple separate control flows group together in a SSIS package. Each container will contain one or more tasks and will run within the control flow of overall package. Its not at all compulsory that every **package** should use Sequence container but there are some benefits of using this **container**. Some of them are described below:-

Benefits:-

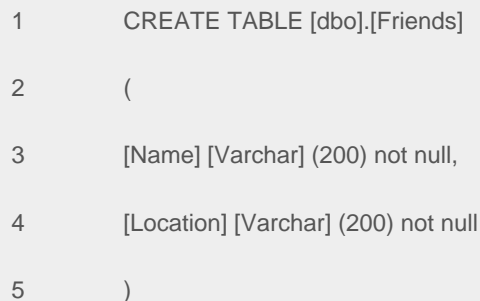
1. They can be huge helpful when **developing** and **debugging** SSIS packages.

2. If a **package** has many tasks then it is easier to **group the tasks** in Sequence Containers and you can **collapse** and **expand** this container for usability.
3. Instead of setting **property** for each individual task, we **group tasks** together that require similar property settings.
4. Providing **scope for variables** that a group of related tasks and containers use.
5. If **one task fails** to succeed inside the container then the process is aborted and all the tasks that were completed successfully get **rolled back** for that container. This depends on **Transaction option property** which can be *Required, Supported and Not Supported* according to your configuration (By default it's **Supported**). It means it creates a Transaction around the components inside Container.

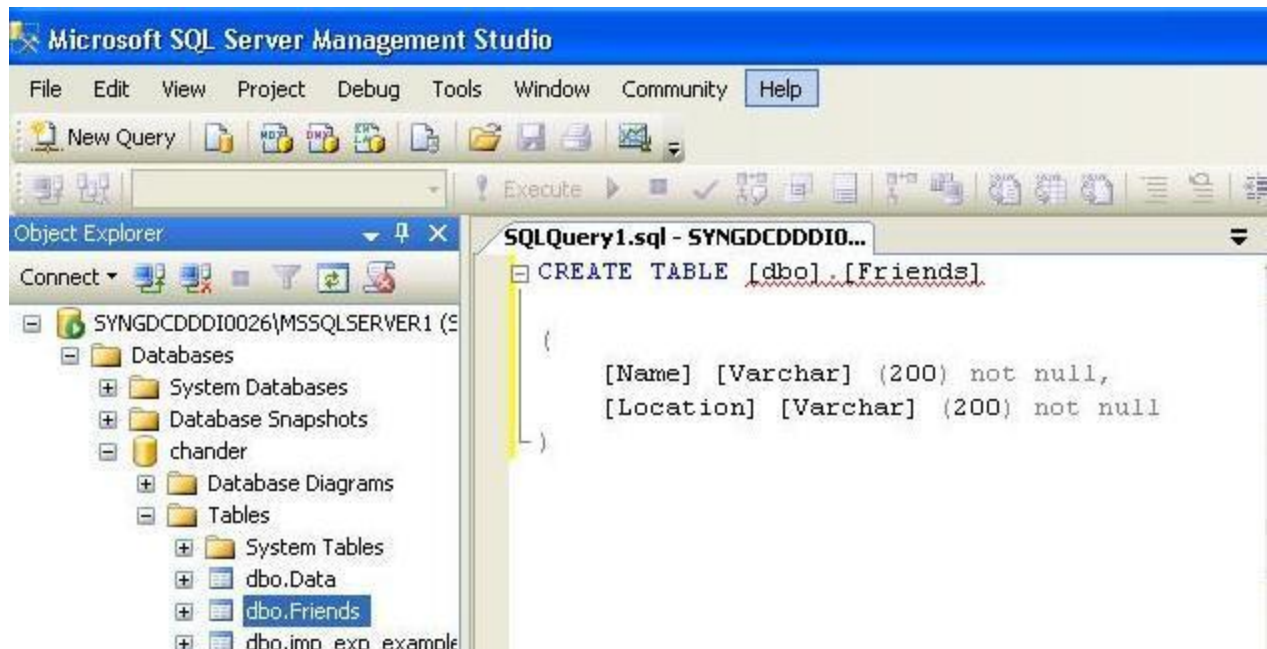
So, now let's learn how to Implement Sequence Container. **Example Scenario:-** We will create **one table** named Friends in SSMS and **one package** in BIDS. Then we will **insert** some rows into our table using **Multiple Execute SQL Tasks** contained in a Sequence container. **STEPS TO FOLLOW**
:- **STEP 1.** Let's **create a table**, say Friends with the fields name and location. Open SQL Server Management Studio (**SSMS**) and Click on **New Query**. Select your present working database (I am using **Chander** as my database) and write the following script in it :-



```
CREATE TABLE [dbo].[Friends]
(
  [Name] [Varchar] (200) not null,
  [Location] [Varchar] (200) not null
)
```



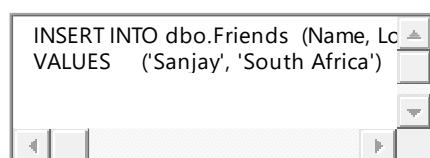
```
1      CREATE TABLE [dbo].[Friends]
2      (
3      [Name] [Varchar] (200) not null,
4      [Location] [Varchar] (200) not null
5      )
```

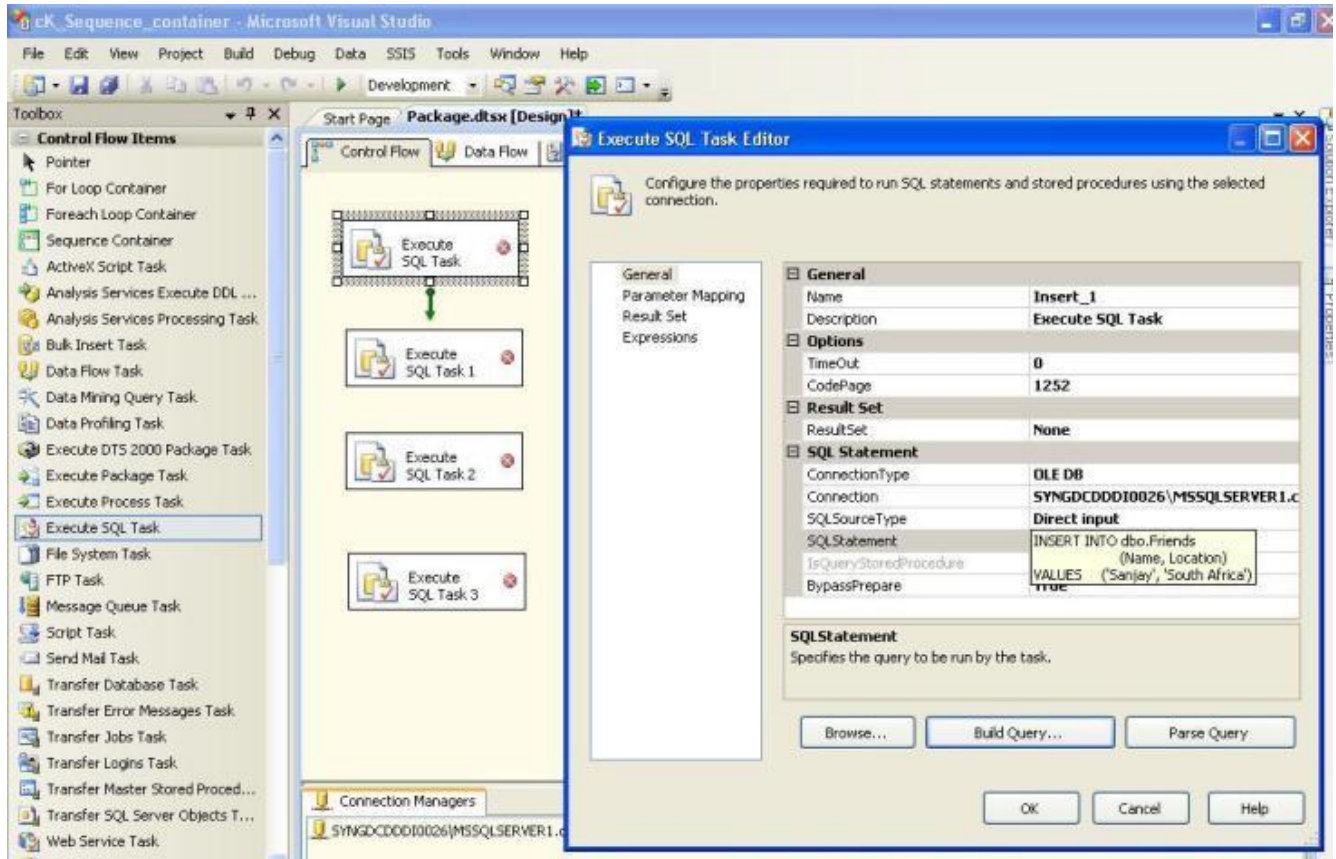
STEP 2. Drag 4 different **Execute SQL Tasks** and Rename them to Insert_1, Insert_2, Insert_3 and Insert_4 respectively by double clicking the **Execute SQL Task**. Configure the SQL Task Editor as:-

- Set **Connection** field to your working database (I am using **Chander** as my database).
- Click on **SQL statement** field and write the following queries inside the respective Execute SQL Tasks. I am showing for Insert_1 Execute SQL Task. You can perform the same for other remaining Execute SQL Tasks.

For Insert_1 Execute SQL Task :-



1	INSERT INTO dbo.Friends (Name, Location)
2	VALUES ('Sanjay', 'South Africa')



For Insert 2 Execute SQL Task :-

```
INSERT INTO dbo.Friends (Name, Location)
VALUES ('Prince', 'England')
```

```
1      INSERT INTO dbo.Friends (Name, Location)
2      VALUES ('Prince', 'England')
```

For Insert 3 Execute SQL Task :-

```
INSERT INTO dbo.Friends (Name, Location)
VALUES ('Amit', 'Australia')
```

```
1      INSERT INTO dbo.Friends (Name, Location)
```

```
2      VALUES      ('Amit', 'Australia')
```

For Insert 4 Execute SQL Task :-

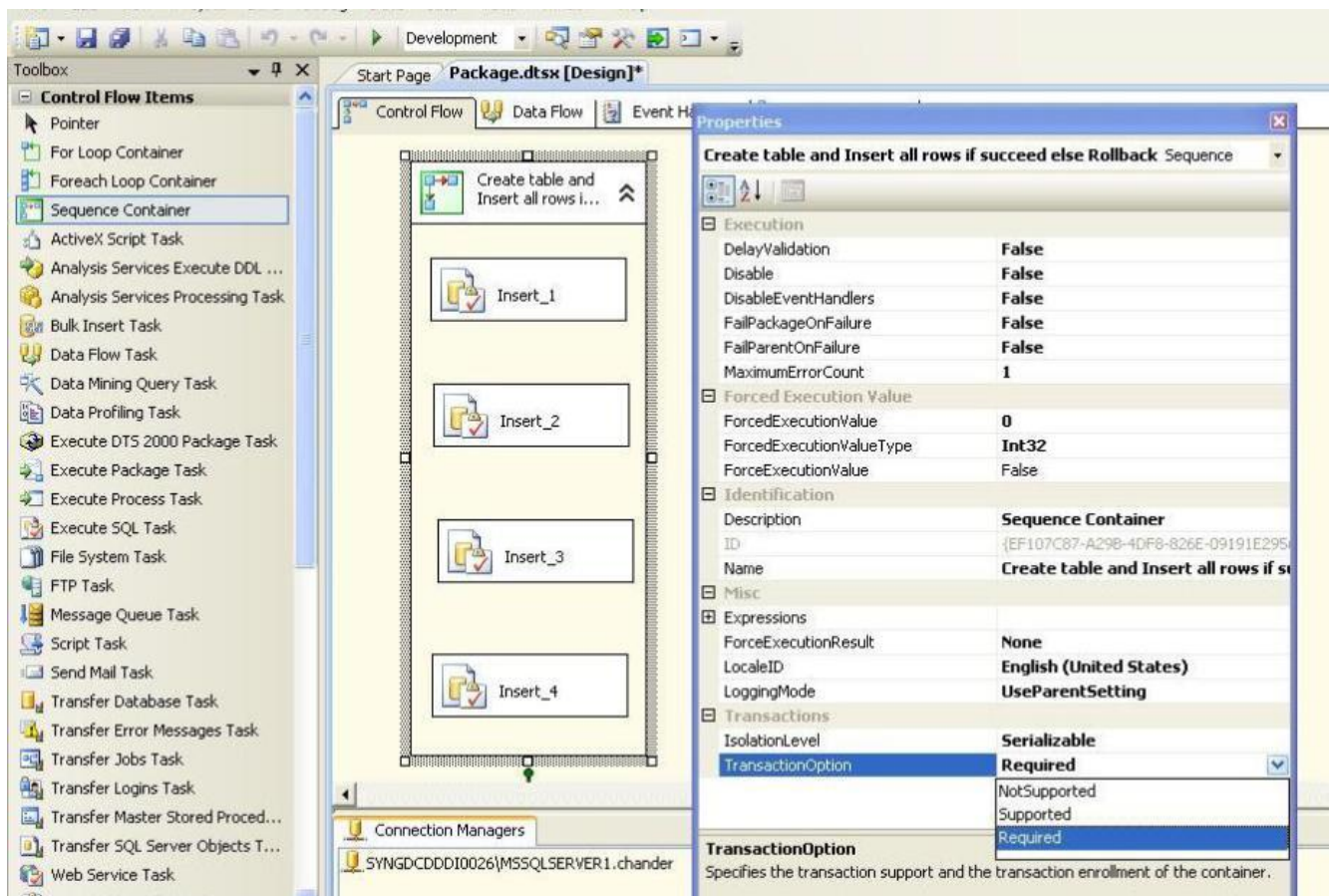
```
INSERT INTO dbo.Friends (Name, Location)
VALUES      ('Aakash')
```

```
1      INSERT INTO dbo.Friends (Name, Location)
```

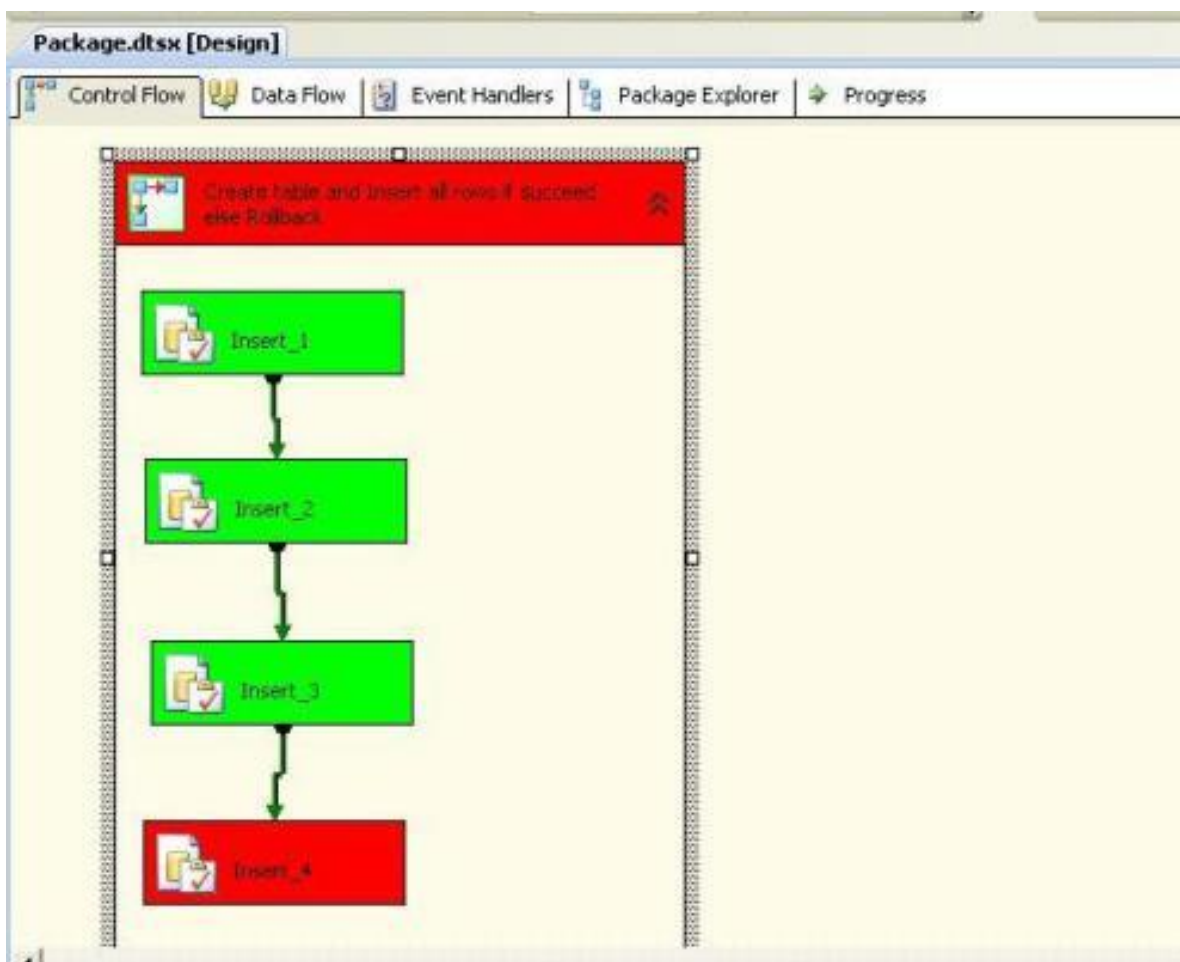
```
2      VALUES      ('Aakash')
```

NOTE :- We are **inserting null** in the location field for **Insert_4 Execute SQL Task**. The purpose is that this task will **fail** as we have mentioned not null for the **Location** field in our Friends Table (STEP 1). This will **halt** the execution of our package which we will see when we will **execute** it.

STEP 3. Now, click on **Sequence container** and Drag it onto Control flow pane. Rename this container to “*Create table and Insert all rows if succeed else Rollback*”.



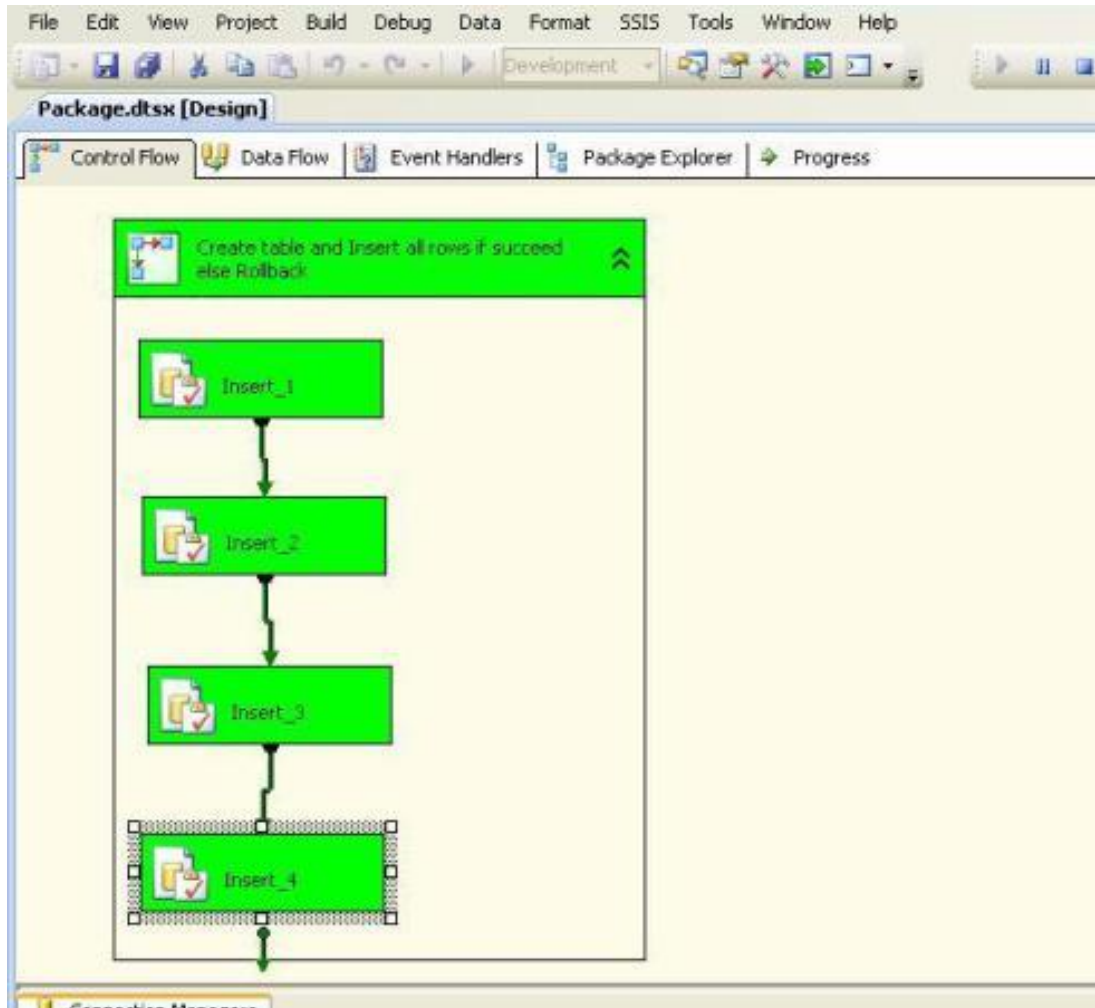
STEP 4. Now, put all the 4 **Execute SQL Tasks** into the Sequence container. By Default Transaction option property for the Container is Supported. Set this property to Required and execute the package. **NOTE:-** Before I could even get the package to fail for the reason I wanted to I got this error. **Error:-** *The SSIS Runtime has failed to start the distributed transaction due to error 0x8004D01B “The Transaction Manager is not available.”. The DTC transaction failed to start. This could occur because the MSDTC Service is not running.* **Sol:-** So to solve this error and to enable the service on my Machine, Try these steps :- Control Panel ->Administrative Tools ->Component Services ->Computers ->My Computer ->Properties and start the Distributed Transaction Coordinator and if it is already running then stop it and start again.



STEP 5. As expected, 4th **Insert fails** and the container fails too. So, it will not create any table in our database. If we go to our Insert_4 Execute SQL Task and **specify location** for that field then it will **execute** our container. So, let's make it **work**.

```
INSERT INTO dbo.Friends (Name, Location)
VALUES ('Aakash', 'America')
```

```
1      INSERT INTO dbo.Friends (Name, Location)
2      VALUES ('Aakash', 'America')
```



STEP 6. It turns to **Green**, means it really does work. As long as the task is set to supported, and the items inside the child package are set to supported as well they will inherit the transaction created by the parent container. Let's check our **database** to see the **table created** by the package.

