|  |
| --- |
| **National University of Computer and Emerging Sciences** |
| Lab Manual 5  “**ETL**” |
|  |
| Data Warehousing and Data Mining |
|  |
|  |
|  |

|  |  |
| --- | --- |
| Course Instructor | Sir Ishaq Raza |
| Lab Instructor(s) | Farhan Azhar  Nouman Ali |
| Section | CS |
| Semester | Fall 2019 |

|  |
| --- |
|  |

Department of Computer Science

FAST-NU, Lahore, Pakistan

ETL Process

**ETL (Extract, Transform and Load)** is a process in data warehousing responsible for pulling data out of the source systems and placing it into a data warehouse. ETL involves the following tasks:  
  
- **extracting the data** from source systems (SAP, ERP, other operational systems), data from different source systems is converted into one consolidated data warehouse format which is ready for transformation processing.   
  
- **transforming the data**may involve the following tasks:

   applying business rules (so-called derivations, e.g., calculating new measures and dimensions),

   cleaning (e.g., mapping NULL to 0 or "Male" to "M" and "Female" to "F" etc.),

   filtering (e.g., selecting only certain columns to load),

   splitting a column into multiple columns and vice versa,

   joining together data from multiple sources (e.g., lookup, merge),

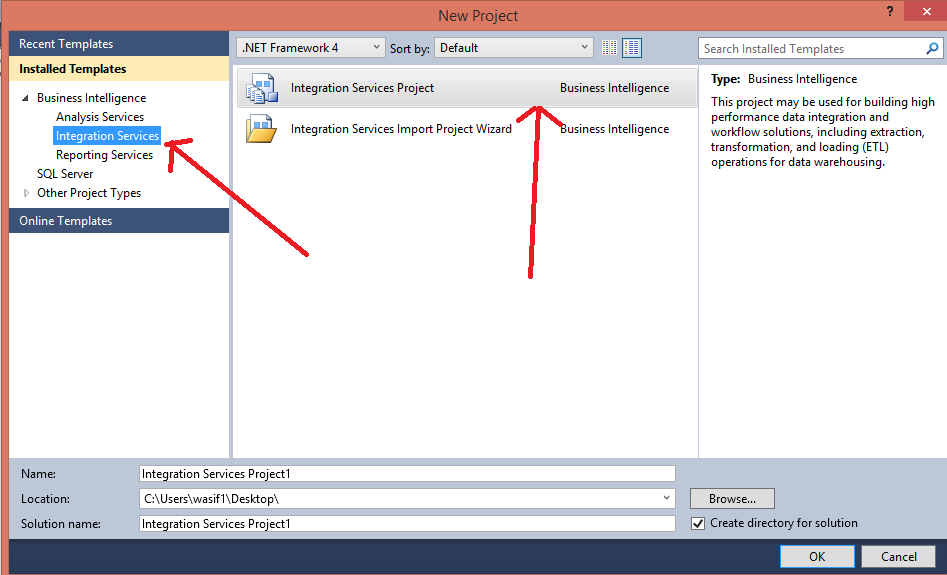
   transposing rows and columns,

   applying any kind of simple or complex data validation (e.g., if the first 3 columns in a row are empty then reject the row from processing)  
- **loading the data** into a data warehouse or data repository other reporting applications.

# **Implementation**

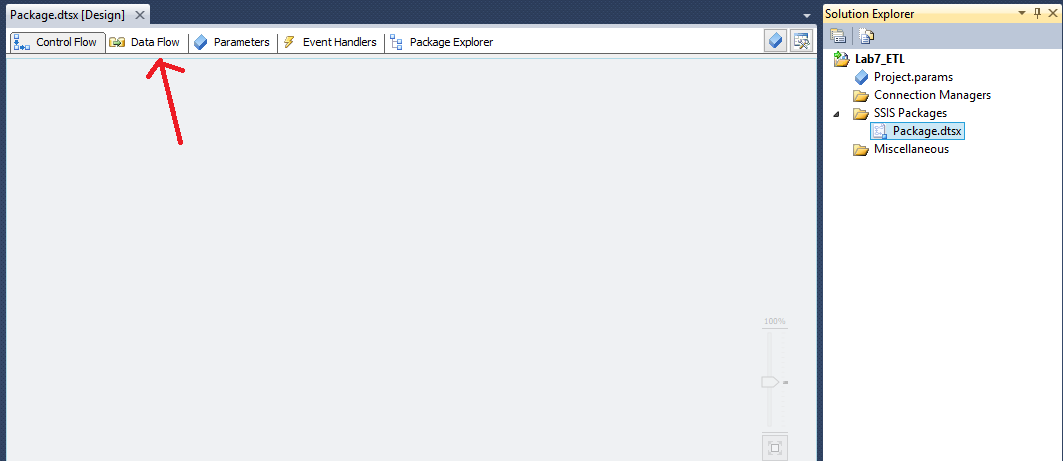
Step 1:

1. Click Start, point to **All Programs**, expand **Microsoft SQL Server 2012**, and click **SQL Server Data Tools**.
2. On the File menu, point to New, and click Project.
3. Expand **Business Intelligence** in the **Installed Templates** pane, and select **Integration Services**. Name the project and click OK.



Step 2:

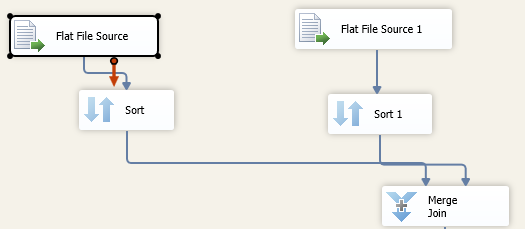
1. Click on the **Data Flow** tab.



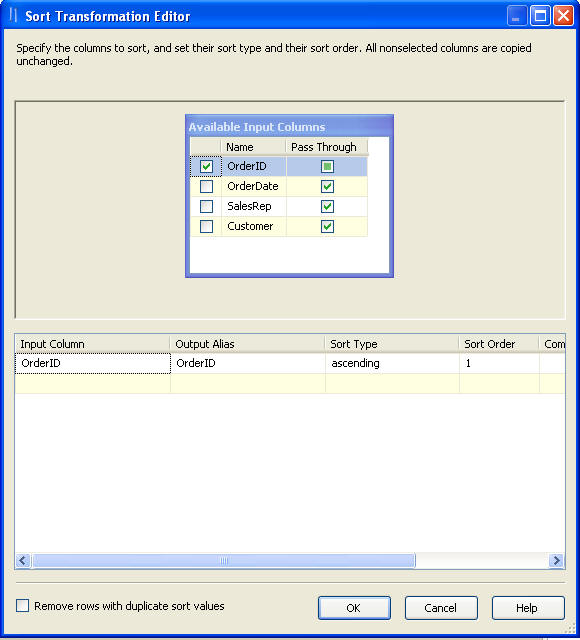
# **Merge Join**

Make two flat file sources one using **OrderHeader** text file and other with **OrderDetails** text file.

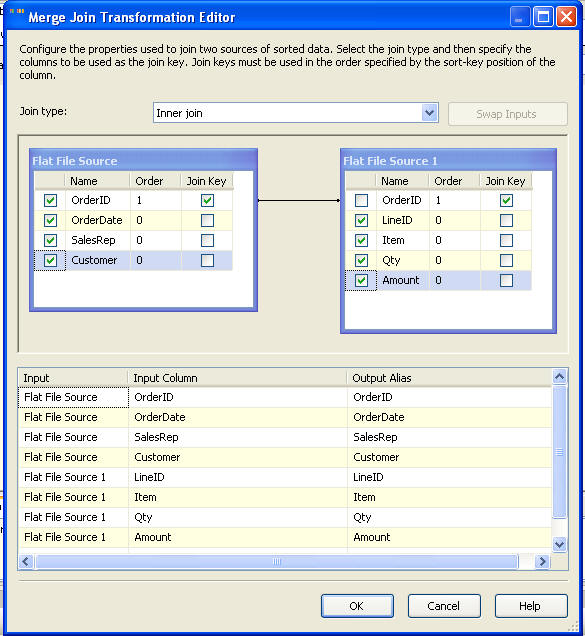
With this load process, let's assume the source data is not sorted first, so we need to use the **SORT** task to sort the data prior to using the **MERGE JOIN** task.  The following shows our Flat File sources and then a SORT task after each one of these and then lastly our MERGE JOIN task.



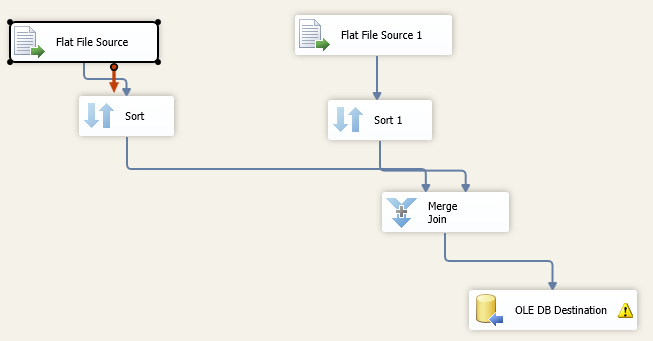
If you right click the Sort task and select Edit you will get a screen such as following.  Here you need to select which column the data should be sorted on.  This needs to be done for both of the flat source files.



Double click the **Merge Join**, from here you can select the columns that you want to have for output as well as determine what type of join you want to employ between these two files.



After this is done you can move on and specify the **destination**.

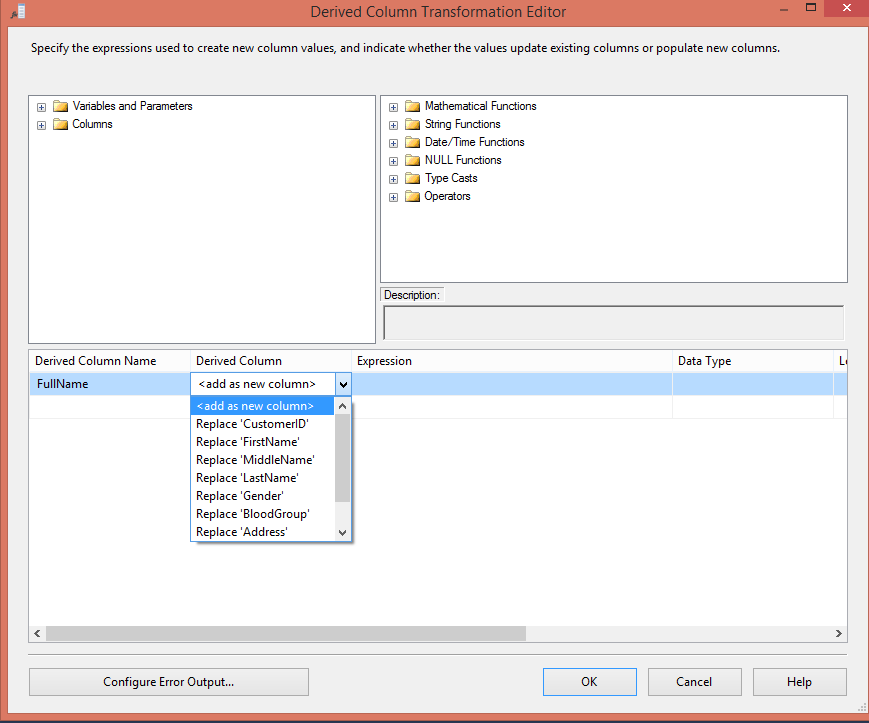


# **Derived Columns**

Drag drop **OLB DB Source**, **OLB DB Destination** and **Derived Columns** component from the tool box.

Set the Source properties and select table **Customer** as source table.

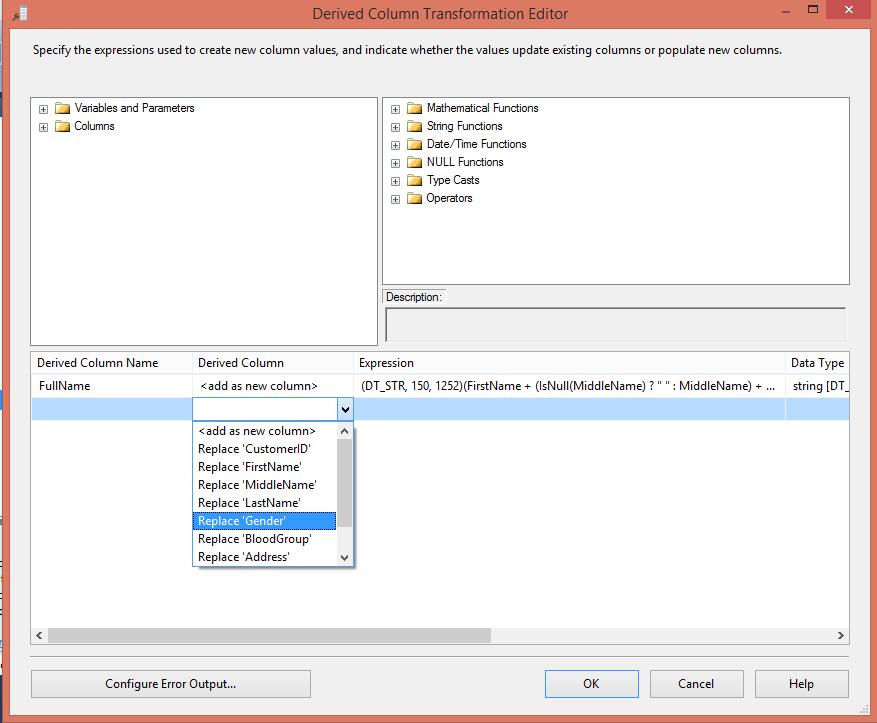
Double click the Derived Column component and select **add as new column** option from the **derived column** tab. Name this new row as **Full Name.**



Click on the empty space under the **Expression** tab and paste the below expression:

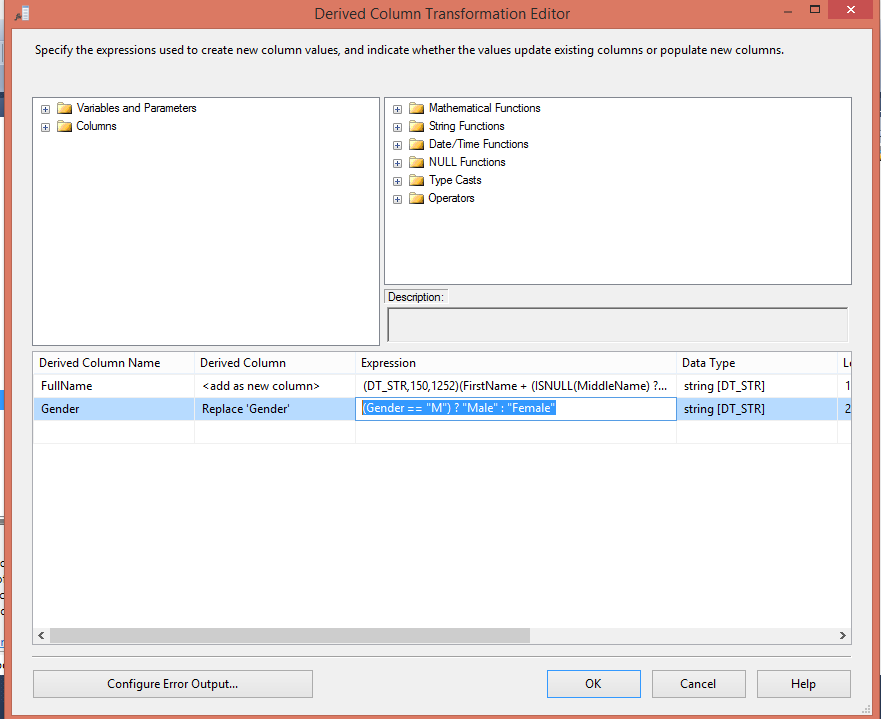
(DT\_STR, 150, 1252)(FirstName + (IsNull(MiddleName) ? " " : MiddleName) + LastName)

Now click one the second empty row below **Derived Column** tab and select **Replace Gender** option.



Paste the following line in the Expression option:

(Gender == "M") ? "Male" : "Female"



Click **Ok** to finish the setting of Derived Column component.

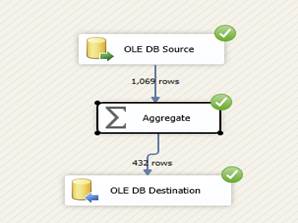
Set the properties of **OLB DB Destination** component and select the **DimCustomer** table as destination table and run the project.

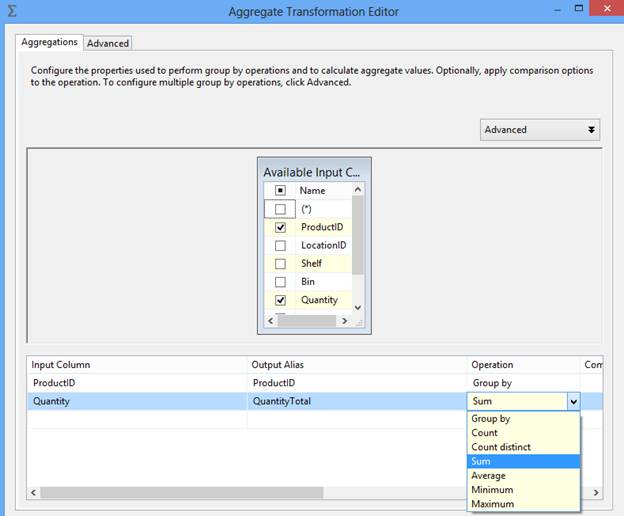
# **Some Other Common Transformations in SSIS**

## 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

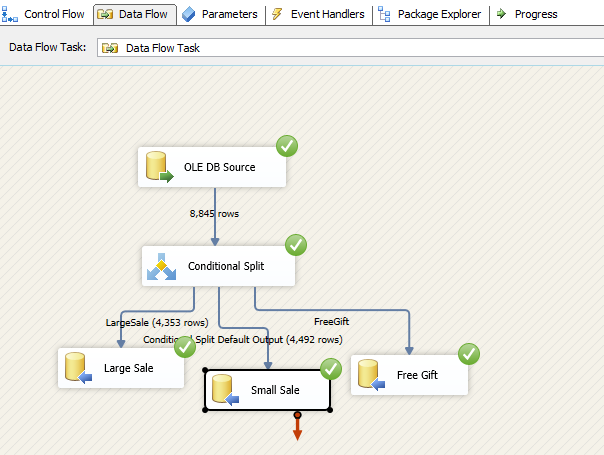
  
Aggregate Transformation Editor

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.

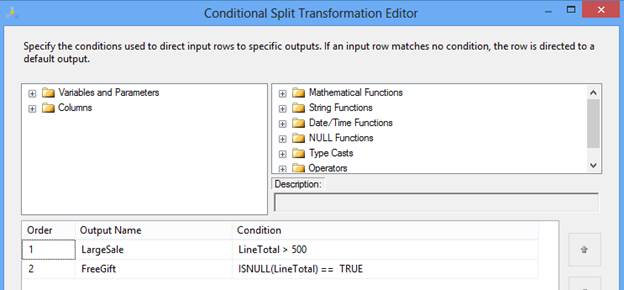
## Conditional Split

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

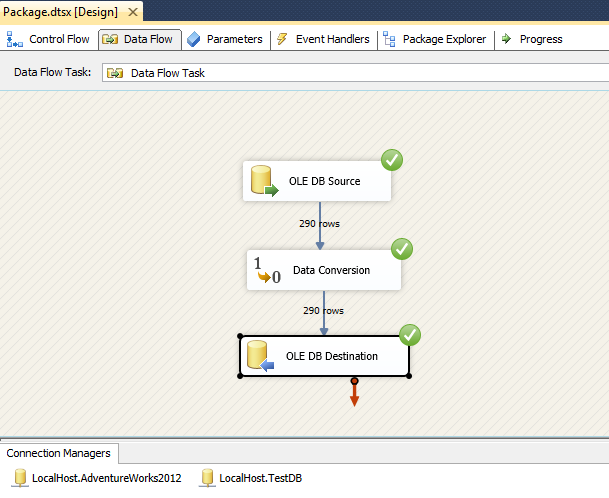
In the above example, we are splitting input records based on total order cost. If cost is more than 500, the record will be considered as part of a large sale. If LineTotal is NULL, we are assuming it's a free gift and no cost is associated with it. The rest we can consider part of small sale, in current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

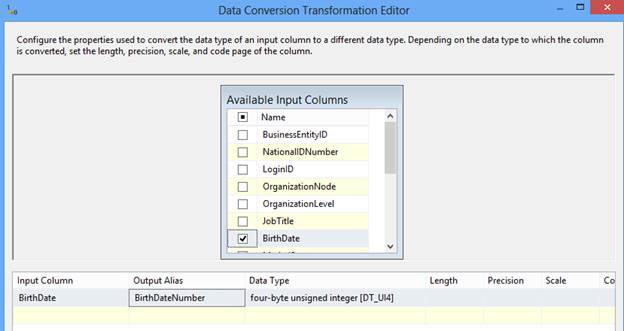
## Data Conversion

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:

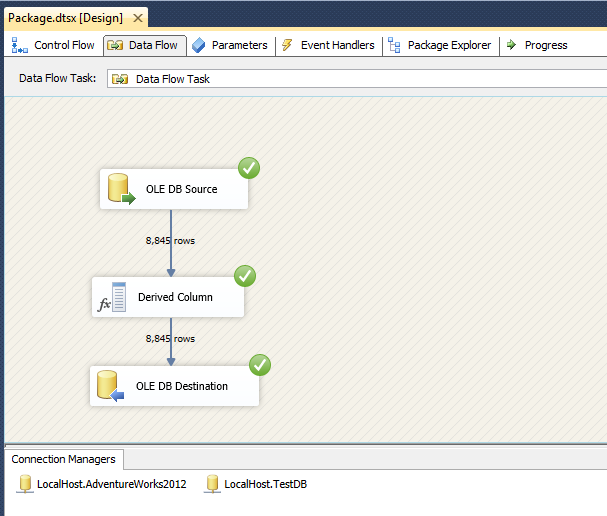
  
Data Conversion Transformation Editor

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

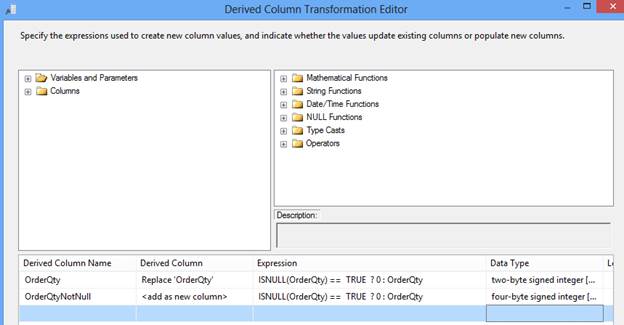
## Derived Column

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:

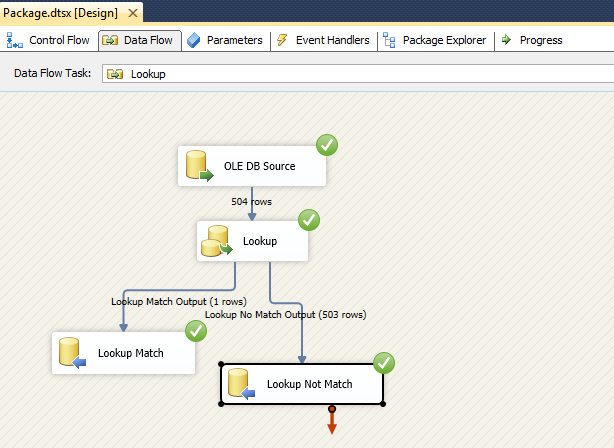
  
Derived Column Transformation Editor

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

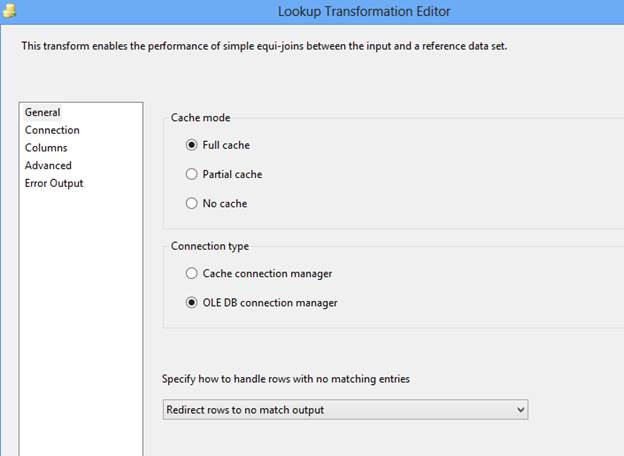
Synchronous transformation, allows you to perform an equi-join between values in the transformation input and values in the reference dataset similar to T-SQL. This transformation is used to join two datasets at a time.  To join more than two datasets we need to put multiple Lookup transformations, similar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

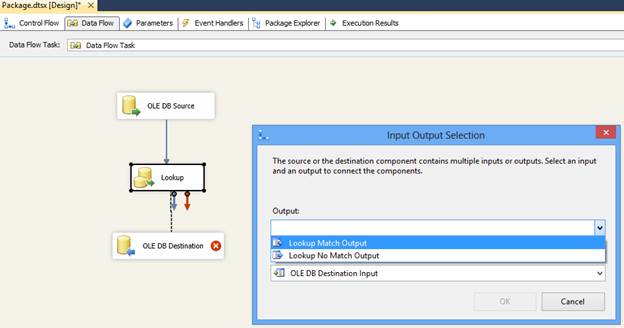
Data Flow task design for Lookup:

  
Lookup Data Flow Task Design

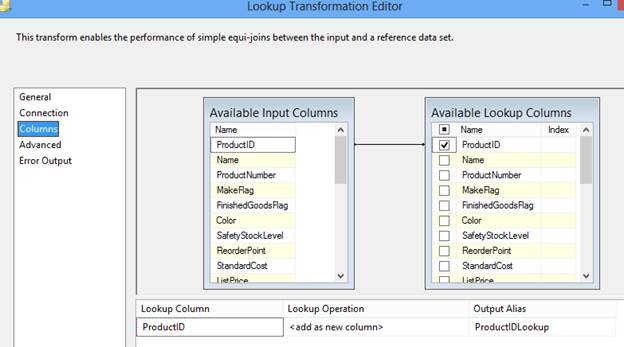
Lookup transformation settings:

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.

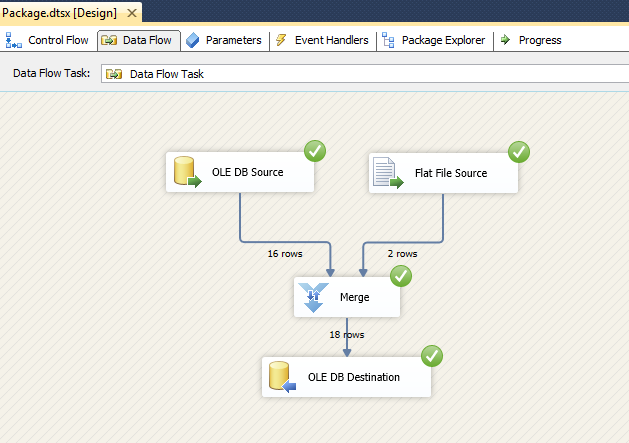
Lookup transformations provides several modes of operations, Full cache, Partial cache or No cache, that allows a trade-off between performance and resource usage.

You can refer to [MSDN](http://technet.microsoft.com/en-us/library/ms141821.aspx) to learn more interesting facts about Lookup transformation.

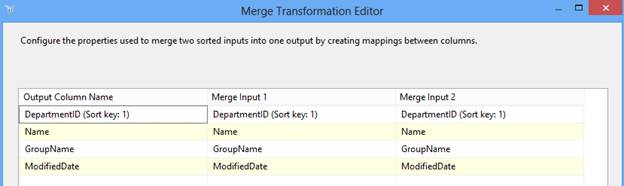
## Merge

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:

  
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.

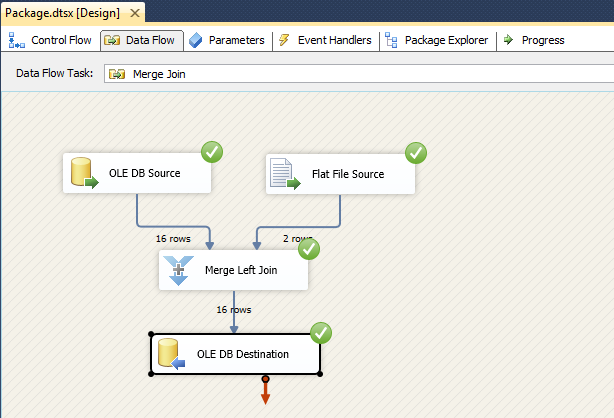
This transformation has two inputs and one output. It does not support an error output.

## Merge Join

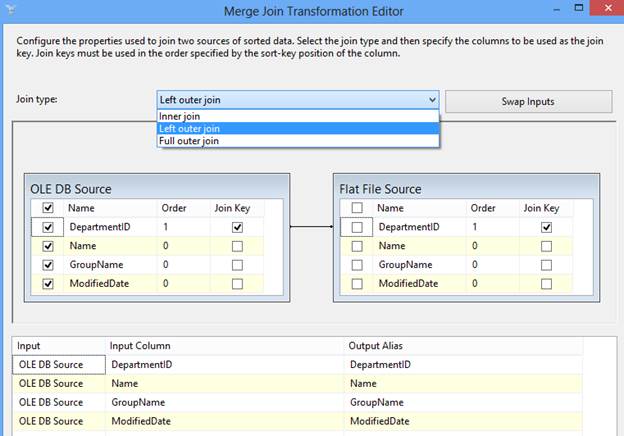
An Asynchronous partial blocking transformation, allows joining data from two sorted datasets using a FULL, LEFT, or INNER join.

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:

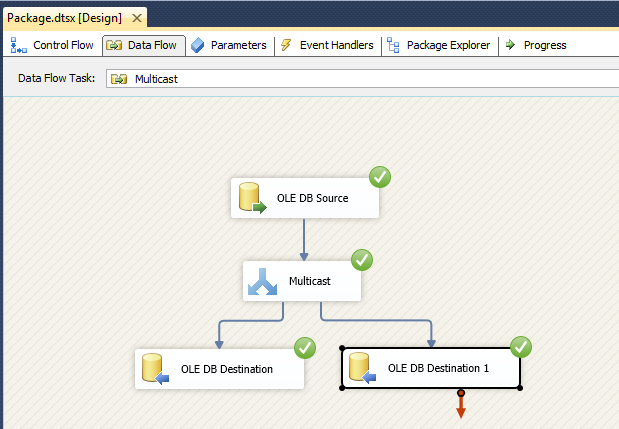
  
Merge Join Transformation Editor

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

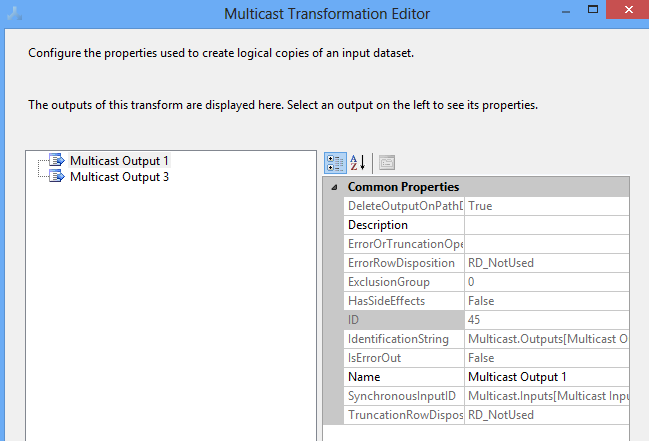
## Multicast

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:

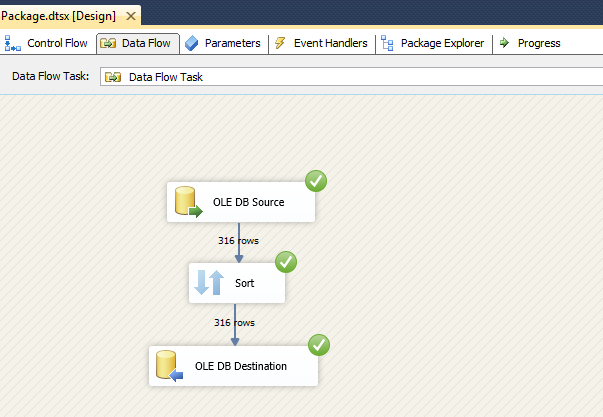
  
Multicast Transformation Editor

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

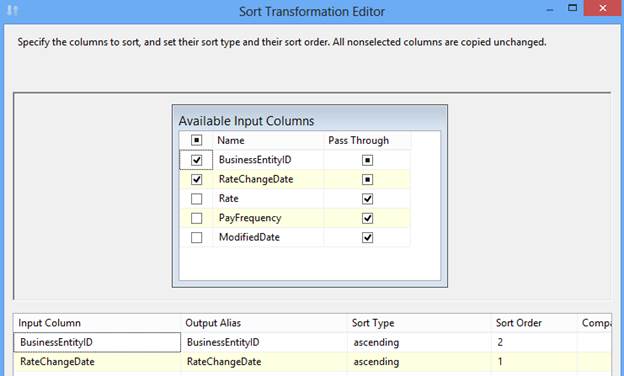
## Sort

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

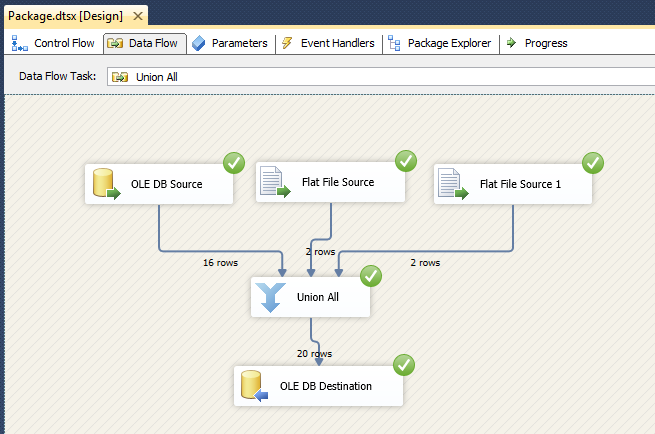
In above example, we arranged input data in ascending order of RateChangeDate first and BusinessEntityID column second.

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