DWBI- Practical 3

1. Introduction to the Worksheet

In the previous worksheet, we extracted data from the data sources to the staging database using SSIS. This worksheet focuses on how to perform data transformations on the staging database and load them into the data warehouse.

2. Create New Package for Staging-to-Data Warehouse ETLs

1. Open Visual Studio Data Tools in 'Administrator' mode and open previously created solution.
2. Right click on ***SSIS Packages*** and select ***New SSIS Package*** and rename it as '**SLIIT_Retail_Load_DW.dtsx**'.

NOTE: It is important to understand the order of execution of tasks. In other words, the order of the tables that we need to load into data warehouse. '**DimProduct**' table contains a reference through a foreign key to '**DimProductSubCategory**' table. Similarly, '**DimProductSubCategory**' table contains a foreign key to '**DimProductCategory**' table.

Thus, first we need to load data to '**DimProductCategory**' table, next the '**ProductSubCategory**' table and finally the '**DimProduct**' table.

3. Product Category Data Transformation and Loading

First, let's handle product category data. Follow the step given below:

1. Drag and drop a ***Data Flow Task***, rename it as '**Transform and Load Product Category Data**' and go to the ***Data Flow*** tab to design the '**Transform and Load Product Category Data**' data flow.
2. Drag and drop ***OLE DB Source*****,** rename as '**Extract from Product Category Staging**' and configure it to extract the '**StgProductCategory**' table. Make sure all the columns are selected.

3. As we are not maintaining history of product category data, we can simply have the updated (latest) record in the data warehouse. This can be achieved in several ways:
    a. Do a full load: truncate '**DimProductCategory**' table, extract all the records from source system table and load to '**DimProductCategory**' table. However, this would not be a good approach as,
        i. There is a possibility that this approach might change Surrogate Keys for existing Product Category records (***delete*** is the worst!), affecting the fact table records referring to those existing product categories in '**DimProductCategory**' table.
        ii. We will have to extract the whole data set of that table in every execution. However, full load will be useful if you want to track deleted records in the source system.
    b. Check for the existence of data and update or insert: if a tuple is already existing in '**DimProductCategory**' table, then we can update the existing tuple. Else, if it is a new record, that it can be inserted. This logic can be developed in many ways Two possible options are,
        i. ETL layer (SSIS) and,
        ii. In the target database layer (as a Stored Procedure in MS SQL Server)

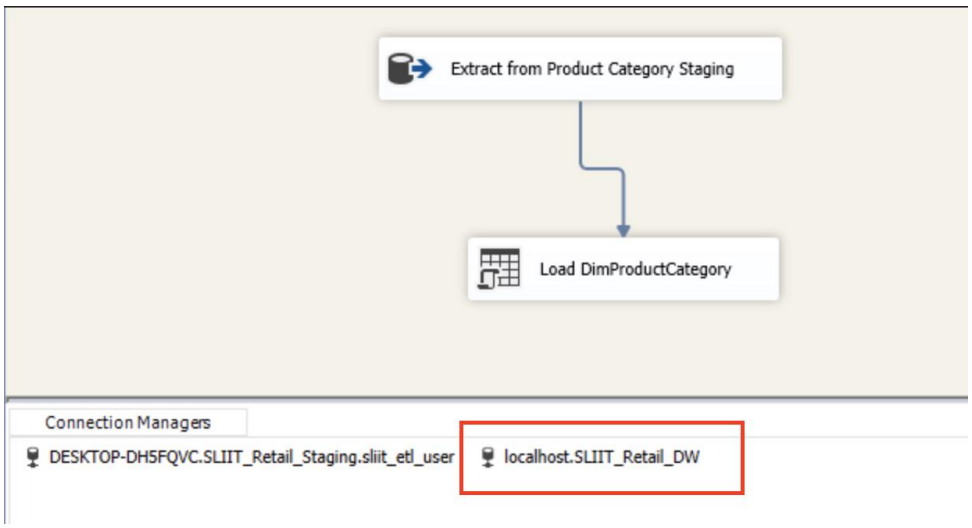
    Let's use the Stored Procedure approach for this requirement. To do this, first we need to create a Stored Procedure in the target database, which is the data warehouse. This will be executed by the ETL process and pass along all the records coming from source side to the data warehouse. Stored Procedure will then decide which records need to be inserted, updated or even ignored.

    Open SQL Server Management Studio, if not already open.

4. Open a new query window and write the code given below to create the Stored Procedure.

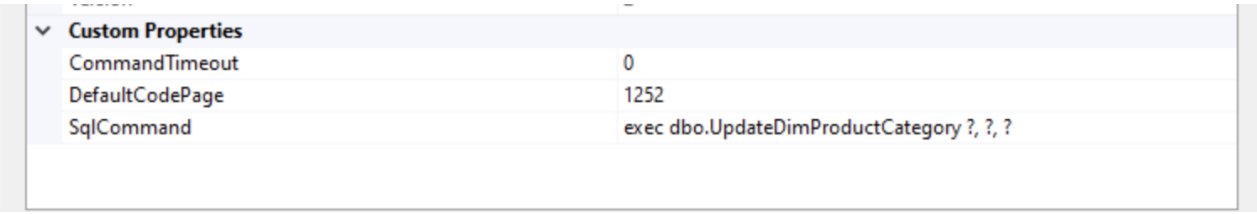
```
CREATE PROCEDURE dbo.UpdateDimProductCategory
@ProductCategoryID int,
@ProductCategoryName nvarchar(50),
@ModifiedDate datetime
AS
BEGIN
if not exists (select ProductCategorySK
from dbo.DimProductCategory
where AlternateProductCategoryID = @ProductCategoryID)
BEGIN
insert into dbo.DimProductCategory
(AlternateProductCategoryID, ProductCategoryName, SrcModifiedDate, InsertDate, ModifiedDate)
values
(@ProductCategoryID, @ProductCategoryName, @ModifiedDate, GETDATE(), GETDATE())
END;
if exists (select ProductCategorySK
from dbo.DimProductCategory
where AlternateProductCategoryID = @ProductCategoryID)
BEGIN
update dbo.DimProductCategory
set ProductCategoryName = @ProductCategoryName,
SrcModifiedDate = @ModifiedDate,
ModifiedDate = GETDATE()
where AlternateProductCategoryID = @ProductCategoryID
END;
END;
```

5. Make sure to select the correct database ('**SLIIT_Retail_DW**') before ***CREATE PROCEDURE*** statement is executed.
6. Now, back in SSIS solution in Data Tools IDE, drag and drop an ***OLE DB Command*** task, rename it as '**Load DimProductCategory**', and link it with '**Extract from Product Category Staging**' using the blue line.
7. Before you configure '**Load DimProductCategory**' task, ensure there is a connection to '**SLIIT_Retail_DW**' database in the ***Connection Managers*** area (usually this is below the design area).
    a. If not, right click on the ***Connection Managers*** area and select ***New OLEDB Connection…*** and create a connection to '**SLIIT_Retail_DW**' database.



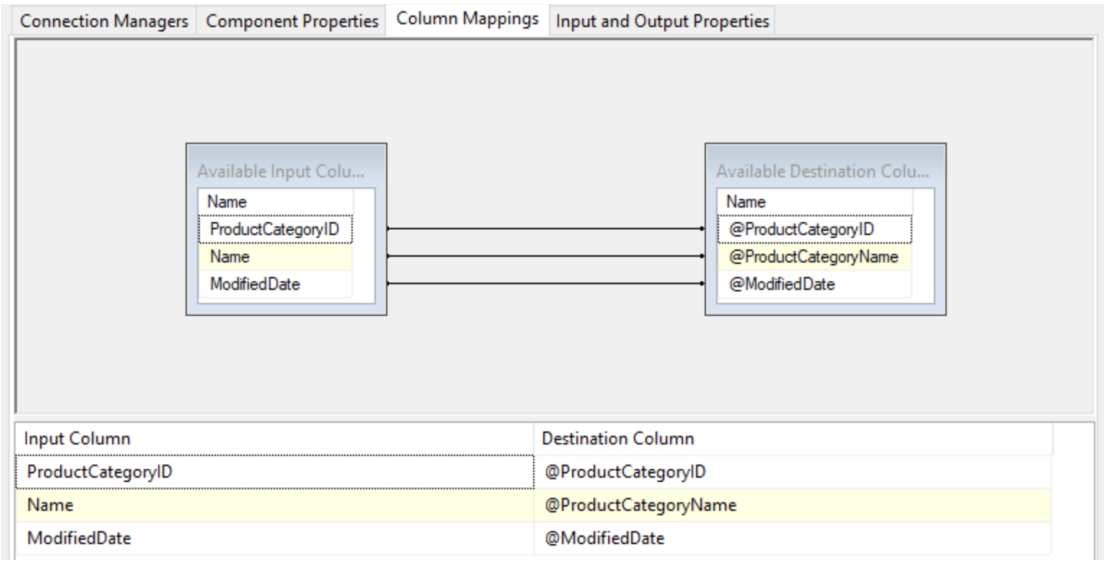
8. Double click on '**Load DimProductCategory**' to open '**Advanced Editor for Load DimProductCategory**' window.
9. In ***Connection Managers*** tab, select the connection manager which points to '**SLIIT_Retail_DW**' database.
10. Go to ***Component Properties*** tab, locate ***SqlCommand*** property, and use the code given below as the command which executes the Stored Procedure created earlier.

    exec dbo.UpdateDimProductCategory ?, ?, ?



Note that we are passing 3 arguments. This Stored Procedure ensures no duplicates are entered into the data warehouse table '**DimProductCategory**'. If there is an existing category record, it will be updated with the latest record coming in from staging table '**StgProductCategory**' else, if it is a new record, just insert it.

11. Go to ***Column Mappings*** tab and map the columns to the variables accordingly and click ***OK*** to complete the configuration.



Execute the '**Transform and Load Product Category Data**' control flow task and see how data is populated to the '**DimProductCategory**' table. Modify records in the staging table '**StgProductCategory**' and run the '**Transform and Load Product Category Data**' control flow task to understand behavior of the stored procedure you created earlier.

4. Product Sub Category Data Transformation and Loading

Now that the product category data is loaded to the data warehouse, next we will load the product sub category data. Note that product sub category table in the source system contains a foreign key to the category table with '**ProductCategoryID**'. However, in the data warehouse, this relationship between '**DimProductCategory**' and '**DimProductSubCategory**' tables are maintained using surrogate keys. Thus, '**DimProductSubCategory**' table should contain the surrogate key of '**DimProductCategory**' table as the foreign key instead of product category ID (value in '**AlternateProductCategoryID**' field).

To load the product sub category data into the '**DimProductSubCategory**' table, we can follow the same approach we followed in loading product category data into the '**DimProductCategory**' table in the data warehouse (using a stored procedure to insert new records and update existing records), as product sub category data also does not maintain history.

1. Using SQL Server Management Studio, create the Stored Procedure given below, in the '**SLIIT_Retail_DW**' database.

```
CREATE PROCEDURE dbo.UpdateDimProductSubCategory
@ProductSubCategoryID int,
@ProductCategoryKey int,
@ProductSubCategoryName nvarchar(50),
@ModifiedDate datetime
AS
BEGIN
if not exists (select ProductSubCategorySK
from dbo.DimProductSubCategory
where AlternateProductSubCategoryID = @ProductSubCategoryID)
BEGIN
insert into dbo.DimProductSubCategory
(AlternateProductSubCategoryID, ProductCategoryKey, ProductSubCategoryName, SrcModifiedDate,
InsertDate, ModifiedDate)
values
(@ProductSubCategoryID, @ProductCategoryKey, @ProductSubCategoryName, @ModifiedDate,
GETDATE(), GETDATE())
END;
if exists (select ProductSubCategorySK
from dbo.DimProductSubCategory
where AlternateProductSubCategoryID = @ProductSubCategoryID)
BEGIN
update dbo.DimProductSubCategory
set ProductCategoryKey = @ProductCategoryKey,
ProductSubCategoryName = @ProductSubCategoryName,
SrcModifiedDate = @ModifiedDate,
ModifiedDate = GETDATE()
where AlternateProductSubCategoryID = @ProductSubCategoryID
END;
END;
```

2. Now, back in SSIS solution, Drag and drop a ***Data Flow Task***, rename it as '**Transform and Load Product Sub Category Data**' and go to the ***Data Flow*** tab to design the '**Transform and Load Product Sub Category Data**' data flow.
3. Drag and drop ***OLE DB Source*****,** rename as '**Extract from Product Sub Category Staging**' and configure it to extract the '**StgProductSubCategory**' table. Make sure all the columns are selected.

4. Now, we need to replace the category ID of sub category records with product category surrogate key value. There are many ways we can achieve this within SSIS, without writing code:
    a. Use another source to extract '**DimProductCategory**' table and join the sub category data with '**DimProductCategory**' to retrieve the matching '**ProductCategorySK**' for respective '**AlternateProductCategoryID**'s.
    b. Use a ***Lookup*** transformation to lookup for the relevant '**ProductCategorySK**' for respective '**AlternateProductCategoryID**'s.

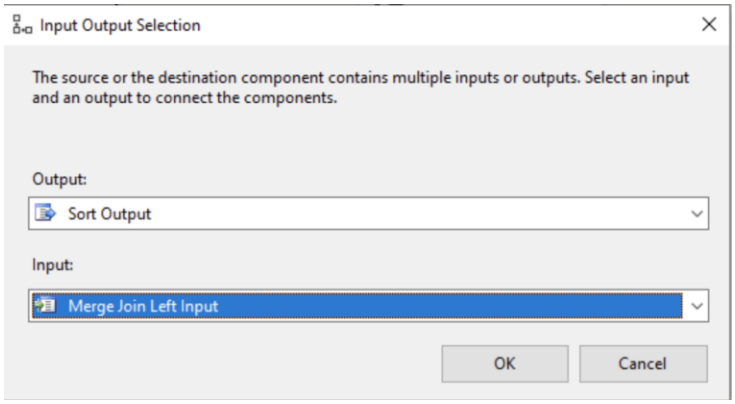
    Let's use method (a) and use two sources to join source data (coming from '**StgProductSubCategory**' table) with '**DimProductCategory**' to retrieve the matching '**ProductCategorySK**' for respective '**AlternateProductCategorID**'s.

    Use another ***OLE DB Source***, rename as '**Extract from Product Category Dimension**' and configure it to extract the '**DimProductCategory**' table in '**SLIIT_Retail_DW**' database. Make sure all the columns are selected

5. Drag and drop a ***Sort*** component from ***SSIS Toolbox***, rename it as '**Product Sub Category Sort**' and connect it with the '**Extract from Product Sub Category Staging**' component using the blue line.
6. Double click '**Product Sub Category Sort**' component and select '**ProductCategoryID**' as the sorting column by ticking on the checkbox in front of '**ProductSubCategoryID**'. Click ***OK***.
7. Drag and drop another ***Sort*** component, rename it as '**Product Category Sort**' and connect it with the '**Extract from Product Category Dimension**' component using the blue line.
8. Double click '**Product Category Sort**' component and select '**AlternateProductCategoryID**' as the sorting column. Click ***OK***.
9. Drag and drop ***Merge Join*** component and link the '**Product Sub Category Sort**' component using the blue line.

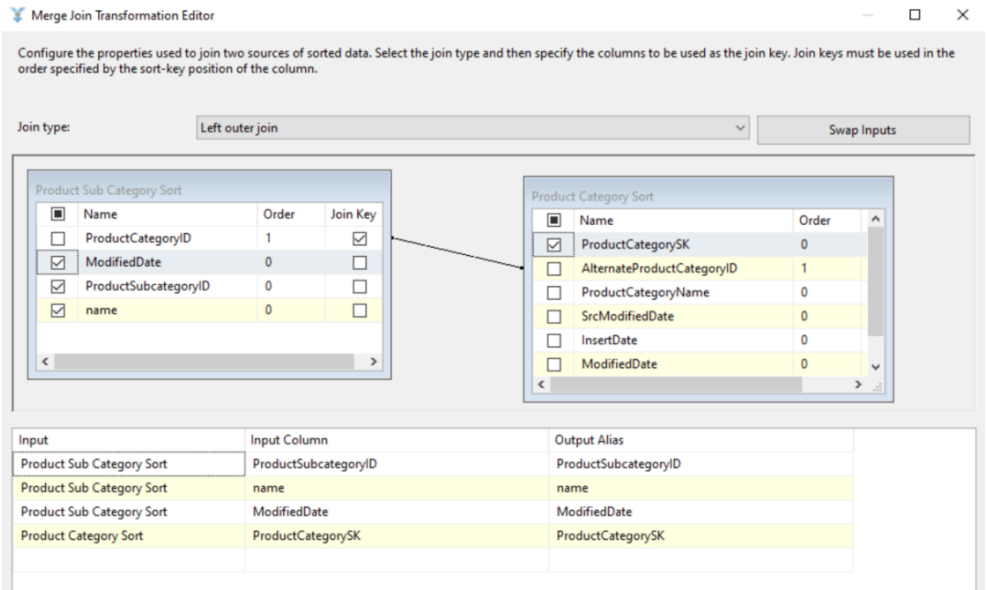
    Read more on ***Merge Join***, ***Merge*** & ***Union All*** SSIS components and how they work.

10. In the ***Input Output Selection*** window, select ***Merge Join Left Input*** as the value for ***Input*** field. Click ***OK***.



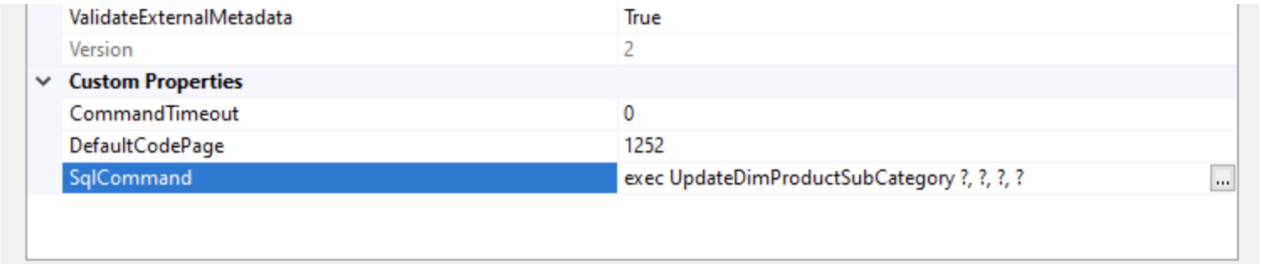
11. Link the '**Product Category Sort**' component to ***Merge Join*** using the blue link of '**Product Category Sort**' and double click ***Merge Join*** to configure the join.
12. In the ***Merge Join Transformation Editor*** window, you should be able to see two sources are connected by '**ProductCategoryID**' (of '**Product Sub Category Sort**') and '**AlternateProductCategoryID**' (of '**Product Category Sort**') and the ***Join Type*** is ***Inner Join*****.** Joining columns are automatically picked based on the sort column you provided in the ***Sort*** components. Let's assume even if there are product sub categories without corresponding product categories, we still load them in to the data warehouse table '**DimProductSubCategory**'. Thus, change the ***Join Type*** to ***Left Outer Join***.

Read more on different join types.



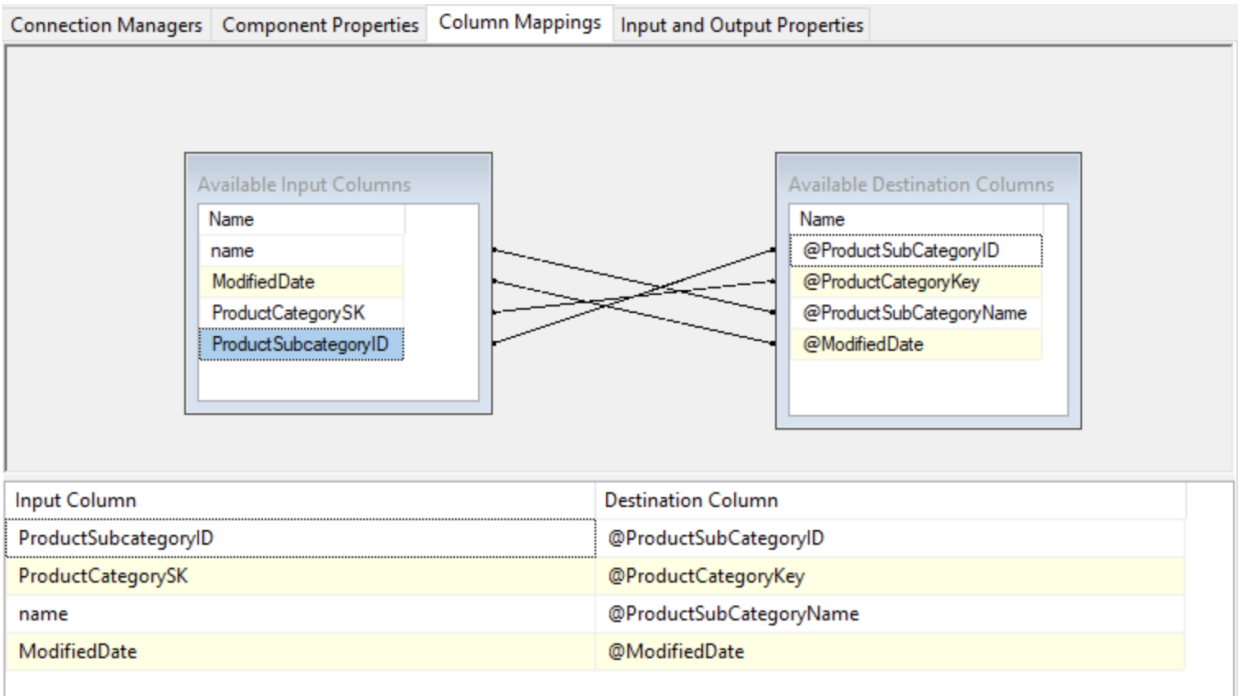
13. Select '**ProductSubCategoryID**', '**name**', '**ModifiedDate**' from '**Product Category Sort**' and '**ProductCategorySK**' from '**Product Category Sort**'. Click ***OK***. This transformation ensures that we replace '**Category ID**' of sub category records with '**Product Category Surrogate Key**'.
14. Drag and drop an ***OLE DB Command*** task, rename it as '**Load DimProductSubCategory**', and link it with ***Merge Join*** using the blue line.
15. Before you configure '**Load DimProductCategory**' task, ensure there is a connection to '**SLIIT_Retail_DW**' database in the ***Connection Managers*** area.
16. Double click on '**Load DimProductSubCategory**' to open '**Advanced Editor for Load DimProductSubCategory**' window.
17. In ***Connection Managers*** tab, select the connection manager which points to '**SLIIT_Retail_DW**' database.
18. Go to ***Component Properties*** tab, locate ***SqlCommand*** property, and use the code given below as the command which executes the Stored Procedure created earlier.

    exec dbo.UpdateDimProductSubCategory ?, ?, ?, ?

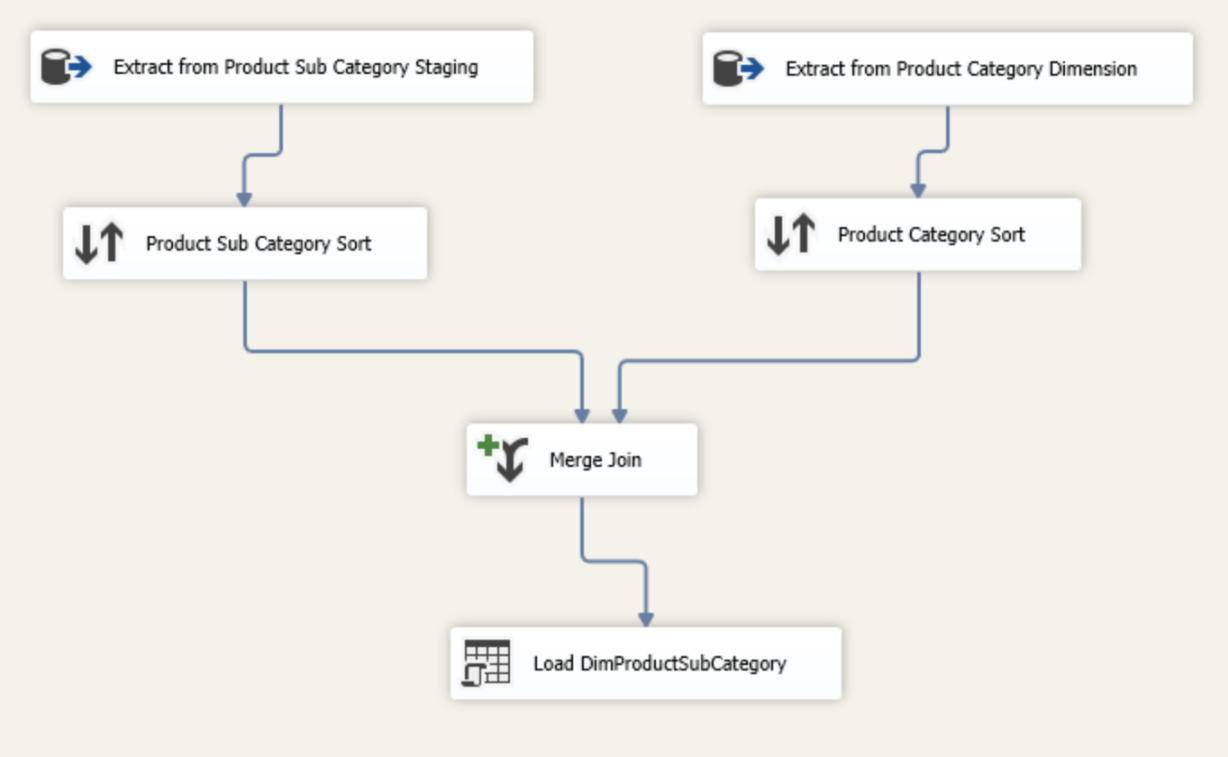


    Note that we are passing 4 arguments. This Stored Procedure ensures no duplicates are entered into the data warehouse table '**DimProductSubCategory**'. If there is an existing sub category record, it will be updated with the latest record coming in from staging table '**StgProductSubCategory**' else, if it is a new record, just insert it.

19. Go to ***Column Mappings*** tab and map the columns to the variables accordingly and click ***OK*** to complete the configuration.

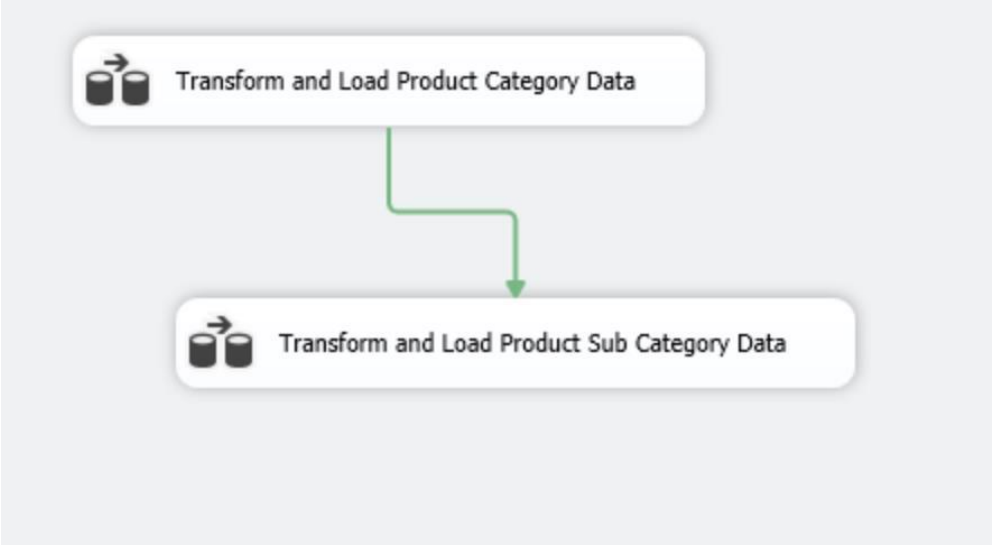


20. Finally, '**Transform and Load Product Sub Category Data**' data flow design should look like below:



Execute the '**Transform and Load Product Sub Category Data**' control flow task and see how data is populated to the '**DimProductSubCategory**' table. Modify records in the staging table '**StgProductSubCategory**' and run the '**Transform and Load Product Sub Category Data**' control flow task to understand behavior of the stored procedure you created earlier.

21. In SSIS Control Flow tab, connect '**Transform and Load Product Category Data**' and '**Transform and Load Product Sub Category Data**' data flows in a way that sub category records are loaded after the category records into the data warehouse.



5. Product Data Transformation and Loading

Now that the product category and sub category data are loaded to the data warehouse, next we will load the product data. Note that product table in the source system contains a foreign key to the sub category table with '**ProductSubCategoryID**'. However, in the data warehouse, this relationship between '**DimProductSubCategory**' and '**DimProduct**' tables are maintained using surrogate keys. Thus, '**DimProduct**' table should contain the surrogate key of '**DimProductSubCategory**' table as the foreign key instead of product sub category ID (value in '**AlternateProductSubCategoryID**' field).

To load the product category data into the '**DimProduct**' table, we can follow the same approach we followed in loading both product category and product sub category data into the data warehouse (using a stored procedure to insert new records and update existing records)., as product data also does not maintain history.

1. Using SQL Server Management Studio, create the Stored Procedure given below, in the '**SLIIT_Retail_DW**' database.

```
CREATE PROCEDURE dbo.UpdateDimProduct
@ProductID int,
@ProductName nvarchar(50),
@ProductNumber nvarchar(25),
@MakeFlag bit,
@FinishedGoodsFlag bit,
@Color nvarchar(15),
@SafetyStockLevel smallint,
@ReorderPoint smallint,
@StandardCost money,
@ListPrice money,
@Size nvarchar(5),
@SizeUnitMeasureCode nvarchar(3),
@Weight decimal(8,2),
@WeightUnitMeasureCode nvarchar(3),
@ProductSubCategoryKey int
AS
BEGIN
if not exists (select ProductSK  from dbo.DimProduct  where AlternateProductID = @ProductID)
BEGIN
insert into dbo.DimProduct (AlternateProductID, ProductName, ProductNumber, MakeFlag, FinishedGoodsFlag, Color,
SafetyStockLevel, ReorderPoint, StandardCost, ListPrice, Size, SizeUnitMeasureCode, [Weight], WeightUnitMeasureCode,
ProductSubCategoryKey, InsertDate, ModifiedDate)
values(@ProductID, @ProductName, @ProductNumber, @MakeFlag, @FinishedGoodsFlag, @Color, @SafetyStockLevel,
@ReorderPoint, @StandardCost, @ListPrice, @Size, @SizeUnitMeasureCode, @Weight, @WeightUnitMeasureCode,
@ProductSubCategoryKey, GETDATE(), GETDATE())
END;
if exists (select ProductSK  from dbo.DimProduct  where AlternateProductID = @ProductID)
BEGIN
update dbo.DimProduct
set ProductName = @ProductName, ProductNumber = @ProductNumber, MakeFlag = @MakeFlag,
FinishedGoodsFlag = @FinishedGoodsFlag, Color = @Color, SafetyStockLevel = @SafetyStockLevel,
ReorderPoint = @ReorderPoint, StandardCost = @StandardCost, ListPrice = @ListPrice,
Size = @Size, SizeUnitMeasureCode = @SizeUnitMeasureCode, [Weight] = @Weight,
WeightUnitMeasureCode = @WeightUnitMeasureCode, ProductSubCategoryKey = @ProductSubCategoryKey, ModifiedDate
= GETDATE()
where AlternateProductID = @ProductID
END;
END;
```

2. Now, back in SSIS solution, Drag and drop a ***Data Flow Task***, rename it as '**Transform and Load Product Data**' and go to the ***Data Flow*** tab to design the '**Transform and Load Product Data**' data flow.
3. Drag and drop ***OLE DB Source*****,** rename as '**Extract from Product Staging**' and configure it to extract the '**StgProduct**' table. Make sure all the columns are selected.
4. Now, we need to replace the sub category ID of product records with product sub category surrogate key value. There are many ways we can achieve this within SSIS, without writing code:
    a. By joining the product data with '**DimProductSubCategory**' table to retrieve the matching '**ProductSubCategorySK**'. *This is what we did in section 4 when loading product sub category data.*
    b. Use a ***Lookup*** transformation to lookup for the relevant '**ProductSubCategorySK**' for respective '**AlternateProductSubCategoryID**'s.

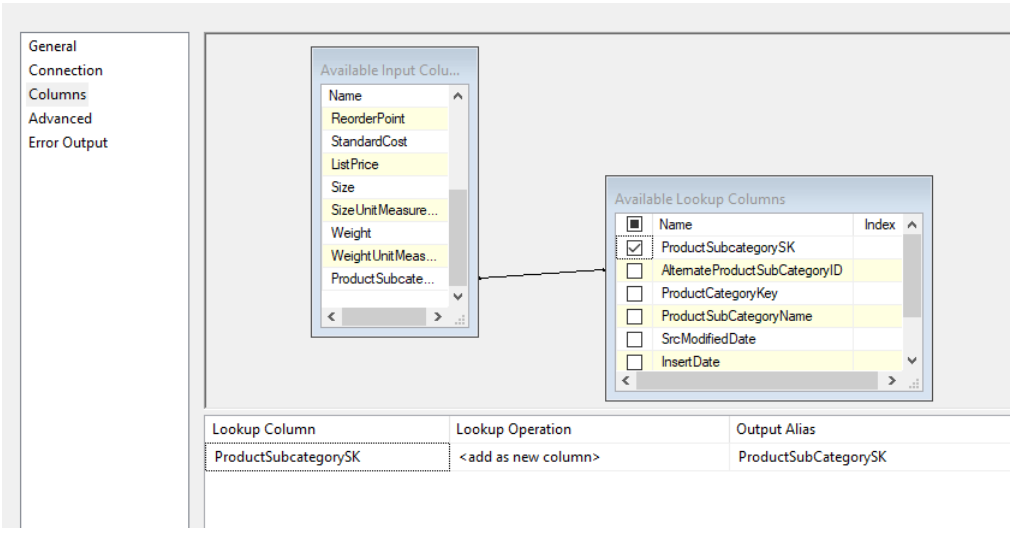
    Let's use method (b) and use a ***Lookup*** transformation to retrieve matching '**ProductSubCategorySK**' for respective '**AlternateProductSubCategorID**'s.

5. Drag and drop a ***Lookup*** component from ***SSIS Toolbox***, rename it as '**Product Sub Category Lookup**' and connect it with '**Extract from Product Staging**' component using the blue line.
6. Double click '**Product Sub Category Lookup**' component to open the ***Lookup Transformation Editor*** window. In ***General*** section, select ***Full cache*** as the ***Cache mode***, ***OLE DB connection manager*** as the ***Connection type*** and select ***Ignore failure*** from the ***Specify how to handle rows with no matching entries*** drop down list.
    Read more on different ***Cache modes*** in SSIS.
7. In ***Connection*** section (second tab in left), ensure connection is set to '**SLIIT_Retail_DW**' and select '**DimProductSubCategory**' as the table. We will connect to '**DimProductSubCategory**' to lookup for matching '**AlternateProductSubCategorID**' and pick the corresponding '**ProductSubCategorySK**'.
8. In **Columns** section, you will see two windows as **Available Input Columns** (columns passing through from the previous component) and **Available Lookup Columns** (columns

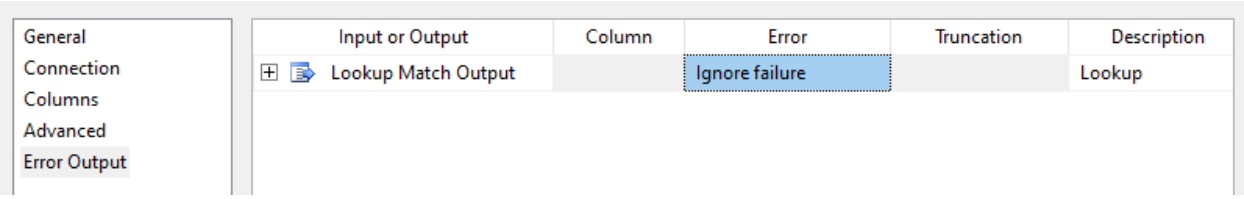
available from the lookup table we selected: '**DimProductSubCategory**'). Map '**ProductSubCategoryID**' in the ***Available Input Columns*** box with '**AlternateProductSubCategorID**' in the ***Available Lookup Columns*** box and select '**ProductSubCategorySK**' by ticking on the box next to the field name. Once you tick the checkbox, selected field will appear in the grid below.

9. In the grid below, set ***Lookup Operation*** as ***<add as new column>*** and provide the ***Output Alias*** as '**ProductSubCategorySK**'; this will take both '**ProductSubCategoryID**' and '**ProductSubCategorySK**' forward to the next step; but we only need '**ProductSubCategorySK**'.

   Optionally, for the ***Lookup Operation*** you can select Replace '**ProductSubCategoryID**'; if you do so, '**ProductSubCategorySK**' will replace '**ProductSubCategoryID**'.

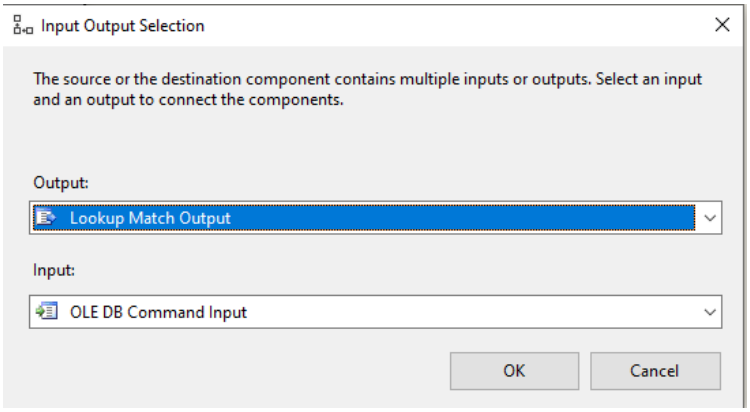


10. In ***Error Output*** section, in the grid on the right side of the window, for ***Lookup Match Output*** entry under ***Input or Output*** column, ensure ***Ignore failure*** is selected as the ***Error*** option. This specifies what to be done if there are errors in matching records. Note that ignoring is not the ideal option to have; but to keep the scenario simple let's use ignore option for now. (This avoids NULL values for sub categories in product table)



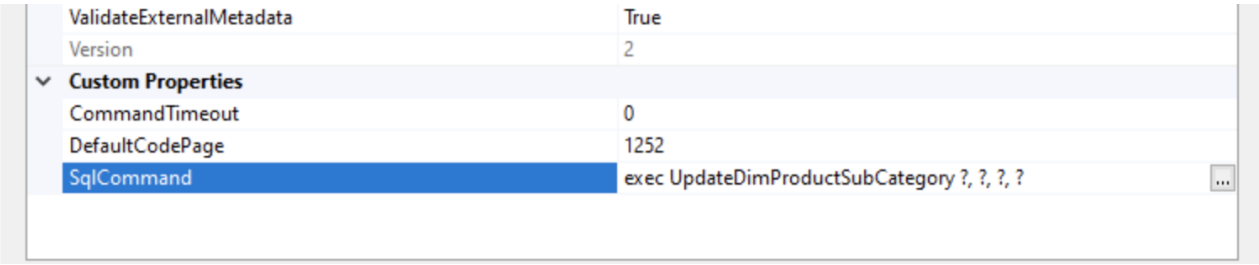
11. Click ***OK***.
12. Drag and drop an ***OLE DB Command*** task, rename it as '**Load DimProduct**', and link it with '**Product Sub Category Lookup**' using the blue line. When you try to link ***Lookup*** component with the ***OLE DB Command*** component, ***Input Output Selection*** window pops-up. A lookup table can return both matching and non-matching records with the lookup column we specify. Select ***Lookup Match Output*** and click ***OK***.

    Optionally, you can send the non-matching columns to a file to monitor records which does not have matching records in sub category table and take corrective actions by following a *Referential Integrity* (RI) approach.



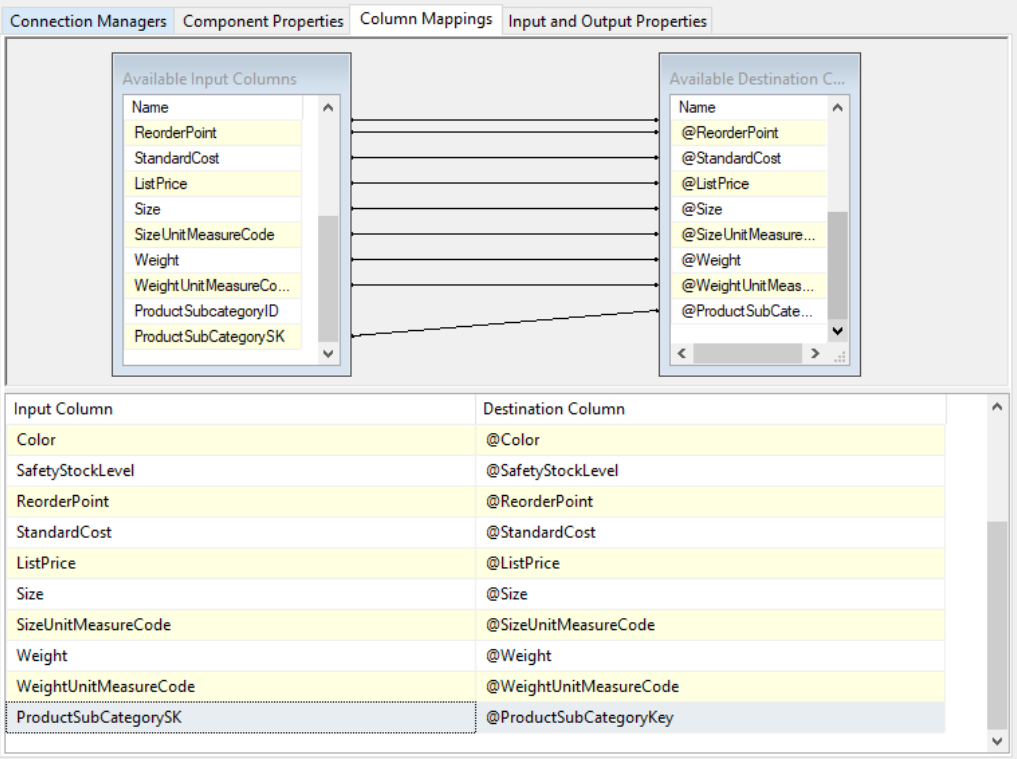
13. Double click on '**Load DimProduct**' to open '**Advanced Editor for Load DimProduct**' window.
14. In ***Connection Managers*** tab, select the connection manager which points to '**SLIIT_Retail_DW**' database.
15. Go to ***Component Properties*** tab, locate ***SqlCommand*** property, and use the code given below as the command which executes the Stored Procedure created earlier. (with 15 parameters)

    exec dbo.UpdateDimProduct ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?

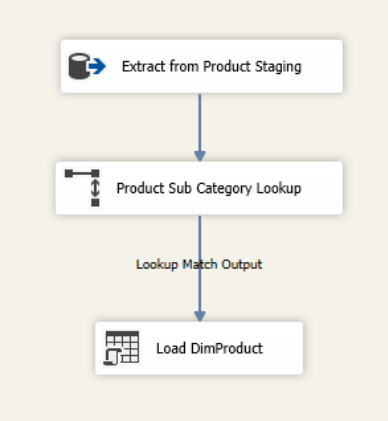


Note that we are passing 15 arguments. This Stored Procedure ensures no duplicates are entered into the data warehouse table '**DimProduct**'. If there is an existing product record, it will be updated with the latest record coming in from staging table '**StgProduct**' else, if it is a new record, just insert it.

16. Go to ***Column Mappings*** tab and map the columns to the variables accordingly and click ***OK*** to complete the configuration. Ensure to map '**ProductSubCategorySK**' as the input for **@ProductSubCategoryKey**, n ot '**ProductSubCategoryID**'.



17. Finally, '**Transform and Load Product Data**' data flow design should look like below:



It is much easier to use Lookup components rather than using multiple sources with ***Sort*** and ***Merge*** components.

Execute the '**Transform and Load Product Data**' control flow task and see how data is populated to the '**DimProduct**' table. Modify records in the staging table '**StgProduct**' and run the '**Transform and Load Product**' control flow task to understand behavior of the stored procedure you created earlier.

18. In SSIS Control Flow tab, connect '**Transform and Load Product Sub Category Data**' and '**Transform and Load Product Data**' data flows in a way that product records are loaded after the sub category records into the data warehouse.