Gyubeom Kim

INFO 498

Randal Root

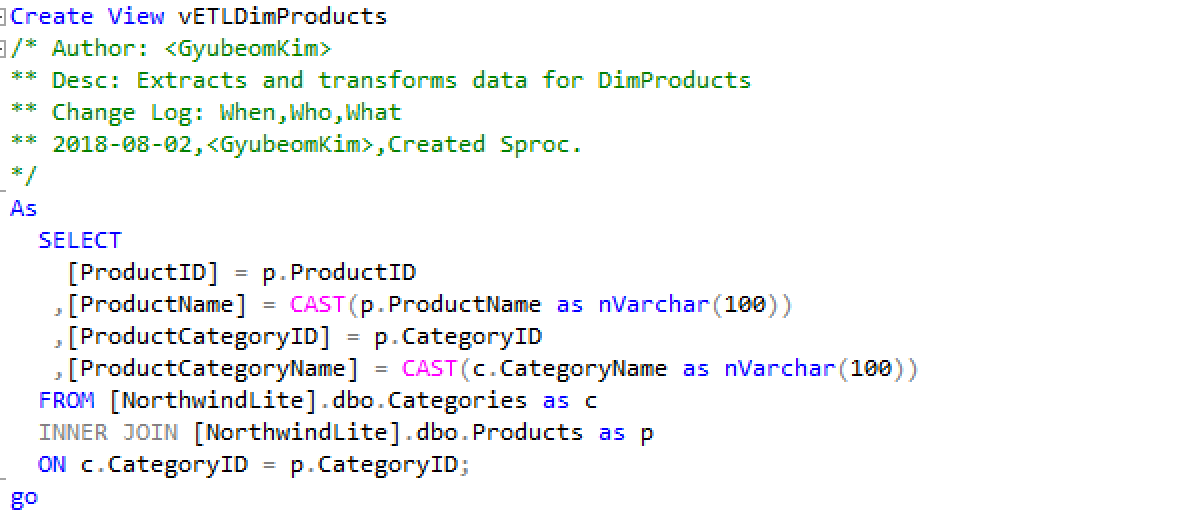
2018/08/02

**Assignment3**

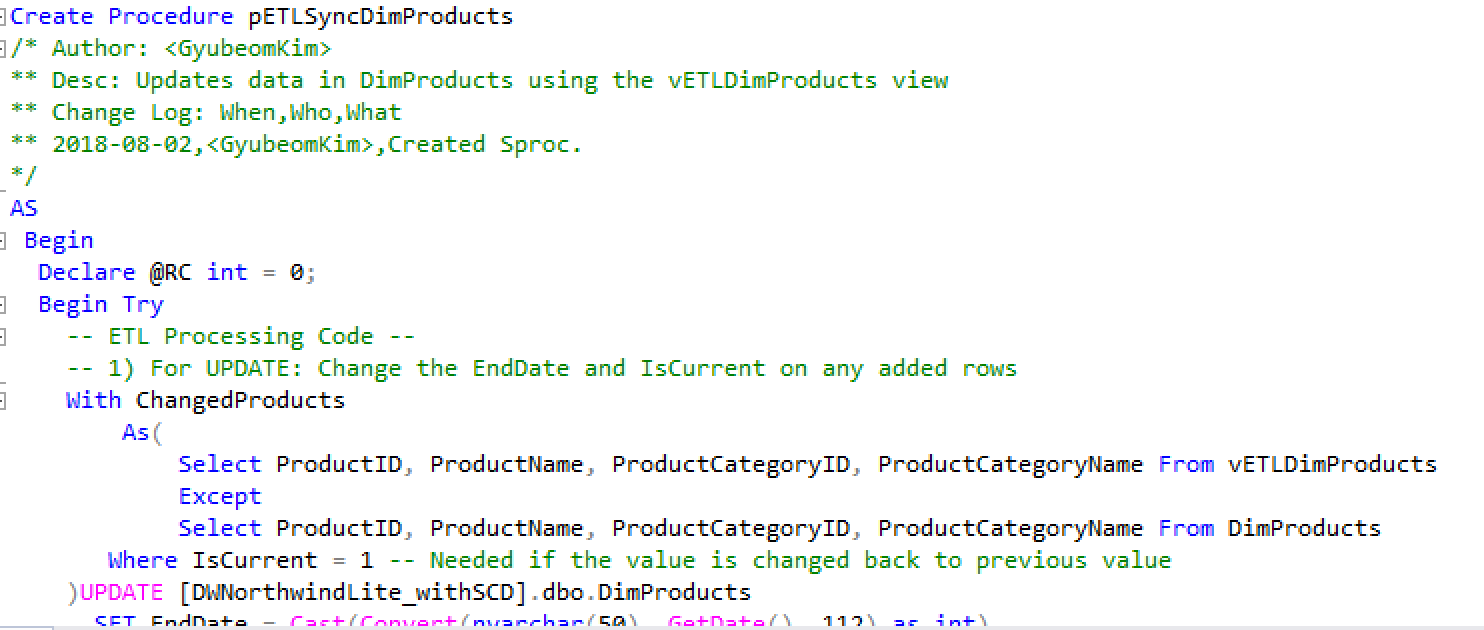
For the third assignment of one of my academic courses, INFO 498, I started to create lite version of the Northwind database by running the provided code. Then, I created a new data warehouse, DWNorthwindLite\_withSCD based on the NorthwindLite DB. There are three dimensional tables and one fact table. For the incremental loading, I created some ETL Processing script with SQL for each table by using different type of SCD, Slowly Changing Dimension.

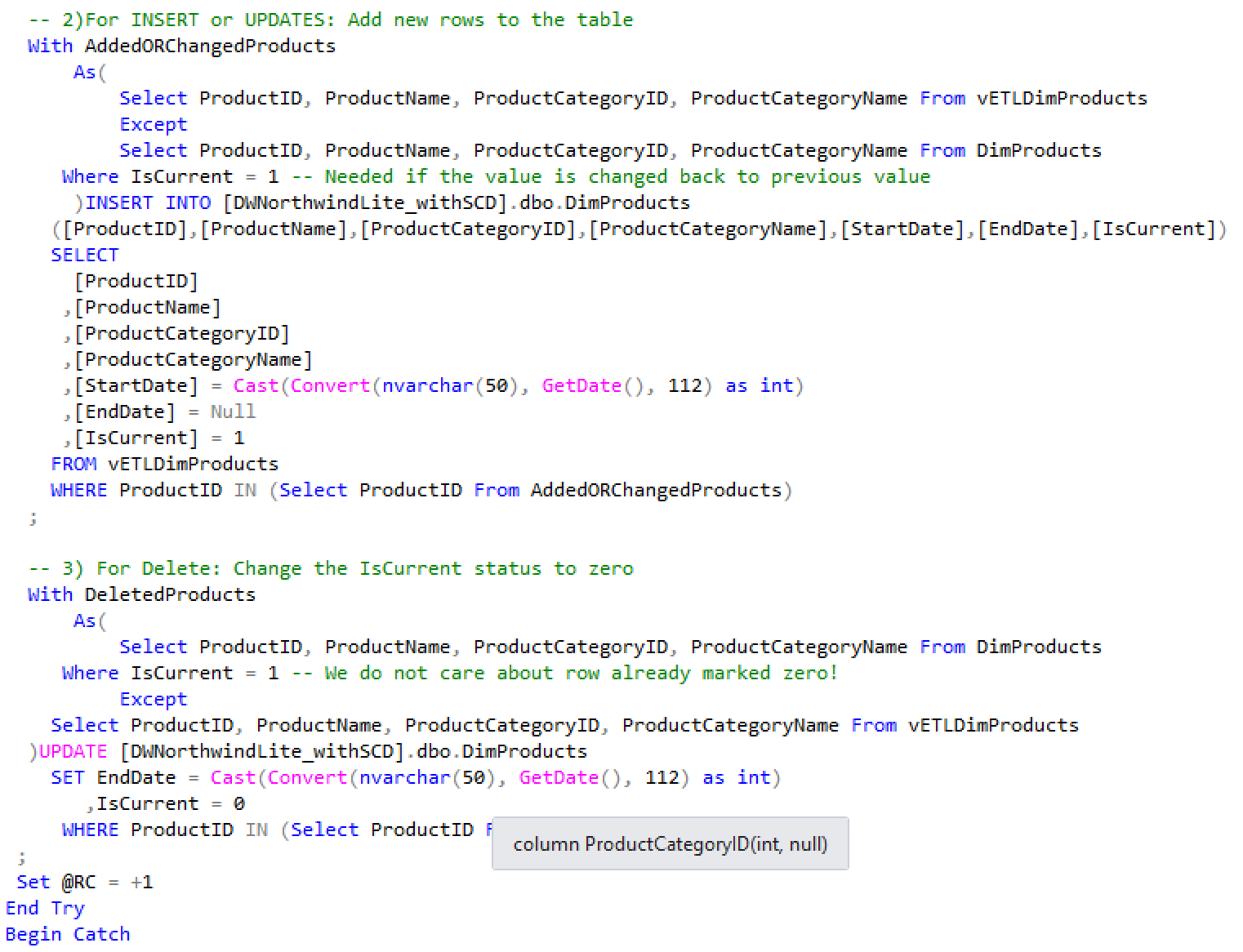
**Summary:**

Before physically creating the ETL script for incremental loading, I carefully started to look the tables first. Then, I compared values between the Source and Target tables. Then, I was possible to insert rows to the target tables where new rows are found in the source table, update rows in the destination that are changed in the source and delete rows from the destination that are removed in the source (Root, 2018). For this process, the different type of SDC is needed. It tracks the status of change in each row or column. Unlike the previous assignment, we did not need to drop the foreign key constraints and clear the tables. However, we have to make an abstraction layer, a view, for the incremental loading process.

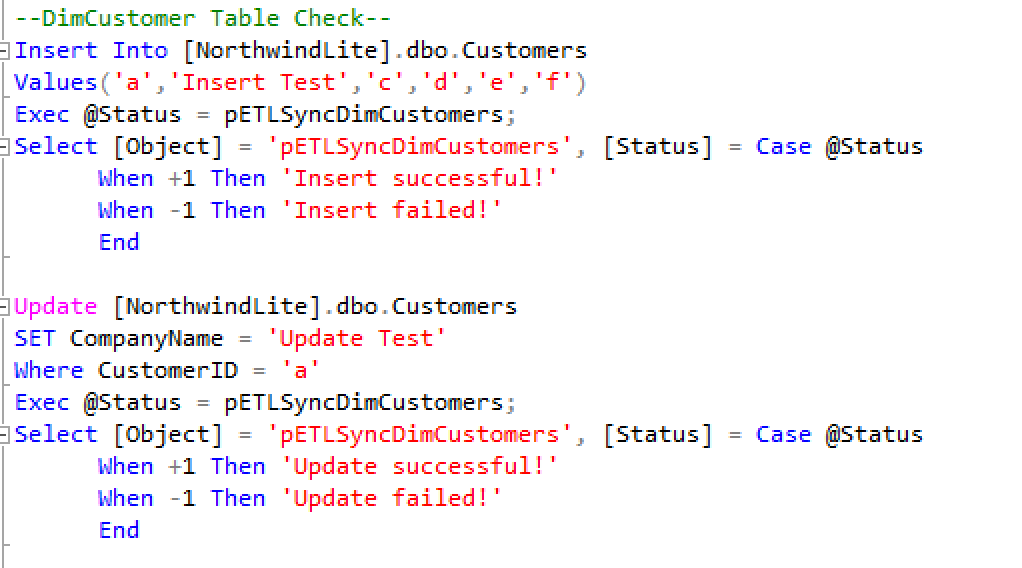


As shown picture above, I created a view for one of the dimensional tables, DimProduct. In this process, I used a view before syncronizing the data in the table. The view is a very simple and effective tool, but it does have some disadvantages. You cannot define parameters on a view. Parameters allow you to pass in specific arguments to get back differing results. This is a useful technique when incrementally loading a data warehouse (Root, 2018).

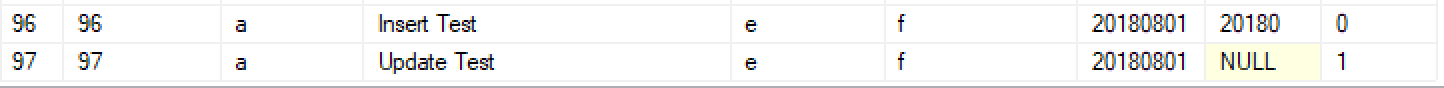


As the picture shown above, I created the stored procedure for syncronizing data in the table. Through the process, we are able to insert, update, and delete. The rest of code picture is below.

As you can see in the picture above, there is column for isCurrent and EndDate. Throught these columns, we can keep track the history of the changes, such as insert, update, and delete. The next step was same. I also created same procedure in DimCustomers table.

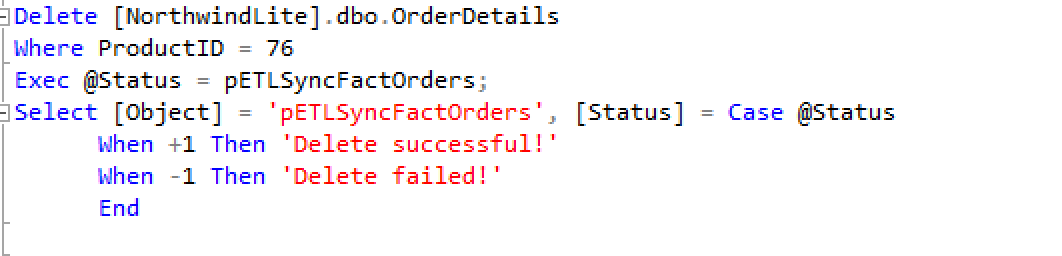


As the picture shown above, I created test code for the procedure. You can see the result of the history of changes, insert and update, in the picture below.

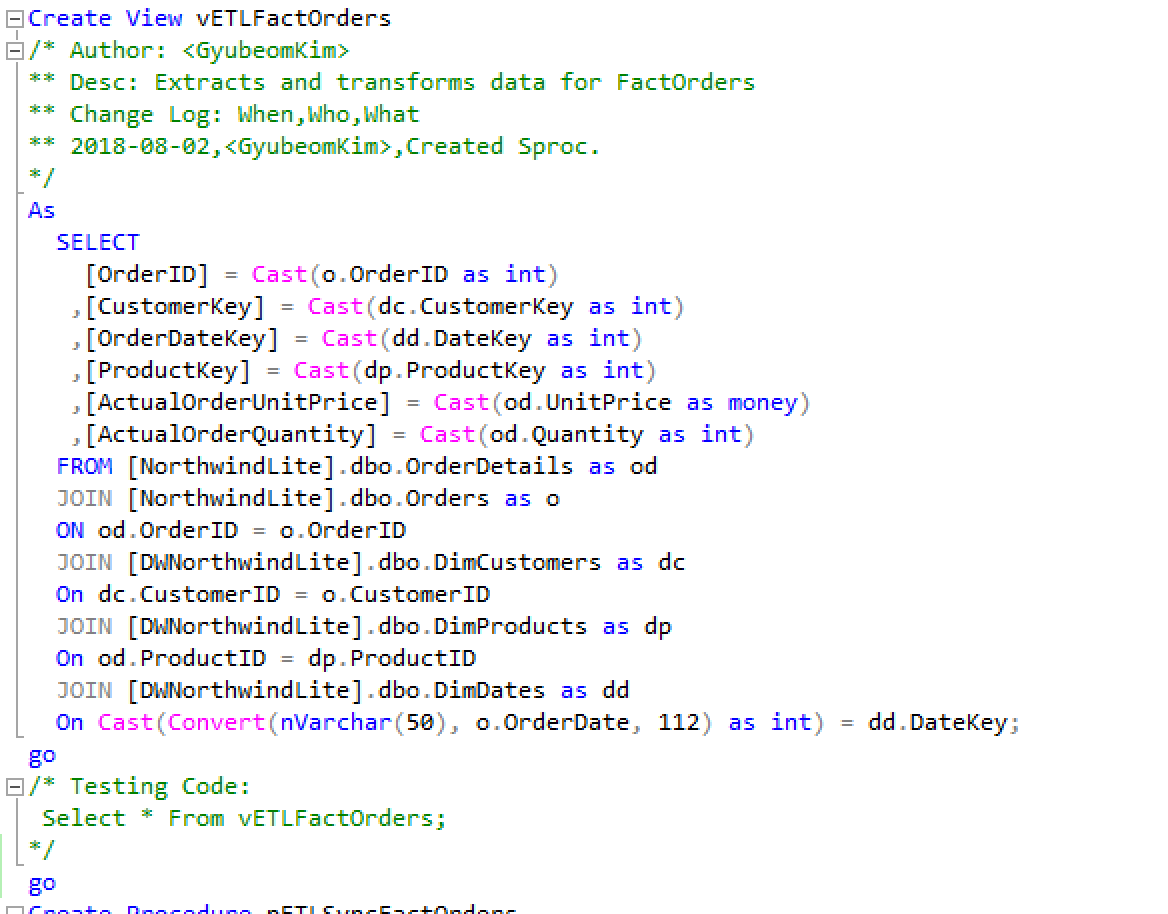


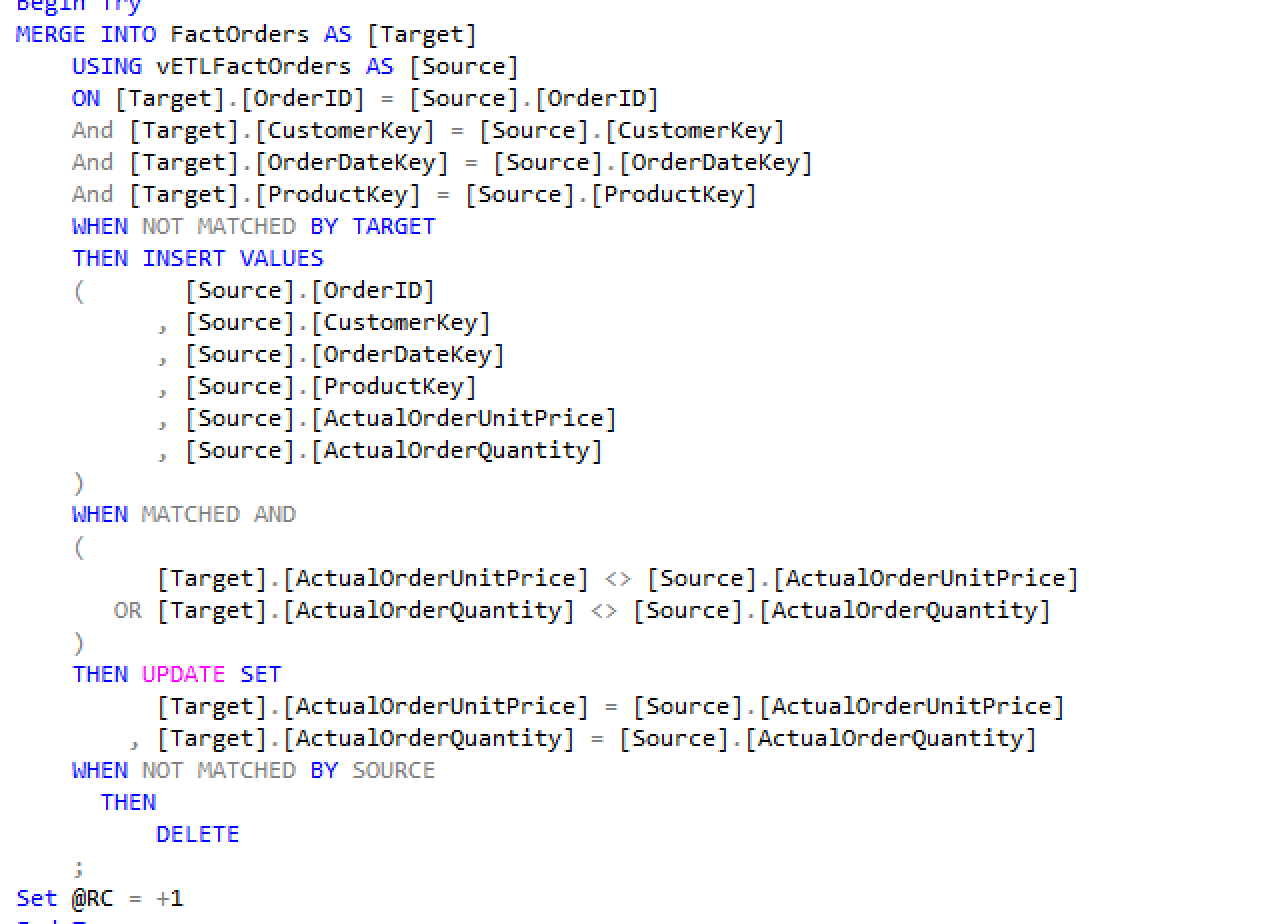
Then, I created test code for delete for the updated data. As you can see the picture below, when the updated data is deleted, isCurrent column became 0.



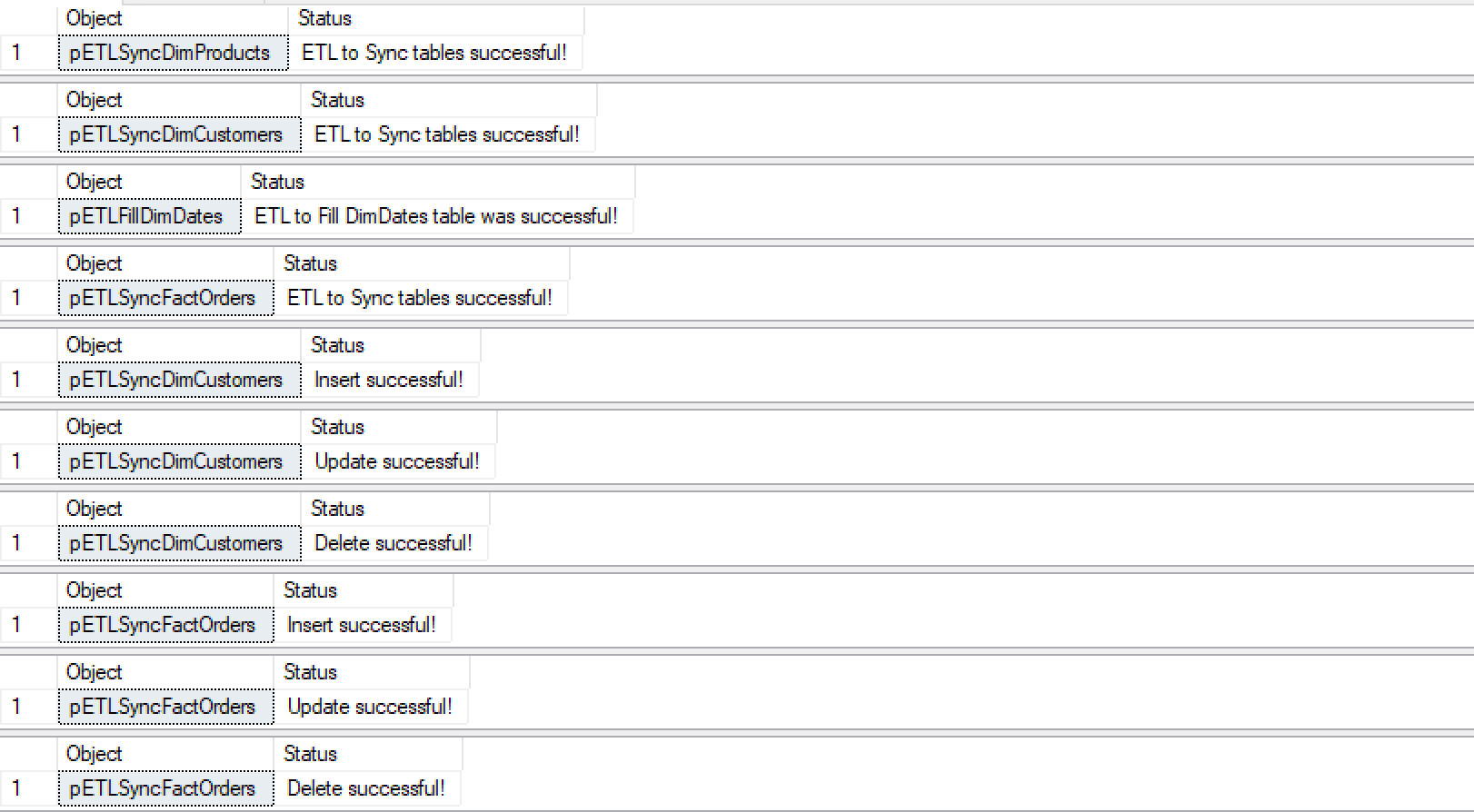
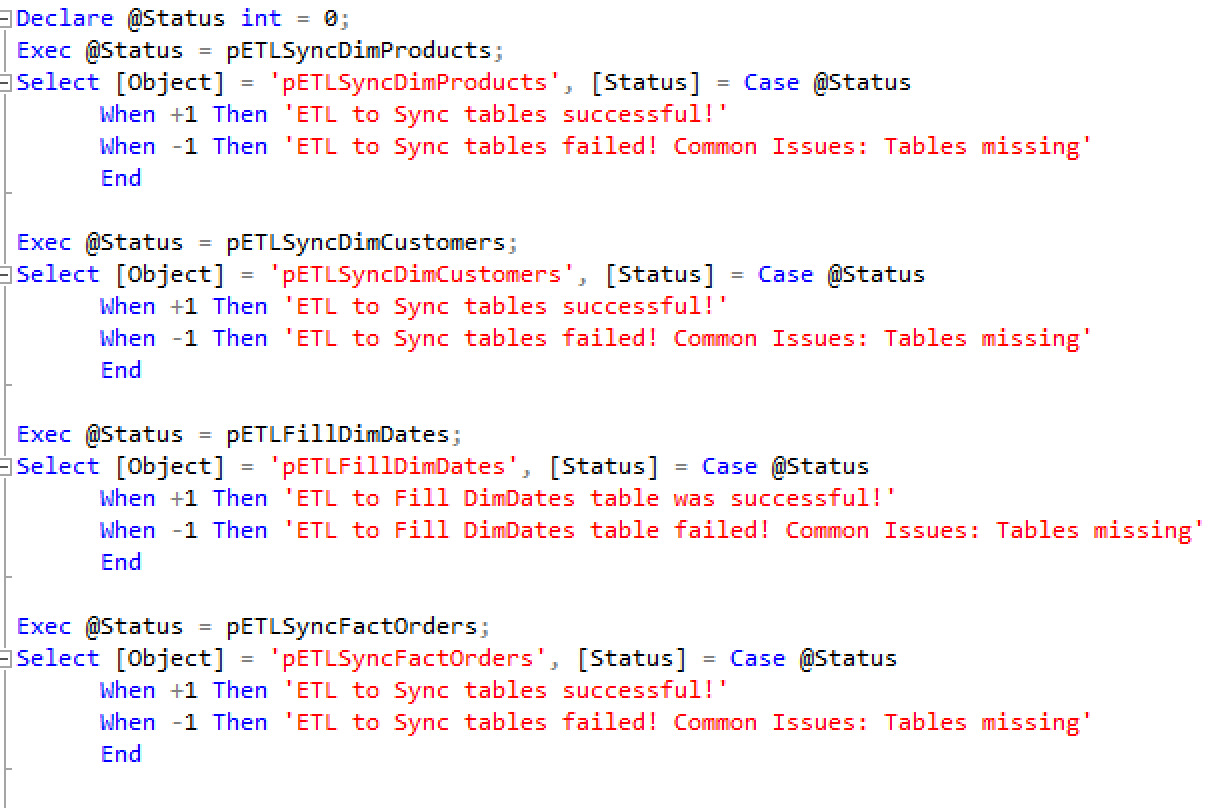


I used type 2 of Slowly Chaning Dimesion processes for the dimesional tables. It tells and tracks the history of changes. However, I used the type 1 of of Slowly Chaning Dimesion processes for the fact table. Unlike the type 2, there is no traking the history of changes in the type 1. I used the way of Merge in this table for syncronizing the data in the table. Before the proces, same as I did on the dimtables, I created the view as the picture shown below.

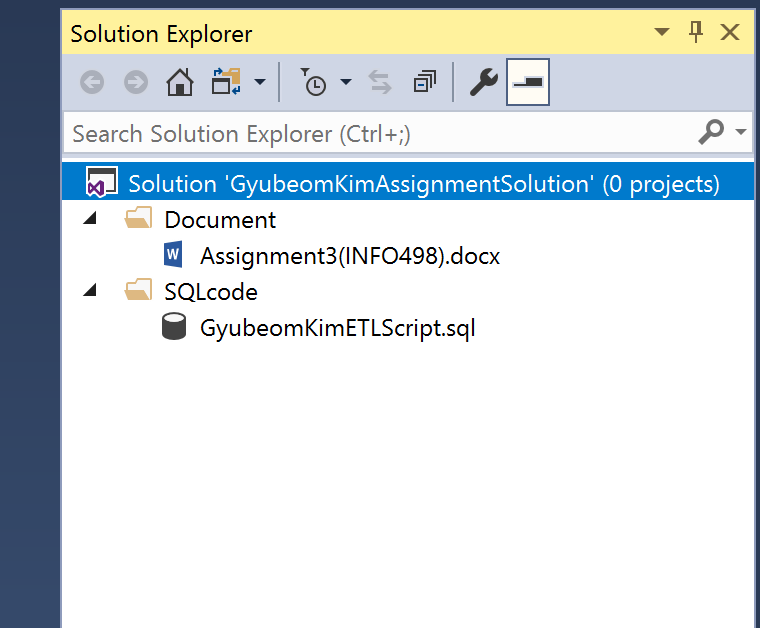




Then, I created the stored procedure for syncronzing data. I chose target table as FactOrders and source table as view table. As you can see in the picture above, it inserts, updates, and deletes on target table Whether it matches with given conditions or not.

****

Then, I created review and test code for the stored procedures as you can see on the picture above.



For the very last step of the assignment, I created a solution for organizing files in Visual Studio. I created a blank solution file, then added files to the respective folders as the picture shown above.

**Details (ETL Process for Synchronizing data):**

**Product Table (Dimesional Table)**

-Created views before Syncronizing data into table, and stored procedure for all tables

EX) Created View [Name]

As Select [Column = CAST(…), ] From [TableName]

Created Procedure [Name]

INSERT INTO [Column,…] SELECT FROM [View]

-In the process of syncronizing, used type 1 of SCD (which tells the history of changes)

EX) See the picture above

-Test Code added

**Customer Table (Dimesional Table)**

-Created views before Syncronizing data into table, and stored procedure for all tables

EX) Created View [Name]

As Select [Column = CAST(…), ] From [TableName]

Created Procedure [Name]

INSERT INTO [Column,…] SELECT FROM [View]

-In the process of syncronizing, used type 1 of SCD (which tells the history of changes)

EX) See the picture above

-Test code added

**Date Table (Dimesional Table)**

-Cofronted with foreign key errors when I runned code multiple times

Ex) 

-Able to solve the conflicted foreign key by dropping key first and adding key later.

EX)

**Order Table (Fact Table)**

- Created views before Syncronizing data into table, and stored procedure for all tables

EX) Created View [Name]

As Select [Column = CAST(…), ] From [TableName]

Created Procedure [Name]

INSERT INTO [Column,…] SELECT FROM [View]

-Used Merge statesment

Ex) The picture shown above

-Type 1 of SCD, no history status of changes.

-Test code added

**Review And Test**

-a script that tells whether all the processes is successful or not.

EX) Declare @Status int;

Exec @Status = Stored Procedure Name;

Select [Object] = ' Stored Procedure Name ', [Status] = Case @Status

When +1 Then ‘ successful!’

When -1 Then ‘failed!’

End

**Work Cited**

Root, R. (2018). *Module2*