Sri Lanka Institute of Information Technology

Data Warehousing and Business Intelligence IT3021

Assignment 1

2025

<u>Assignment 1 Report</u>

Student Name – Gunathilaka P. A. S. R

IT Number – IT22136824

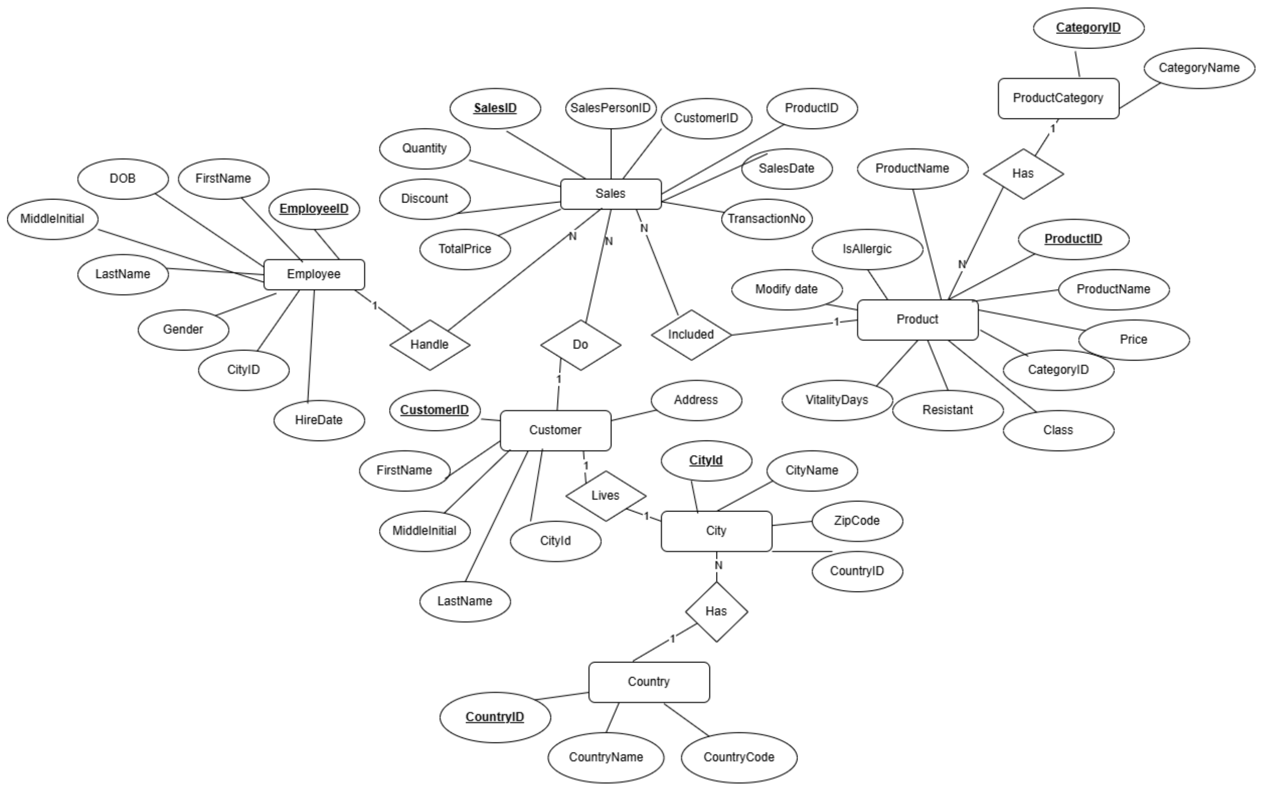
Table of Contents

1 Dataset Selection

1.1 Description

Dataset selected – Grocery Sales Data
The Grocery Sales Dataset contains data designed for analyzing sales transactions by Customers, Products, Employees and Geographical information. The original dataset consists of **7 csv files** and **1 year of sales data** and will enable us to explore sales trends, customer behaviors and business insights.
The original data tables of the dataset have been edited, configured and rearranged to suit the requirements of the project. Hence, 7 data tables have been identified.

1. Sales.csv – Contains Transactional data for each sale.
2. Products.csv – Contains details of the products being sold.
3. Employees.csv – Stores details of employees handling transactions.
4. Customers.csv – Contains details about the customers who make purchases.
5. Countries.csv – Stores country related metadata.
6. Cities.csv – Contains city level demographic data.
7. Categories.csv – Contains product categories.



ER-Diagram

2 Preparation of the Data Sources

All the 7 data files in the original dataset were in the csv format. Therefore, those data files were separated into multiple data source types to meet project requirements.
Three types of data sources were utilized: **csv, txt, database**.

1. Database –

- A **source database (DWBI_Assignment1_Source**) was created by importing the Sales.csv, Employees.csv, Customers.csv and Products.csv files.
- Sales Table – contains transactional data (Sales Id, quantity, Sales Date, Total price etc..) for each sale.
- Employee Table – contains employee data (Employee Id, First Name, Hire Date etc..) who handles transactions.
- Customer Table – contains customer data (Customer id, First Name, Last Name, Address etc..).
- Product Table – contains details of products (Productid, Product name, Category Id, Unit price etc..) being sold.

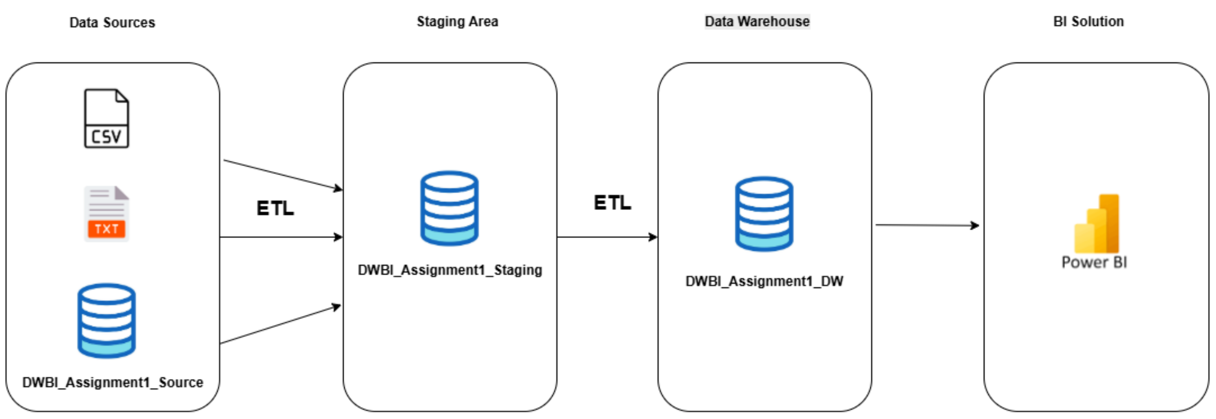
2. .csv –

- The **product category data** were kept in the csv file format. This contains data about product categories (Product category id, category name).

3. .txt –

- Demographic information data such as country and city data were saved in the .txt file format. Hence 2 .txt files (**cities.txt, country.txt**) files were generated. These files contain data such as country name, country code, city name, zip codes.

3 Solution Architecture



High Level Solution Architecture Diagram

The above is a high-level Data Warehousing and Business Intelligence architectural solution for the chosen dataset and topic. We can identify four main layers here:

1. Data Sources –

- The first step in a Data Engineering project is to establish a solid architecture to gather data from various sources such as Cloud data, APIs, Operational Databases, Flat files etc. depending on the goal and resources available. In this scenario, two data sources, a source database and flat files (.csv, .txt) have been used.
- Source Database (**DWBI_Assignment1_Source**) contains Sales, Employee, Customer and Product tables.
- CSV (**ProductCategory.csv**) contains product category data.
- TXT files (**Country.txt, Cities.txt**) contain demographic data.

2. Staging area –

- Used to store data temporarily during the ETL process between data sources and data warehouse.
- Store data from all the data sources at one place so they can be transformed and loaded into the data warehouse while minimizing the effect on the sources. In this scenario, **DWBI_Assignment1_Staging** acts as the data staging area.

3. Data warehouse –

- A data warehouse is a large collection of historical business data that is used to enhance internal decision-making using Business Intelligence and Machine Learning. In this scenario **DWBI_Assignment1_DW** is used as the data warehouse.

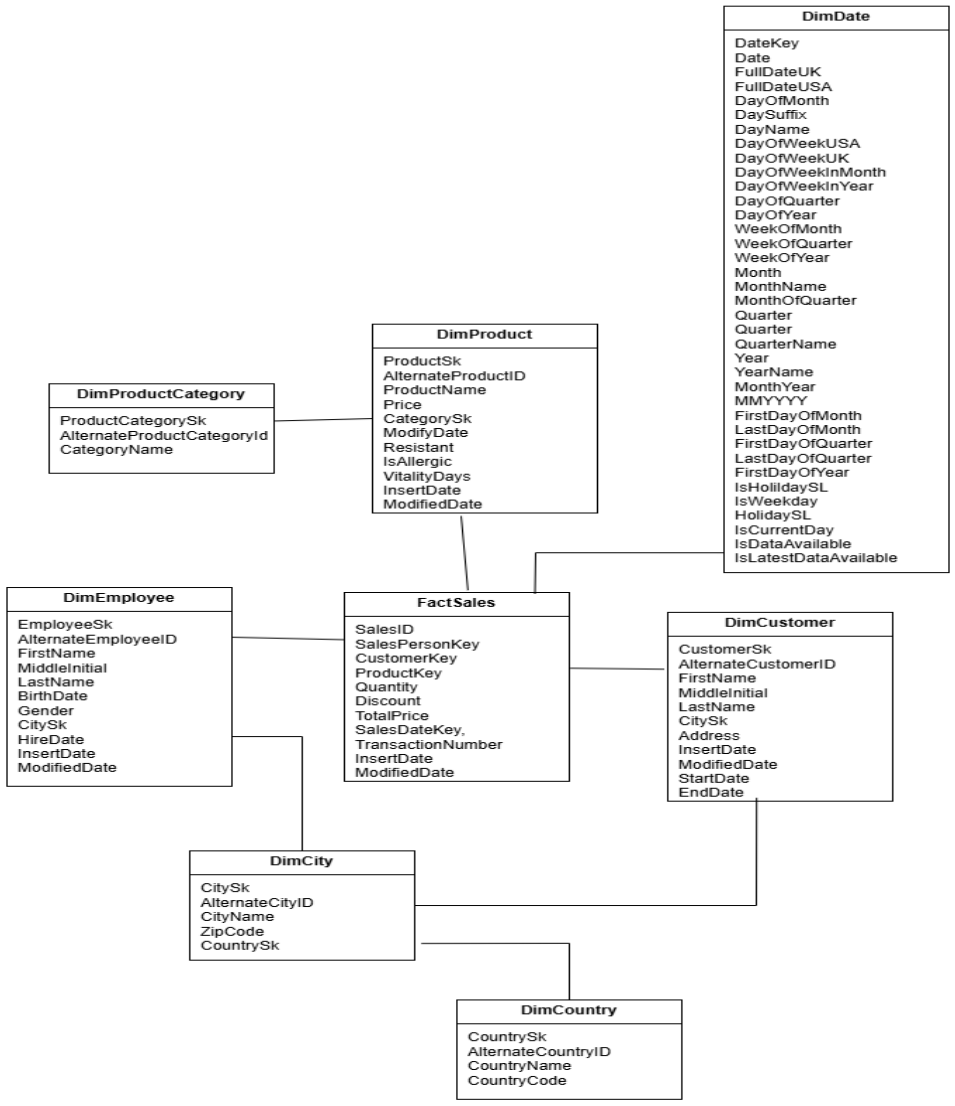
4. BI solution (consumption) –

- This employs technology and services to transform data into actionable insights that assist organizations in making better decisions. In this scenario, SQL Server Analysis Service **(SSAS)** is used to create data cubes and **Power BI** is used to create dashboards.

5. ETL –

- ETL is the process of extracting from various data sources, transforming those data and loading them to a destination. In this scenario, SQL Server Integration Service **(SSIS)** is used as the ETL tool.

4 Data Warehouse Design and Development



Dimension Model

The above is the dimension model used for this scenario. 6 dimensions tables excluding date dimension and a single fact table have been identified and designed.

- **Schema used – Snowflake Schema**
  Snowflake schema is used to reduce redundancy through normalization. As seen in the diagram above, City dimension and Product dimension tables have been normalized resulting in Country and Product Category Dimension tables. The central fact table, FactSales which stores sales transactions, includes metrics like total price and discount, is connected to Product, Customer, Date, Employee dimensions through surrogate keys.

- **Dimention And Fact Table**

Seven-dimension tables and a single fact table were created:

- DimProduct - The Product Dimension Table contains product information and ProductSk acts as the surrogate key and there is a reference from DimProduct to DimProductCategory table.
- DimProductCategory - The ProductCategory Dimension table contains product category information. ProductCategorySk acts as the surrogate key.
- DimEmployee - The Employee Dimension contains employee's detail. EmployeeSk acts as the surrogate key and there is a reference from DimEmployee to DimCity table.
- DimCustomer - The Customer Dimension contains customer details. CustomerSk acts as the surrogate key and there is a reference from DimCustomer to DimCity table.
- DimCity - The City Dimension contains information about cities. CitySk acts as the surrogate key and there is a reference from DimCity to DimCountry tables.
- DimCountry - The Country Dimension contains information about countries. CountrySk acts as the surrogate key.
- DimDate - This is a common dimension. It can be role playing and static dimensions as well. DateKey acts as the surrogate key.
- FactSales – Contains all sales transactions data. There is no surrogate key and there are references to DimProduct, DimCustomer, DimEmployee and DimDate dimension tables.

Assumptions – It was assumed that Product, Product category, Employee, City and Country data will not change with time.

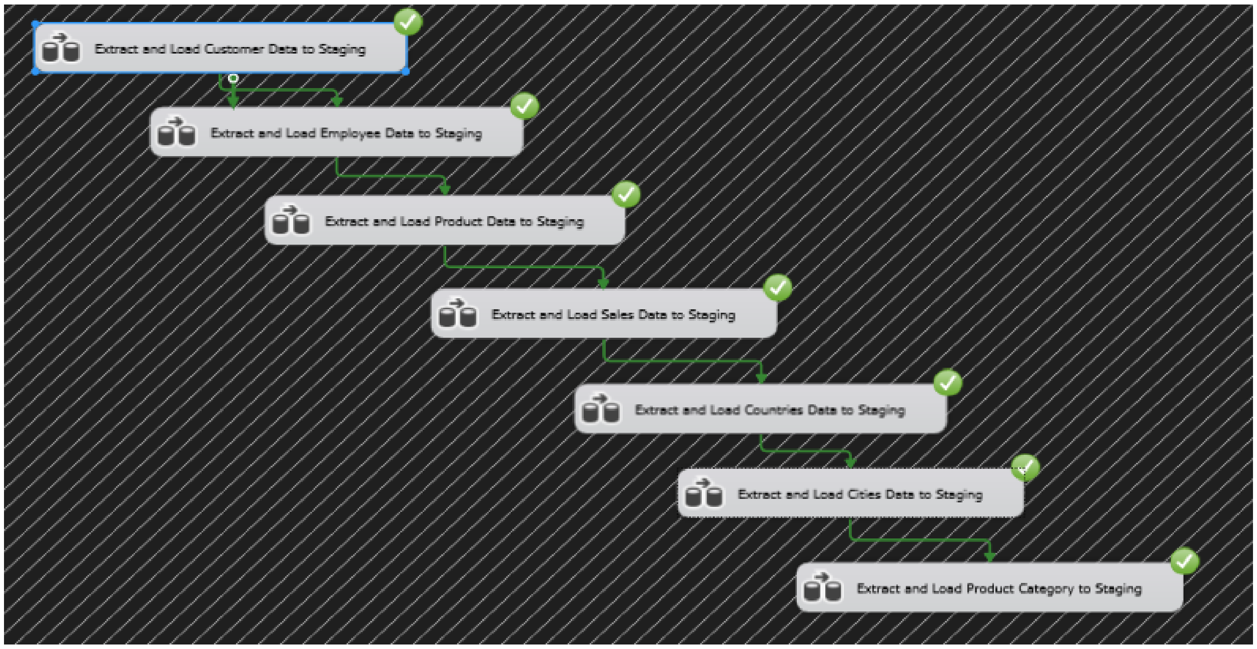
- **Slowly Chaning Dimention**

It was assumed that Customer details such as **last name and address** may change over time. Hence, DimCustomer was considered as a **slowly changing dimension**. **Last name** attribute is considered as **changing attribute (SCD Type1) and Address attribute** is considered as a **historical attribute (SCD Type2**).

5 ETL Development

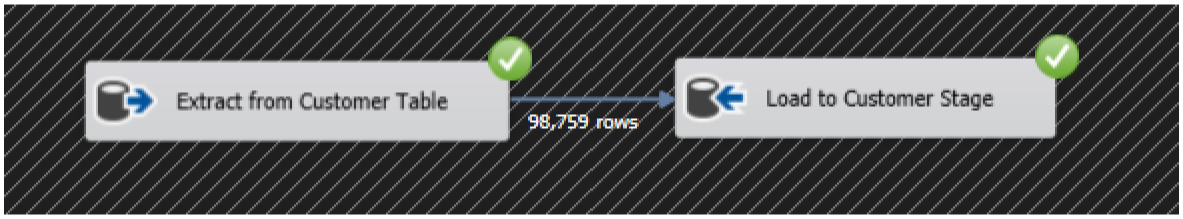
5.1 Extract Data from Source to Staging

- Order of Execution

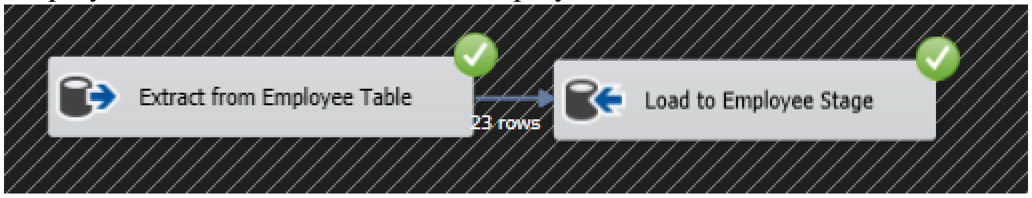

Extract and Load Customer Data to Staging
Extract and Load Employee Data to Staging
Extract and Load Product Data to Staging
Extract and Load Sales Data to Staging
Extract and Load Countries Data to Staging
Extract and Load Cities Data to Staging
Extract and Load Product Category to Staging

Data is loaded from a database source and flat file sources and is staged in an intermediate location until being loaded into the data warehouse. Individual extractions into the staging database happen as below images:

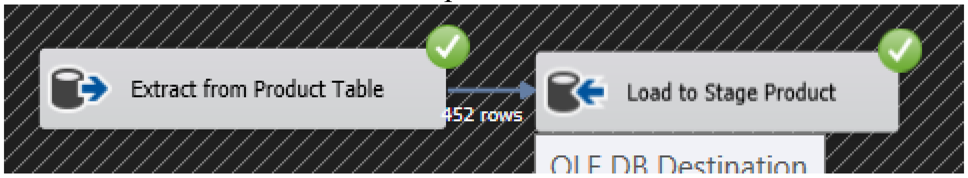
- Customer data are extracted from the customer data table in the source database.



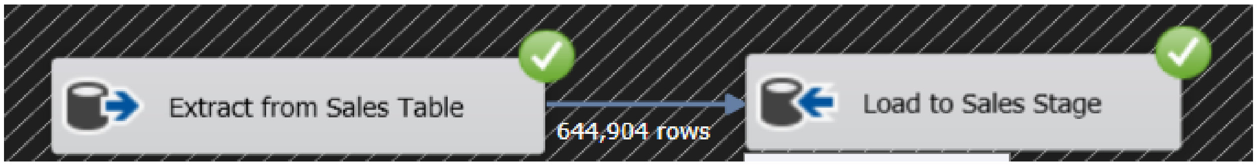
- Employee data is extracted from the employee table in the source database.



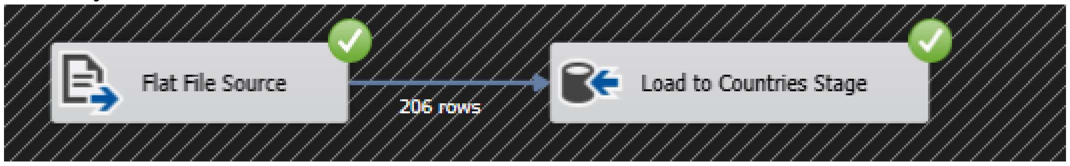
- Product data is extracted from the product table in the source database.



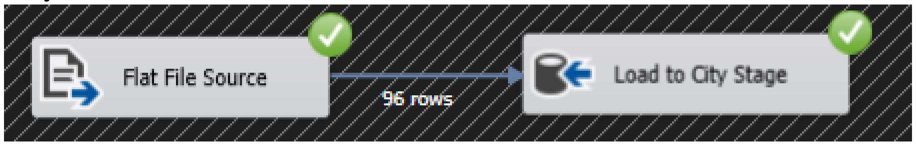
- Sales data is extracted from the sales table in the source database.



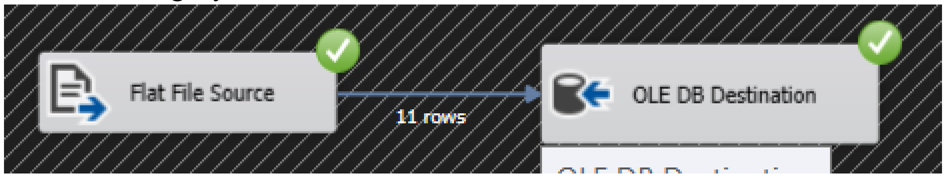
- Country data is extracted from a .txt flat file source



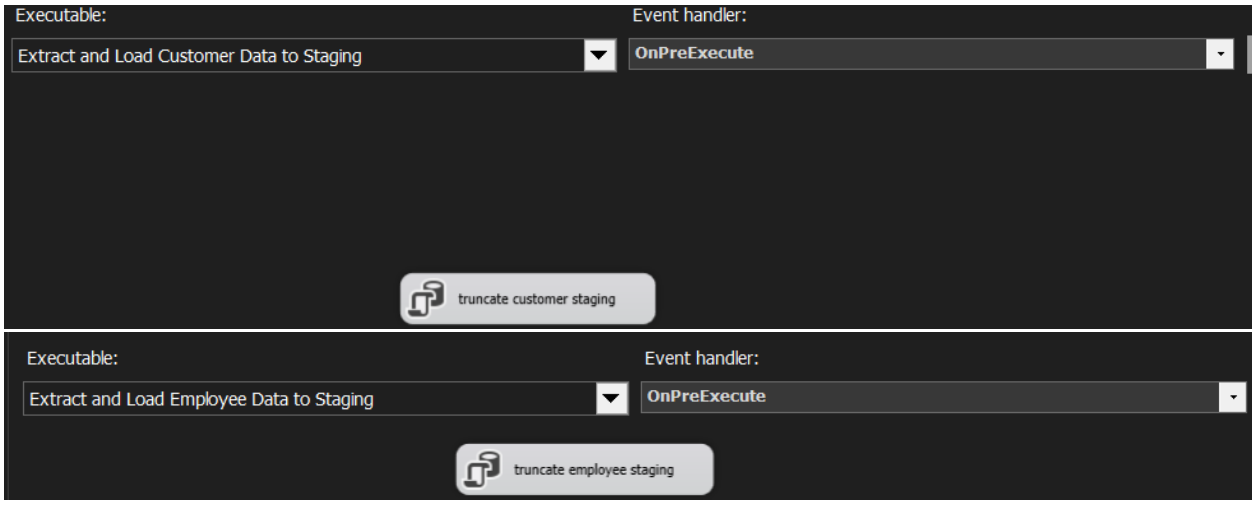
- City data is extracted from a .txt flat file source.



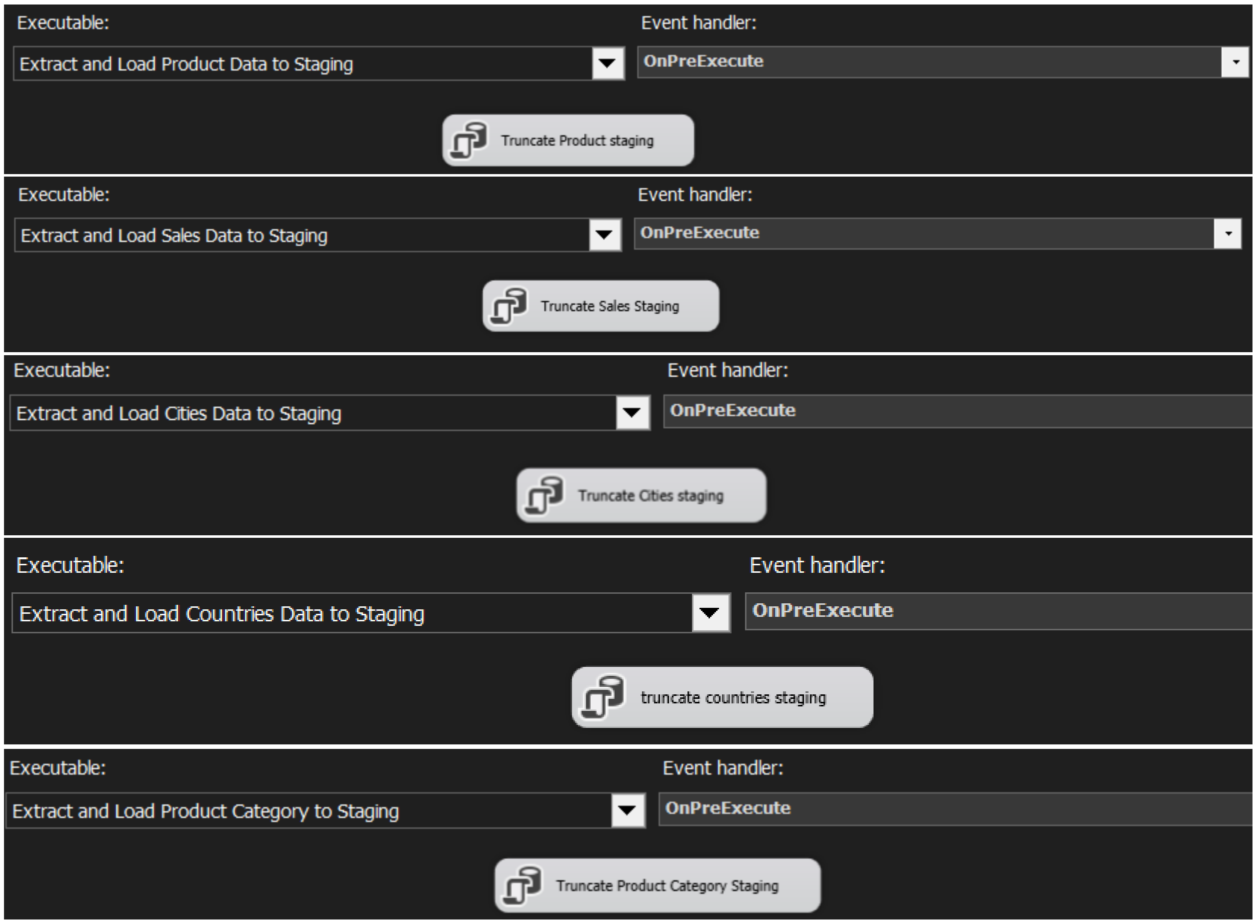
- Product Category data is extracted from .csv flat file source.



- If we run the package multiple times, the staging tables will be repeatedly loaded with the source data without truncating the data already within it. To avoid this **OnPreExecute** event handlers with **execute sql tasks** are used for each staging table.


Executable: Extract and Load Customer Data to Staging
Event handler: OnPreExecute

Executable: Extract and Load Employee Data to Staging
Event handler: OnPreExecute


Executable: Extract and Load Product Data to Staging
Event handler: OnPreExecute

Executable: Extract and Load Sales Data to Staging
Event handler: OnPreExecute

Executable: Extract and Load Cities Data to Staging
Event handler: OnPreExecute

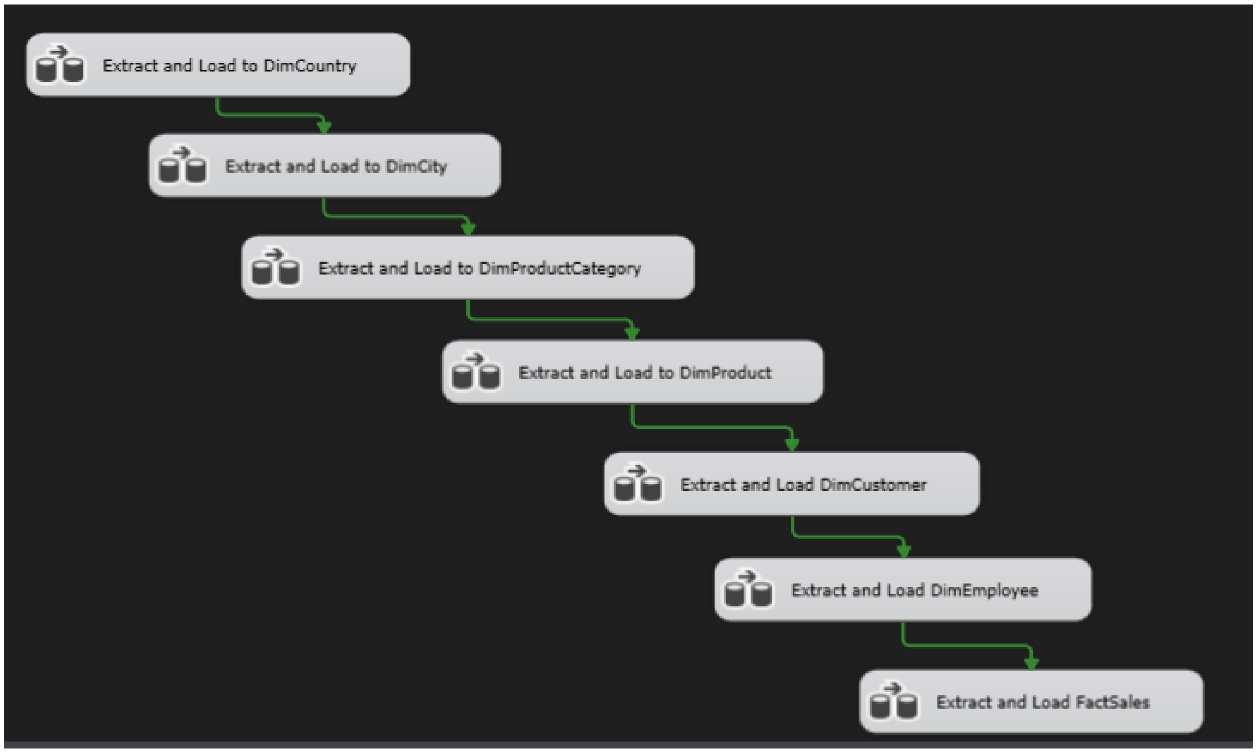
Executable: Extract and Load Countries Data to Staging
Event handler: OnPreExecute

Executable: Extract and Load Product Category to Staging
Event handler: OnPreExecute

•

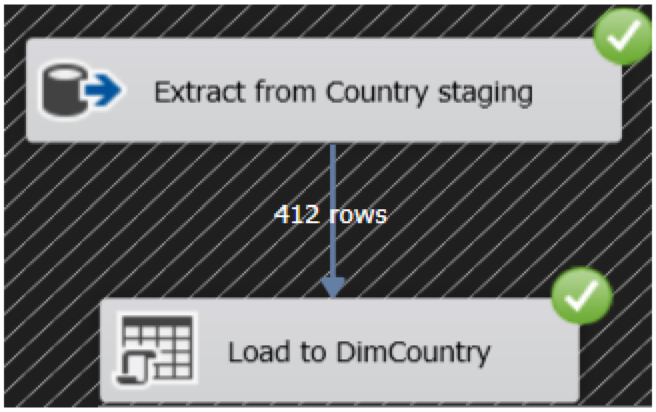
5.2 Transforming the Staged Data and Loading to the Data warehouse.

- Order of Execution


Extract and Load to DimCountry
Extract and Load to DimCity
Extract and Load to DimProductCategory
Extract and Load to DimProduct
Extract and Load DimCustomer
Extract and Load DimEmployee
Extract and Load FactSales

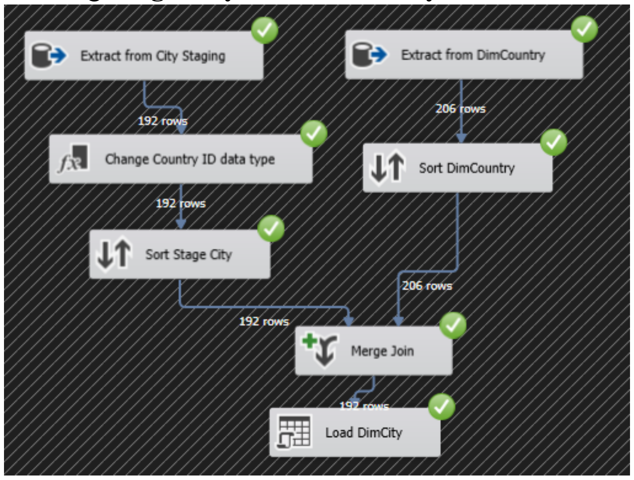
The staged data then undergoes some transformations before they get loaded into the data warehouse as shown below and the order of execution matters here since surrogate key from one-dimension may reference to another dimension table. Hence, those tables should be loaded first. Stored procedures have been used for dimension tables that history is not tracked to make sure only the new records or updated records will only be added to dimension table.

- Loading **Stage Country** data to **DimCountry.** No transformations were there to be made to country data.

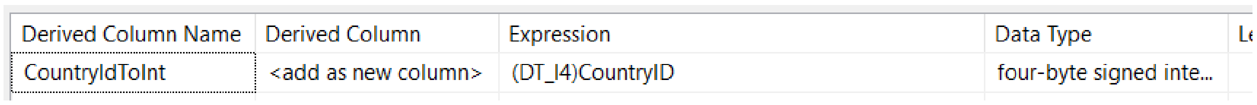


```
create procedure dbo.UpdateDimCountry
@CountryId int,
@CountryName nvarchar(50),
@CountryCode nvarchar(50)
as
begin
if not exists (select CountrySk
from dbo.DimCountry
where AlternateCountryID = @CountryId)
begin
insert into DimCountry (AlternateCountryID, CountryName, CountryCode, InsertDate,ModifiedDate)
values (@CountryId, @CountryName, @CountryCode, GETDATE(), GETDATE())
end;
if exists (select CountrySk
from dbo.DimCountry
where AlternateCountryID = @CountryId)
begin
update dbo.DimCountry
set CountryName = @CountryName,
CountryCode = @CountryCode,
ModifiedDate = GETDATE()
where AlternateCountryID = @CountryId and (CountryName != @CountryName or CountryCode != @CountryCode)
end;
end;
```

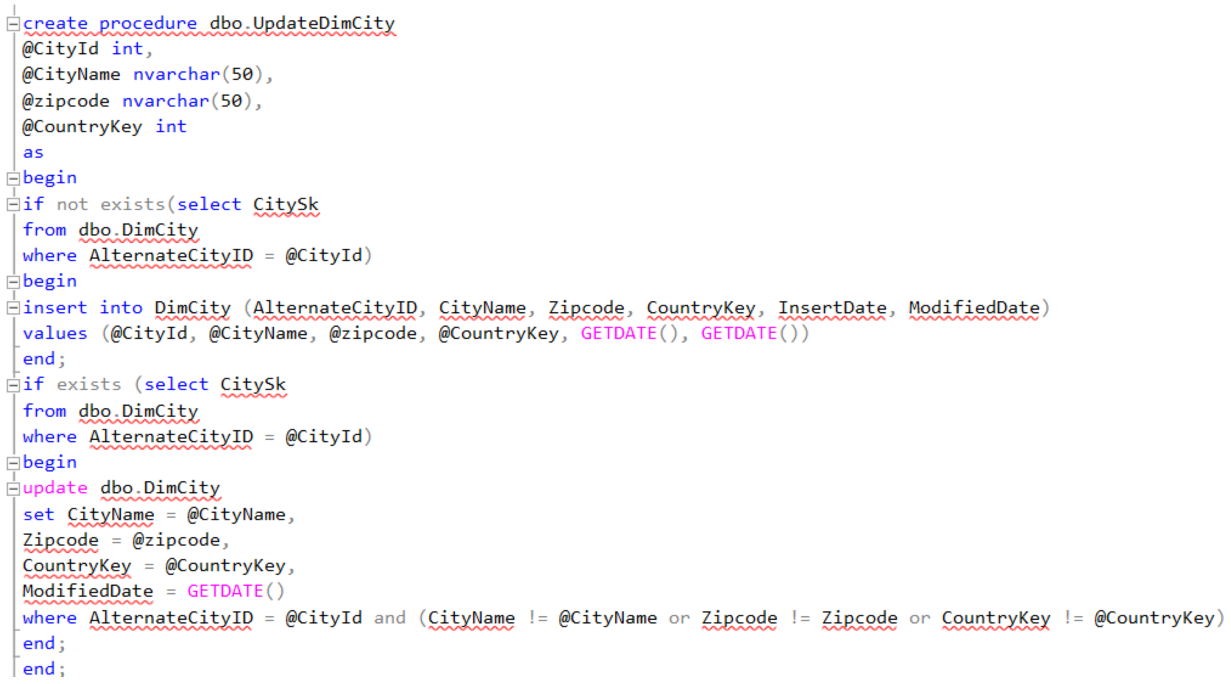
- Loading **Stage City** data to **DimCity**.



Two **sort** components have been used to sort City staging and DimCountry data by city id and get **proper country surrogate keys**.
CountryID data type had to be changed to DT_I4. This was done using a **derived column**.

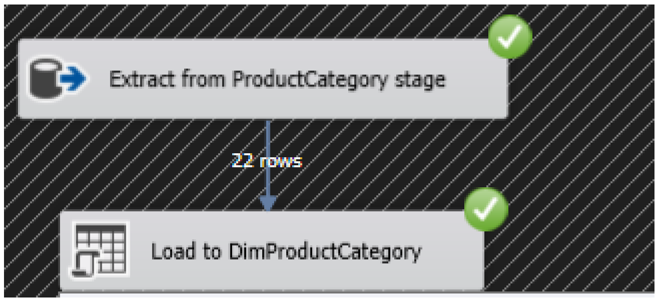


Derived Column Name	Derived Column	Expression	Data Type	Le
CountryIdToInt	<add as new column>	(DT_I4)CountryID	four-byte signed inte...	



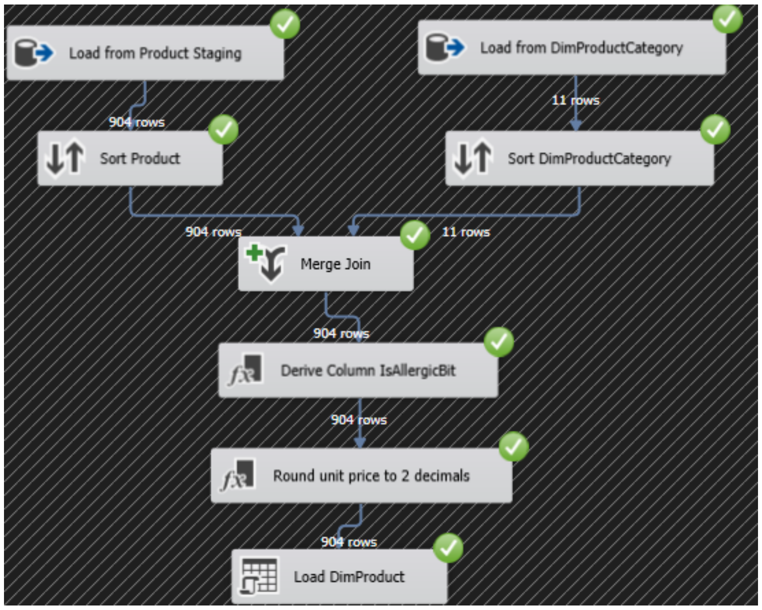
```
create procedure dbo.UpdateDimCity
@CityId int,
@CityName nvarchar(50),
@zipcode nvarchar(50),
@CountryKey int
as
begin
if not exists(select CitySk
from dbo.DimCity
where AlternateCityID = @CityId)
begin
insert into DimCity (AlternateCityID, CityName, Zipcode, CountryKey, InsertDate, ModifiedDate)
values (@CityId, @CityName, @zipcode, @CountryKey, GETDATE(), GETDATE())
end;
if exists (select CitySk
from dbo.DimCity
where AlternateCityID = @CityId)
begin
update dbo.DimCity
set CityName = @CityName,
Zipcode = @zipcode,
CountryKey = @CountryKey,
ModifiedDate = GETDATE()
where AlternateCityID = @CityId and (CityName != @CityName or Zipcode != Zipcode or CountryKey != @CountryKey)
end;
end;
```

- Load Stage Product Category data to DimProductCategory.


Extract from ProductCategory stage
22 rows
Load to DimProductCategory

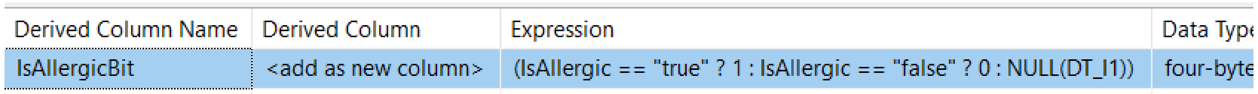
```
create procedure dbo.UpdateProdCategory
@CategoryId int,
@CategoryName nvarchar(50)
as
begin
if not exists(select ProductCategorySk
from dbo.DimProductCategory
where AlternateCategoryID = @CategoryId)
begin
insert into DimProductCategory (AlternateCategoryID, ProductCategoryName, InsertDate, ModifiedDate)
values (@CategoryId, @CategoryName, GETDATE(), GETDATE())
end;
if exists (select ProductCategorySk
from dbo.DimProductCategory
where AlternateCategoryID = @CategoryId)
begin
update dbo.DimProductCategory
set ProductCategoryName = @CategoryName,
ModifiedDate = GETDATE()
where AlternateCategoryID = @CategoryId and (ProductCategoryName != @CategoryName)
end;
end;
```

- Load **Stage Product** data to **DimProduct.**



Two sort components are used to sort Product Stage and DimProductCategory data using **CategoryID** and get the proper **CategorySk.**

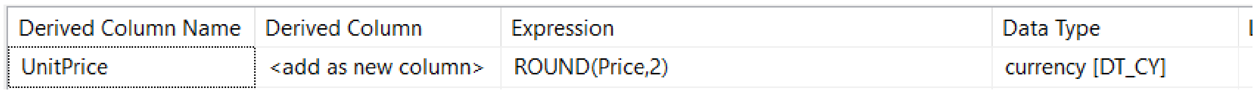
The Allergic column in product data was of type DT_WSTR. Therefore, using a **derived column**, I derived a new column **IsAllergicBit** which has 1 for True, 0 for False and NULL for unknown.



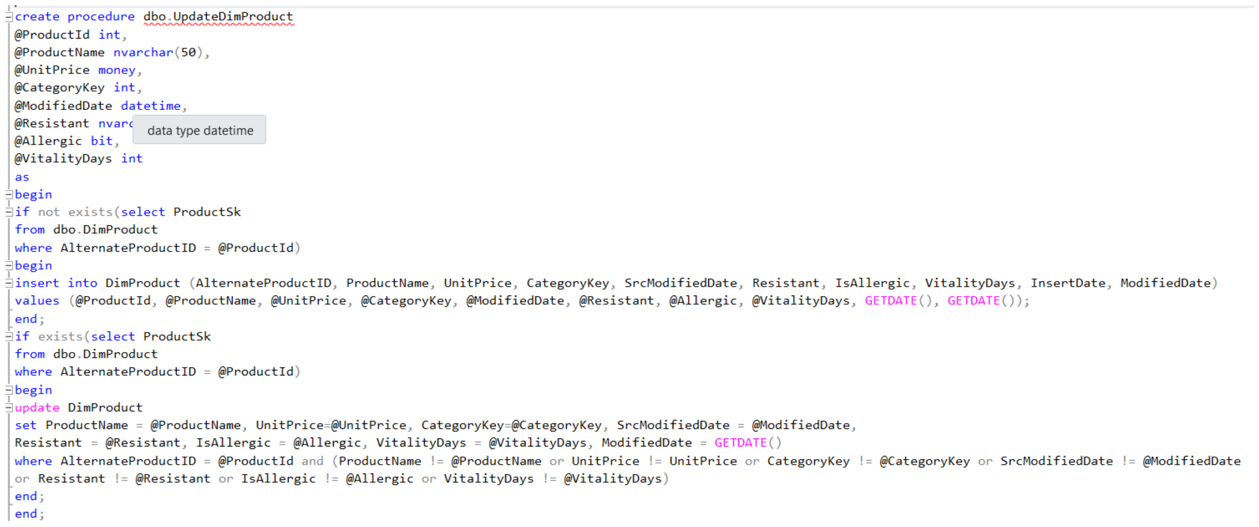
Derived Column Name	Derived Column	Expression	Data Type
IsAllergicBit	<add as new column>	(IsAllergic == "true" ? 1 : IsAllergic == "false" ? 0 : NULL(DT_I1))	four-byte

The **class column** in product data was **dropped** since it had no meaning.

The **unit price** of products was **rounded** to 2 decimal places.

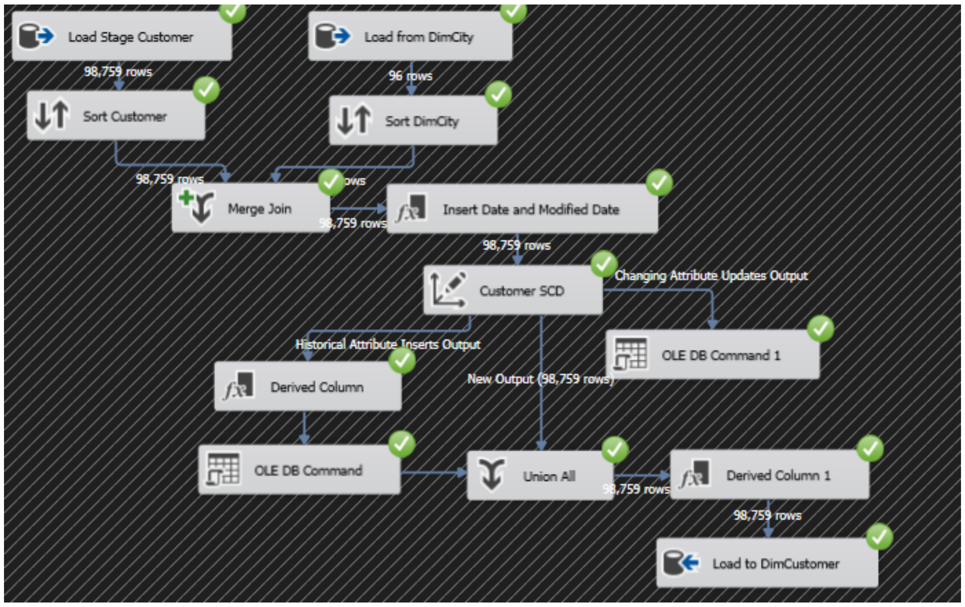


Derived Column Name	Derived Column	Expression	Data Type
UnitPrice	<add as new column>	ROUND(Price,2)	currency [DT_CY]

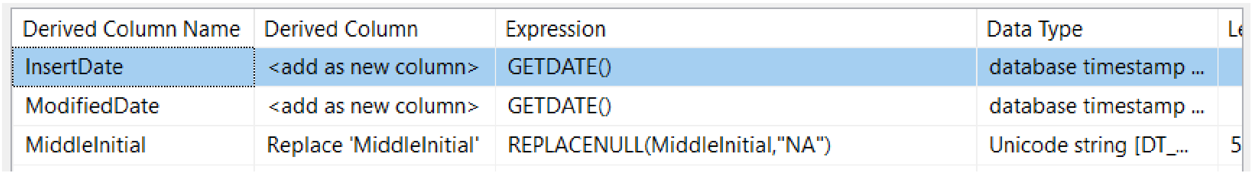


```
create procedure dbo.UpdateDimProduct
@ProductId int,
@ProductName nvarchar(50),
@UnitPrice money,
@CategoryKey int,
@ModifiedDate datetime,
@Resistant nvarc
@Allergic bit,
@VitalityDays int
as
begin
if not exists(select ProductSk
from dbo.DimProduct
where AlternateProductID = @ProductId)
begin
insert into DimProduct (AlternateProductID, ProductName, UnitPrice, CategoryKey, SrcModifiedDate, Resistant, IsAllergic, VitalityDays, InsertDate, ModifiedDate)
values (@ProductId, @ProductName, @UnitPrice, @CategoryKey, @ModifiedDate, @Resistant, @Allergic, @VitalityDays, GETDATE(), GETDATE());
end;
if exists(select ProductSk
from dbo.DimProduct
where AlternateProductID = @ProductId)
begin
update DimProduct
set ProductName = @ProductName, UnitPrice=@UnitPrice, CategoryKey=@CategoryKey, SrcModifiedDate = @ModifiedDate,
Resistant = @Resistant, IsAllergic = @Allergic, VitalityDays = @VitalityDays, ModifiedDate = GETDATE()
where AlternateProductID = @ProductId and (ProductName != @ProductName or UnitPrice != UnitPrice or CategoryKey != @CategoryKey or SrcModifiedDate != @ModifiedDate
or Resistant != @Resistant or IsAllergic != @Allergic or VitalityDays != @VitalityDays)
end;
end;
```

- Load **Stage Customer** data to **DimCustomer** as a slowly changing dimension.

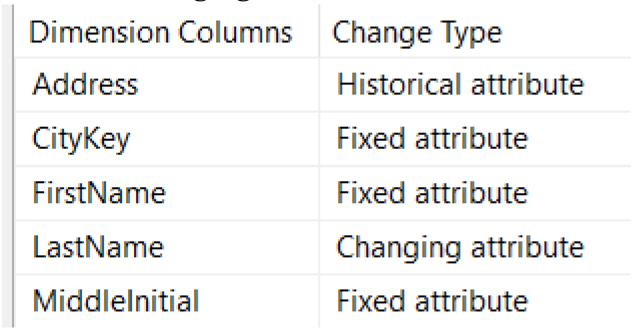


Data from stage customer and DimCity tables were sorted using **sort components** by cityid and merged to get the city surrogate key. Moreover, **Insert and Modified dates** columns were derived and **the null values** in the **MiddleInitial column** was also replaced by "NA" by using a **derived column**.



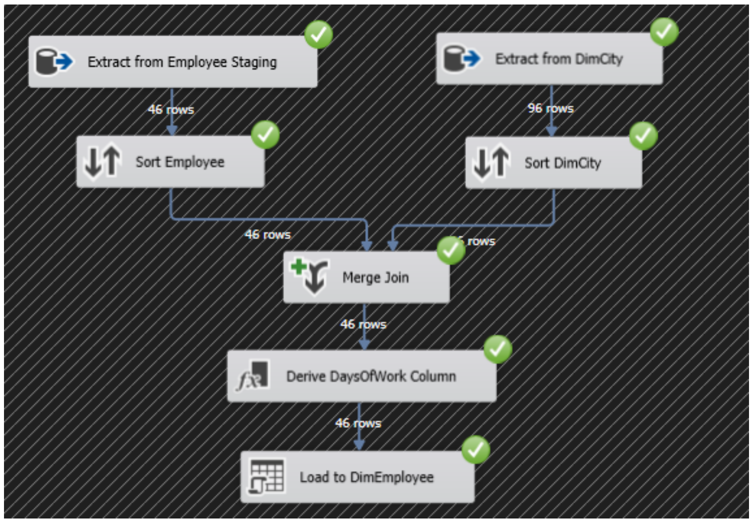
Derived Column Name	Derived Column	Expression	Data Type	Le
InsertDate	<add as new column>	GETDATE()	database timestamp ...	
ModifiedDate	<add as new column>	GETDATE()	database timestamp ...	
MiddleInitial	Replace 'MiddleInitial'	REPLACENULL(MiddleInitial,"NA")	Unicode string [DT_...	5

Next, **a SCD component** was used to implement slowly changing dimensions. **Customer ID** was selected as the **business key**. **Address** was used as a **historical attribute** and the **last name** was used as a **changing attribute**.

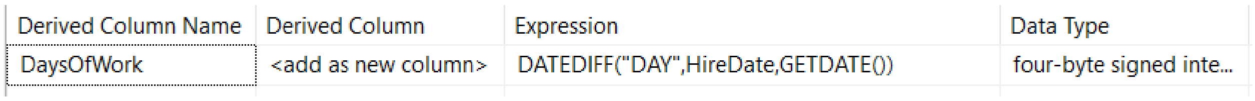


Dimension Columns	Change Type
Address	Historical attribute
CityKey	Fixed attribute
FirstName	Fixed attribute
LastName	Changing attribute
MiddleInitial	Fixed attribute

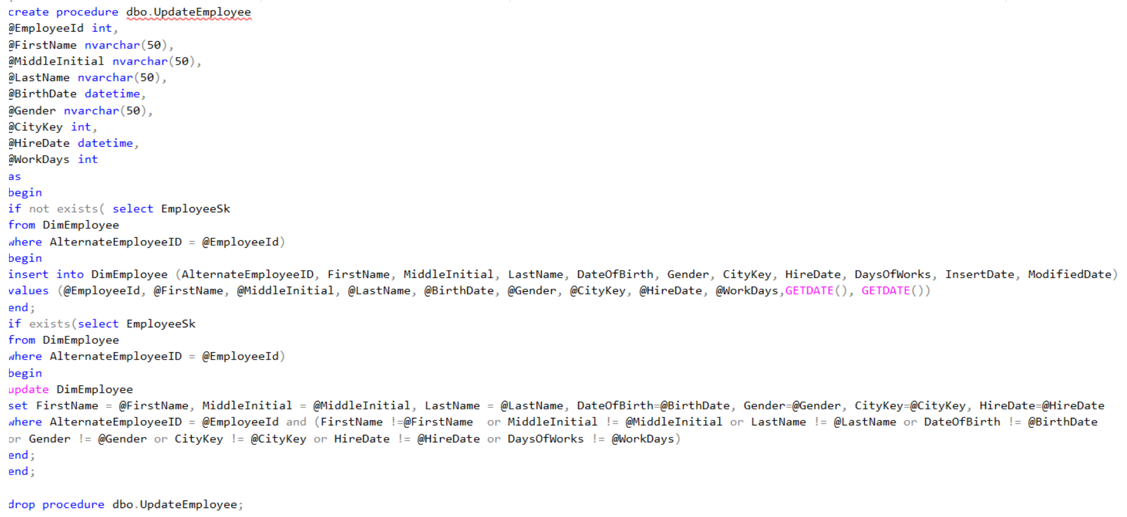
- Load **Stage Employee** data to **DimEmployee**.



Data from Stage Employee and DimCity tables are sorted using CityId and merged using a **merge join** to get City surrogate key. A **derived column** is used to derive **DaysOfWork** column.



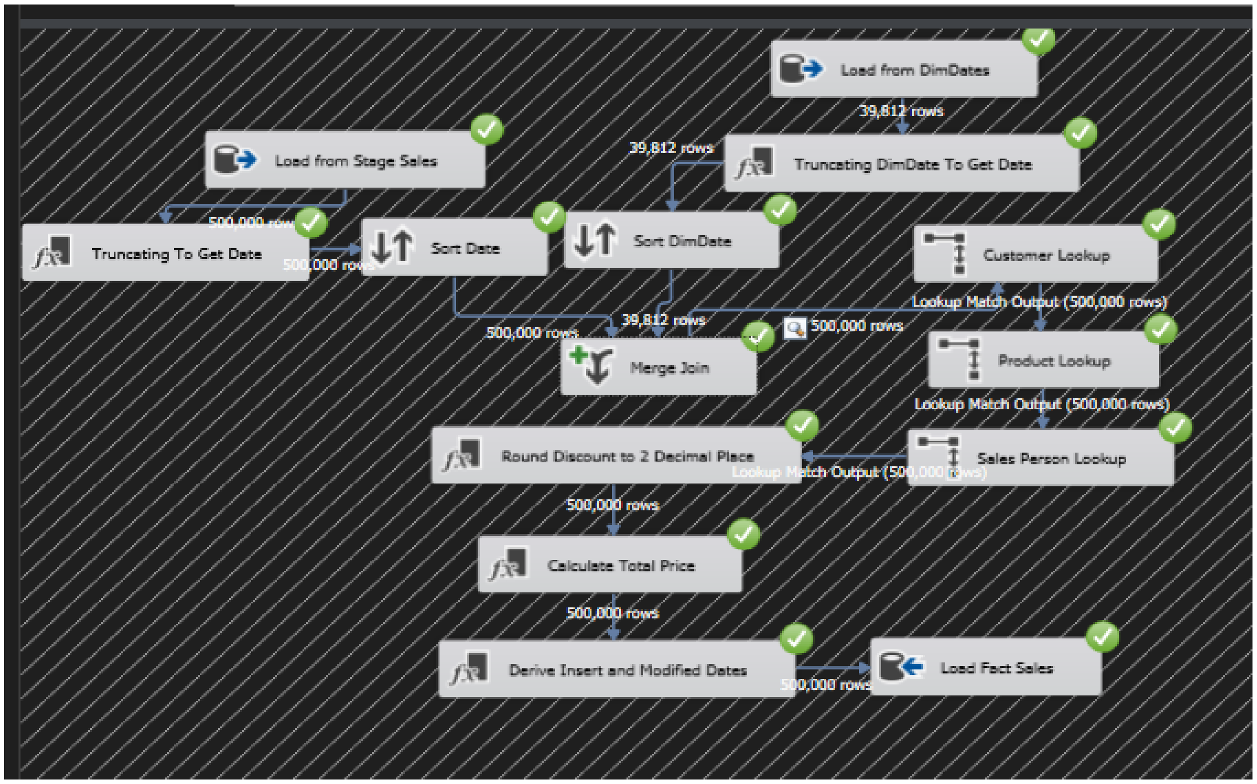
Derived Column Name	Derived Column	Expression	Data Type
DaysOfWork	<add as new column>	DATEDIFF("DAY",HireDate,GETDATE())	four-byte signed inte...



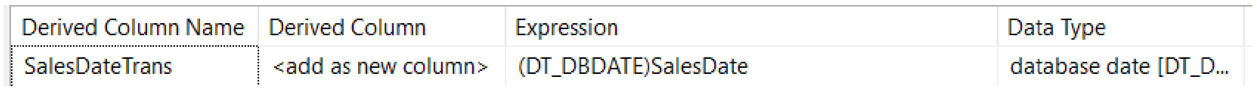
```
create procedure dbo.UpdateEmployee
@EmployeeId int,
@FirstName nvarchar(50),
@MiddleInitial nvarchar(50),
@LastName nvarchar(50),
@BirthDate datetime,
@Gender nvarchar(50),
@CityKey int,
@HireDate datetime,
@WorkDays int
as
begin
if not exists( select EmployeeSk
from DimEmployee
where AlternateEmployeeID = @EmployeeId)
begin
insert into DimEmployee (AlternateEmployeeID, FirstName, MiddleInitial, LastName, DateOfBirth, Gender, CityKey, HireDate, DaysOfWorks, InsertDate, ModifiedDate)
values (@EmployeeId, @FirstName, @MiddleInitial, @LastName, @BirthDate, @Gender, @CityKey, @HireDate, @WorkDays,GETDATE(), GETDATE())
end;
if exists(select EmployeeSk
from DimEmployee
where AlternateEmployeeID = @EmployeeId)
begin
update DimEmployee
set FirstName = @FirstName, MiddleInitial = @MiddleInitial, LastName = @LastName, DateOfBirth=@BirthDate, Gender=@Gender, CityKey=@CityKey, HireDate=@HireDate
where AlternateEmployeeID = @EmployeeId and (FirstName !=@FirstName  or MiddleInitial != @MiddleInitial or LastName != @LastName or DateOfBirth != @BirthDate
or Gender != @Gender or CityKey != @CityKey or HireDate != @HireDate or DaysOfWorks != @WorkDays)
end;
end;

drop procedure dbo.UpdateEmployee;
```

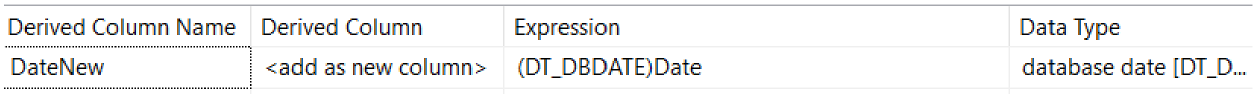
- Load **Stage Sales** data to **FactSales**.



Sales date from Stage Sales and Date from DimDate are **truncated** to get date by removing timestamps because timestamps of sales date do not match with timestamps of the date in DimDate, therefore cannot be merged.



Derived Column Name	Derived Column	Expression	Data Type
SalesDateTrans	<add as new column>	(DT_DBDATE)SalesDate	database date [DT_D...

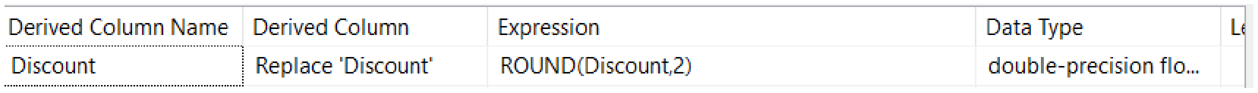


Derived Column Name	Derived Column	Expression	Data Type
DateNew	<add as new column>	(DT_DBDATE)Date	database date [DT_D...

Sort components with **merge** join has been used to get the datekey as the SalesDateKey from DimDate.

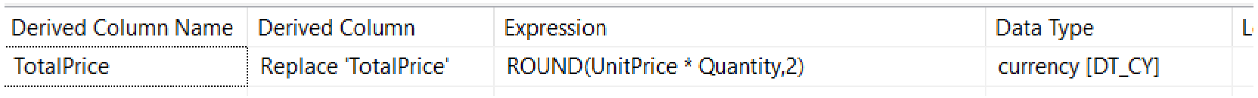
Lookups have been used to get ProductSk, CustomerSk, EmployeeSk from DimProduct, DimCustomer and DimEmployee tables.

Discounts are **rounded** to 2 decimal places.



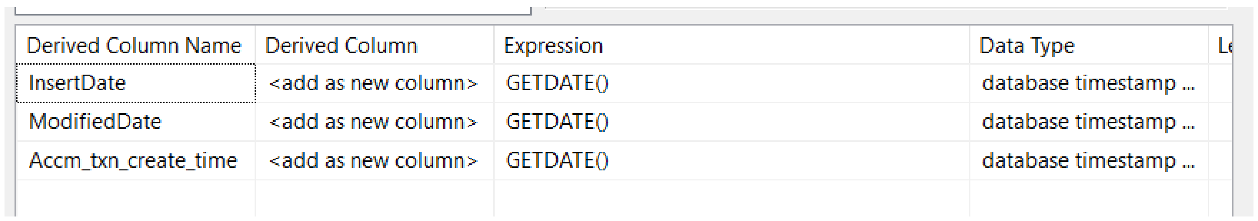
Derived Column Name	Derived Column	Expression	Data Type	Le
Discount	Replace 'Discount'	ROUND(Discount,2)	double-precision flo...	

In the source data, all the values in the **Total price column were 0.0.** Therefore, calculations were made when loading to Fact table to calculate values for the total price by multiplying quantity in stage sales table with unit price in the product table and rounded to 2 decimal places.



Derived Column Name	Derived Column	Expression	Data Type	L
TotalPrice	Replace 'TotalPrice'	ROUND(UnitPrice * Quantity,2)	currency [DT_CY]	

Finally**, Insert, Modified dates** and **Accm_txn_create_time** were created using derived columns.



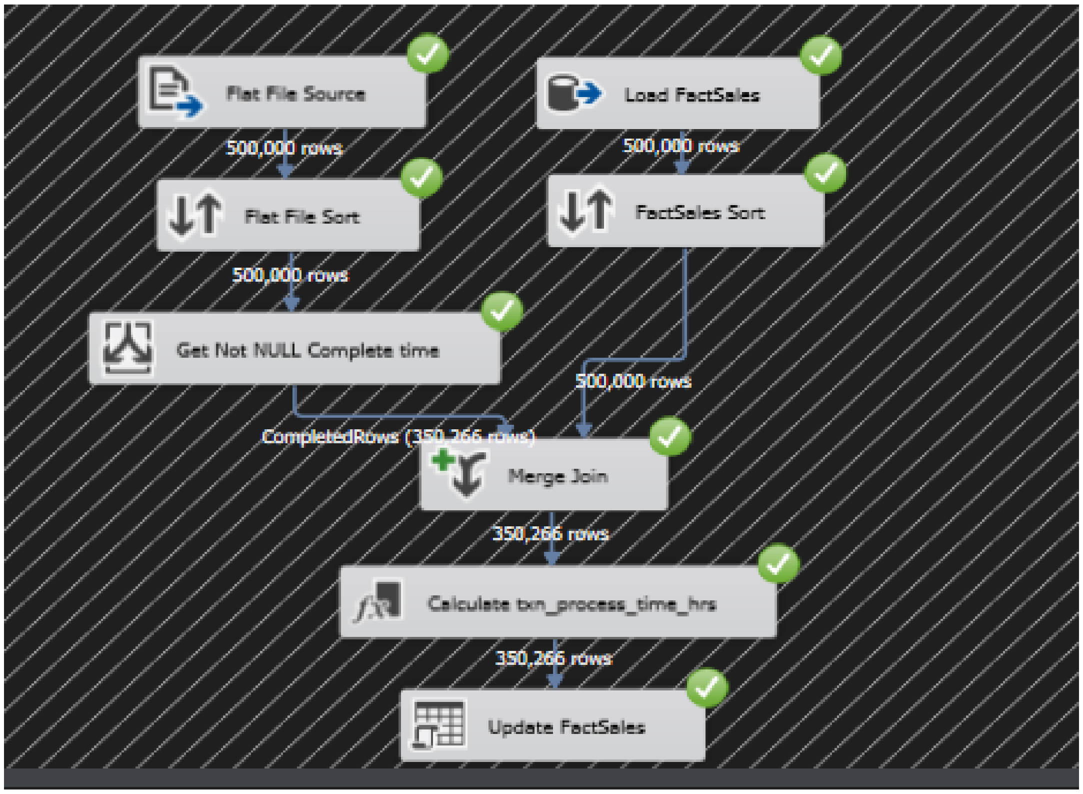
Derived Column Name	Derived Column	Expression	Data Type	Le
InsertDate	<add as new column>	GETDATE()	database timestamp ...	
ModifiedDate	<add as new column>	GETDATE()	database timestamp ...	
Accm_txn_create_time	<add as new column>	GETDATE()	database timestamp ...	

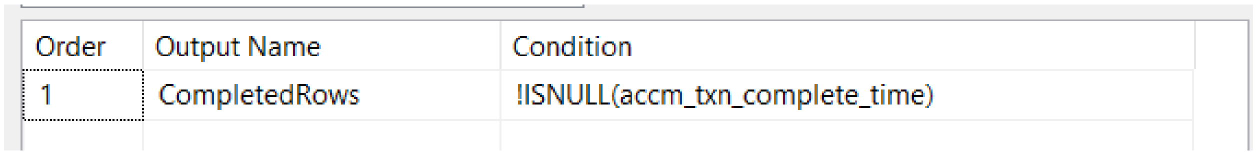
•

6 ETL Development – Accumulating Fact Tables

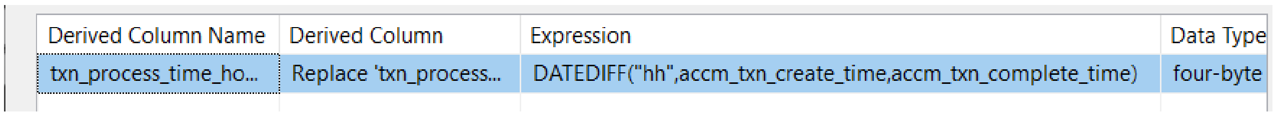
The process time in hours is calculated with the difference of create time and complete time. This task is performed in a separate package where a derived column calculates the process time and updates the **txn_process_time_hours** column, and the **accm_txn_complete_time** column data is loaded with a csv file which includes sales id with respective accm_txn_complete_time.

Since some transactions still might not have been completed, their complete time will be null. Therefore, I have used a **conditional split** to get rows only where **transaction complete time is not null**. Then using a **derived column**, the difference between complete time and create time was calculated and updated in the fact table.





Order	Output Name	Condition
1	CompletedRows	!ISNULL(accm_txn_complete_time)



Derived Column Name	Derived Column	Expression	Data Type
txn_process_time_ho...	Replace 'txn_process...	DATEDIFF("hh",accm_txn_create_time,accm_txn_complete_time)	four-byte