
Sri Lanka Institute of Information Technology

# Starbucks Offers Data Warehouse Solution

## Assignment-1 Document

IT3021 - Data Warehousing and Business Intelligence
Assignment 1

Submitted by: IT19069432 – Dissanayake D.M.I.M.

Date of submission: 11/05/2021

## Table of Contents

## 1. Dataset Selection

A Transactional Dataset about offer-based transactions and as well as regular transactions done by customers through various online platforms and the Starbucks reward mobile app provided by the Starbucks company was selected as the base dataset to build the Data Warehouse. The original source files can be found using the link provided below.

Source Link: https://www.kaggle.com/blacktile/starbucks-app-customer-reward-program-data

The dataset can be further described as a list of transactions and offer interactions done by customers using Starbucks online apps. A given customer in a list of 17000 pool of customers may either perform a transaction, view an offer, receive an offer, or else complete a received offer sent by the Starbucks. All these four types are taken as customer interactions that has a business value and are listed as under a common name called "Event". Ultimately, the events are a list of transactions that my belong to different types of transactions. Moreover, for offer-related events, offer type and the medium which was used to transmit the offer are also given with detailed offer data. Upon completing an offer, a customer gets a specific reward which is deducted from the difficulty of the offer. (difficulty is the full value of the product if the offer was not in place given in USD and the reward is the price deducted from the difficulty if the customer completes the offer.) All the events had recorded with relevant dates.

There were originally three source files provided in the JSON format, namely, "portfolio.json", "profile.json", and "transcript.json". These source files were carefully analyzed and confirmed that they contain enough records and columns that matches the requirements given in the assignment specification.

While analyzing the original files it was observed that the dataset contained transactions expanded across about closely two years. (dates were given as integers in a column called "time" in transcript.json file). Portfolio.json file had all the customer data. Each customer was given a unique 32 characters long string ID whereas the names of the customers were not disclosed due to confidentiality issues that might arise. Addresses of the customers are also missing in the original source files that were found, thus there was a lot of data enrichment to be done in the customer data file.

Since these files had enough records and columns to be divided into several source files and hierarchies such as [year, month, date], [offer, offer type, channel type] were identified, they were identified as a valid dataset and the data source preparation was started. The entities identified in the original source files and their relationships are shown in the following simplified ER diagram. A comprehensive detailed table where each source file is described further can be found in the section 2.2 of this report.

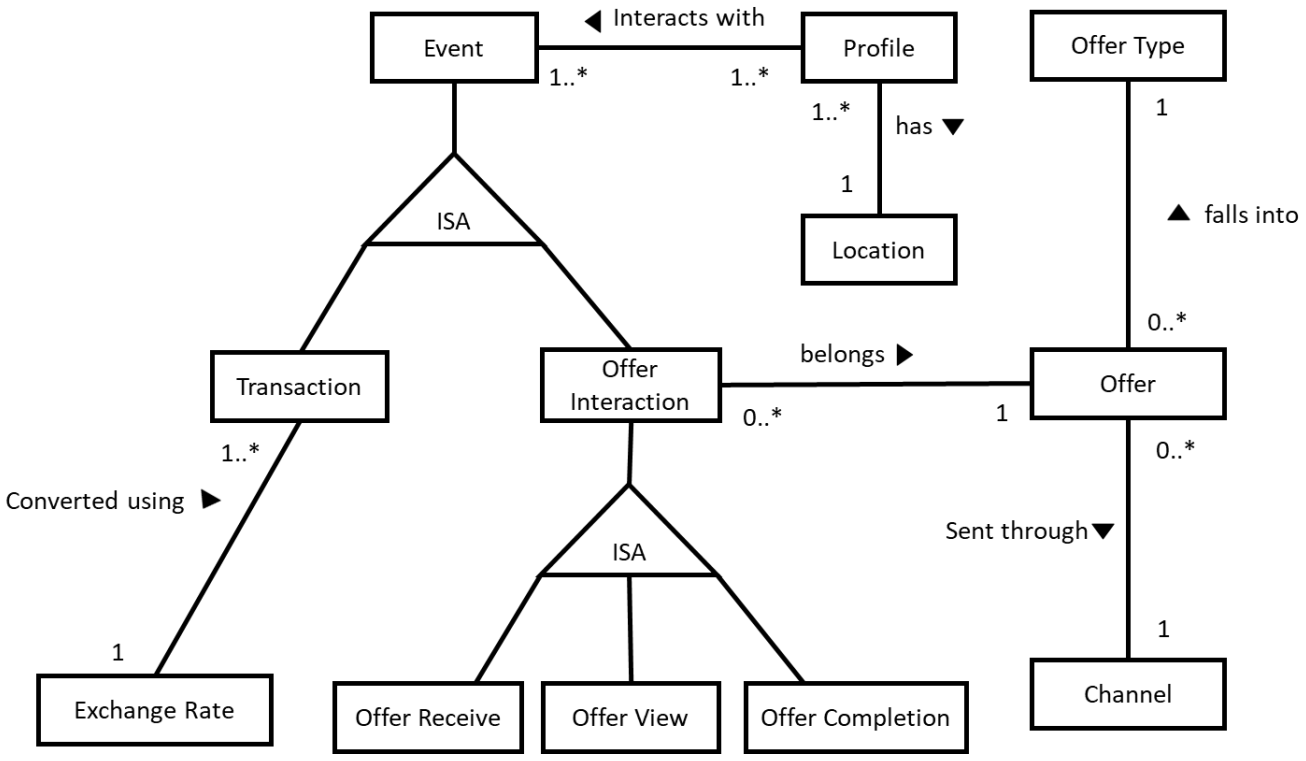



Figure 1. 1 A simplified EER diagram of the potential entities identified in the source files

As shown in the simplified EER diagram above, there are enough entities identified to create Dimension and Fact tables, Calculations to be performed and included as measurable columns in the fact table, potential Lookups that can be used to figure out foreign key like scenarios when connecting dimension tables and fact tables, and Sorting and merging possibilities when joining entities like profile (customer) and location (addresses).

Thus, it is ensured beforehand that there will be enough data for later, when creating the SSAS cubes and generating reports, based on the above source file data. Moreover, since there are possibilities to use Merge joins, Sorts, Lookups, and a lot of derived columns which will be explained the section 2.2 of the report in detail, have ensured to be able to implement a rich set of ETL tasks in both ETL processes from sources to staging and staging to the Data Warehouse.

## 2. Preparation of data sources

Three JSON source files introduced in the previous section were then divided into two database files, two text files and a CSV file to prepare a set of compatible source files with different file types to be imported as source files for the Data Warehouse solution.

The prepared source files out of the original json source files are as follows. Note that following descriptions are only brief introductions to each source file prepared. A Detailed analysis of the source files can be found in the Excel sheet given with the report which contains all the mapping related details of source tables and dimension and fact tables as well.

### 2.1 Prepared Source Files

#### 2.1.1. starbucks_offers MySQL database

This source file is a MySQL database which has two table, namely, “offer” and “offer_type”. Table offer contains distinct set of offers sent by Starbucks to its customers and additional data about the offers such as duration, difficulty and rewards offered. The offer id which was identified to be the business key of the offer table was a 32 characters long string which was unique for each offer. Offers and offer types were originally included in the portfolio.json file offered by Starbucks. The JSON file was imported into MySQL and then the offer and offer_type tables were created as two separate tables.

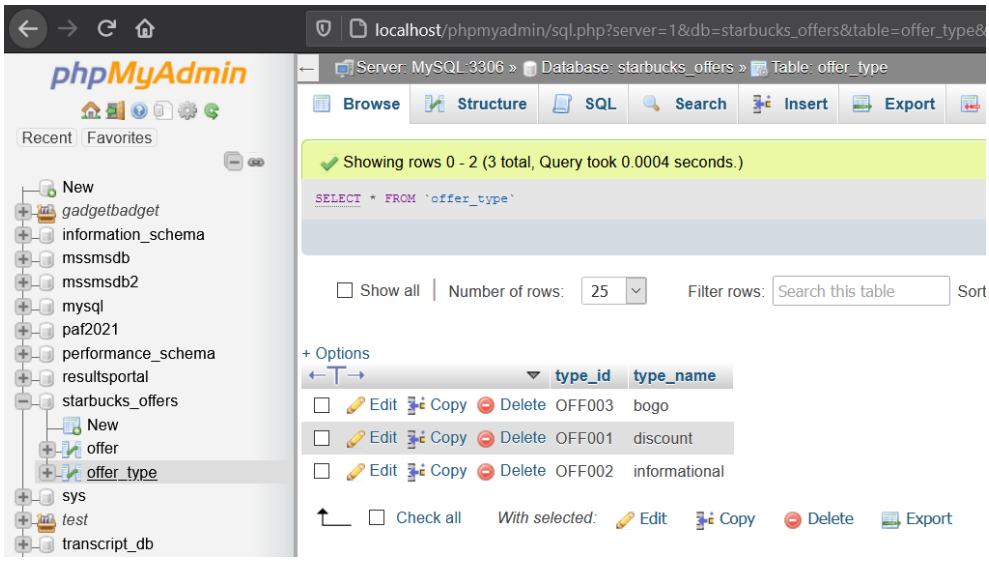



Figure 2.1.1. 1. offer_type source table in phpMyAdmin

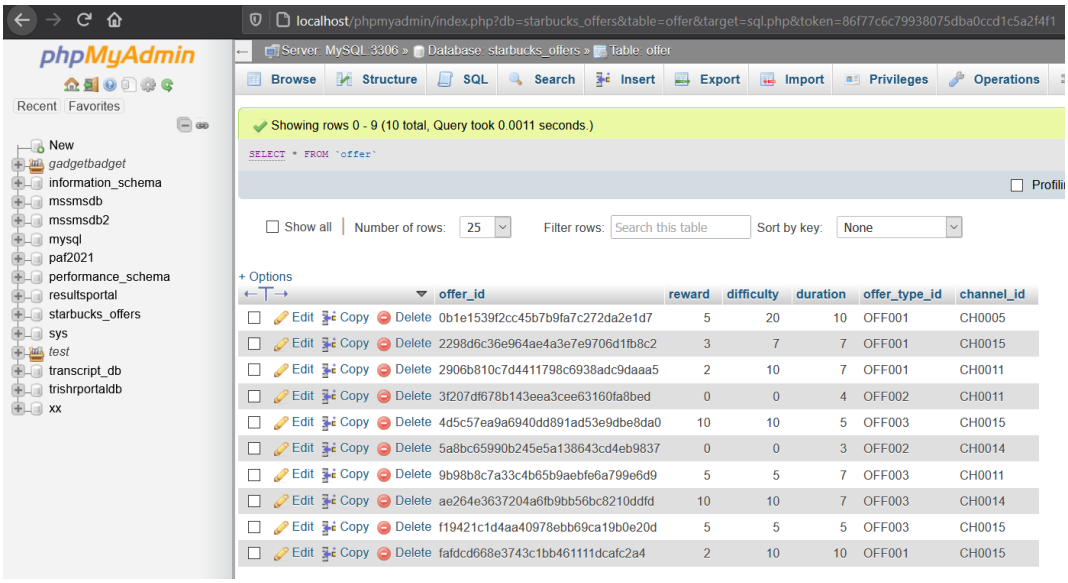



Figure 2.1.1. 2. offer source table in phpMyAdmin

### 2.1.2. Starbucks_Offers MSSQL database

Originally the portfolio.json file contained a list of channels that was used to send offers to the customers. After offer and offer_type was prepared as MySQL tables, the list of channels was taken out and all possible combinations that can occur was made from the given channel list for each offer. Then these combinations were given a unique channel id and a Microsoft SQL Database was created with only one table, which was namely, “channels”.

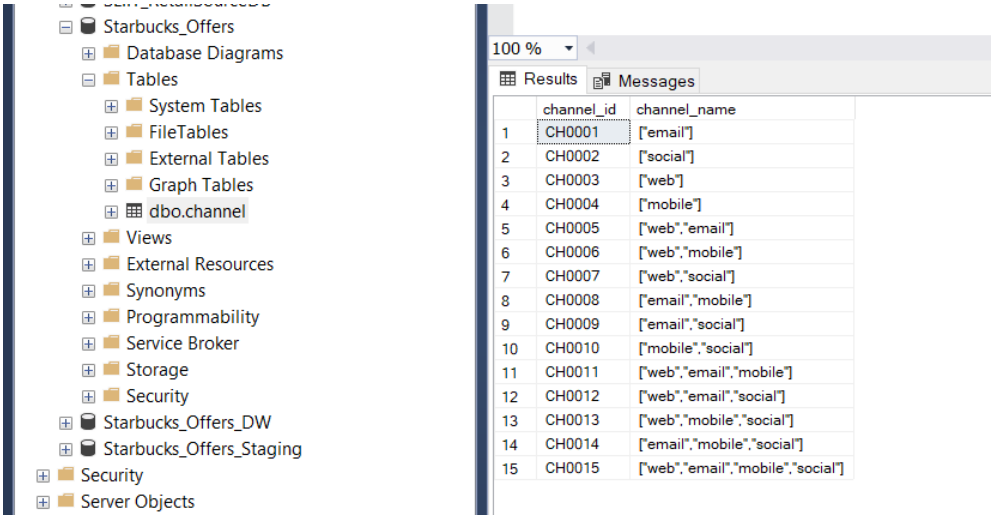



Figure 2.1.2. 1. channel table in Starbucks_Offers in SSMS

### 2.1.3. Profile.txt

Profile.txt is a flat file prepared using the data that were present in the profile.json file. This text file contains customer details without some attributes such as name and address as they were not included in the original source file as well. File contains tab delimited data across five columns, namely, person id, gender, age, date became a member and income of each customer. Had a total of **17000** unique customer details in the file. It was noticed that default values of gender and income were null/empty, while the default value of the age column was "118" which is not a real age depending on the nature of the records and the dataset itself. These values were required to be managed to enrich the customer dataset further.

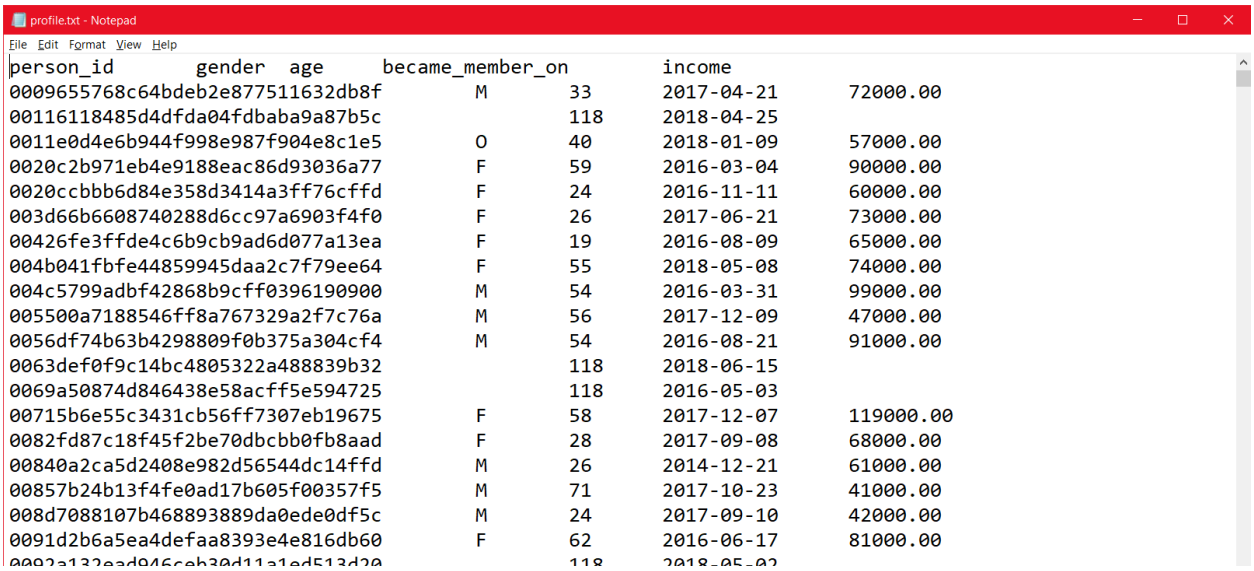
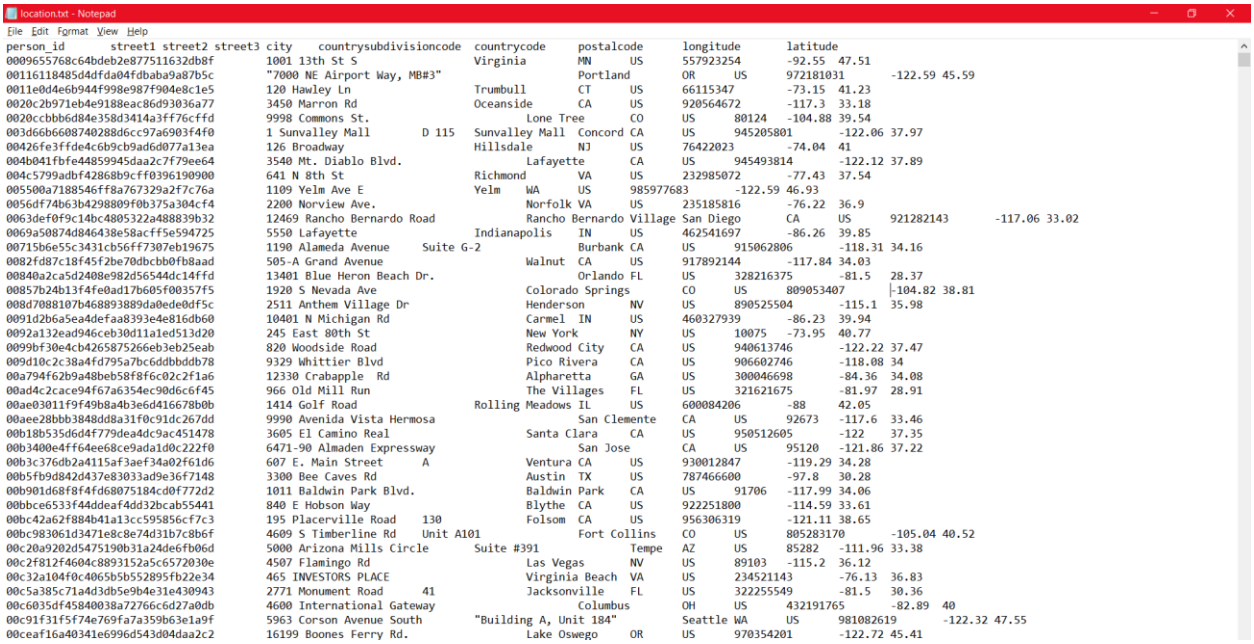



Figure 2.1.3. 1. Profile.txt file contents [Notepad]

### 2.1.4. Location.txt

Since the addresses were not given in the original json source files, a valid set of addresses were assigned to each uniquely identified customer and joined with the person id to have a valid address per each customer. Note that the list of addresses was separately downloaded but contains addresses of Starbucks customers based within United States of America. Location file contains 17000 unique addresses along with geo location of each address in longitudes and latitudes.



location.txt - Notepad

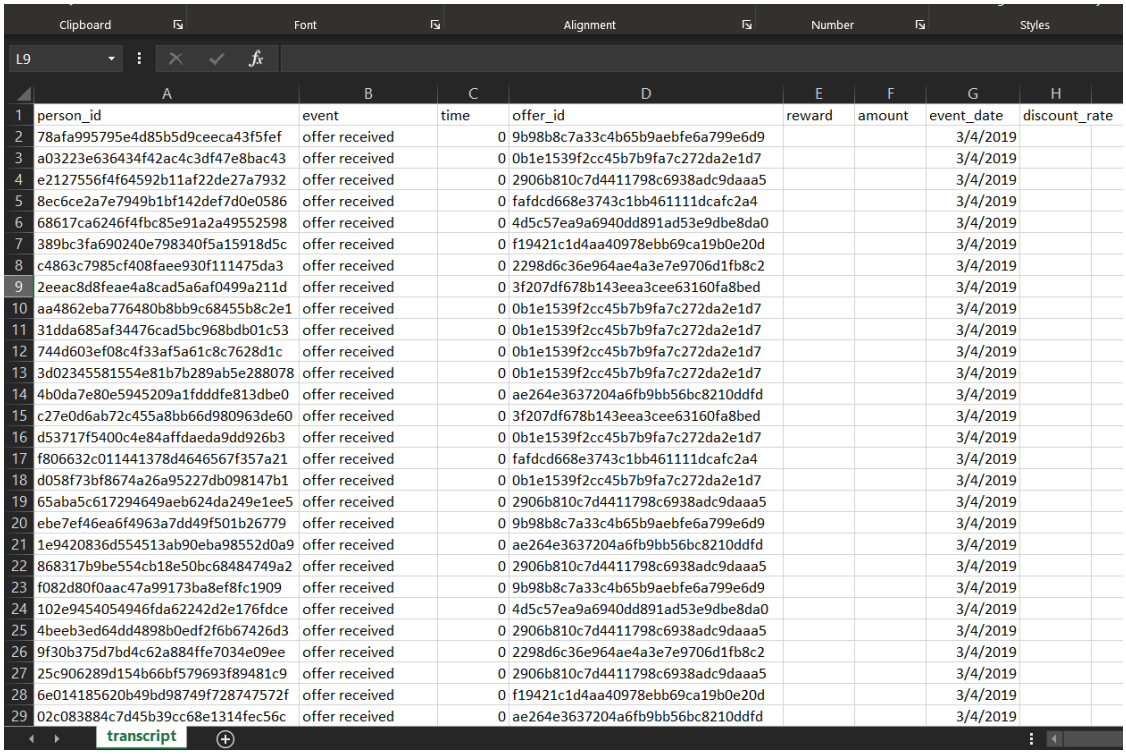
File Edit Format View Help

| person_id | street1 | street2 | street3 | city | countrysubdivisioncode | countrycode | postalcode | longitude | latitude |
|---|---|---|---|---|---|---|---|---|---|
| 0009655768c64bdeb2e877511632db8f | 1001 13th St S | | | Virginia | MN | US | 557923254 | -92.55 | 47.51 |
| 00116118485d4dfda04fdbaba9a87b5c | "7000 NE Airport Way, MB#3" | | | Portland | OR | US | 972181031 | -122.59 | 45.59 |
| 0011e0d4e6b944f998e987f904e8c1e5 | 120 Hawley Ln | | | Trumbull | CT | US | 66115347 | -73.15 | 41.23 |
| 0020c2b971eb4e9188eac86d93036a77 | 3450 Marron Rd | | | Oceanside | CA | US | 920564672 | -117.3 | 33.18 |
| 0020ccbbb6d84e358d3414a3ff76cffd | 9998 Commons St. | | | Lone Tree | CO | US | 80124 | -104.88 | 39.54 |
| 003d66b66087402880d6cc97a6903f4f0 | 1 Sunvalley Mall | D 115 | | Sunvalley Mall | Concord | CA | US | 945205801 | -122.06 | 37.97 |
| 00426fe3ffde4c6b9cb9ad6d077a13ea | 126 Broadway | | | Hillsdale | NJ | US | 76422023 | -74.04 | 41 |
| 004b041fbfe44859945daa2c7f79ee64 | 3540 Mt. Diablo Blvd. | | | Lafayette | CA | US | 945493814 | -122.12 | 37.89 |
| 004c5799adbf42868b9cff0396190900 | 641 N 8th St | | | Richmond | VA | US | 232985072 | -77.43 | 37.54 |
| 005500a7188546ff8a767329a2f7c76a | 1109 Yelm Ave E | | | Yelm | WA | US | 985977683 | -122.59 | 46.93 |
| 0056df74b63b4298809f0b375a304cf4 | 2200 Norview Ave. | | | Norfolk | VA | US | 235185816 | -76.22 | 36.9 |
| 0063def0f9c14bc4805322a488839b32 | 12469 Rancho Bernardo Road | | | Rancho Bernardo Village | San Diego | CA | US | 921282143 | -117.06 | 33.02 |
| 0069a50874d846438e58acff5e594725 | 5550 Lafayette | | | Indianapolis | IN | US | 462541697 | -86.26 | 39.85 | |
| 00715b6e55c3431cb56ff7307eb19675 | 1190 Alameda Avenue | Suite G-2 | | Burbank | CA | US | 915062806 | -118.31 | 34.16 | |
| 0082fd87c18f45f2be70dbcbb0fb8aad | 505-A Grand Avenue | | | Walnut | CA | US | 917892144 | -117.84 | 34.03 | |
| 00840a2ca5d2408e982d56544dc14ffd | 13401 Blue Heron Beach Dr. | | | Orlando | FL | US | 328216375 | -81.5 | 28.37 | |
| 00857b24b13f4fe0ad17b605f00357f5 | 1920 S Nevada Ave | | | Colorado Springs | CO | US | 809053407 | -104.82 | 38.81 | |
| 008d7088107b468893889da0ede0df5c | 2511 Anthem Village Dr | | | Henderson | NV | US | 890525504 | -115.1 | 35.98 | |
| 0091d2b6a5ea4defaa8393e4e816db60 | 10401 N Michigan Rd | | | Carmel | IN | US | 460327939 | -86.23 | 39.94 | |
| 0092a132ead946ceb30d11a1ed513d20 | 245 East 80th St | | | New York | NY | US | 10075 | -73.95 | 40.77 | |
| 0099bf30e4cb4265875266eb3eb25eab | 820 Woodside Road | | | Redwood City | CA | US | 940613746 | -122.22 | 37.47 | |
| 009d10c2c38a4fd795a7bc6ddbbddb78 | 9329 Whittier Blvd | | | Pico Rivera | CA | US | 906602746 | -118.08 | 34 | |
| 00a794f62b9a48beb58f8f6c02c2f1a6 | 12330 Crabapple Rd | | | Alpharetta | GA | US | 300046698 | -84.36 | 34.08 | |
| 00ad4c2cace94f67a6354ec90d6c6f45 | 966 Old Mill Run | | | The Villages | FL | US | 321621675 | -81.97 | 28.91 | |
| 00ae03011f9f49b8a4b3e6d416678b0b | 1414 Golf Road | | | Rolling Meadows | IL | US | 600084206 | -88 | 42.05 | |
| 00aee28bbb3848dd8a31f0c91dc267dd | 9990 Avenida Vista Hermosa | | | San Clemente | CA | US | 92673 | -117.6 | 33.46 | |
| 00b18b535d6d4f779dea4dc9ac451478 | 3605 El Camino Real | | | Santa Clara | CA | US | 950512605 | -122 | 37.35 | |
| 00b3400e4ff64ee68ce9ada1d0c222f0 | 6471-90 Almaden Expressway | | | San Jose | CA | US | 95120 | -121.86 | 37.22 | |
| 00b3c376db2a4115af3aef34a02f61d6 | 607 E. Main Street | A | | Ventura | CA | US | 930012847 | -119.29 | 34.28 | |
| 00b5fb9d842d437e83033ad9e36f7148 | 3300 Bee Caves Rd | | | Austin | TX | US | 787466600 | -97.8 | 30.28 | |
| 00b901d68f8f4fd68075184cd0f772d2 | 1011 Baldwin Park Blvd. | | | Baldwin Park | CA | US | 91706 | -117.99 | 34.06 | |
| 00bbce6533f44ddeaf4dd32bcab55441 | 840 E Hobson Way | | | Blythe | CA | US | 922251800 | -114.59 | 33.61 | |
| 00bc42a62f884b41a13cc595856cf7c3 | 195 Placerville Road | 130 | | Folsom | CA | US | 956306319 | -121.11 | 38.65 | |
| 00bc983061d3471e8c8e74d31b7c8b6f | 4609 S Timberline Rd | Unit A101 | | Fort Collins | CO | US | 805283170 | -105.04 | 40.52 | |
| 00c20a9202d5475190b31a24de6fb06d | 5000 Arizona Mills Circle | | Suite #391 | Tempe | AZ | US | 85282 | -111.96 | 33.38 | |
| 00c2f812f4604c8893152a5c6572030e | 4507 Flamingo Rd | | | Las Vegas | NV | US | 89103 | -115.2 | 36.12 | |
| 00c32a104f0c4065b5b552895fb22e34 | 465 INVESTORS PLACE | | | Virginia Beach | VA | US | 234521143 | -76.13 | 36.83 | |
| 00c5a385c71a4d3db5e9b4e31e430943 | 2771 Monument Road | 41 | | Jacksonville | FL | US | 322255549 | -81.5 | 30.36 | |
| 00c6035df45840038a72766c6d27a0db | 4600 International Gateway | | | Columbus | OH | US | 432191765 | -82.89 | 40 | |
| 00c91f31f5f74e769fa7a359b63e1a9f | 5963 Corson Avenue South | | "Building A, Unit 184" | Seattle | WA | US | 981082619 | -122.32 | 47.55 | |
| 00ceaf16a40341e6996d543d04daa2c2 | 16199 Boones Ferry Rd. | | | Lake Oswego | OR | US | 970354201 | -122.72 | 45.41 | |

Figure 2.1.4. 1. locations of each customer combined with their unique id [Notepad]

### 2.1.5. transcript.csv

Transcript CSV file is a comma delimited flat file which has the transaction details of the dataset. Transactions are called “Events” in this context and There can be several types of events. Customer interacting with an offer, customers receiving an offer, customer doing a transaction, all falls under Events. All events provided in the file are transaction/offer interaction events occurred during 2019-03-04 and 2021-02-20 which is almost two years’ worth transaction data. Since originally only the date was given in the format of number of hours since the first event occurred, “event date” column had to be manually populated based on the given value in the time column of the original source. Moreover, the number of columns must be filled had varied based on the type of the event. That means this is a single table mapped for a generalized relationship. (Event can be either transaction or offer interaction. Offer interactions can be of 3 sub types. All these were recorded in one single table.) Therefore, based on the specific event of a record, some of the columns are left as NULL, which means those columns do not belong to the event record. All relevant fields are filled without any NULL values.



| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | person_id | event | time | offer_id | reward | amount | event_date | discount_rate |
| 2 | 78afa995795e4d85b5d9ceeca43f5fef | offer received | 0 | 9b98b8c7a33c4b65b9aebfe6a799e6d9 | | | 3/4/2019 | |
| 3 | a03223e636434f42ac4c3df47e8bac43 | offer received | 0 | 0b1e1539f2cc45b7b9fa7c272da2e1d7 | | | 3/4/2019 | |
| 4 | e2127556f4f64592b11af22de27a7932 | offer received | 0 | 2906b810c7d4411798c6938adc9daaa5 | | | 3/4/2019 | |
| 5 | 8ec6ce2a7e7949b1bf142def7d0e0586 | offer received | 0 | fafdcd668e3743c1bb461111dcafc2a4 | | | 3/4/2019 | |
| 6 | 68617ca6246f4fbc85e91a2a49552598 | offer received | 0 | 4d5c57ea9a6940dd891ad53e9dbe8da0 | | | 3/4/2019 | |
| 7 | 389bc3fa690240e798340f5a15918d5c | offer received | 0 | f19421c1d4aa40978ebb69ca19b0e20d | | | 3/4/2019 | |
| 8 | c4863c7985cf408faee930f111475da3 | offer received | 0 | 2298d6c36e964ae4a3e7e9706d1fb8c2 | | | 3/4/2019 | |
| 9 | 2eeac8d8feae4a8cad5a6af0499a211d | offer received | 0 | 3f207df678b143eea3cee63160fa8bed | | | 3/4/2019 | |
| 10 | aa4862eba776480b8bb9c68455b8c2e1 | offer received | 0 | 0b1e1539f2cc45b7b9fa7c272da2e1d7 | | | 3/4/2019 | |
| 11 | 31dda685af34476cad5bc968bdb01c53 | offer received | 0 | 0b1e1539f2cc45b7b9fa7c272da2e1d7 | | | 3/4/2019 | |
| 12 | 744d603ef08c4f33af5a61c8c7628d1c | offer received | 0 | 0b1e1539f2cc45b7b9fa7c272da2e1d7 | | | 3/4/2019 | |
| 13 | 3d02345581554e81b7b289ab5e288078 | offer received | 0 | 0b1e1539f2cc45b7b9fa7c272da2e1d7 | | | 3/4/2019 | |
| 14 | 4b0da7e80e5945209a1fdddfe813dbe0 | offer received | 0 | ae264e3637204a6fb9bb56bc8210ddfd | | | 3/4/2019 | |
| 15 | c27e0d6ab72c455a8bb66d980963de60 | offer received | 0 | 3f207df678b143eea3cee63160fa8bed | | | 3/4/2019 | |
| 16 | d53717f5400c4e84affdaeda9dd926b3 | offer received | 0 | 0b1e1539f2cc45b7b9fa7c272da2e1d7 | | | 3/4/2019 | |
| 17 | f806632c011441378d4646567f357a21 | offer received | 0 | fafdcd668e3743c1bb461111dcafc2a4 | | | 3/4/2019 | |
| 18 | d058f73bf8674a26a95227db098147b1 | offer received | 0 | 0b1e1539f2cc45b7b9fa7c272da2e1d7 | | | 3/4/2019 | |
| 19 | 65aba5c617294649aeb624da249e1ee5 | offer received | 0 | 2906b810c7d4411798c6938adc9daaa5 | | | 3/4/2019 | |
| 20 | ebe7ef46ea6f4963a7dd49f501b26779 | offer received | 0 | 9b98b8c7a33c4b65b9aebfe6a799e6d9 | | | 3/4/2019 | |
| 21 | 1e9420836d554513ab90eba98552d0a9 | offer received | 0 | ae264e3637204a6fb9bb56bc8210ddfd | | | 3/4/2019 | |
| 22 | 868317b9be554cb18e50bc68484749a2 | offer received | 0 | 2906b810c7d4411798c6938adc9daaa5 | | | 3/4/2019 | |
| 23 | f082d80f0aac47a99173ba8ef8fc1909 | offer received | 0 | 9b98b8c7a33c4b65b9aebfe6a799e6d9 | | | 3/4/2019 | |
| 24 | 102e9454054946fda62242d2e176fdce | offer received | 0 | 4d5c57ea9a6940dd891ad53e9dbe8da0 | | | 3/4/2019 | |
| 25 | 4beeb3ed64dd4898b0edf2f6b67426d3 | offer received | 0 | 2906b810c7d4411798c6938adc9daaa5 | | | 3/4/2019 | |
| 26 | 9f30b375d7bd4c62a884ffe7034e09ee | offer received | 0 | 2298d6c36e964ae4a3e7e9706d1fb8c2 | | | 3/4/2019 | |
| 27 | 25c906289d154b66bf579693f89481c9 | offer received | 0 | 2906b810c7d4411798c6938adc9daaa5 | | | 3/4/2019 | |
| 28 | 6e014185620b49bd98749f728747572f | offer received | 0 | f19421c1d4aa40978ebb69ca19b0e20d | | | 3/4/2019 | |
| 29 | 02c083884c7d45b39cc68e1314fec56c | offer received | 0 | ae264e3637204a6fb9bb56bc8210ddfd | | | 3/4/2019 | |

Figure 2.1.5. 1. transcript.csv [ Excel]

### 2.1.6. Exchange Rates API

Other than the original sources, An API was used as a source to capture the current exchange rate from USD to LKR and LKR to USD, daily. An API call must be made to get the response with the relevant exchange rates as a JSON object. The API requires an API key to verify the device, thus a non-expiring API key was requested and attached to the URL. API has a free version which allows 100 API calls per hour along with limited to 2 Currency Types which satisfies the requirements of the Data Warehouse, thus was selected as a valid source.[4] The format of the API call is given below.

USD to LKR conversion API call:
https://free.currconv.com/api/v7/convert?q=LKR_USD&compact=ultra&apiKey=960591cbee556d44e235

LKR to USD conversion API call:
https://free.currconv.com/api/v7/convert?q=USD_LKR&compact=ultra&apiKey=960591cbee556d44e235

API Response Format [4]:

```
{"LKR_USD":0.005065}
{"USD_LKR":198.485851}
```

## 2.2. Detailed Description of Source Files

| Source Name | Source Type | Object Name (If Any) | Scheme | Object Type | Description |
|---|---|---|---|---|---|
| starbucks_offers | MySQL Database | offer | InnoDB | MySQL Database Relation (Table) | Includes all offer details offered by Starbucks Rewards currently |
| | | offer_type | InnoDB | MySQL Database Relation (Table) | Includes all offer_type details offerred by Starbucks Rewards |
| | | | | | |
| starbucks_offers | Microsoft SQL Database | channel | dbo | Microsoft SQL Database Relation (Table) | Includes all the combinations of channels used to |
| | | | | | |
| profile.txt | Text File | profile | flat file | Tab delimited .TXT file | Includes Tab-separated customer information of Starbucks |
| | | | | | |
| location.csv | CSV File | location | flat file | Comma-delimited .CSV file | Includes all location details of the above customers |
| | | | | | |
| transcript.csv | CSV File | transcript | flat file | Comma-delimited .CSV file | Includes all offer/transaction related details customers have involved with |
| | | | | | |
| https://free.currconv.com/api/v7 | API | API Response | http | .JSON outputs | returns up-to-date currency exchange rates upon request. Requires a private non-expiring API key. Response is in the format of a JSON object. This process has to be done daily as the source only returns current exchanging rate only and no historical values can be retrieved) Although this has been taken as a source, It can be directly inserted/updated to the Database warehouse without loading to the staging layer as well. (Historical values are randomely generated and loaded for the purpose of doing calculations) |
| | | | | | |
| | | | | | |
| * Initial Source files were only in CSV and JSON formats and they were carefully divided into several file types and databases in order to prepare various types of source files indicated above. JSON files were first imported using MSSQL Server and converted into respective types later by exporting the imported tables. | | | | | |
| | | | | | |
| * There are no files generated by the given API above. The response is taken during the staging/ DW ETL processes accordingly. | | | | | |

Figure 2.2.1. Source File Descriptions [Excel]

| Source | Source Type | Table Name | Column Name | Data Type | Linking Table | Linking Column | Description |
|---|---|---|---|---|---|---|---|
| starbucks_offers | MySQL Database | offer | offer_id | nvarchar(50) | | | Unique Business ID |
| | | | reward | tinyint | | | Maximum reward can be received by a customer as a USD value |
| | | | difficulty | tinyint | | | Total Amount have to pay if offer is absent (in USD) |
| | | | duration | tinyint | | | time span in days that the given offer will last after receiving |
| | | | offer_type_id | nvarchar(6) | offer_type | type_id | offer type id of the corresponsing type of the offer |
| | | | channel_id | nvarchar(6) | channel | channel_id | channel id of the combination of channels used to send a particular offer |
| | | | | | | | |
| | | offer_type | type_id | nvarchar(6) | | | Unique Business ID |
| | | | type_name | nvarchar(15) | | | name of the offer type (discount, bogo, or informational) |
| | | | | | | | |
| starbucks_offers | Microsoft SQL Database | channel | channel_id | nvarchar(6) | | | Unique Business ID |
| | | | channel_name | nvarchar(50) | | | channel or set of channels which the offer will sent through to starbucks customers Channels are presented as a list (python list style). Channel combinations are unique. (possible channels are: web, email, social, and |
| | | | | | | | |
| profile.txt | Text File | profile | person_id | nvarchar(50) | | | Unique Business ID given to each customer of starbucks |
| | | | gender | nvarchar(2) | | | gender of the customer (M - Male , F - Female, or, O - Other). Null values are present as a default value. Needs to be properly cleaned beforehand. |
| | | | age | tinyint | | | age of the customer (either in a valid range or 118 has been given as the default value where age is unknown. Needs cleaning if possible.) |
| | | | became_member_on | datetime | | | date that the customer officially registered at starbucks online |
| | | | income | money | | | income of the customer. Default value is set to NULL where income is |
| | | | | | | | |
| location.csv | CSV File | location | person_id | nvarchar(50) | profile | person_id | Unique Business ID reffering the cutomer in the profile table |
| | | | street1 | nvarchar(60) | | | Firs line of the customer's address |
| | | | street2 | nvarchar(60) | | | Second line of the customer's address |
| | | | street3 | nvarchar(60) | | | Third line of the customer's address |
| | | | city | nvarchar(50) | | | customer's city |
| | | | countrysubdivisioncode | nvarchar(4) | | | state/province code (Two letter code) |
| | | | countrycode | nvarchar(4) | | | country code (all given addresses are US based) |
| | | | postalcode | nvarchar(50) | | | actual postal code |
| | | | longitude | nvarchar(50) | | | longitude of the exact position that the adress refers to |
| | | | latitude | nvarchar(50) | | | lattitude of the exact position that the adress refers to |
| | | | | | | | |
| transcript.csv | CSV File | transcript | person_id | nvarchar(50) | profile | person_id | Unique Business ID reffering the cutomer in the profile table |
| | | | event | nvarchar(50) | | | Transactional event occurred (can be either a offer-related task or an actual transaction done by the customer. Based on the above types, amount, offer_id, and reward columns may be set to NULL to indicate the relationship |
| | | | time | tinyint | | | original time given in number of days since the transaction recoding started by the starbucks. This has been used to create an exact date when source table preparation and will be dropped and event_date will be taken as the time measure. (NOT an ETL transformation. Its rather done in staging layer) |
| | | | offer_id | nvarchar(50) | offer | offer_id | offer id of the corresponsing offer customer has interacted with. Can be NULL in case the event is a transaction |
| | | | reward | tinyint | | | reward received by a customer as a USD value |
| | | | amount | money | | | amount as a USD value that involves with a customer transaction. Can be NULL in case the event is not a transaction. |
| | | | event_date | datetime | | | event date is the date time the transaction/offer interactions have been |
| | | | discount_rate | decimal(3,2) | | | dicount rate as a percentage, given for transactions made. ([>=2% if amount<=5 USD] or [>=5% if amount>5 USD]) |
| | | | | | | | |
| https://free.currconv.com/api/v7 | API | exchange_rate | source_currency | nvarchar(3) | | | Unique Partial Business ID, Used to identify the exchange rates uniquely. This is a three-letter universal code given to each currency type. (Ex: USD) |
| | | | destination_currency | nvarchar(3) | | | Unique Partial Business ID, Used to identify the exchange rates uniquely. This is a three-letter universal code given to each currency type. (Ex: LKR) |
| | | | exchange_rate | decimal(18,6) | | | daily exchange rate (USD to LKR or LKR to USD rate as a decimal) |
| | | | retrieved_date | date | | | Unique Partial Business ID, , Used to identify the exchange rates uniquely. This is the exact date of the exchange rate retreival. |

Figure 2.2.2. Detailed source file column descriptions

Note that above table can be also found either in the Starbucks ETL Mapping document placed in the documents folder of the submission or in the visual studio solution as an attachment (in the solution explorer).

Types of Files available in the Prepared source files: (6 Sources in Total)

- 1 MySQL and 1 Microsoft SQL Databases
- 2 Text Files
- 1 CSV File
- 1 API

## 3. Solution architecture

The Data Warehouse solution architecture can be shown as in the following figure 3.1. Some components in the Core Storage Layer and the whole BI-Layer are not drawn since they have not been implemented in this solution.

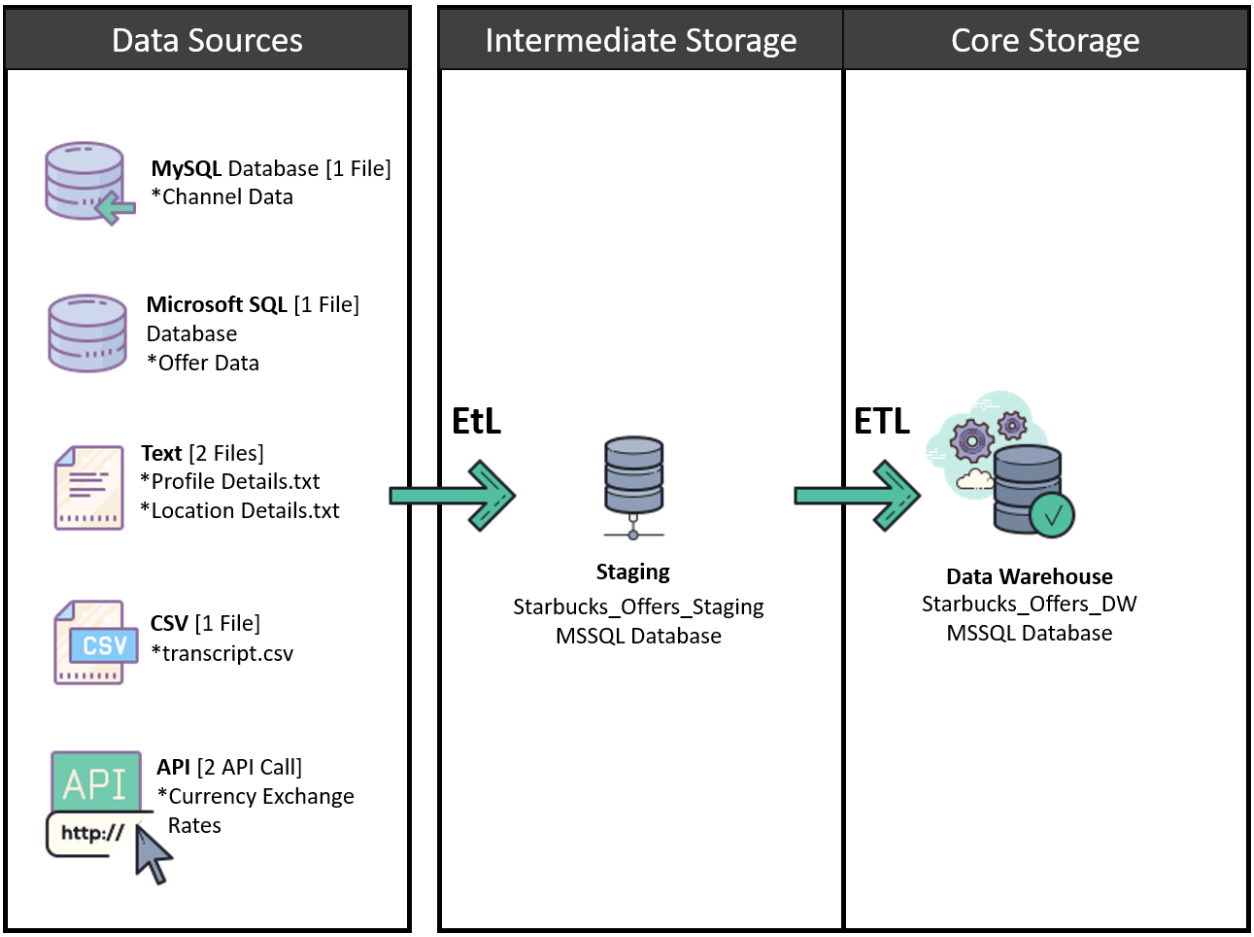



Figure 3. 1. The high-level architecture of the Starbucks Offers Data Warehouse Solution. OLAP Servers, Data Marts, SSAS Cubes, and the BI-Layer is not included.

### Data Sources

Data Sources column of the figure 3.1 contains all the source files prepared previously. From the source files, business data are taken into the staging layer through the first **EtL** process which has a smaller number of transformations which mainly focuses on populating the staging layer tables in the staging database. The exact steps taken in the EtL process has been included in the section 5, “ETL Development”.

Various source data taken from the source files/locations will be taken into the staging layer (Intermediate storage) and saved in the staging database's tables. Data enrichment took place here for columns with much higher rates of containing NULL values which were meaningless. Gender column of the profile table was having almost 2000 records as NULLs and they were assigned "NA" during the first EtL process as a technique of data enrichment. However, other than that, no major transformations were done during the first EtL process while extracting and loading data from source files to the staging database.

**Intermediate Storage (Staging Database only) and EtL**

Staging database in the Intermediate storage layer, as its name suggests, acts as an intermediate storage between the Data warehouse and the source files. The purpose of having a stage layer was to leave complex transformations during the first EtL process. The reason is that the EtL from source files/locations is preferred to be faster since there can be other delays such as network traffic, thus performing complex transformations during the extraction from the sources might slow down the whole process which will ultimately result in spending more time than required in that phase, which means the OLTP systems will be busy for more time than they really must. The solution for this was to divide the ETL process into two ETL processes with the staging database in the middle, which will speed up the first EtL process due to less transformations done during the process execution.

Note that the staging database tables are almost identical to the source tables which the data were originally taken from. The data types might have been slightly altered to get rid of data truncation while loading source data. These tables will get fully truncated (or deleted) which will flush out all existing data before the next EtL from sources to staging to only retain up-to-date data in the staging tables. The process has been explained in detail in the section 5, "ETL Development". The complete list of staging database tables is shown in the figure 2.2.2. attached in section 2.2. Note that the tables are not created beforehand and will get automatically created during the EtL process.

**Core Storage (Data Warehouse only) and ETL**

In the core storage layer, there is the Data Warehouse, and it contains the pre-created Dimension and Fact Tables. In this context, there is only one fact table present. The dimension tables have a special type of auto incrementing unique integer key which is known as the surrogate key to identify table records uniquely, while this key is also used to connect dimension tables and fact table and in the case of a snowflake scheme, the dimension tables and normalized dimension tables will also be connected using these surrogate keys instead of using the default business key(primary key of the source tables). In the case of Starbucks Offers Data Warehouse, it follows a snowflake scheme and thus, contains normalized dimension tables.

The data loaded into the staging database are taken into the Data Warehouse using the **ETL** process which is the second ETL process shown in the figure 3.1 from the left. The second ETL process has a set of complex Transformations and may take more time to finish the execution than the first EtL process. All the Derived column addition, data enrichments, data cleaning, table merging,

sorting and many other transformations are done in the second ETL process and are explained in detail in section 5, “ETL Development”. A list of tables and their columns in the data warehouse solution is given below and are explained in detail.

| Dimension Name | Truncate Before Update? | Dimension Attributes | Derived Attribute? | Data Type | Null/ Not Null | Key Column? | Data Length | Derived Logic | Description |
|---|---|---|---|---|---|---|---|---|---|
| DimChannel | No | ChannelSK | Yes | Int | NOT NULL | Yes. SK | Arbitary | Auto Increment | Dimension Table Unique Surrogate Key |
| | | AlternateChannelID | No | nvarchar(6) | NOT NULL | No | 6 | | Business Key in the Source Table |
| | | ChannelName | No | nvarchar(50) | NOT NULL | No | 50 | | Name of the Channel Group offeres send through |
| | | InsertDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Insertion Date |
| | | ModifiedDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Modified Date |
| | | | | | | | | | |
| DimOfferType | No | OfferTypeSK | Yes | Int | NOT NULL | Yes. SK | Arbitary | Auto Increment | Dimension Table Unique Surrogate Key |
| | | AlternateOfferTypeID | No | nvarchar(6) | NOT NULL | No | 6 | | Business Key in the Source Table |
| | | TypeName | No | nvarchar(15) | NOT NULL | No | 15 | | Offer Type of offers offered by Starbucks |
| | | InsertDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Insertion Date |
| | | ModifiedDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Modified Date |
| | | | | | | | | | |
| DimOffer | No | OfferSK | Yes | Int | NOT NULL | Yes. SK | Arbitary | Auto Increment | Dimension Table Unique Surrogate Key |
| (Considered as SDC) | | AlternateOfferID | No | nvarchar(50) | NOT NULL | No | 50 | | Business Key in the Source Table |
| | | Reward | No | tinyint | NULL | No | 4 | | Reward offered on Offer completion as a USD value |
| | | Difficulty | No | tinyint | NULL | No | 4 | | Total Amount have to pay if offer is absent (in USD) |
| | | Duration | No | tinyint | NULL | No | 4 | | Duration till the offer is available after generated |
| | | OfferTypeKey | No | int | NOT NULL | Yes. FK | Arbitary | | Surrogate Key of the Offer Type |
| | | ChannelKey | No | int | NOT NULL | Yes. FK | Arbitary | | Surrogate Key of the Channel |
| | | StartDate | Yes | datetime | NOT NULL | No | N/A | This Record Insert Time | This Record Insert Time |
| | | EndDate | Yes | datetime | NULL | No | N/A | Next Record Insert Time | Next Record Insert Time |
| | | InsertDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Insertion Date |
| | | ModifiedDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Modified Date |
| | | | | | | | | | |
| DimProfile | No | PersonSK | Yes | Int | NOT NULL | Yes. SK | Arbitary | Auto Increment | Dimension Table Unique Surrogate Key |
| (Considered as SDC) | | AlternatePersonID | No | nvarchar(50) | NOT NULL | No | 50 | | Business Key in the Source Table |
| | | Gender | No | nvarchar(2) | NULL | No | 2 | | Gender of the Customer |
| | | Age | No | tinyint | NULL = 118 | No | 4 | | Age of the Customer |
| | | BecameMemberOn | No | datetime | NULL | No | N/A | | BecameMemberOn of the Customer |
| | | Income | No | money | NULL | No | N/A | | Income of the Customer |
| | | YearOfBirth | Yes | Int | NULL | No | 4 | (YEAR([BecameMemberOn]) - Age) | YearOfBirth of the Customer |
| | | Street1 | No | nvarchar(60) | NOT NULL | No | 60 | | Street1 of the Customer's Address |
| | | Street2 | No | nvarchar(60) | NULL | No | 60 | | Street2 of the Customer's Address |
| | | Street3 | No | nvarchar(60) | NULL | No | 60 | | Street3 of the Customer's Address |
| | | City | No | nvarchar(50) | NOT NULL | No | 50 | | City of the Customer's Address |
| | | CountrySubDivisionCode | No | nvarchar(4) | NOT NULL | No | 4 | | CountrySubDivisionCode of the Customer |
| | | CountryCode | No | nvarchar(4) | NOT NULL | No | 4 | | CountryCode of the Customer's Address |
| | | PostalCode | No | nvarchar(50) | NOT NULL | No | 50 | | PostalCode of the Customer's Address |
| | | Longitude | No | nvarchar(50) | NULL | No | 50 | | Longitude of the Customer's Location |
| | | Latitude | No | nvarchar(50) | NULL | No | 50 | | Latitude of the Customer's Lacation |
| | | StartDate | Yes | datetime | NOT NULL | No | N/A | This Record Insert Time | This Record Insert Time |
| | | EndDate | Yes | datetime | NULL | No | N/A | Next Record Insert Time | Next Record Insert Time |
| | | InsertDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Insertion Date |
| | | ModifiedDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Modified Date |
| | | | | | | | | | |
| FactEvents | No | EventSK | Yes | Int | NOT NULL | Yes. SK | Arbitary | Auto Increment | Dimension Table Unique Surrogate Key |
| | | PersonKey | No | int | NOT NULL | Yes. FK | Arbitary | | Business Key in the Source Table |
| | | EventName | No | nvarchar(50) | NOT NULL | No | 50 | | EventName of the Customer's Address |
| | | EventDateKey | No | int | NULL | Yes. FK | Arbitary | | EventDateKey of the Customer |
| | | OfferKey | No | int | NULL | Yes. FK | Arbitary | | OfferKey of the Customer |
| | | Reward | No | tinyint | NULL | No | 4 | | Reward of the Customer |
| | | AmountUSD | No | money | NULL | No | N/A | | AmountUSD of the Customer |
| | | CurrencyRateKey | Yes | int | NULL | Yes. FK | Arbitary | Relevent exchange rate is retrieved based on Event date if the event name is equal to transaction. | CurrencyRateSK of the Customer |
| | | DiscountRate | No | decimal(3,2) | NULL | No | 3,2 | | DiscountRate of the Customer's Dication |
| | | DiscountAmountUSD | Yes | money | NULL | No | N/A | ([AmountUSD]*([DiscountRate]/100)) | DiscountAmountUSD of the Customer |
| | | TotalAmountAfterDiscountUSD | Yes | money | NULL | No | N/A | ([AmountUSD]-([AmountUSD]*([DiscountRate]/100))) | TotalAmountAfterDiscountUSD of the Customer |
| | | InsertDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Insertion Date |
| | | ModifiedDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Modified Date |
| | | | | | | | | | |
| DimDate | N/A | DateKey | Yes | int | NOT NULL | Yes. SK | N/A | Static Dimensional Table | Dimension Table Unique Surrogate Key |
| | | More Attributes... | | | | | | | ... |
| | | | | | | | | | |
| DimCurrency | No | CurrencyRateSK | Yes | Int | NOT NULL | Yes. SK | Arbitary | Auto Increment | Dimension Table Unique Surrogate Key |
| (Considered as SDC but NO sd/ed) | | SourceCurrencyType | No | nvarchar(4) | NOT NULL | No | 4 | | Partial Business Key in the Source Table |
| | | DestinationCurrencyType | No | nvarchar(4) | NOT NULL | No | 4 | | Partial Business Key in the Source Table |
| | | ExchangeRate | No | decimal(18,6) | NOT NULL | No | 18,6 | | Actual Exchange Rate |
| | | ExchangeDate | No | date | NOT NULL | No | N/A | | Partial Business Key in the Source Table |
| | | InsertDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Insertion Date |
| | | ModifiedDate | Yes | datetime | NOT NULL | No | N/A | System Date | Timestamp of the Modified Date |

Figure 3. 2. Details of the List of Data Warehouse tables in Starbucks_Offers_DW database. 200% Zoom is recommended to read the text in the image clearly.

Note that the list of details given above in the figure 3.2 is also available in the Starbucks ETL Mapping Document.xlsx file submitted along with the report. The Mapping file also can be found in the visual studio SSIS project solution.

# 4. Data warehouse design & development

The Dimensional Model of the Data Warehouse solution was ultimately designed according to the Snowflake Schema. Since the dataset chosen was about offer and transaction details of customers, the relevant dimensions were identified according to the answers for the questions.

Who? : customers [Profile]
What? : offers and Transactions [Event]
Why? : identify customer interactions with offers sent by Starbucks
Where? : Address of the customers [Location]
When? : Transaction dates [Event dates]
How? : based on offer types, event, event dates, transactions in LKR or USD, etc.

List of Dimension and Fact tables are described in detail in figure 3.2 of the previous section. While doing the mapping from source files/tables to the Data Warehouse tables offer table was considered as normalized and was connected to channel and offer_type tables which is why the schema of the solution became a snowflake schema.

## 4.1. Dimensional Model Schema

The Data warehouse design of the Starbucks_Offers_DW is given below, which was designed according to the snowflake schema.

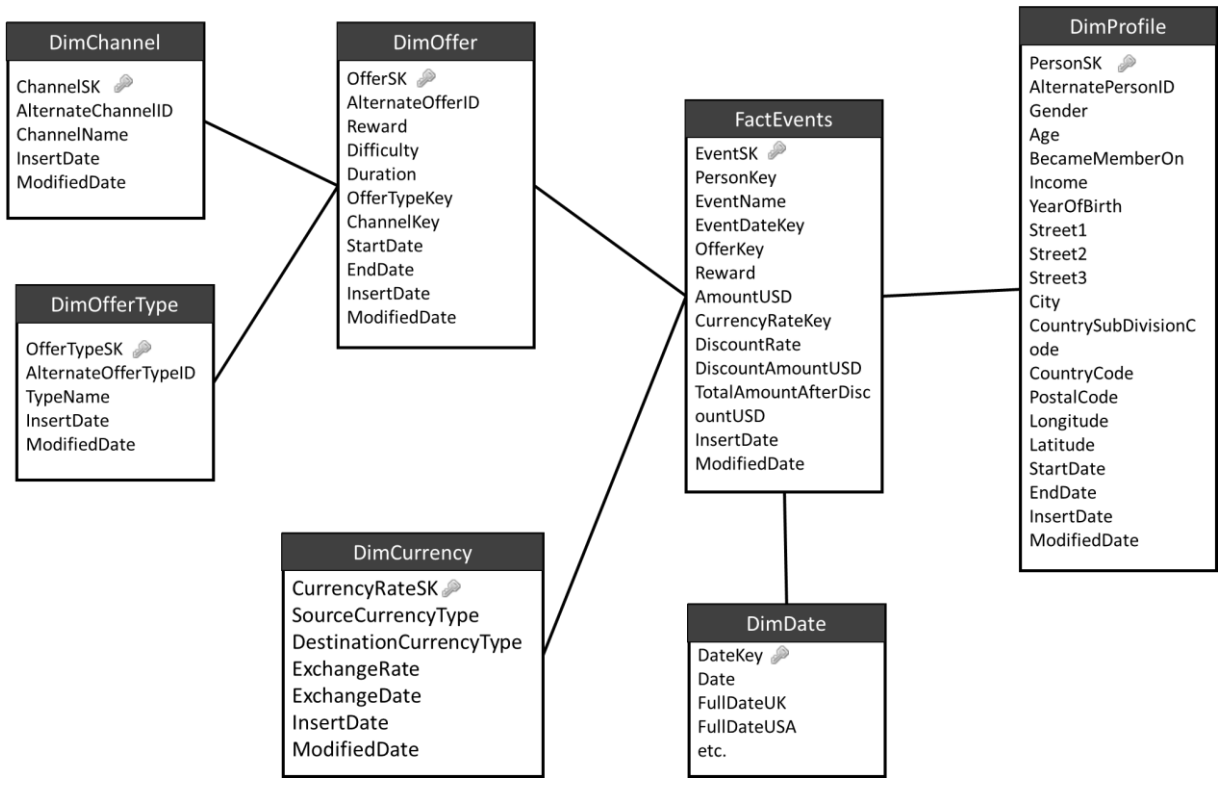



Figure 4. 1. Snowflake schema dimensional model used by Starbucks Offers Data Warehouse Solution

### 4.2. Dimensional Tables

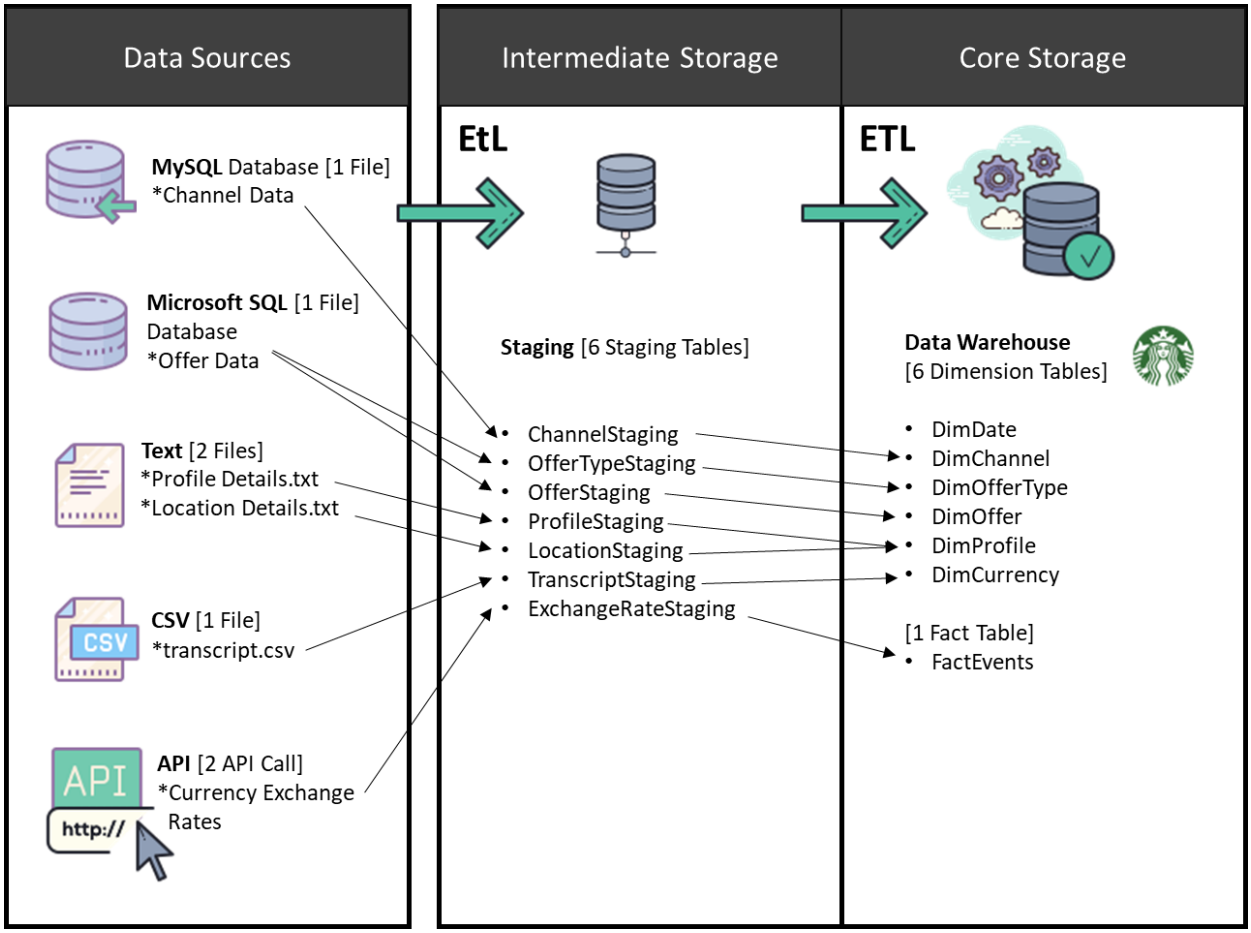



Figure 4. 2. 1. Table Mapping from Source, Staging to Data Warehouse

**Slowly Changing Dimensions (SCD)**

Notice that DimProfile, DimCurrency, and DimOffer have treated as **Slowly Changing Dimensions.** Both DimProfile and DimOffer dimension tables have two derived fields to store start date and end date to maintain and indicate historical records. **(Type 2).** The dimension table DimCurrency behaves in a different manner and it is explained below.

DimCurrency table is used to record current and historical values of daily exchange rates of USD to LKR and LKR to USD to enable currency conversions between USD and LKR in the data warehouse for analytics [1][2]. To make the calculations easier, it was decided to store the surrogate key of the relevant exchange rate using a lookup while loading the fact table records to the data warehouse rather than recording the exchange rate in a column in the fact table directly. The reason for this decision is that in case a given exchange rate of a certain date is not available,

the fact table may have to be checked for empty exchange rate cells and replace and update the fact table from time to time which can be a quite tedious task considering the number of rows that will get inserted into the fact table.

As a solution for this issue, what was done is treating the DimCurrency dimension table as a static table at very first and store default values for an extended period (just like the DimDate dimension table). This way, the fact table will be initialized with default value of exchange rate of a given date as 0.00 (maybe from fact table record start date until a year like 2098) and each record will have a unique surrogate key. When the real exchange rates are taken from the API, the existing records will get updated or else if it is not already inserted in the table, a new exchange rate record will get inserted (a stored procedure is used).

This avoids a complex scenario where the ETL developers might have to consider inserting a default exchange rate at the ETL execution time if a certain exchange rate is not available, thus, inserting default values first will reduce the number of ETL tasks have to be performed. An exchange rate is retrieved using the business key of the DimCurrency table, which is three columns, SourceCurrencyType, DestinationCurrencyType, and ExchangeDate combined. When a new transaction is inserted into the fact table, always a corresponding surrogate key will be returned using a lookup whether the exchange rate is a default value or a valid exchange rate, avoiding a complex scenario of having to insert new records to the DimCurrency when inserting a new record to the fact table.

For the reasons explained above, although DimCurrency is a Slowly Changing Dimension, default values were pre-loaded and are getting updated daily using the values retrieved using the API in the staging layer EtL. More about these EtL can be read in the section 5, “ETL Development”. Since the DimCurrency values are recorded daily, start date and end date columns were not used, and although it usually updates an existing default value, the SCD type cannot be necessarily said to be Type 0 because in practice, it maintains history.

**Static Dimension tables**

Both DimDate and DimCurrency tables are treated as static dimension tables and the values have been pre-inserted. Note that DimCurrency pre-inserted values are the same default values which will remain unchanged until the real exchange rates are retrieved and loaded. DimCurrency is only considered as a static table at the time of Data Warehouse creation. After that, its values will get periodic (daily) updates thus no longer valid to be considered as a static dimension.

**Role Playing Dimension tables**

There are no role-playing dimensions present in the Starbucks Data Warehouse design since the DimDate is also referred only one tome by the fact table, which otherwise has a high chance of being a role-playing dimension table.

### 4.3. Fact Tables

In the data warehouse design, there is only one fact table, namely, "FactEvents" which is falling under Transaction Fact Table Type since it keeps a record of every even/transaction that happens. The table "TranscriptStaging" in the staging database was mapped to FactEvents table in the data warehouse and several derived columns have been added including measures and timestamps.

#### 4.3.1. Types of Measures

Following is a list of Measures there exists in the FactEvents Fact table in Starbucks_Offers_DW data warehouse solution.

**Fully Additive Measures**

- Reward [reward received by a customer for offer completed events, given in USD]
- AmountUSD [original amount had to be paid for a transaction event, given in USD]
- TotalAmountAfterDiscountUSD [amount paid after deduction the discount, in USD]

**Semi-Additive Measures**

There are no semi-additive measures in the FactEvents fact table.

**Non-Additive Measures**

- Discount Rate [Discount rate granted for transactions done by the customers]

### 4.4. Identified Hierarchies

The following Hierarchies were identified between columns of the given tables.

- Year > Month > Day [DimDate]
- SourceCurrencyType > DestinationCurrencyType [DimCurrency]
- OfferType > Offer [DimOffer and DimOfferType]
- Channel > Offer [DimOffer and DimChannel]
- Age > YearOfBirth > Person_ID [DimProfile]
- Countrycode > CountrySubDivisionCode > City > PostalCode > Street1 > Street2 > Street3 [DimProfile]
- City > PostalCode > Street1 > Street2 > Street3 > geolocation (latitude and longitude) [DimProfile]

### 4.5. Derived Columns

- **YearOfBirth** – derived from using age and membership acquired date [DimProfile]
- **Surrogate Keys** – given to each data warehouse dimension table as a unique PK [int]
- **InsertDate and ModifiedDate** – given to all dimensional model dim and fact tables to record timestamps of insertions and modifications.
- **StartDate, EndDate** – given to each SCD table except DimCurrency, to keep track of historical value changes
- **DiscountAmountUSD** – (AmountUSD * discount rate) [FactEvents Table]
- **TotalAmountAfterDiscountUSD** – (AmountUSD – DiscountAmountUSD) [FactEvents Table]

### 4.6. Data Warehouse Implementation

#### 4.6.1. Staging Layer SQL Database Implementation

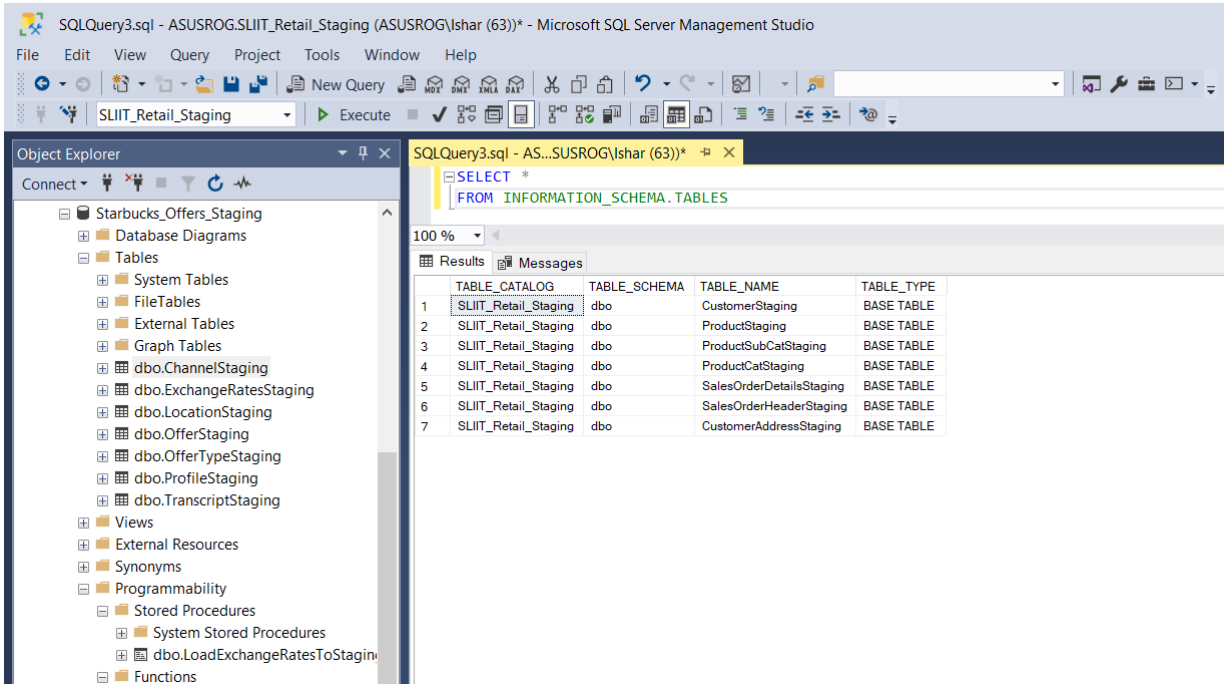



Figure 4.6.1. 1. List of tables implemented in Staging database "Starbucks_Offers_Staging"

Creating the Staging layer tables was done during the Fast Load in Staging EtL.

### 4.6.2. Data Warehouse SQL Database Implementation

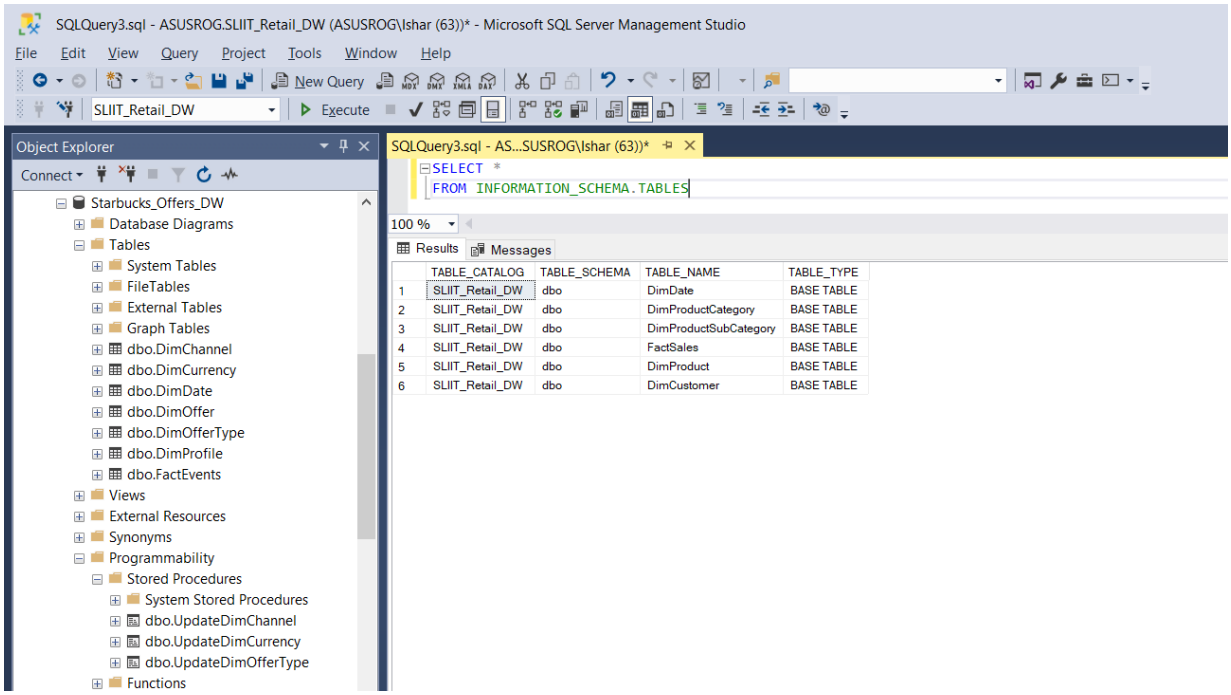



Figure 4.6.2. 1. Starbucks_Offers_DW data warehouse Dimensional model Dim and Fact Tables

## Individual Dim and Fact Table Designs

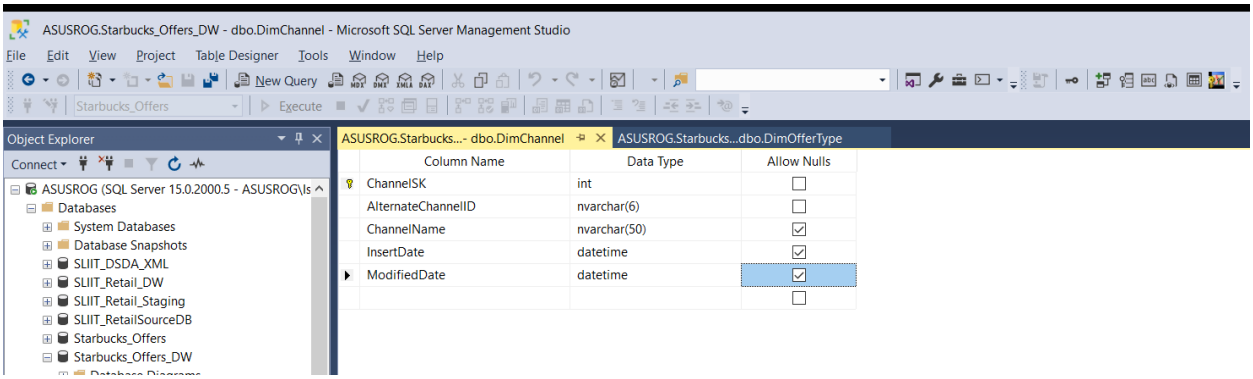



Figure 4.6.2. 2. DimChannel Table

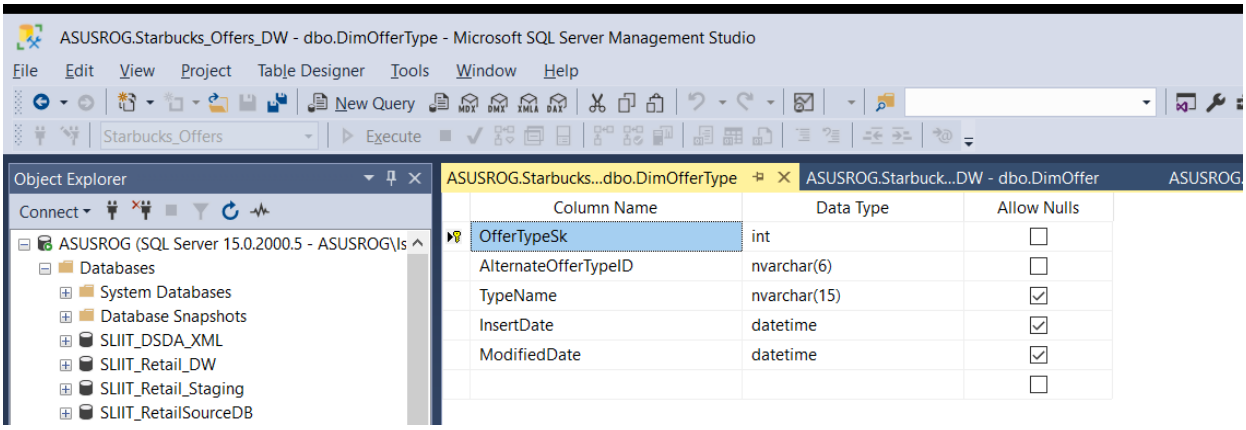



Figure 4.6.2. 3. DimOfferType Table

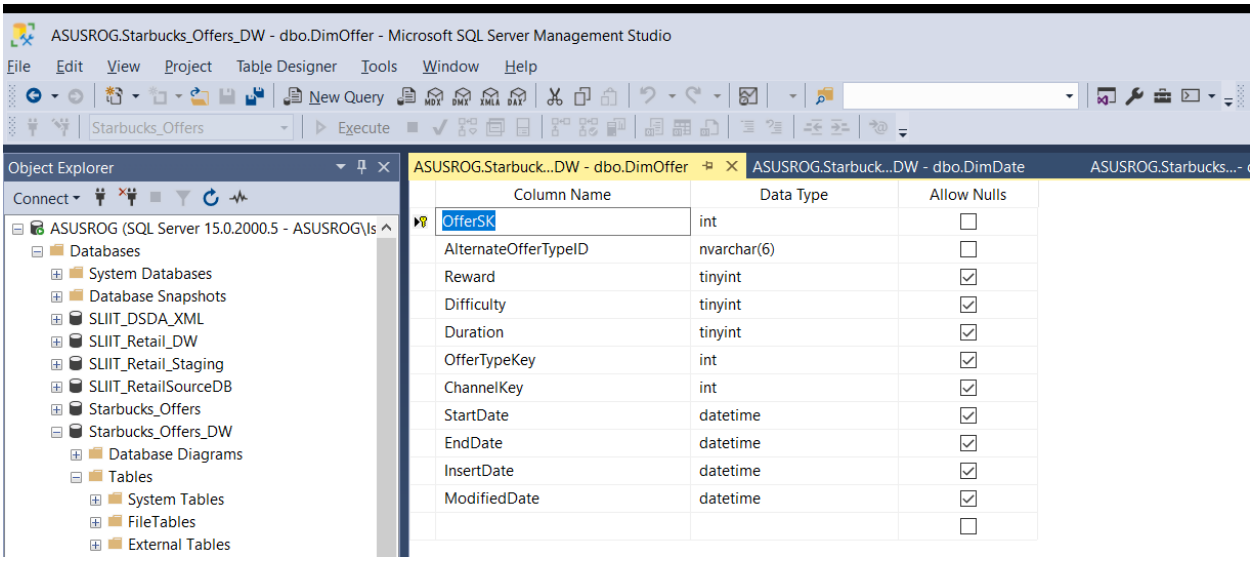



Figure 4.6.2. 4. DimOffer SCD Table

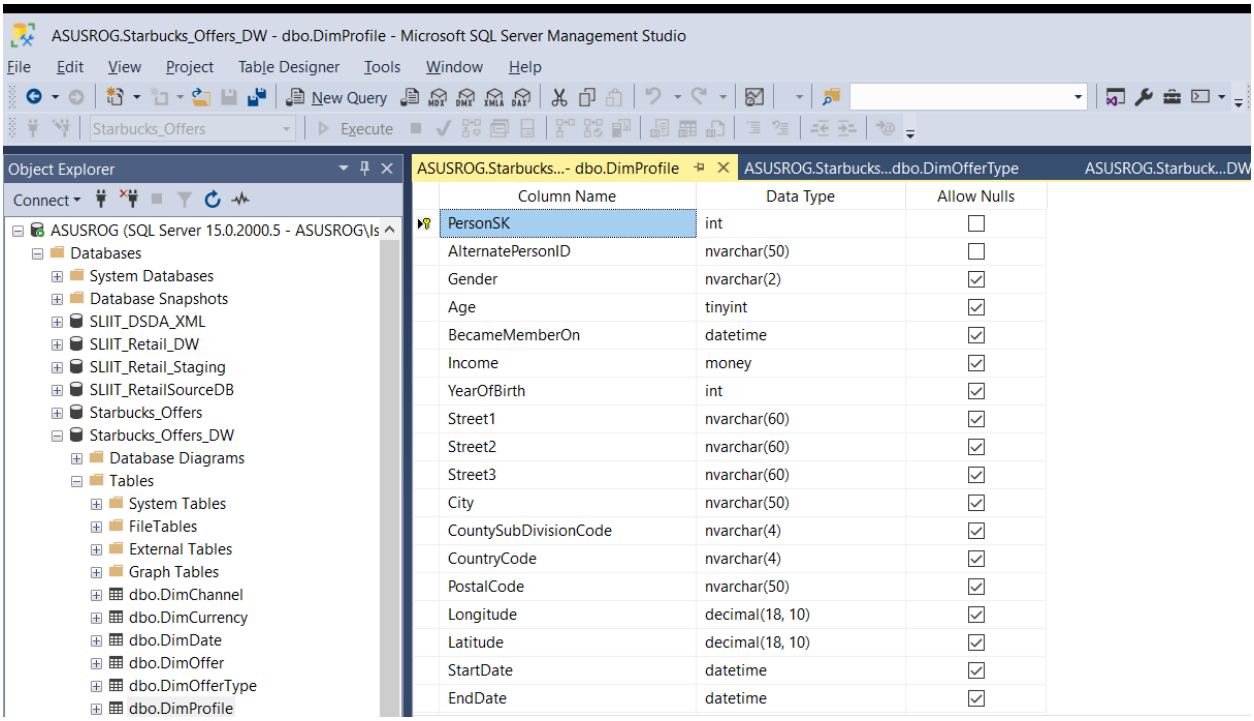



Figure 4.6.2. 5. DimProfile SCD Table

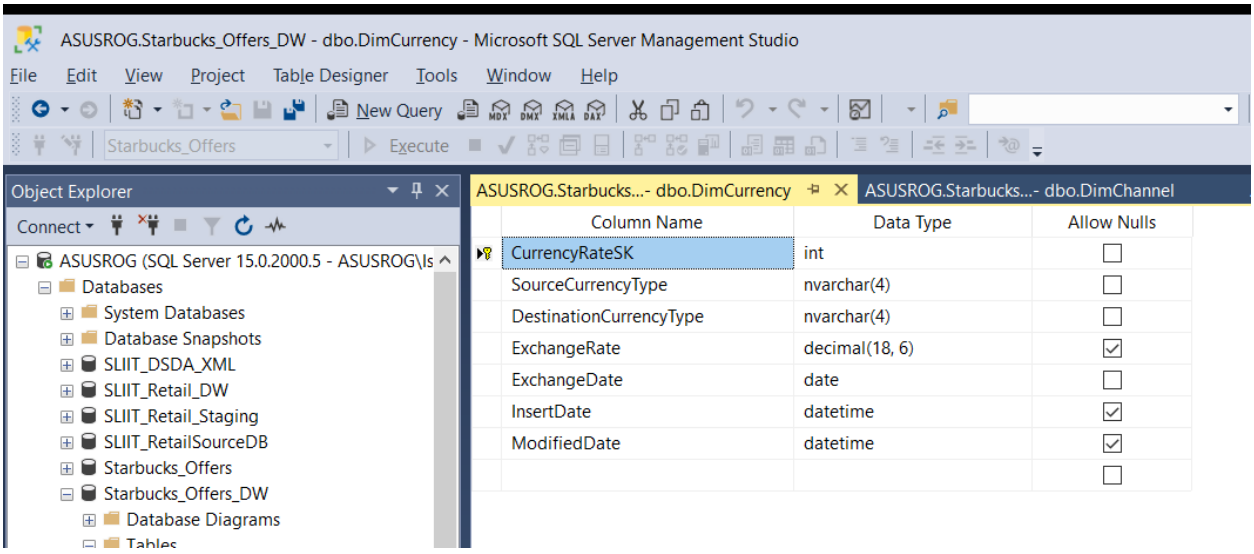



Figure 4.6.2. 6. DimCurrency SCD Table

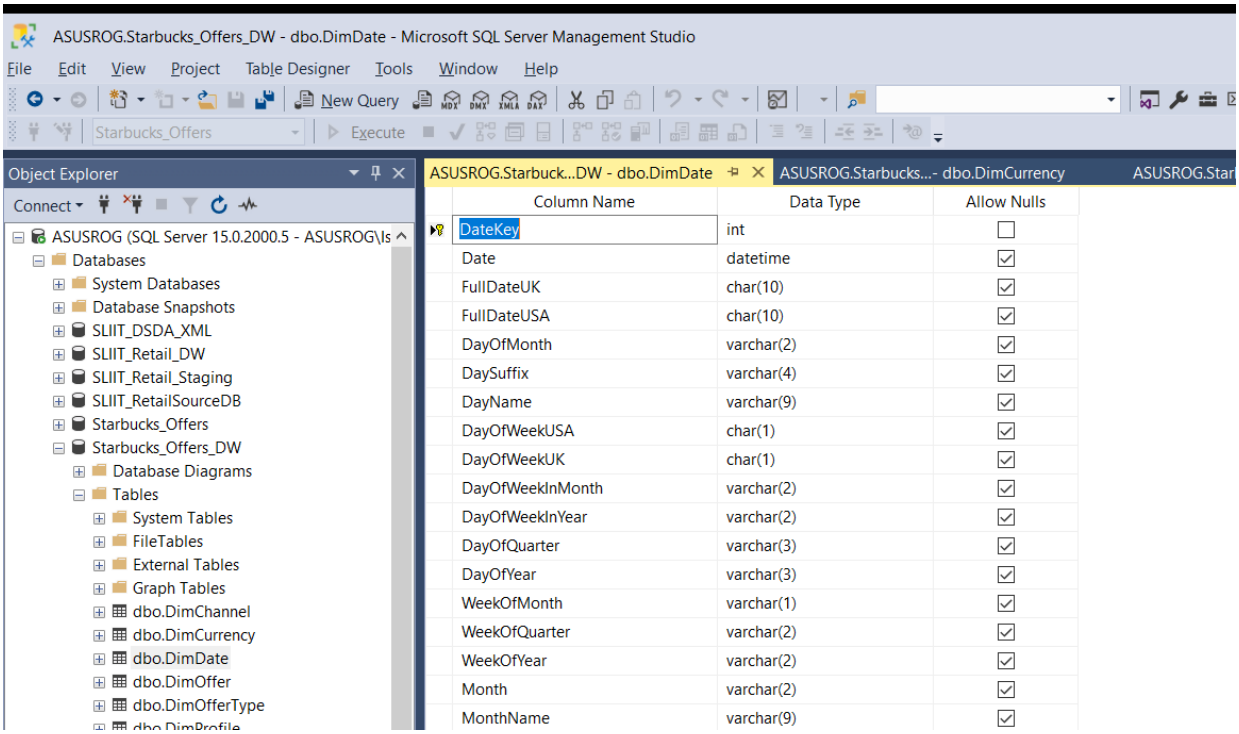



Figure 4.6.2. 7. DimDate Static Dimension Table

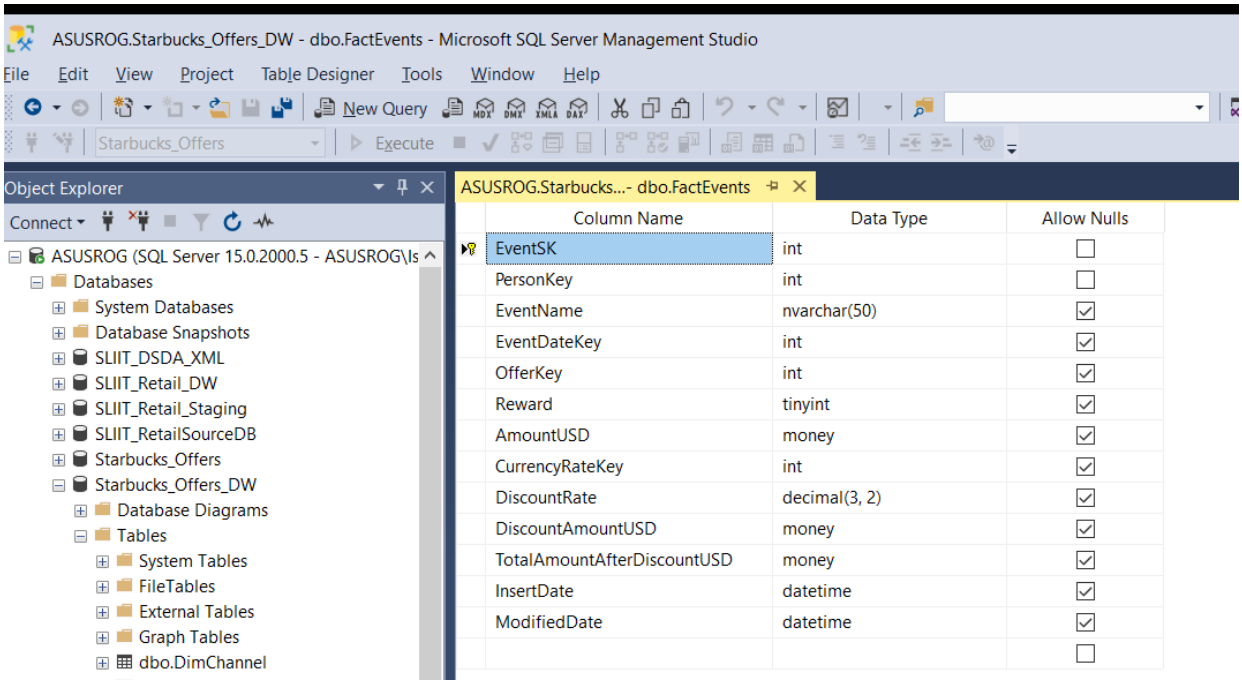



Figure 4.6.2. 8. FactEvents Fact table

# 5. ETL development

As shown in the high-level architectural design (figure 3.1) there are two distinct ETL processes. The first EtL is from Source files/locations to Staging EtL (less transformations) and the next ETL is from Staging to Data Warehouse that was already designed and implemented with empty set of Dimension and Fact tables. Both ETL processes have been developed according to the Mapping document generated and considering the order of execution of each task in a particular ETL. The second ETL process which Loads staging data to warehouse is configured to get executed right after the sources-to-staging EtL has finished execution. Within each independent ETL process, the control flow order has been created considering the order of execution as well.

First, a SSIS project has been created using Visual Studio 2019 and all the SQL Server databases related to the Staging and Data Warehouse has been created using SQL Server 2019 Developer edition. Before explaining each ETL process, there were some pre-requisites to be completed.

## 5.1. Pre-requisites

- Visual Studio 2019
- VSIX Packages of SSIS, SSAS, and SSRS compatible with Visual Studio 2019
- SQL Server 2019
- Registering **System.Text.Json.dll** in Windows GAC using Windows PowerShell to enable JSON Deserialization within script tasks (Required when setting up the API calls)
- MySQL Connector version 8.0 to establish connections to the MySQL databases.

Note that the required DLL file has been included in the Solution folder of the submission. Instructions on how to register it in windows GAC using PowerShell and Installing MySQL connection has been given below.

### Registering the required DLL File

First, use the given DLL or download the DLL using NuGet Package Manager in Visual Studio and then copy and paste the DLL into a well-known easily accessible directory such as C:\ drive.

Then open PowerShell with Administrator privileges and type the following command.

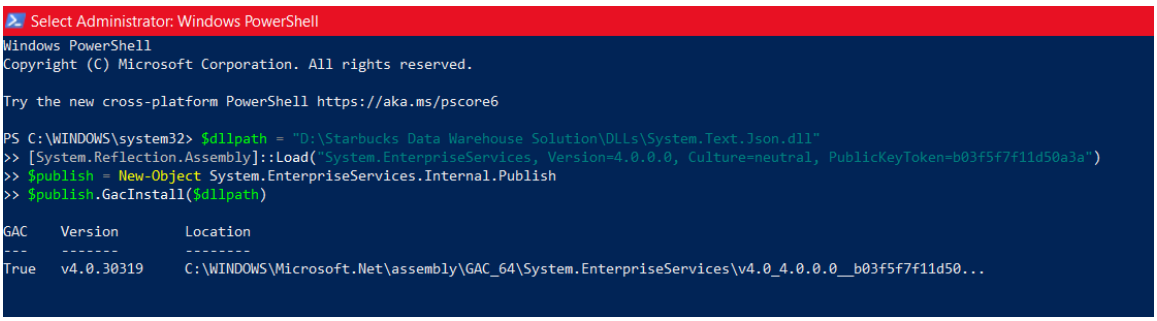



Figure 5.1. 1. Registering System.Text.Json.dll using PowerShell

Replace the $dllpath with the location of the DLL that was copied earlier and after running all the commands the GAC status should indicate TRUE if the task was a success. Without registering the DLL file, the Script task of Starbucks_Offers_Staging package which supposed to make the API call will not function properly.

**Installing MySQL Connector**

After downloading and installing the MySQL connecter, preferably version 8, Go to the start menu and search for ODBC Data Sources (64-bit) (depends on the OS version) and Add a new System or User DSN > choose MySQL ODBC 8.0 Unicode Driver.

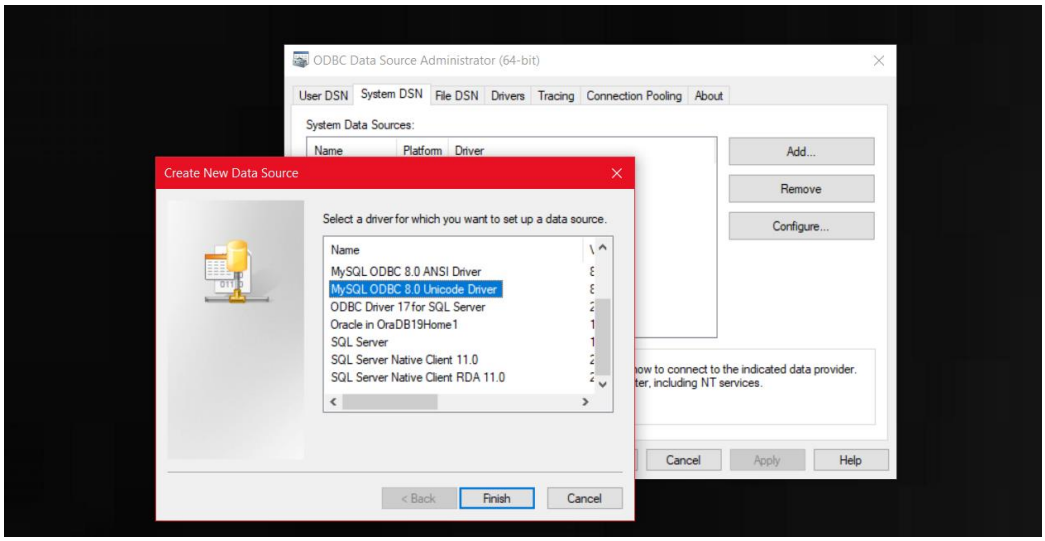



Figure 5.1. 2. Selecting the Proper MySQL Driver in ODBC Data Sources(64-bit) on Windows 10

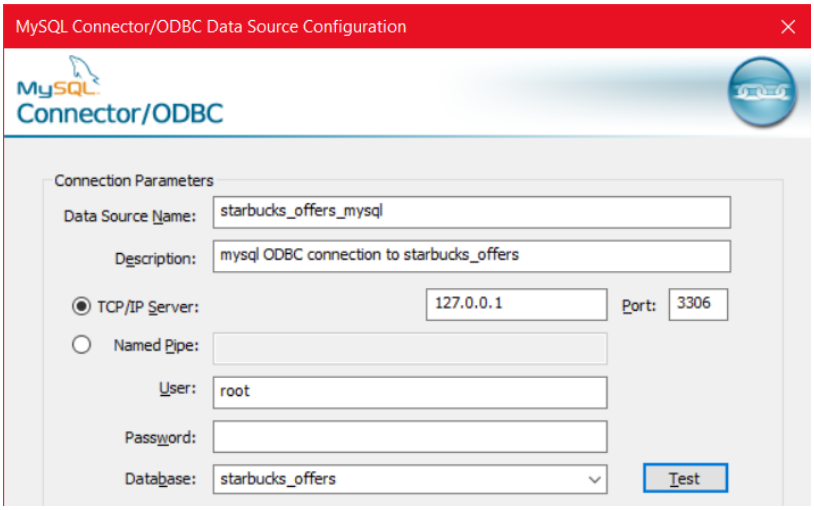



Figure 5.1. 3 Providing valid connection details to establish a new connection with an existing MySQL server instance

Then the proper connection details and authentication should be provided, and the connection should be tested. Note that without the above two specific prerequisites explained, some features will not work and the Staging ETL will fail.

### 5.2. Additional Features in both ETL processes

In addition to the required main components in each ETL that have been explained in the following sections, an additional feature of ability to send an Email has been implemented in both ETL processes either when they get executed successfully or fails to get executed for some reason. Email sending was configured using a Script Task in SSIS control flow for success Emails and the failures will get notified via email from the Script Tasks implemented in the Even Handlers on "OnError" for each ETL SSIS package[10]. The code can be found in the Appendix (B-1).

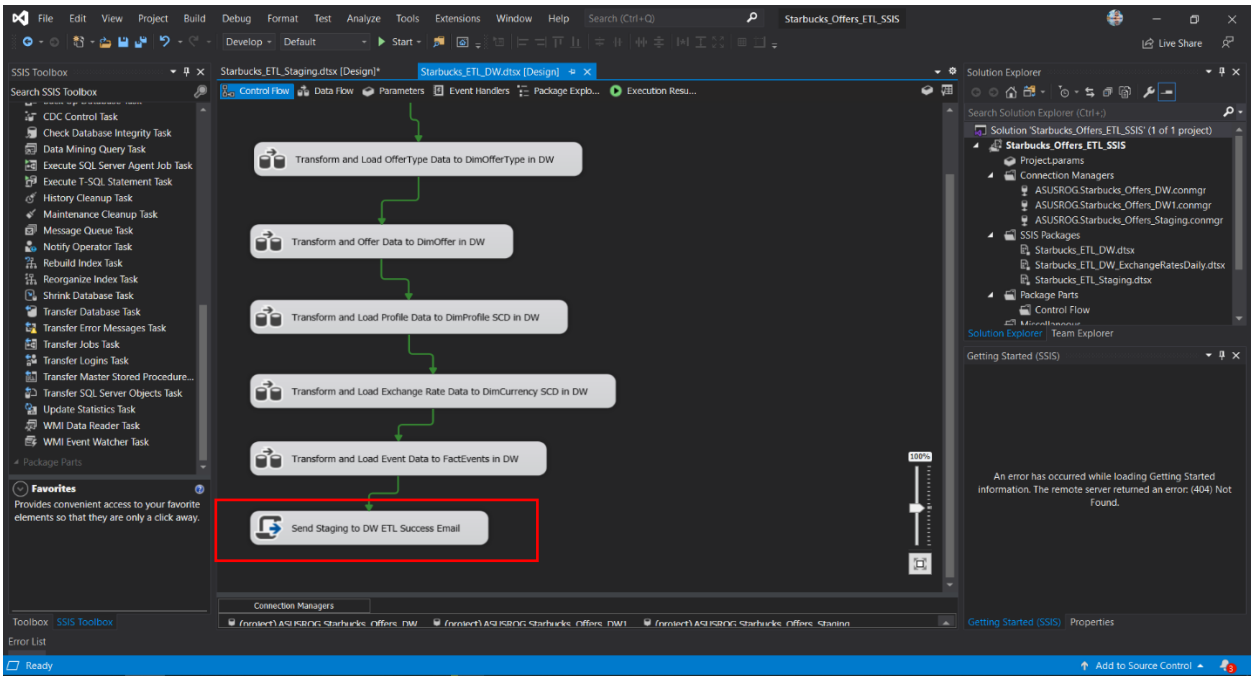



Figure 5.2. 1. Example of a Script task configured to send email notifications at the end of Staging to DW ETL execution

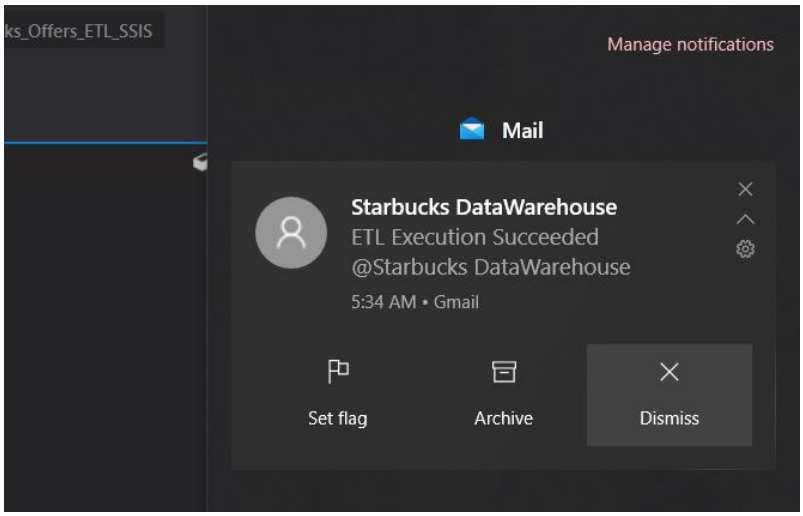



Figure 5.2. 2 Email Notification received after ETL execution [Windows 10]

### 5.3. SSIS Solution, Packages, and Project Connections

The Project has been named as Starbucks_ETL_SSIS and there are two main packages as follows.

**SSIS Packages**

- Starbucks_ETL_Staging.dtsx – Includes set of ordered Tasks which performs EtL from sources to the staging layer database tables.
- Starbucks_ETL_DW.dtsx – contains all tasks which gets executed in a specific order which extract data from staging layer tables, do the necessary transformations, and load to the dimensions and fact tables in the data warehouse database.

**Project Connections**

Project connections are a set of connections made to connect to staging and data warehouse databases. One or more connections to the same database have created in scenarios depending on the type of the connection manager, namely, ADO.NET or ODBC

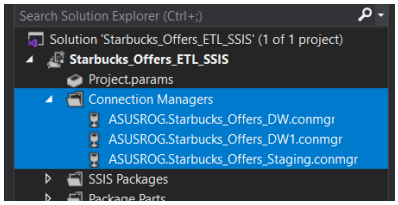



Figure 5.3. 1. Project Connection Managers

Project connection Managers are common for every package included in the solution while there are additional connections created in each package as per the necessity especially when connecting to data sources from the staging package.

### 5.4. Sources to Staging layer EtL

In the first EtL process, the number of transformations is quite close to zero. It is mainly focusing on loading the source data to the staging layer database tables for many valid reasons, including, release source databases/ files as soon as possible within a less amount of time, make staging layer available faster, lower the network traffic cost if the EtL is implemented over a network and more.

In this EtL process the order does not that many matters because constraints (business keys) are not validated during the data loading to staging tables. However, to avoid two tasks running at once which may require more system resources, the control flow was implemented by connecting all Task components together making a pipeline kind of order in execution.

Another valid reason for implementing an order of execution in this EtL was the fact that there is an additional email notification which should get triggered when all tasks are completed without any issue and to be able to execute the second ETL process only after all tasks in the first EtL are executed fully. Sources to Staging EtL has been designed in Starbucks_ETL_Staging.dtsx SSIS Package.

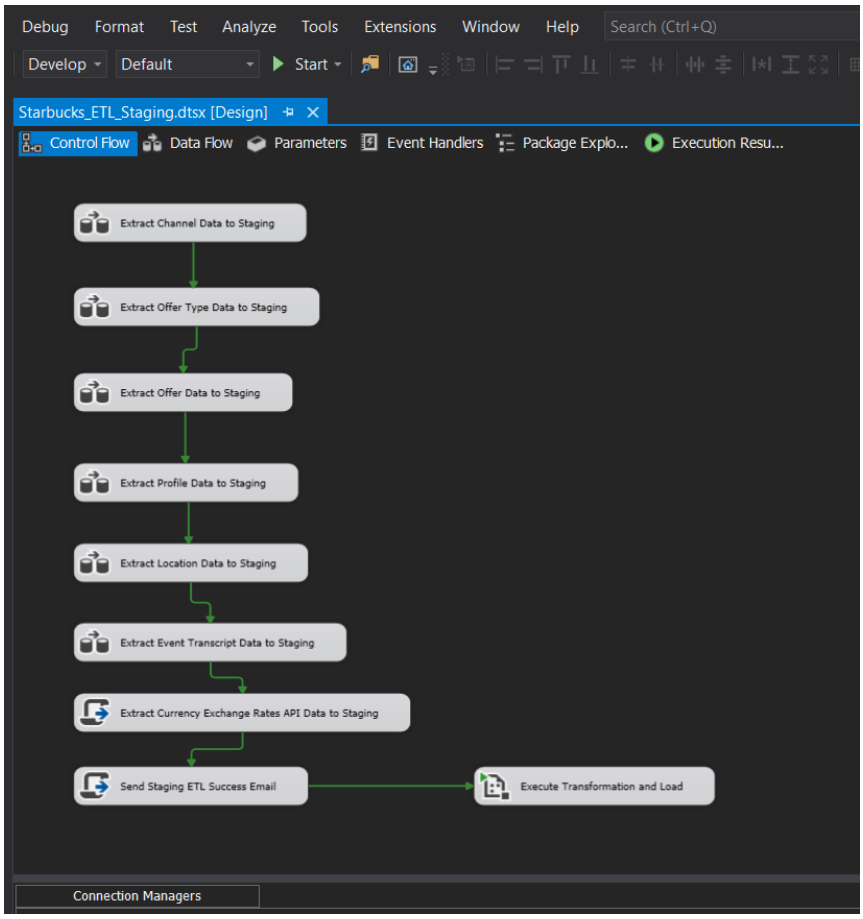



Figure 5.4. 1. Overall Control Flow of the Staging Package Tasks

### 5.4.1. Connections used

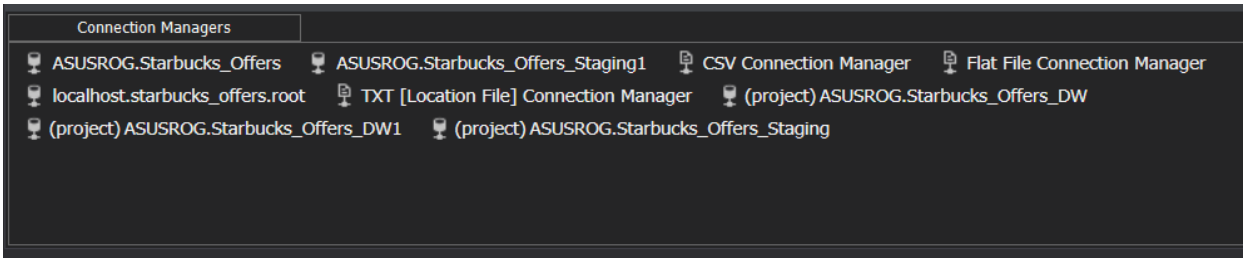



Figure 5.3. 2. List of connection Managers created to connect with the source file and the staging database in the Starbucks_ETL_Staging.dtsx package. Different types of sources required different types of connection managers

| **Connection Manager Name** | **Target** |
|---|---|
| ASUSROG.Starbucks_Offers | MSSQL source database (channel table) |
| CSV Connection Manager | Flat File connection used to load CSV Source (Transcript.csv) |
| Flat File Connection Manager | Flat File connection used to load profile.txt source file |
| TXT [location File] Connection Manager | Flat File connection used to load location.txt source file |
| localhost.starbucks_offers.root | ADO.NET Connection to connect to MySQL source database (offer_type and offer tables) |
| ASUSROG.Starbucks_Staging1 | ADO.NET Connection used by API Call Script task to insert currency exchange rates to staging table |
| (project) ASUSROG.Starbucks_Offers_DW | ODBC Connection to the Data Warehouse database |
| (project) ASUSROG.Starbucks_Offers_DW1 | ADO.NET Connection to the Data Warehouse database |
| (project) ASUSROG.Starbucks_Offers_Staging | ODBC Connection to the Staging database |

Notice that in some cases like running script tasks, additional ADO.NET connection to the same destinations/sources have been created. The reason is that the script tasks was required a ADO.NET connection manager when trying to insert the currency exchange rates to the staging layer's relevant table. Although that is the approach taken while coding the script task in this scenario, there might be ways to use the ODBC connections as well.

### 5.4.2. Tasks of EtL from Sources to Staging in their Execution Order

#### 5.4.2.1. Extract Channel Data to Staging

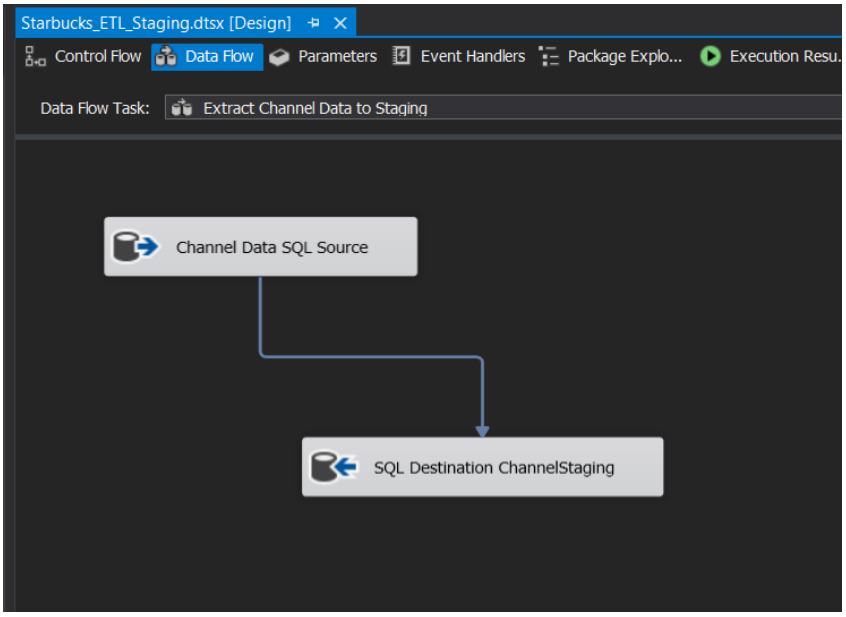



Figure 5.4. 2.1.1. Data Flow of the "Extract Channel Data to Staging" Data Flow Task

As shown in the figure 4.4.2.1.1, an OLE DB Source and OLE DB Destination components were used to extract from channel data from MSSQL database to the ChannelStaging table of the MSSQL Staging database "Starbucks_Offers_Staging". No transformations were done during the process.

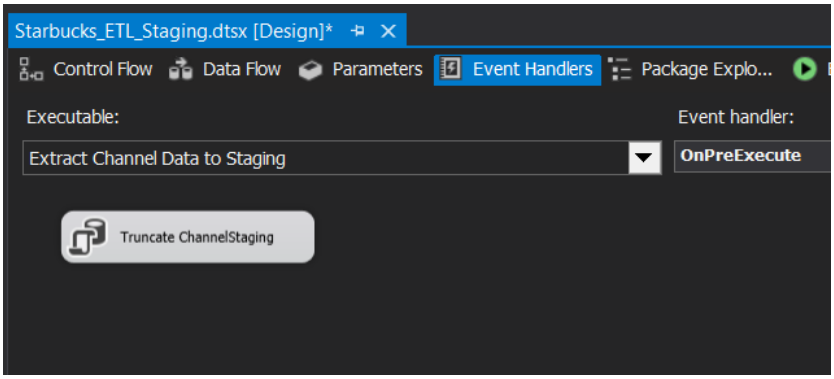



Figure 5.4. 3 OnPreExecute Event Handler of the data flow

An "Execute SQL Task was used inside the relevant event handler of the above-mentioned data flow task on Pre-execution to truncate the ChannelStaging table beforehand the loading takes place to clear any previously staged data since they are simply outdated.

### 5.4.2.2. Extract Offer Type Data to Staging

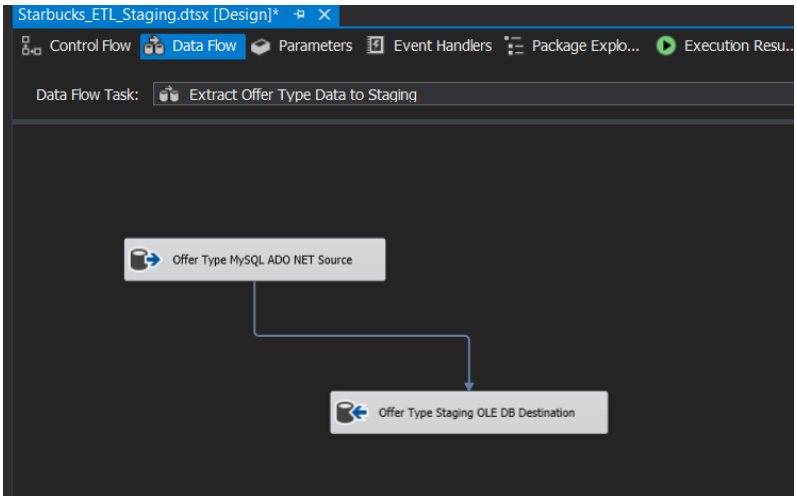



Figure 5.4.2.2. 1. Data Flow of "Extract Offer Type Data to Staging" Data Flow Task

Data Flow ensures that source data from the MySQL database table “offer_type” are extracted using an ADO.NET Source and loaded to the OfferTypeStaging table in the Staging database in MSSQL Server using an OLE DB Destination. No transformations were done to extracted data.

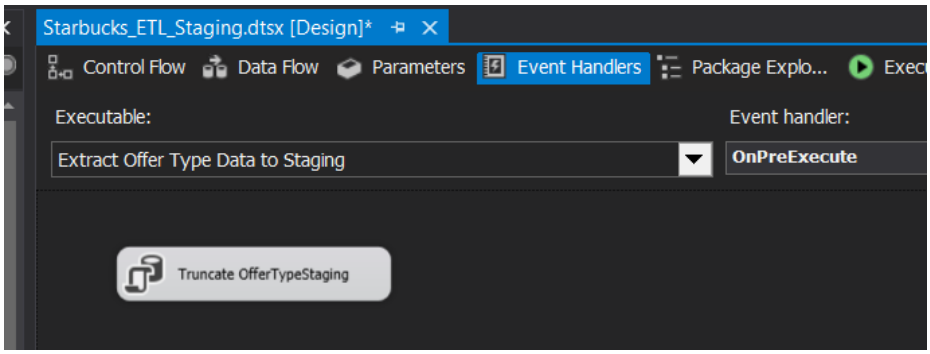



Figure 5.4.2.2. 2. OnPreExecute Event handler which has an Execute SQL Task to truncate the relevant table

Similar in the previous case, an OnPreExecurte event handler was used to create a Execute SQL Task to truncate the OfferTypeStaging table before loading the newly extracted data.

### 5.4.2.3. Extract Offer Data to Staging

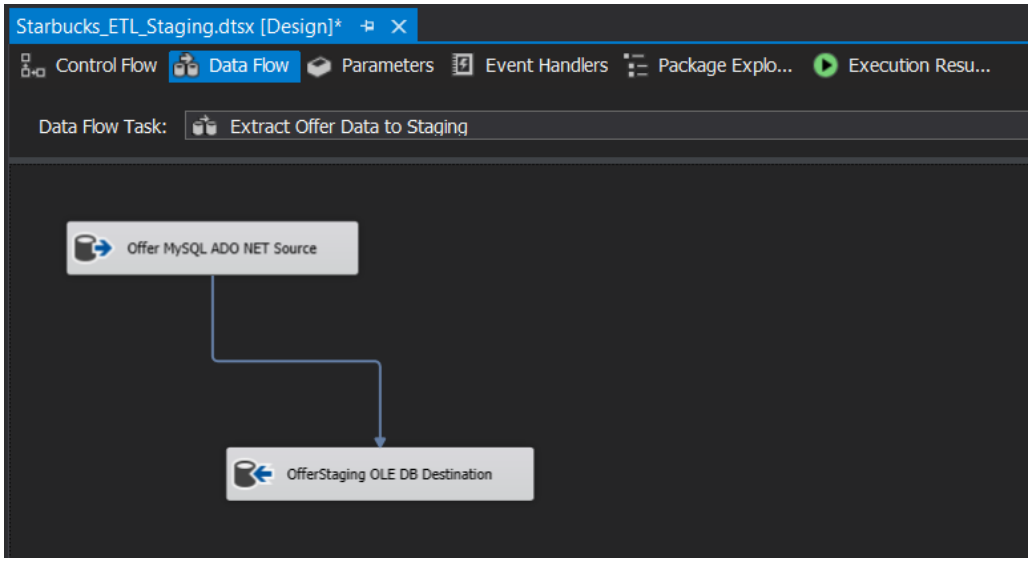



Figure 5.4.2.3. 1. Data Flow with the ADO.NET source and OLE DB Destination

Data Flow of the "Extract Offer Data to Staging" Data Flow Task uses a ADO.NET Source and connects to source MySQL database and extracts data and then loads them to OfferStaging table of the OLE DB destination which is the Staging database "Starbucks_Offers_Staging" in MSSQL Server.

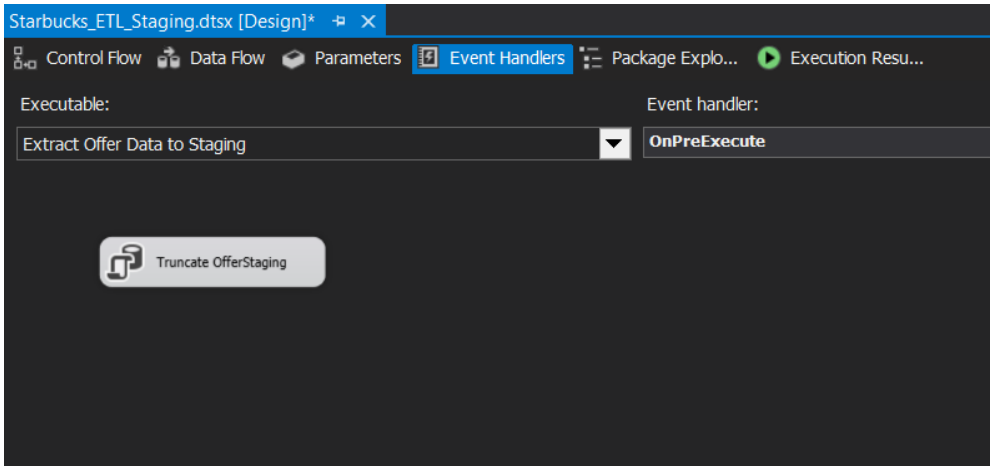



Figure 5.4.2.3. 2. OnPreExecute event handler that truncates OfferStaging table before data is extracted and loaded

An event handler was used in the similar way shown in previous cases, to truncate the OfferStaging table of the staging database prior to the commencement of the EtL Task by specifying "OnPreExecute".

### 5.4.2.4. Extract Profile Data to Staging

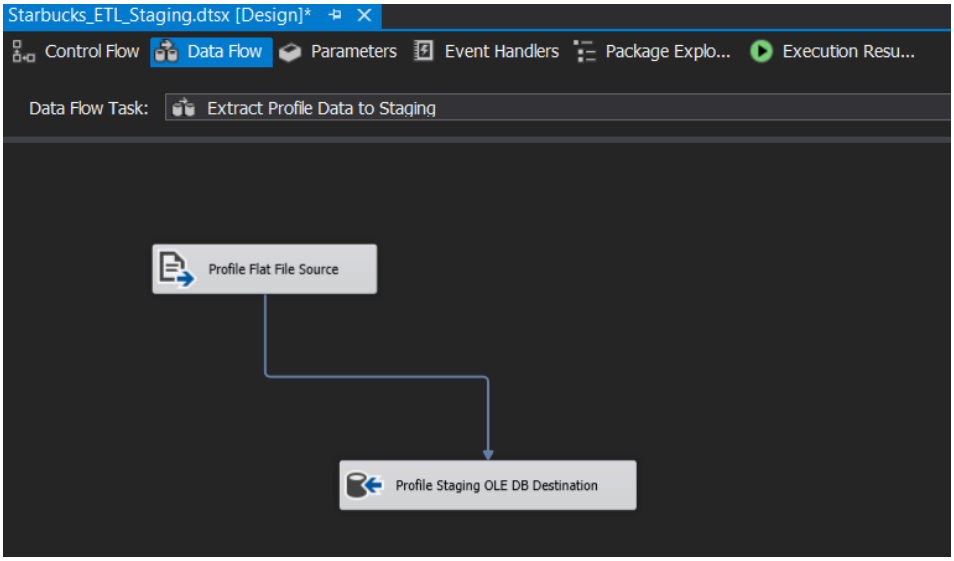



Figure 5.4.2.4. 1. Data Flow of "Extract Profile Data to Staging" to extract and load data from profile.txt to ProfileStaging table

Profile data are getting extracted using a flat file source that connects to the file and using a OLE DB destination that is used to load (insert) into the staging database table, “ProfileStaging”.

An event handler is in place and configured to run OnPreExecute to truncate the ProfileStaging table before loading. The event handler is called automatically before the Extract Profile Data to Staging Data Flow Task gets executed.

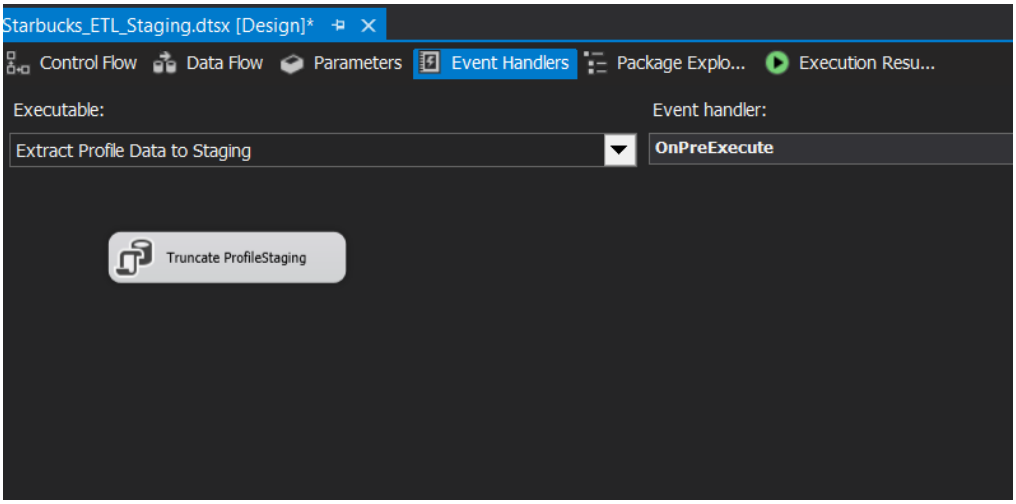



Figure 5.4.2.4. 2. ProfileStaging table truncating SQL Task in event handler

### 5.4.2.5. Extract Location Data to Staging

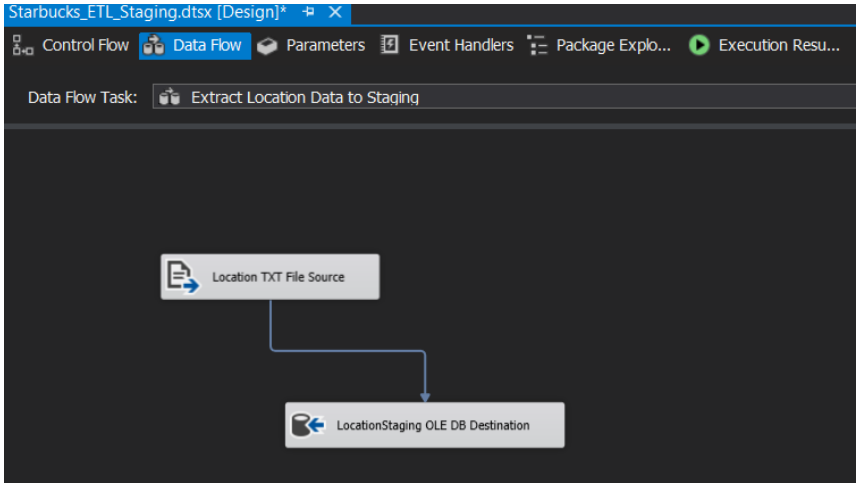



Figure 5.4.2.5. 1. location.txt file data extracting and loading data flow

A Flat File Source is used to extract data from location.txt using the relevant connection manager and then the data is sent to OLE DB Destination which is responsible for loading the extracted location data into LocationStaging destination table in the staging database in MSSQL Server.

An event handler is in place to truncate the LocationStaging table before loading data to it. No transformations are done during this data flow.

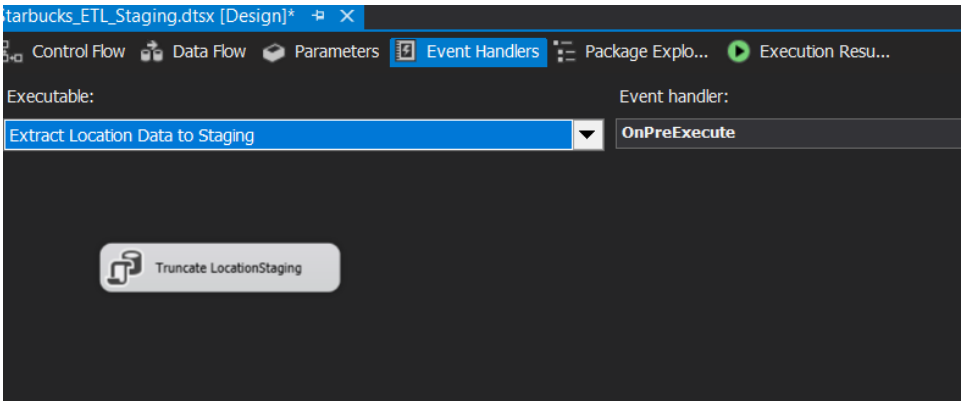



Figure 5.4.2.5. 2. The event handler which runs before the data flow gets executed, to truncate LocationStaging table using the added “Execute SQL Task” component.

### 5.4.2.6. Extract Event Transcript Data to Staging

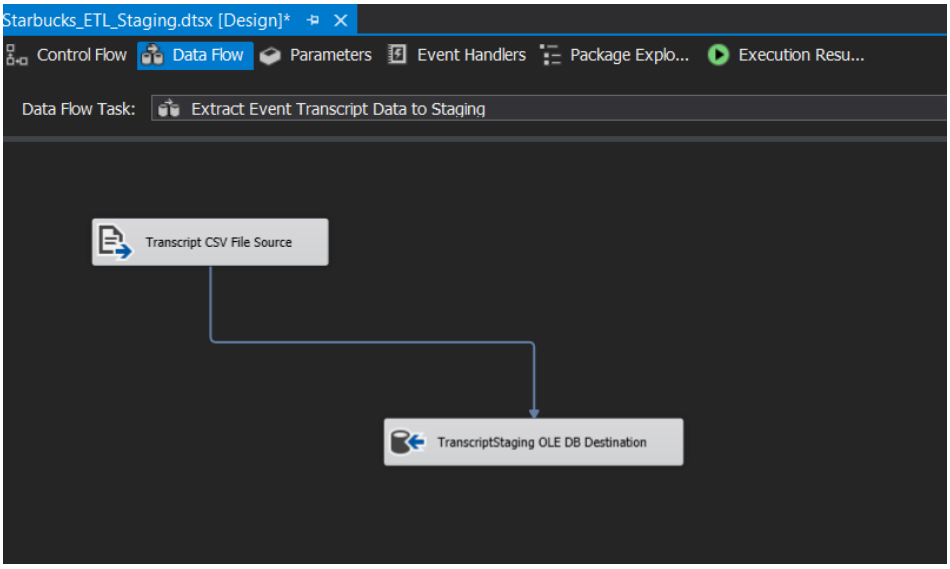



Figure 2.4.2.6. 1. Event Transcript Extraction and Load Data Flow

Since Event Transcript is a CSV file, again a flat file source was taken as The Source to get the data extracted and an OLE DB destination component has been placed to load the extracted event details to the staging layer database table, “TranscriptStaging”.

Similarly, an event handler is placed on pre-execution to clear the previously stored event/transaction data from the staging database table before the newly extracting data are stored.

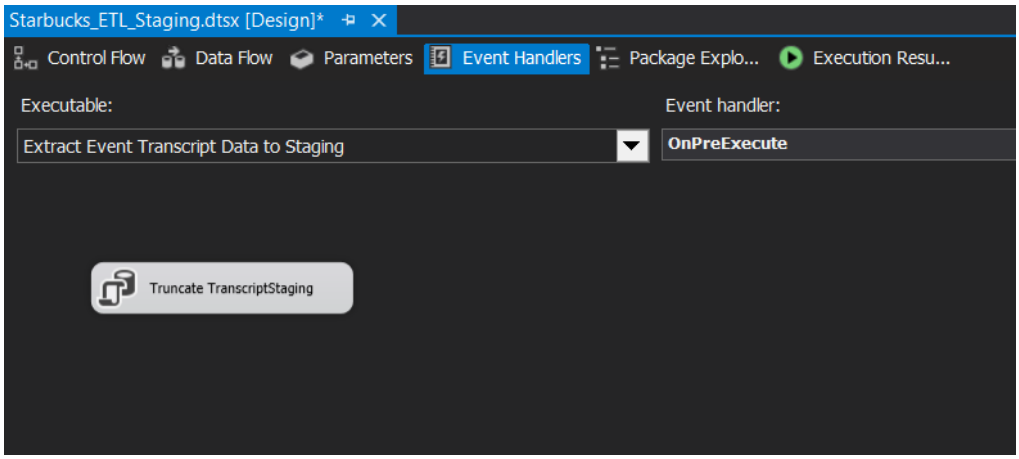



Figure 2.4.2.6. 2. Truncate TransactionStaging "execute SQL task" added to the list of event handlers as a OnPreExecute event handler

### 5.4.2.7. Extract Currency Exchange Rates API Data to Staging

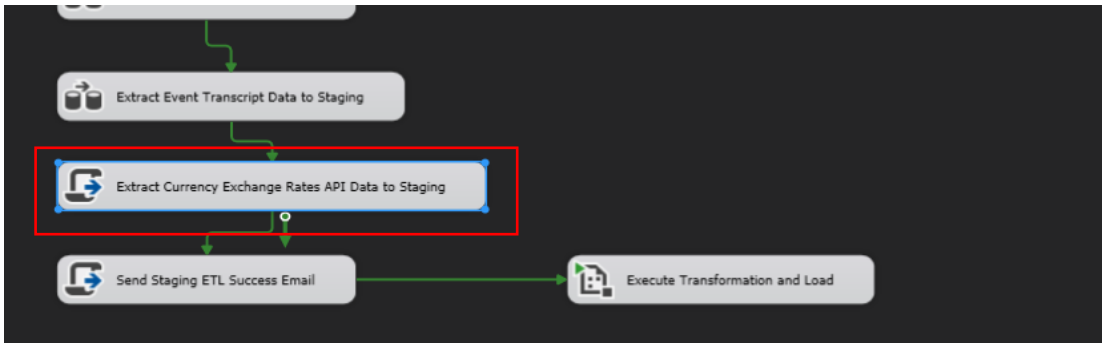



Figure 5.4.2.7. 1. Script task in place to get currency rates from an API

Since extracting and loading current exchange rate for USD to LKR and vice-versa is depending on an API call, a script task was used instead of a data flow task.[3][4][5] The script task does the API calling and writes the values taken from the API response to the staging layer database called "ExchangeRatesStaging" table. There are no event handlers for this particular Script task. Instead of using an event handler, the ExchangeRatesStaging table is being truncated via the same stored procedure which the script task calls and passes the newly retrieved currency exchange rates to be written to the relevant table.[6]

Stored Procedure called by the Script task has been named "`LoadExchangeRatesToStaging`" and the values that are being passed each time it is called are `@USDLKR, @LKRUSD, and @CURDATE` which are simply the USD to LKR rate, LKR to USD rate and the date of retrieval in that order.[6][8][9]

### 5.4.2.8. Send Staging ETL Success Email

The next task of the first EtL process is another script task which is used to send an email notification about the EtL task execution. If all the tasks up to this task have been completed means that all sources-to-staging data extraction and loading tasks have been executed successfully, thus, an email will be sent from the scrip task saying so and the process is explained in section 5.2 with one of the email notifications received during the testing of the EtL process.

In case if one of the tasks in above pipeline fails along the way, a slightly different email will be received by the recipients saying that the EtL from sources to staging has been failed. It is done by a different script task that was added to the list of event handlers OnError.

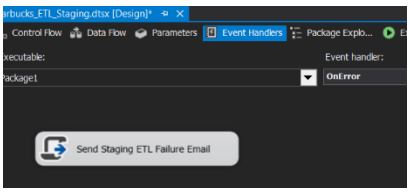



Figure 5.4.2.8. 1. Error Email triggering Script task in Package OnError event handlers list

### 5.4.2.9. Execute Transformation and Load

At the end of the first EtL process, (at the end of the execution of Starbucks_ETL_Staging.dtsx package) an Execute Package Task is added to execute the next package which contains the second ETL process from staging layer to data warehouse tables.


Figure 5.4.2.9. 1. Execute Package Task in place to trigger the execution of the next ETL package

## 5.5. Staging layer to Data Warehouse ETL

The second ETL process does more work than the first EtL explained earlier. As mentioned before, the earlier set of tasks had almost no transformations, data cleaning, data enrichments, addition of derived columns, performing measures, or not even merges and sorts. However, this process has a lot of transformations to be done to the data before they can be loaded into the data warehouse dimension and fact tables. Although the first EtL Task order was not that much important, the second set of ETL task execution order is the most important part. Thus, the order of the tasks has been well analyzed and planned rather than just having a set of interconnected tasks which will be simply useless and eventually they will fail due to having no logic while loading data.

To ensure that the integrity of the data will be saved, an execution order was planned and the ETL tasks were added accordingly. Unlike in the first set of EtL tasks in the previous section, the business keys and surrogate keys are used to populate derived key columns with relevant surrogate keys to preserve the links between dim and fact tables. Before any of these were done, a separate package was created to implement these named, Starbucks_ETL_DW.dtsx.

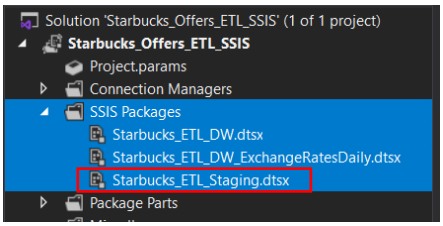



Figure 5.5. 1. Package for staging to data warehouse ETL implementation

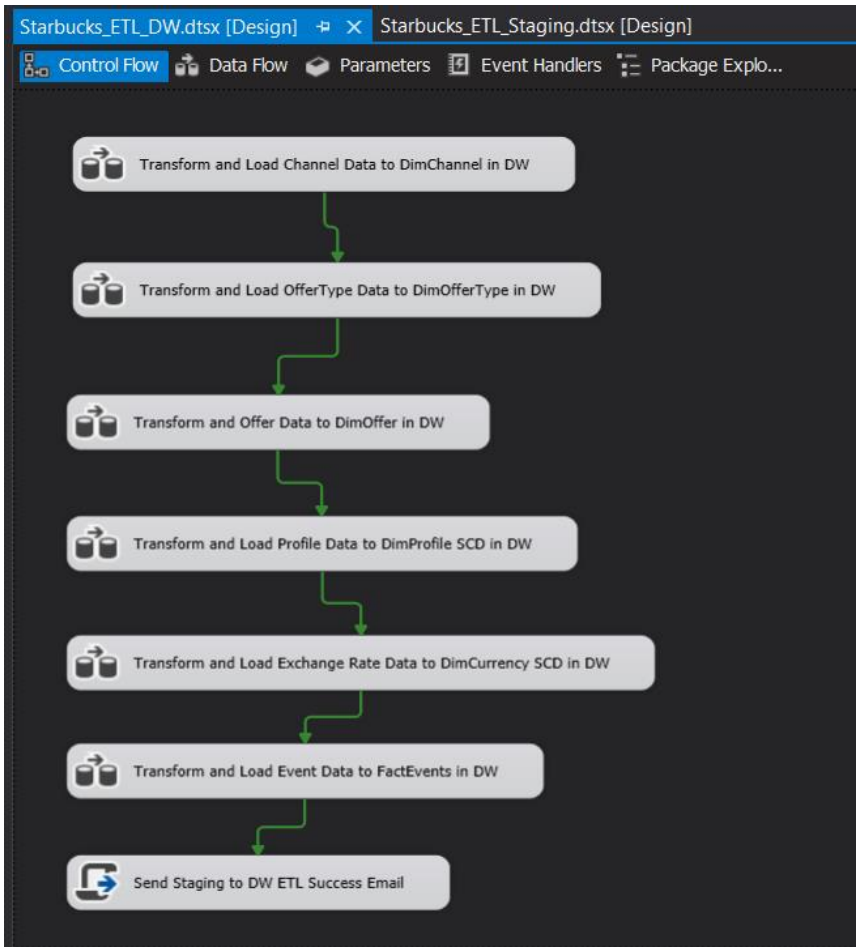



Figure 5.5. 2. overall flow of ETL tasks in the Starbucks_ETL_DW.dtsx package which is responsible for loading data to data warehouse after performing a set of complex transformations on the extracted data from the staging layer tables before loading.

Please note that the Data Flow tasks have not been assigned any event handlers to truncate any of the data warehouse tables because truncating data warehouse tables may cause issues of unexpected surrogate key changes which might be the root cause to end up with a faulty set of data in the data warehouse. Data for all data warehouse tables are either updated or inserted only.

The only event handler present is a OnError event handler which runs a script task to send email notifications upon the package execution failures.

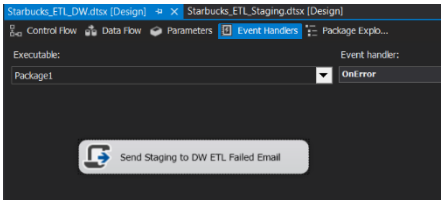



Figure 5.5. 3. Execute SQL Task in the OnError list of even handlers to send failure email notifications

### 5.5.1. Connections used

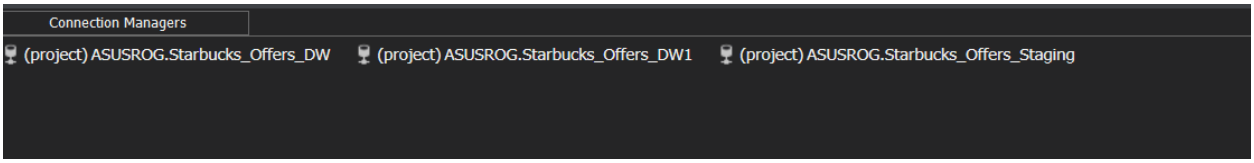



Figure 5.5.1. 1. List of connection Managers created to connect with the source file and the staging database in the Starbucks_ETL_DW.dtsx package. Note that no package level connections have been created and all connections shown are already available project level connection managers previously explained.

### 5.5.2. Tasks of ETL from Staging to Data Warehouse in their Execution Order

#### 5.5.2.1. Transform and Load Channel Data to DimChannel in DW

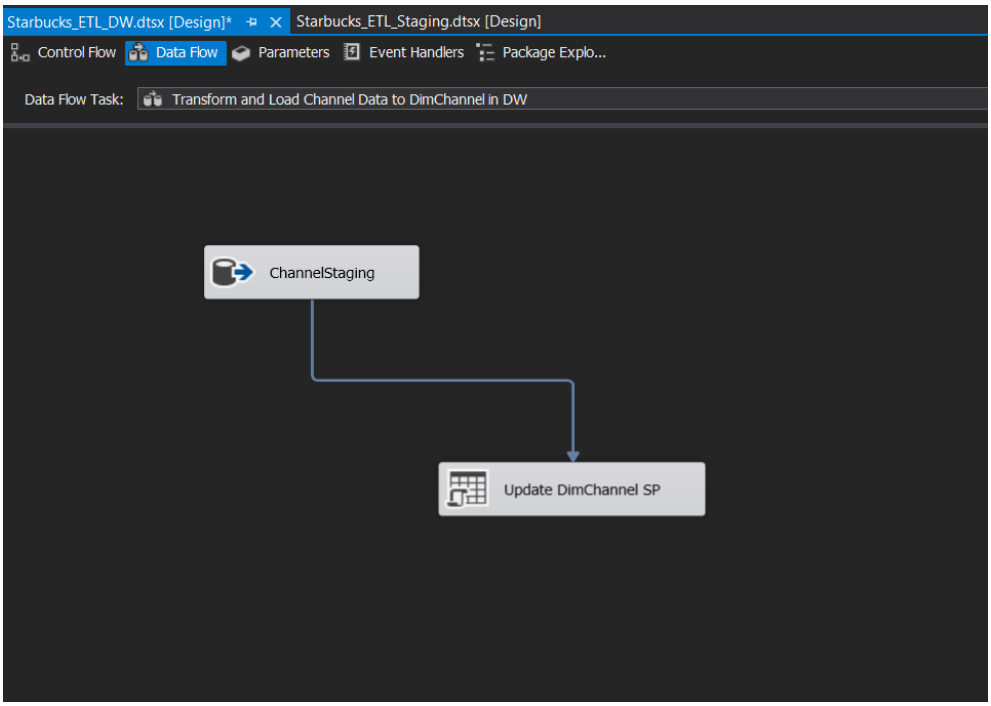



Figure 5.5.2.1. 1. Data Flow of the Data Flow Task "Transform and Load Channel Data to DimChannel in DW

As shown in the figure 5.5.2.1.1, an OLE DB source was added to extract the data from staging layer table “ChannelStaging” and since there are no transformations to be done, the extracted data is sent to an OLE DB command component to pass the data to a stored procedure in the data warehouse to do the update or deletion accordingly. The relevant stored procedure in this case is “UpdateDimChannel”. Stored procedure call: EXEC dbo.UpdateDimChannel ?, ?

### 5.5.2.2. Transform and Load OfferType Data to DimOfferType in DW

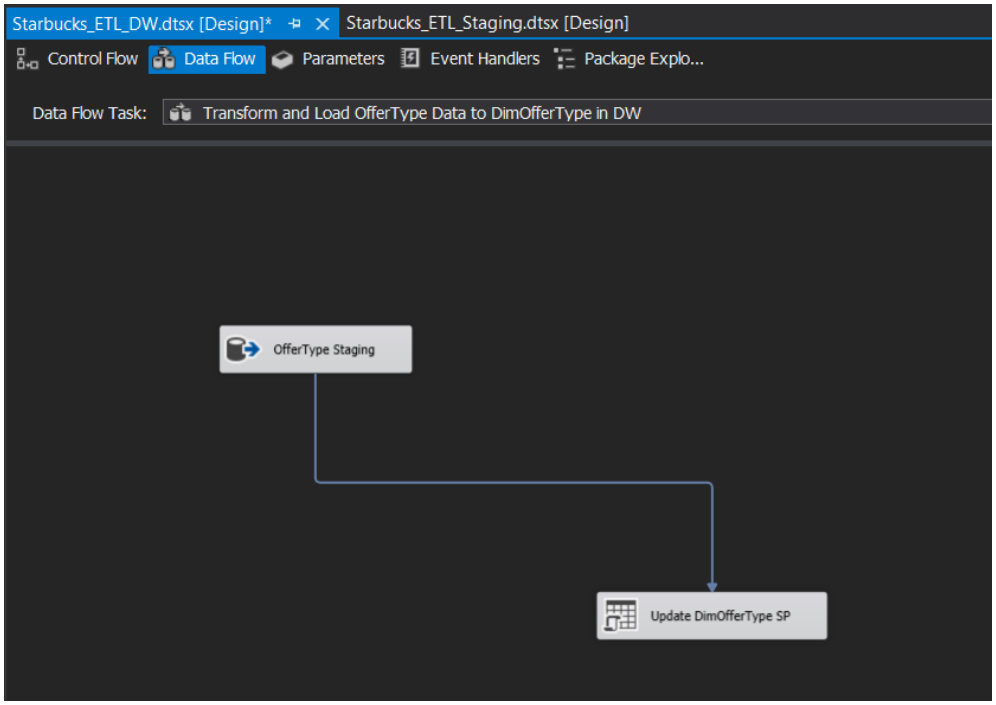



Figure 5.5.2.2. 1. Data Flow of "Transform and load OfferType Data to DimOfferType in DW" Data Flow Task

Like in the first data flow process, here also the Offer Type data extracted from the staging table "OfferTypeStaging" using the OLE DB Source component will not be transformed in any way. The extracted data is sent to an OLE DB Command to pass to the stored procedure in place within the Data Warehouse database, named "UpdateDimOfferType". The stored procedure is used to figure out if the incoming data is already present in the data warehouse table "DimOfferType" using the business key provided. Based on the existence of data, they will be either updated or inserted.

Stored procedure call:

```
EXEC dbo.UpdateDimOfferType ?, ?
```

Based on the passed offer_type_id, which is the business key of the offer_type table in source tables, (Same name in the staging layer database table as well), the stored procedure will run a simple if condition and check if there exists a matching row. If exists, the data will get updated and in the opposite case where the row is not found, the stored procedure will simply run an insert query to insert the newly received data. InsertDate and ModifiedDate are also either inserted or updated based on the same logic.

### 5.5.2.3. Transform and Offer Data to DimOffer in DW

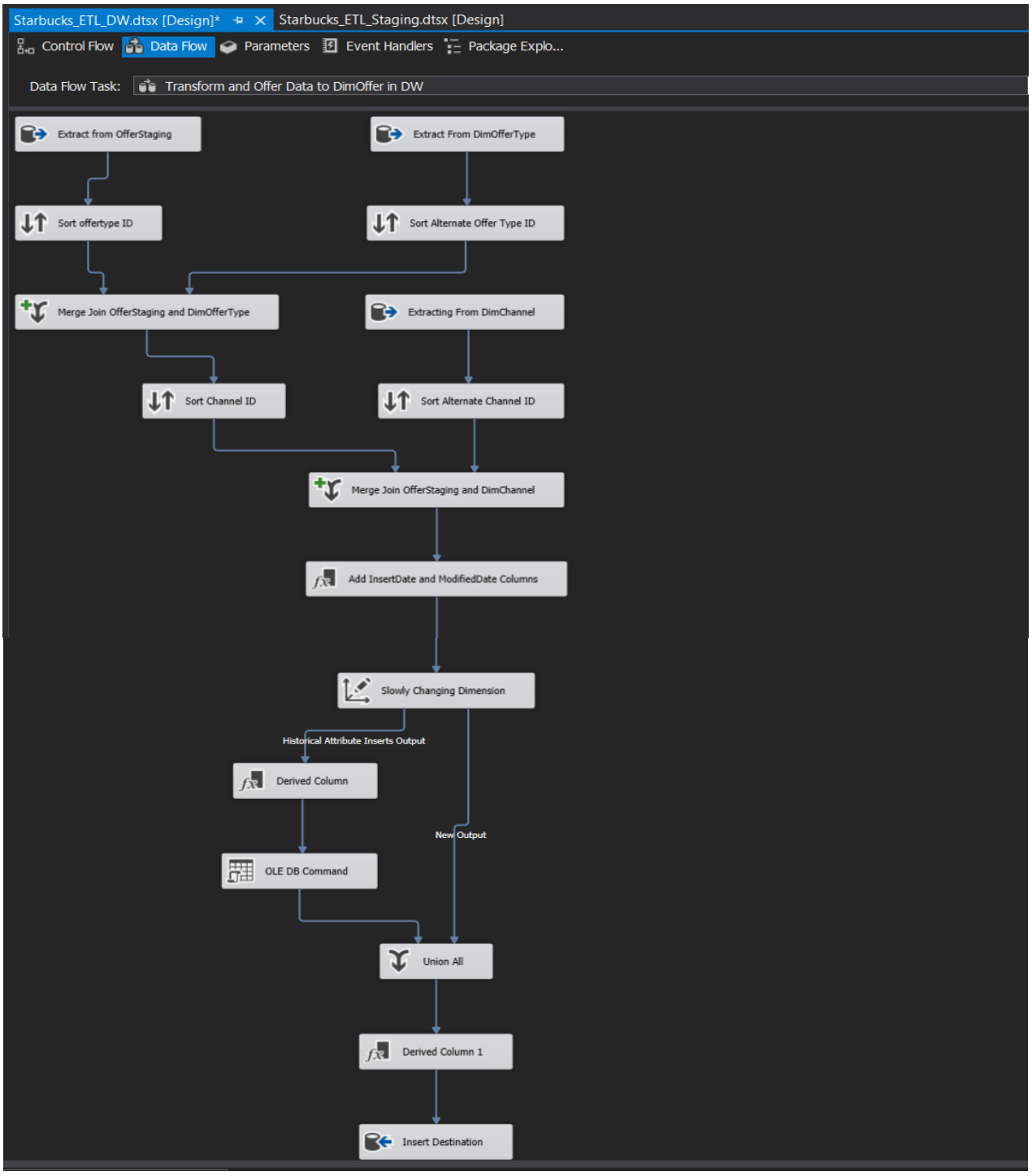



Figure 5.5.2.3. 1. Data Flow of "Transform and Offer Data to DimOffer in DW" with Data Transformations in place which is loaded to the DimOffer Slowly Changing Dimension.

Data Flow of the "Transform and Offer Data to DimOffer in DW" Data Flow Task is a bit complicated than the previously explained data flows since it has to load data in to a slowly changing dimension table in the data warehouse, namely, the "DimOffer" table.

Since DimOffer table has connected to DimOfferType and DimChannel dimensions as shown in the Figure 4.1 – the snowflake schema followed by the Starbucks_Offers_ DW data warehouse, DimOffer table required the Surrogate keys of each table. To do this there was two ways, 1- using lookups, or 2-using sort + merge join. In this case, the method used is sort + merge join.

**Getting required keys using Sort and Merge join**

Thus, at first, offer data is extracted from the OfferStaging table using an OLE DB source which is connected to the staging database and DimOfferType data which were already loaded to the data warehouse in the previous step are also extracted using another OLE DB source from the data warehouse. Both table data extracted are then sorted using the **Sort** component by offer type id (AlternateOfferTypeId in the DimOfferType table) to speed up the merging process. After that both sorted data of two tables are joined using a **Merge Join** component which performs a left outer join to preserve offer data in case a matching row in the DimOfferType table is not found. Then from the output, OfferTypeSK as the key and the offer_type_id column is ignored since its no longer needed. Similar task is done for retrieving the matching ChannelSK from the DimChannel table in the data warehouse and the resultant columns are sent through a **Derived Column** component to create necessary derived columns.

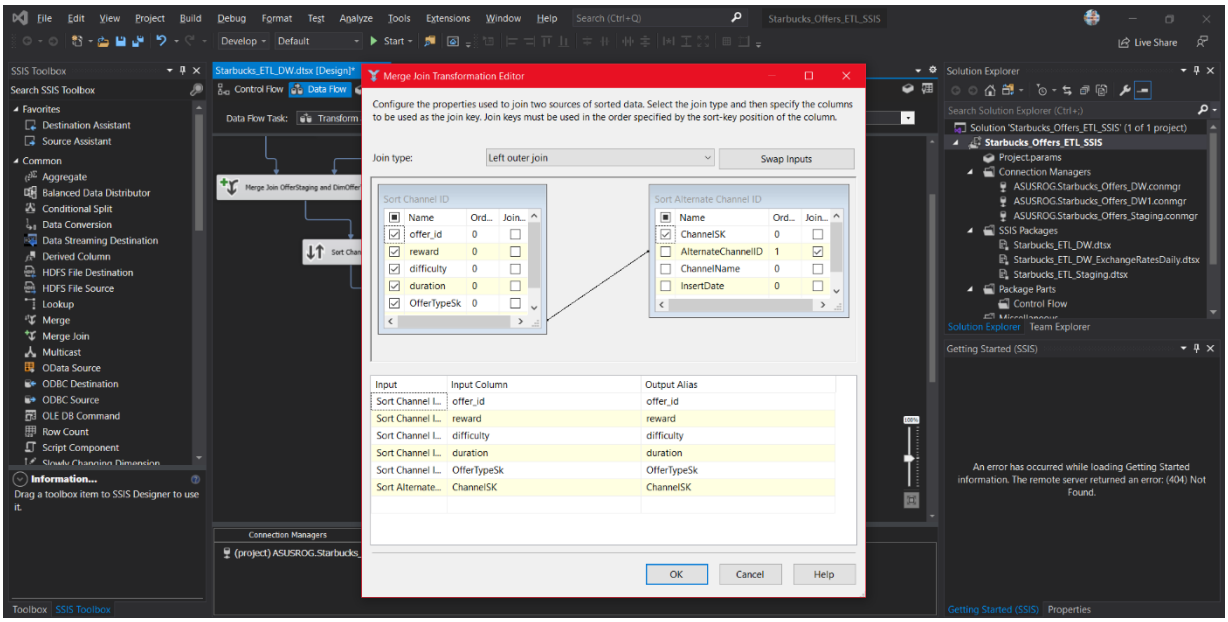



Figure 5.5.2.3. 2. Selecting ChannelSK from the merge join component after a left outer join of tables OfferStaging and DimChannel to get relevant channel SK and ignore the original channel id as it is not inserted at last.

### Adding derived timestamp columns

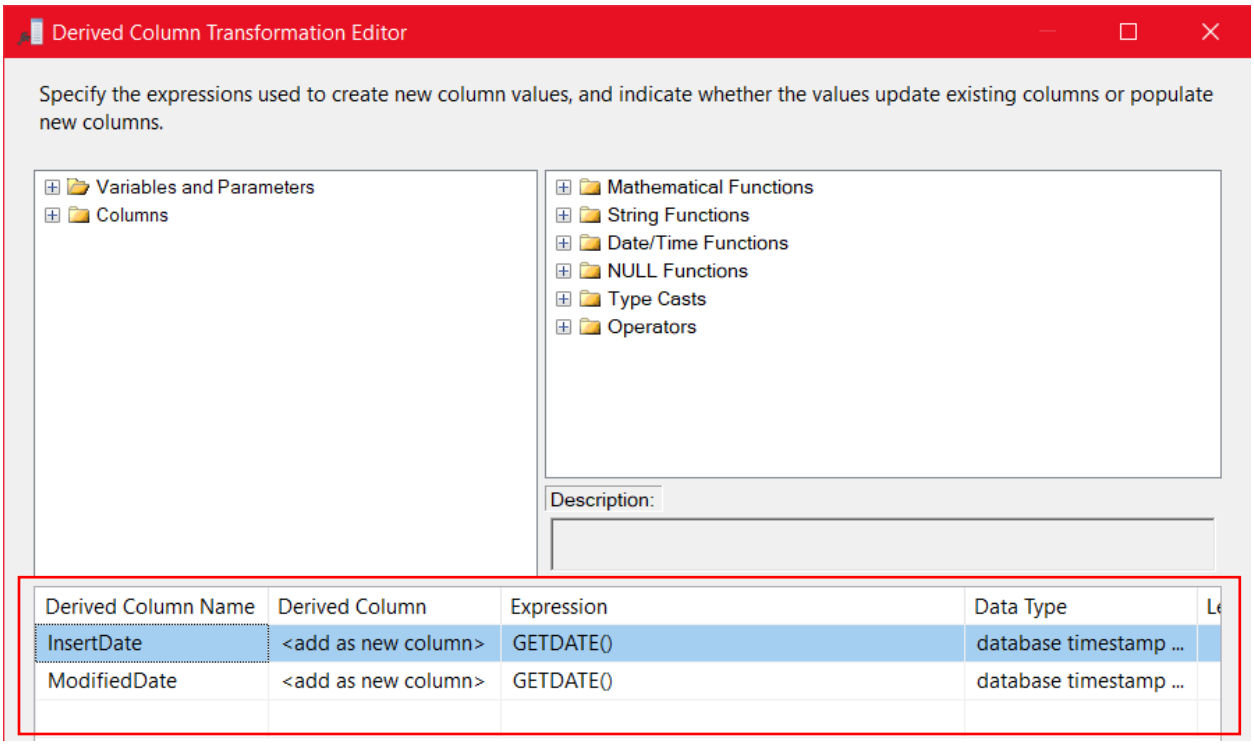



Figure 5.5.2.3. 3. new derived columns with current timestamp assigned

Derived columns have been simply specified and the value is taken as the current timestamp from the provided list of functions by the Derived Column Component by default.

### Allowing Slowly Changing Feature

It is assumed that historical values of DimOffer table should be preserved since the same offer can have different number of rewards, difficulties, or durations from time to time. This will allow the data analytics to be performed based on offers and how their attributes have changed overtime, which period an offer had best interactions and find out what were the rewards, difficulties, and durations back then, and more.

To make enable the feature, a slowly changing dimension component has been used next, and it was configured by specifying business key and the set of historical values to keep an eye on when loading new set of data into the warehouse table, “DimOffer”. The type of the DimOffer table record is set to Type 2 by configuring two timestamp columns, StartDate and EndDate. After that, the auto generated OLE DB Command which updates an existing row when a new record is about

to get inserted, the SQL Command was modified to update the ModifiedDate column as well which is the derived column added in the previous step. The Modified SQL Command is:

```
UPDATE [dbo].[DimOffer] SET [EndDate] = ?, [ModifiedDate]=? WHERE [AlternateOfferID] = ?
AND [EndDate] IS NULL
```

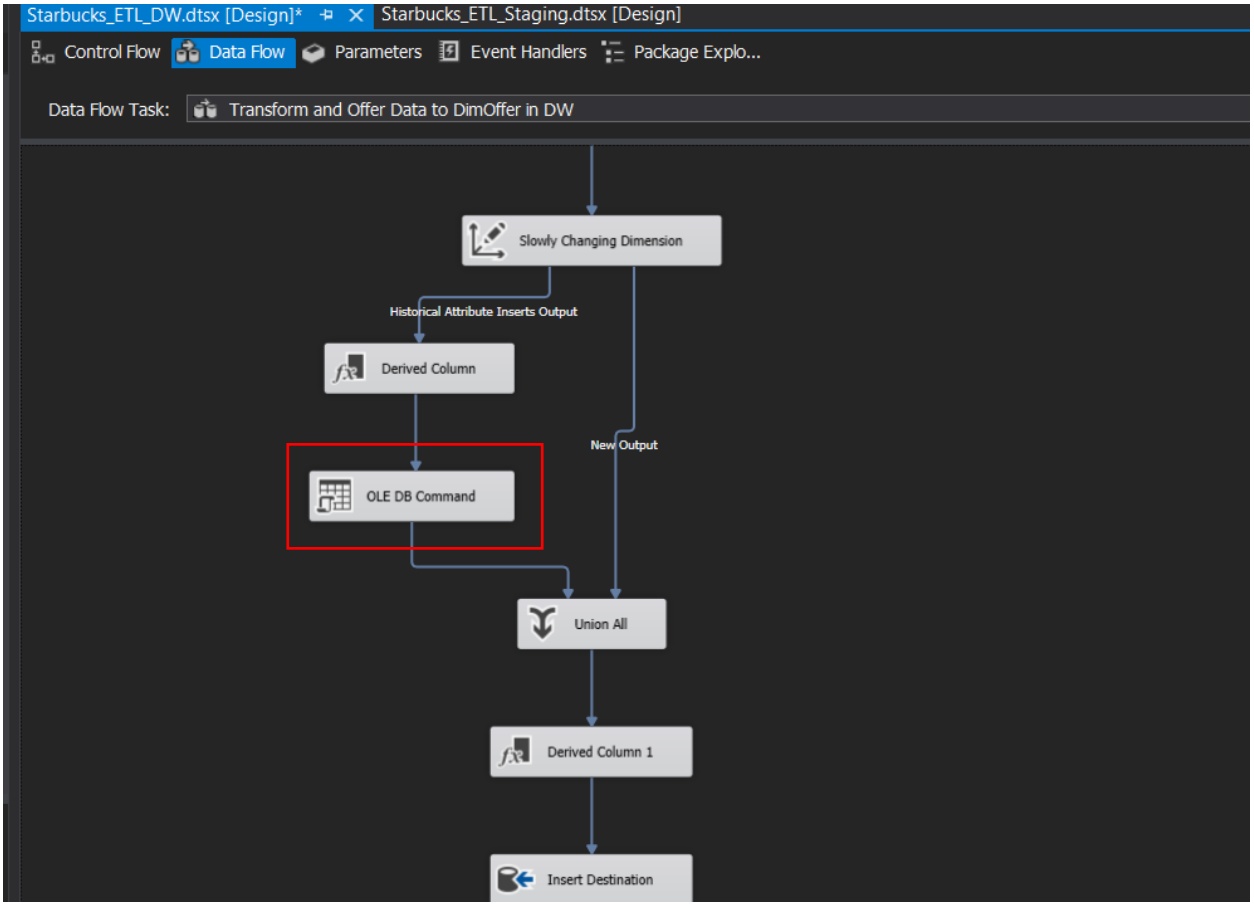



Figure 5.5.2.3. 4 OLE DB Command where the SQL Query modifications were done in order to update the ModifiedDate as well in the expiring row when a new row with updated values are about to get inserted.

Since DimOffer table is assumed not to have any changing columns (only have fixed and historical columns), there are only two outputs have come out of the slowly changing dimension component. In case there are changing columns are present, there will be three outputs out of the SCD component where two OLE DB SQL queries should be modified to correctly insert the ModifiedDate of the relevant rows.

### 5.5.2.4. Transform and Load Profile Data to DimProfile SCD in DW

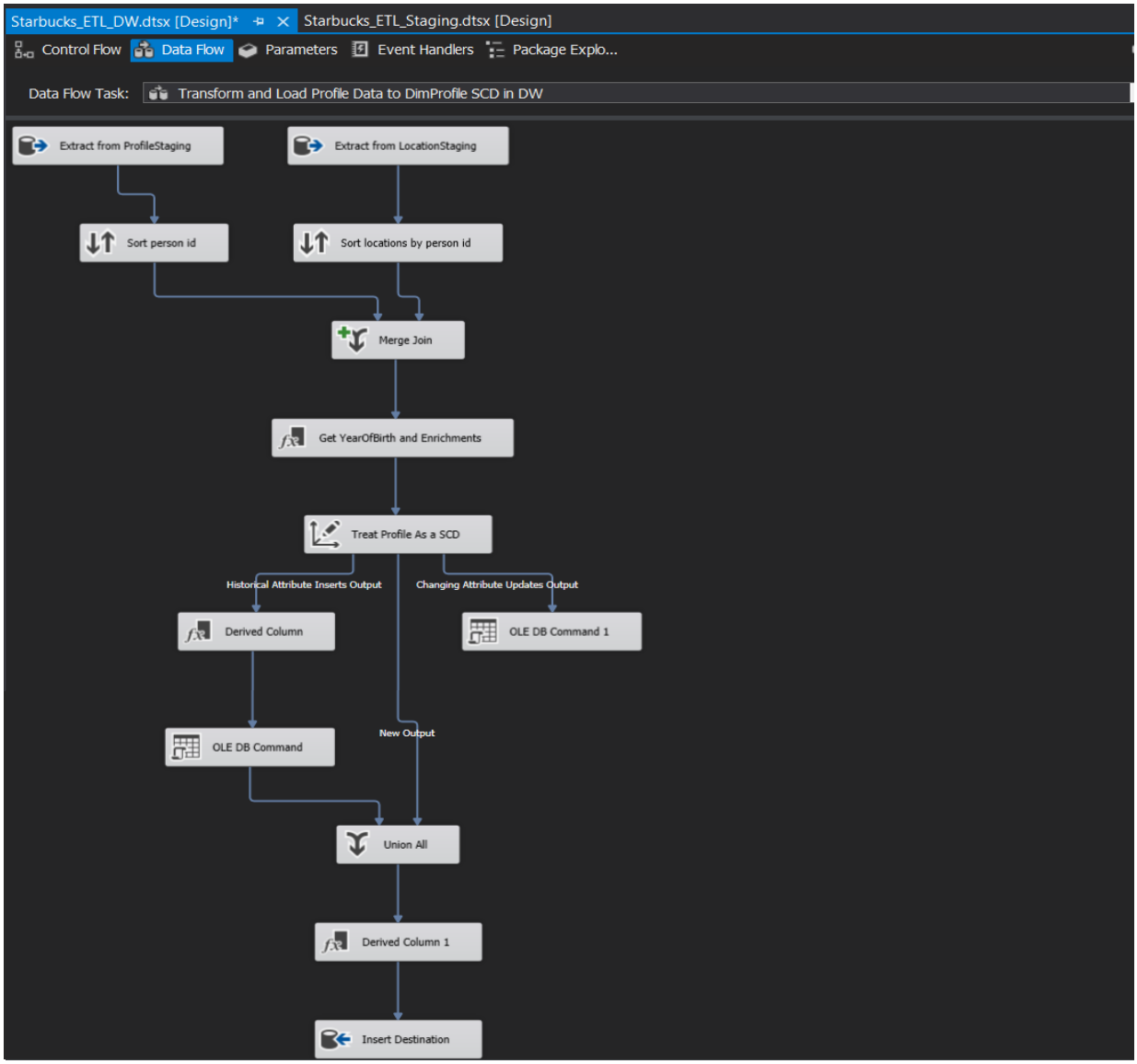



Figure 5.5.2.4. 1. overall data flow of "Transform and Load Profile Data to DimProfile SCD in DW" Data Flow Task component which is used to load ProfileStaging and LocationStaging data to DimProfile SCD

This Data Flow Task is focused on joining the ProfileStaging and LocationStaging tables extracted from the Staging database in order to bring the customer and address data together and prepare them to be loaded to the DimProfile dimension table in the Data Warehouse. There are no other dimensions that the DimProfile is connected to get any surrogate keys, thus, lookups are not necessary in this case.

### Merging Profiles and Locations

First two staging database tables were extracted using two OLE DB Source components and both are sorted using a dedicated **Sort** component at each end. Sorting is done to aid faster joining of the two tables. Sorting is particularly done according to the person_id in both tables and the sorted rows of two tables are joined using a **left outer join** configured using a **merge join** component where the left table is chosen as ProfileStaging table to preserve customer data even if a matching address is not found in the LocationStaging table data.

In the merge join, the list of output columns was carefully ticked, leaving out the recurring person_id by only ticking one person_id column.

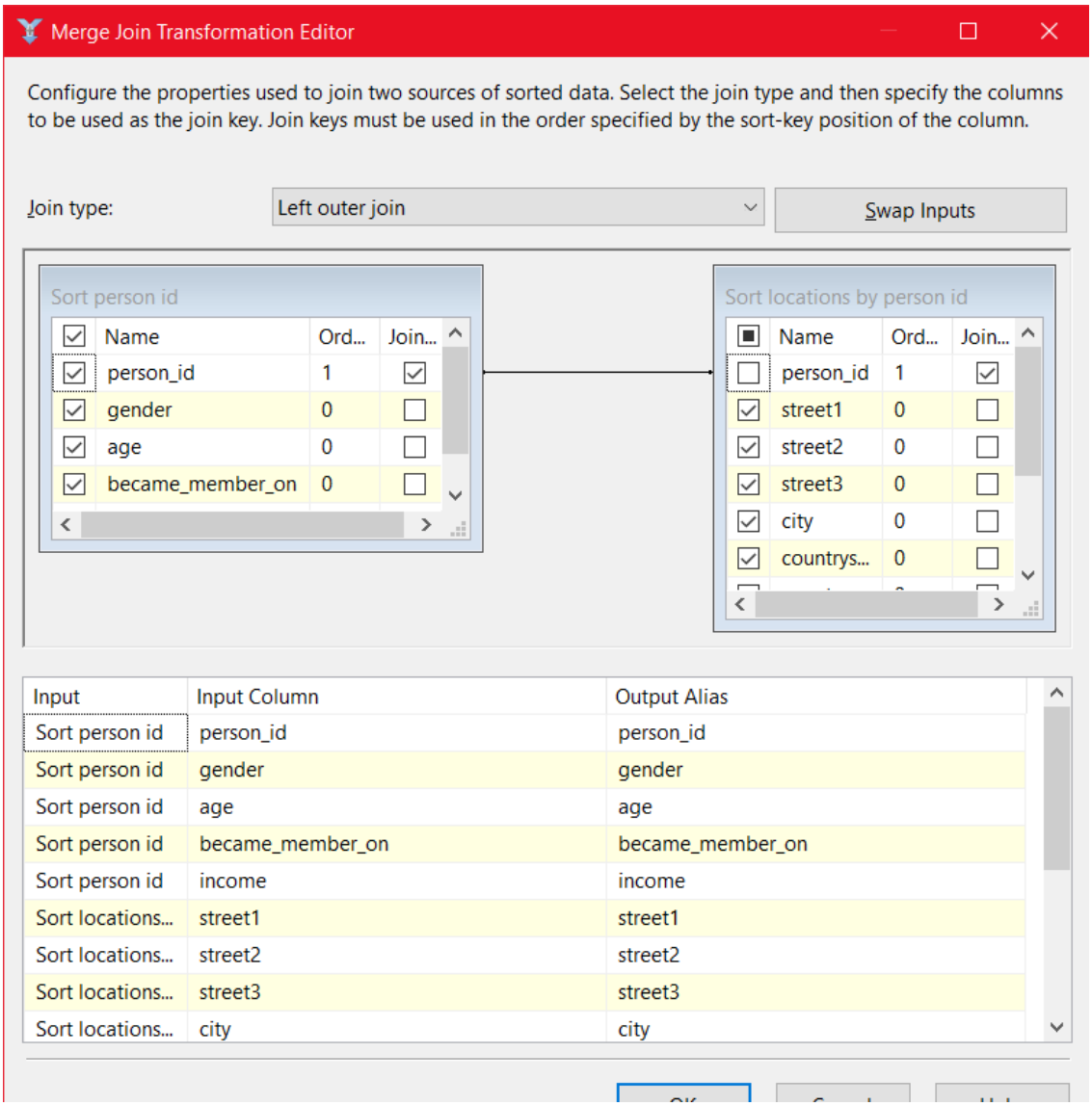



Figure 5.5.2.4. 2. selecting the output columns taken after joining two staging tables using person_id

**Derived Columns and Data Enrichment**

Next, the resultant output of the merge join above were given as input to a new Derived Column component named “Get YearOfBirth and Enrichments” which is used to derive a new column based on the age and the membership date when a customer created a Starbucks account, assuming that the age was provided during the registration process. Other derived columns such as InsertDate, and ModifiedDate are also created in this component alone.

Since previously it was identified that the Gender column of the ProfileStaging table had no character assigned for customers who have not provided a Gender at the time of the registration, as a step of **data enrichment**, the gender was set to “NA” (which stands for Not Assigned) where it was empty.

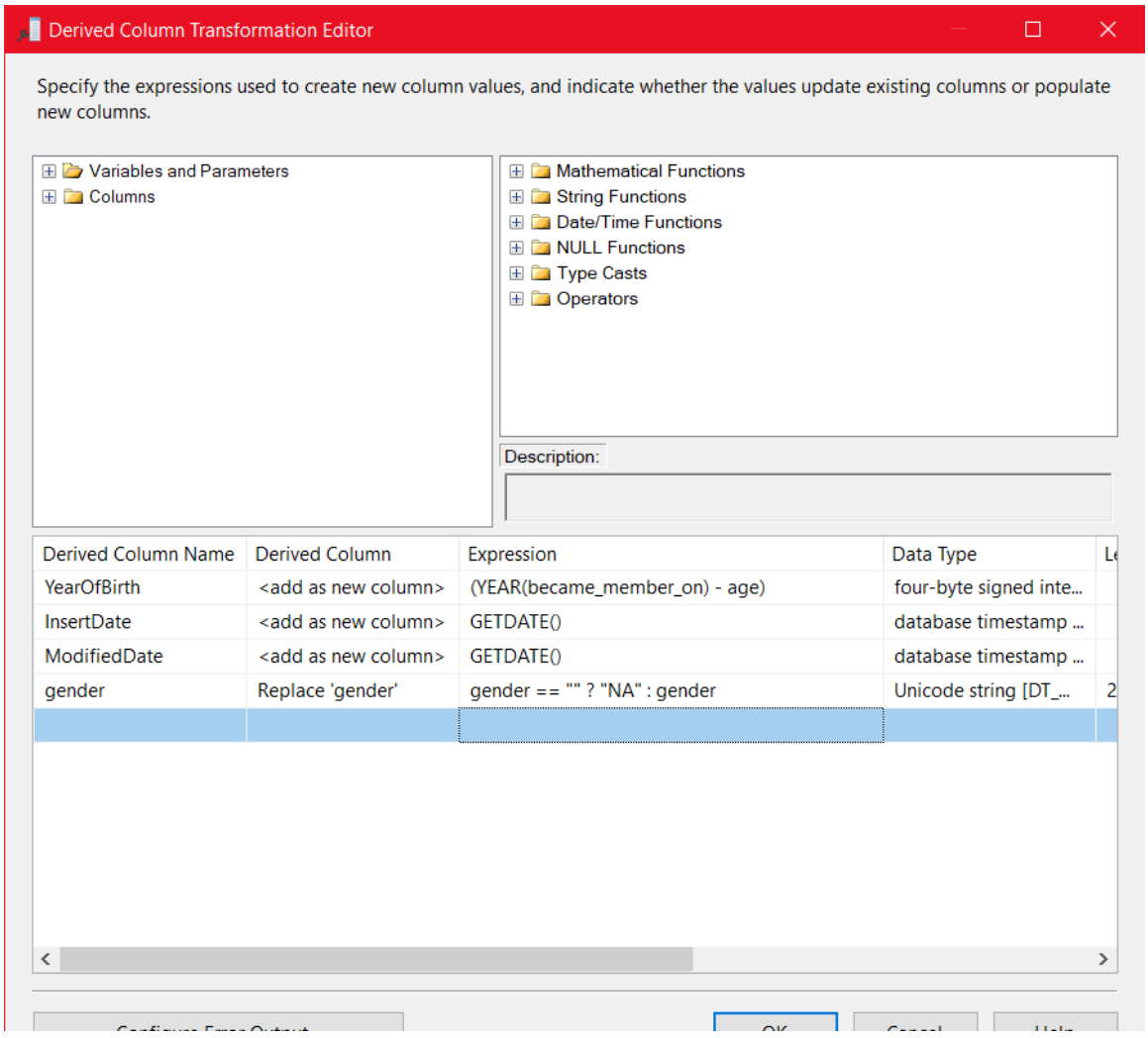



Figure 5.5.2.4. 3. Generating derived columns and Enrich data provided in the Gender column using a proper Expression.

**Enable DimProfile Slowly Changing Feature**

Like in the previous DimOffer table, DimProfile Dimension table is also a Slowly Changing dimension as it was assumed that Starbucks is interested in the locations of the customers and performs analysis based on the locations (Addresses). DimProfile table has fixed, changing and historical columns thus the slowly changing component used here has given three outputs based on those columns specified during the configuration.

Age, Gender, BecameMemberOn, Income, Latitude, Longitude, PostalCode, Street1, Street2, Street3, and YearOfBirth columns were set to be Changing Columns which means if one these column values were changed, they will simply get updated in the existing record itself.

City, CountryCode, CountySubDivisionCode were set as Historical Columns which means if at least one of these columns were updated in an existing row in DimProfile, the existing row will be expired and a new row will be inserted, preserving the history of the location based on main columns mentioned earlier. All other non-specified columns were considered as Fixed Columns.

Type of the SCD is set to Type-2 by indicating a StartDate and an EndDate for expiring the rows upon historical column value changes.

OLE DB Command and OLE DB Command 1 components which were auto generated to handle above-mentioned scenario were update with the ModifiedDate included in their SQL Query to be able to correctly update the ModifiedDate of the DimProfile table records.

Updated SQL Queries are as follows:

OLE DB Command Component

```
UPDATE [dbo].[DimProfile] SET [EndDate] = ?, [ModifiedDate]=? WHERE [AlternatePersonID] = ? AND [EndDate] IS NULL
```

This update the modified date of the expiring record when a new record is about to be inserted when historical column updates have been identified.

OLE DB Command 1 Component

```
UPDATE [dbo].[DimProfile] SET [Age] = ?,[BecameMemberOn] = ?,[Gender] = ?,[Income] = ?,[Latitude] = ?,[Longitude] = ?,[PostalCode] = ?,[Street1] = ?,[Street2] = ?,[Street3] = ?,[YearOfBirth] = ?, [ModifiedDate]=? WHERE [AlternatePersonID] = ? AND [EndDate] IS NULL
```

This SQL query was designed to update changing column values when updated values are received and it was modified to update the ModifiedDate as well.

Finally, the changes will get either inserted or updated in the DimOffer dimension according to the specified mappings done in the OLE DB Destination component in the end.

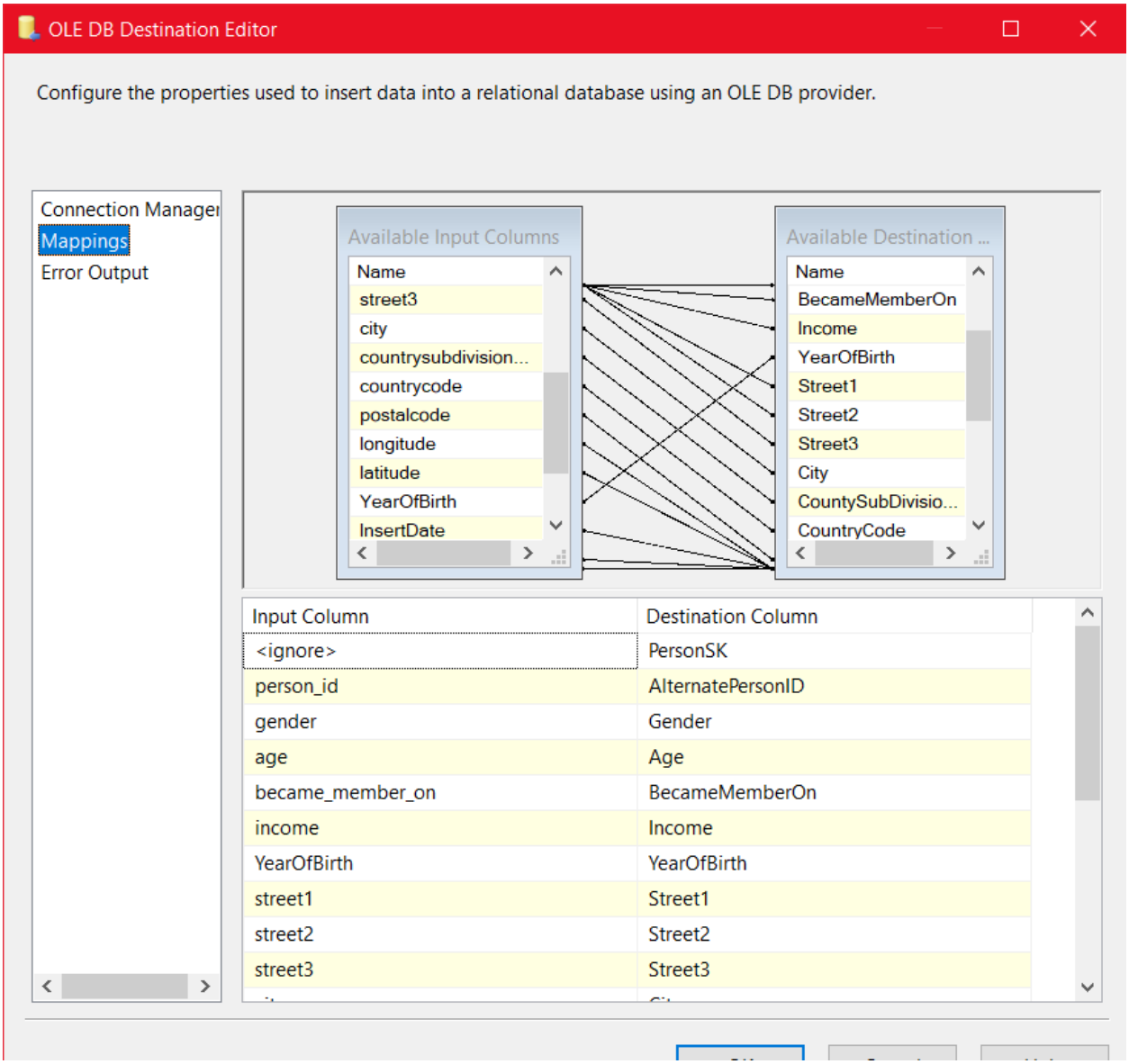



Figure 5.5.2.4. 4. Final column mapping done after the slowly changing dimension component was configured

### 5.5.2.5. Transform and Load Exchange Rate Data to DimCurrency SCD in DW

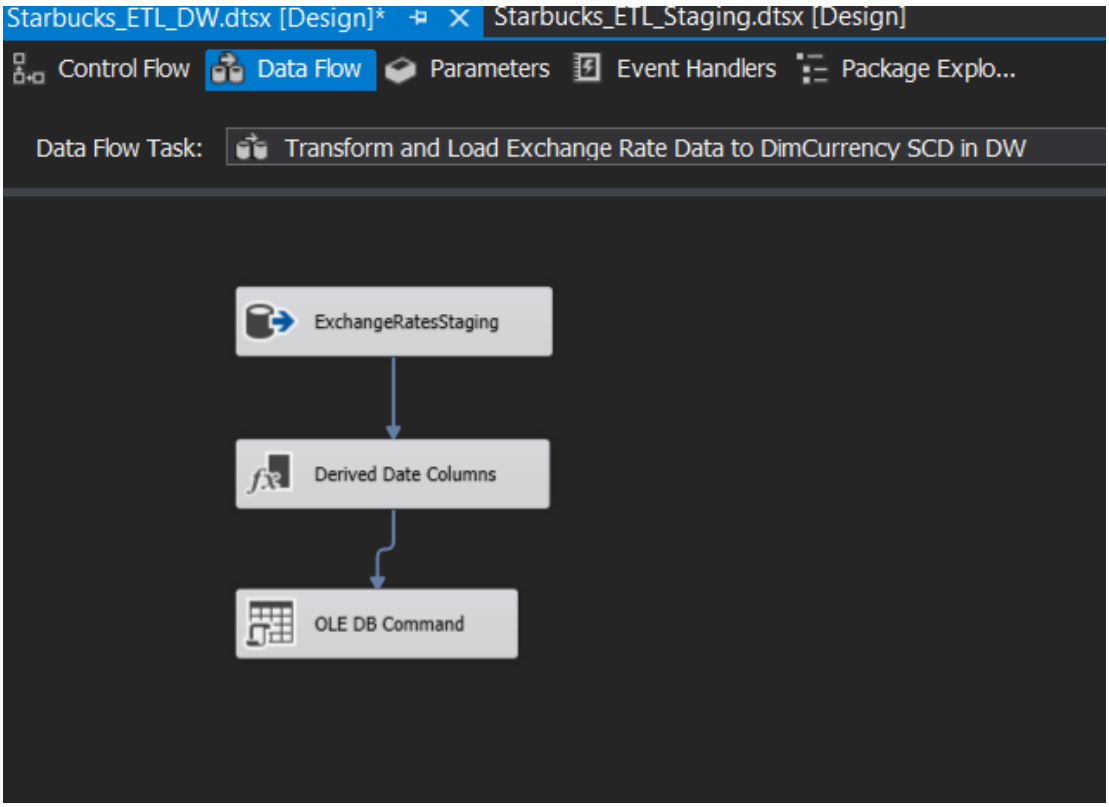



Figure 5.5.2.5. 1. Data Flow of "Transform and Load Exchange Rate Data to DimCurrency SCD in DW" Data Flow Task

In the above Data Flow, the data from ExchangeRatesStaging table are extracted which will be the updated exchange rates. Then two derived columns InsertDate and ModifiedDate columns were specified to get the current timestamp. Finally, instead of using an OLE DB Destination to load data, an OLE DB Command has been used. The reason for this is DimCurrency dimension table having default exchange rate value 0 for dates until 2098 pre-inserted just like in the case of a static dimension.

Exchange rates for all upcoming dates for a large period is pre-inserted specially for the fact table to be able to get a surrogate key reference even if there are no valid exchange rate specified at the time. This eliminates the complicated scenario where there should be a complicated logic should be implemented to insert a exchange default exchange rate first before continue to insert the record in the fact table just to be able to have a surrogate key reference to reference the DimCurrency table later when calculations will be done.

Having pre-inserted static default values will provide a valid surrogate key as a reference even when a valid exchange rate is not specified and when the exchange rate is finally retrieved it can be updated in the relevant row in the DimCurrency table. Thus, the OLE DB Command is used to

call a stored procedure which enables updating an already existing value for a newly retrieved exchange rate.

DimCurrency table can be considered as a Slowly Changing Dimension although the StartDate or the EndDate have not been configured using a slowly changing dimension component. Historical values can be easily identified using the ExchangeDate which acts as either the StartDate or EndDate where past dates in ExchangeDate are considered as Expired by default.

### 5.5.2.6. Transform and Load Event Data to FactEvents in DW

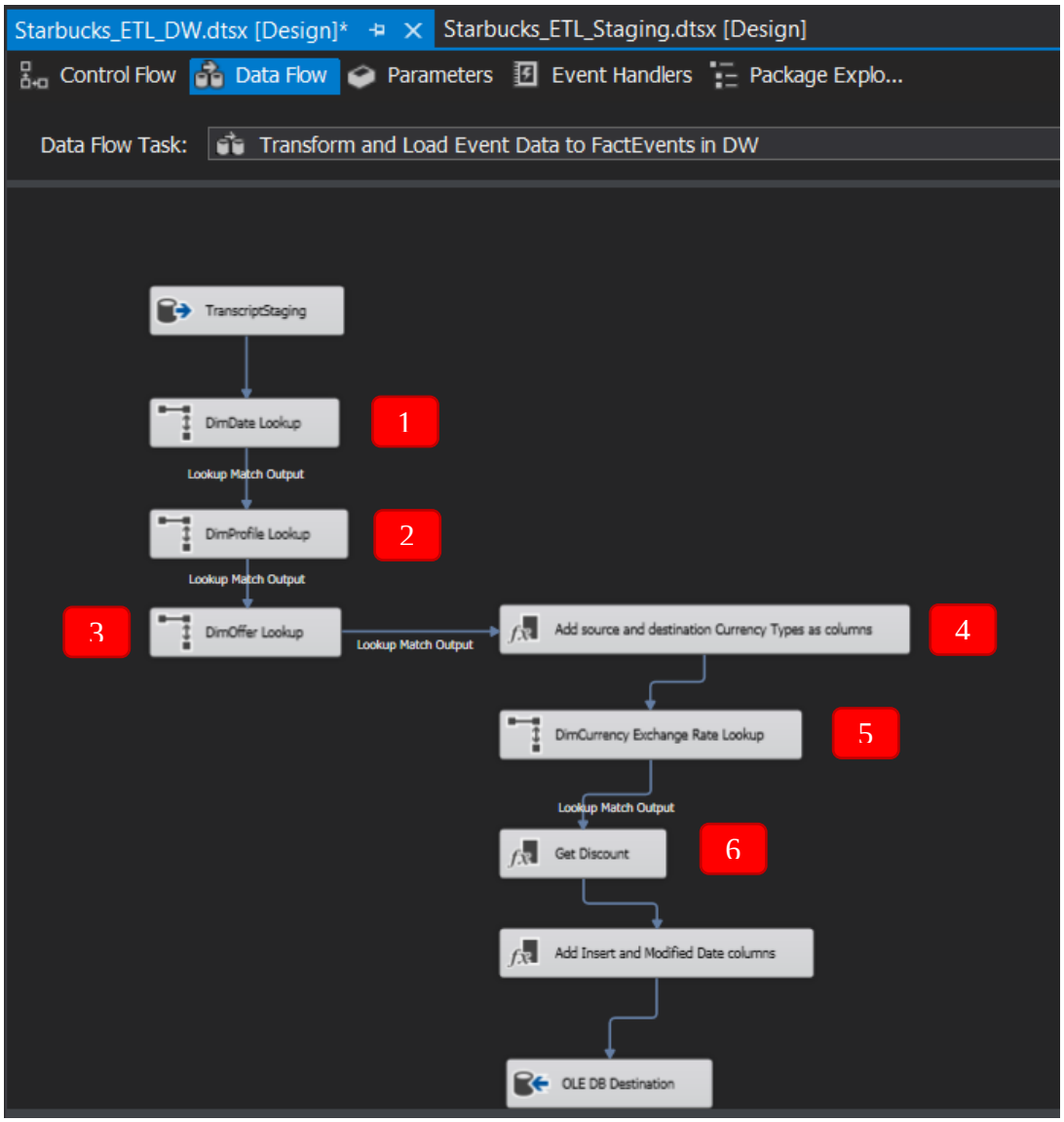



Figure 5.5.2.6. 1. Data Flow of " Transform and Load Event Data to FactEvents in DW " Data Flow Task. Special steps have been numbered to easily refer later in the description

This Data Flow belongs to the ETL step of loading the Fact Table. Since the fact table refers a lot of dimension tables lookups were used (instead of using sort + merge join like in previous DimOffer case) to get the corresponding surrogate keys to establish table references.

After extracting event records from TranscriptStaging table in the staging database using OLE DB Source, 1st Lookup is done to retrieve relevant date key for the corresponding event date.

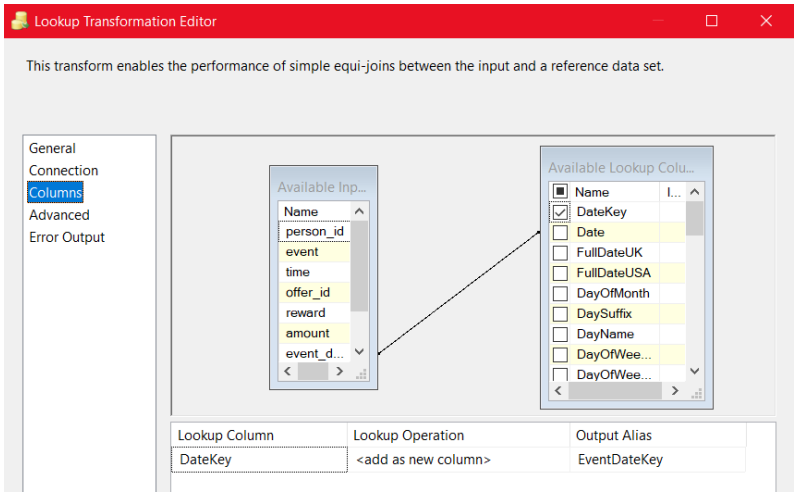



Figure 5.5.2.6. 2. Retrieving DateKey from DimDate using a lookup by looking up the event_date

2nd Lookup is done to retrieve the PersonSK from DimProfile by comparing the business key of a customer which is person_id.

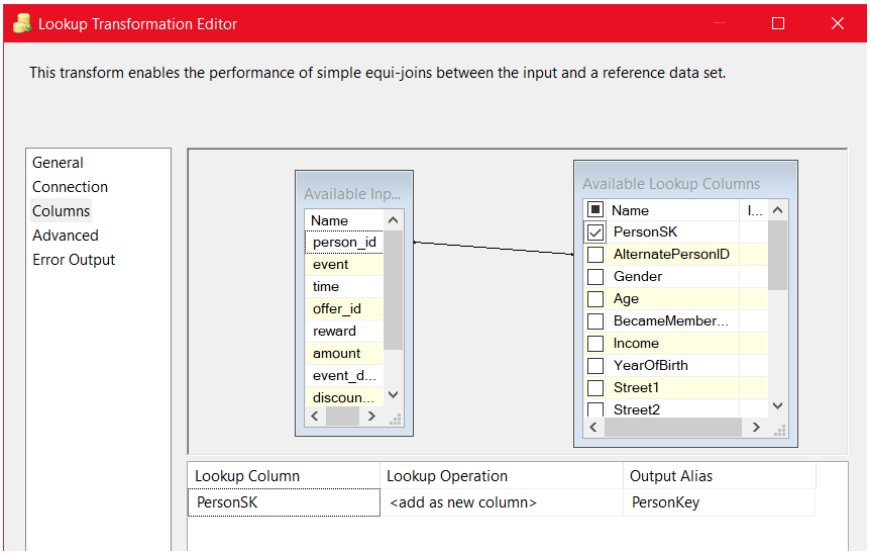



Figure 5.5.2.6. 3. Retrieving surrogate key from DimProfile using its business key, person_id using the Lookup

3rd Lookup is done to retrieve relevant OfferSK from DimOffer by providing the business key "offer_id"

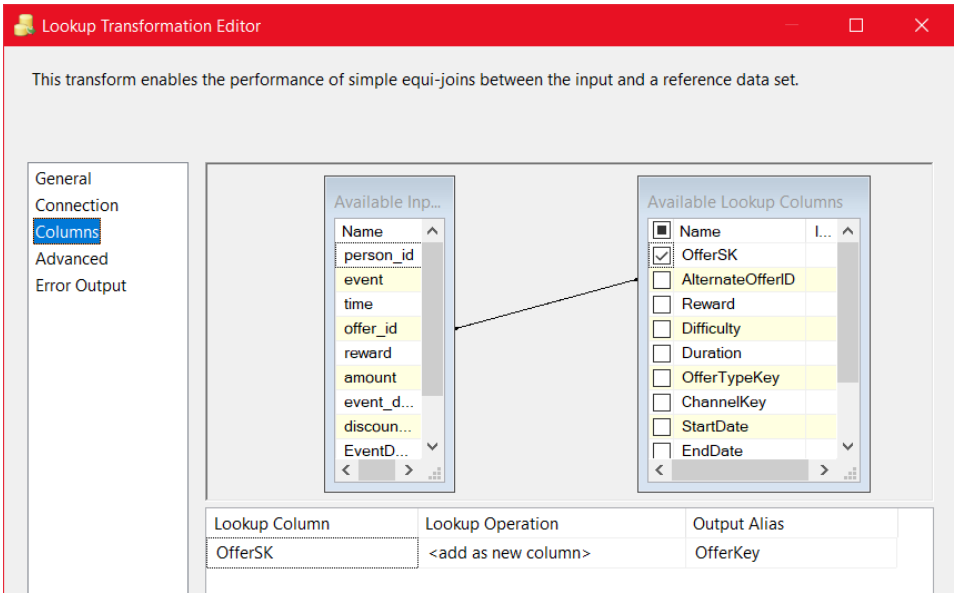



Figure 5.5.2.6. 4. Looking up and retrieving the OfferSK from DimOffer

Before getting the relevant exchange rate from USD to LKR, to be able to get that, first it is required to have a valid business key. In DimCurrency, the business key is SourceCurrencyType + DestinationCurrencyType + ExchangeDate. (a partial business key). Since it is always required the exchange rate for USD to LKR since all the transactions are done in USDs, the required business key of the DimCurrency can be easily generated by adding USD and LKR as values for two new temporary derived columns called SourceCurrency and DestinationCurrency while the ExchangeDate is the same as event_date which is already present.

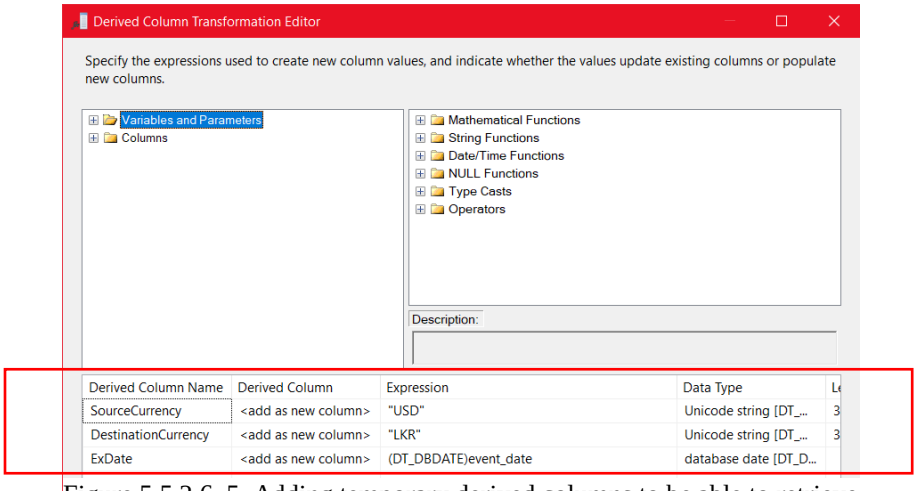



Figure 5.5.2.6. 5. Adding temporary derived columns to be able to retrieve the CurrencyRateSK in the next step

As shown in the figure 5.5.2.6.5., three temporary derived columns were created especially with event_date converted to DT_DBDATE since time portion is not considered when comparing the dates.

5th Step (Numbered above) Looks up the DimCurrency table using the newly derived columns provided as the business key and retrieves the CurrencyRateSK which will be used in future currency conversions.

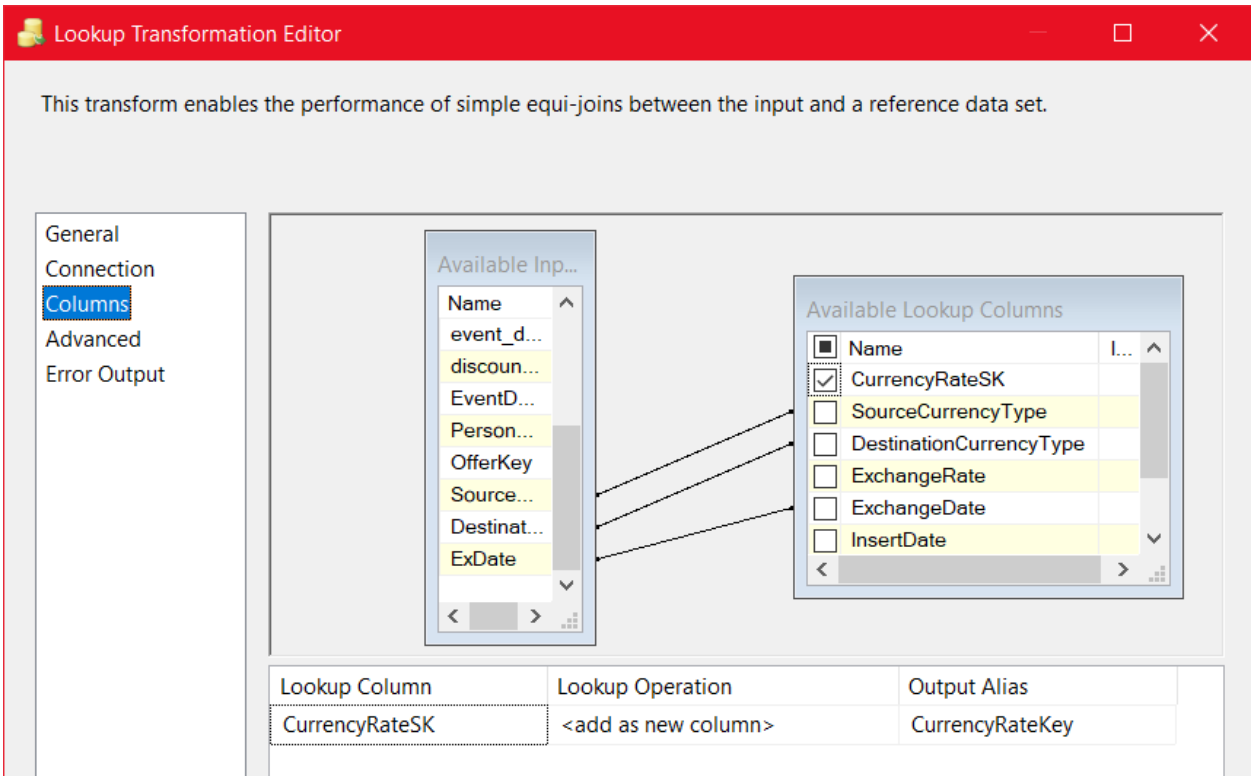



Figure 5.5.2.6. 6. Looking up the CurrencyRateSK using the derived columns as the business key

6th step mentioned in the Data Flow figure is referred to calculate measures using values available in the FactEvents fact table. Main calculation done is based on the discount rate and according to the given rate the discounted amount and the actual transaction amount paid by the customers are identified. These values are included in a new set of derived columns so that they can be inserted into the FactEvent Fact Table at the end.

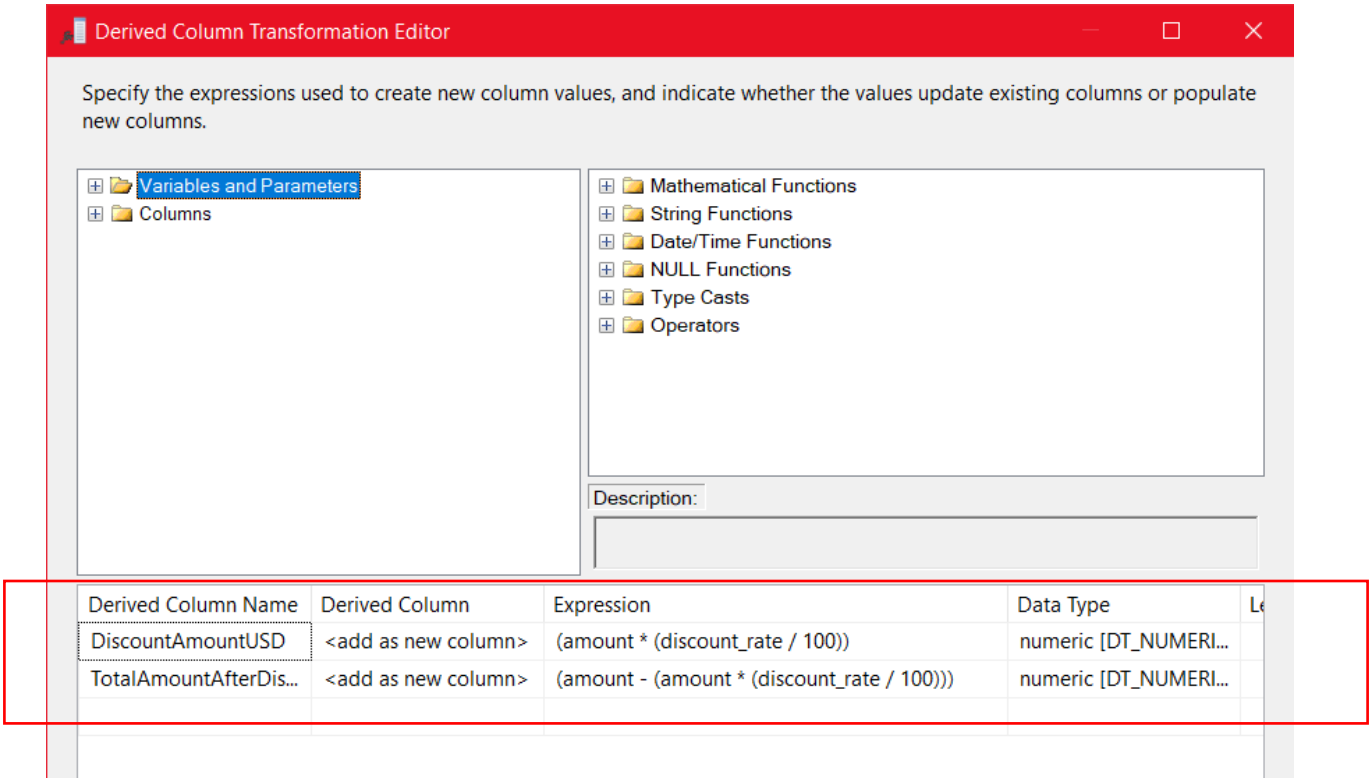



Figure 5.5.2.6. 7. Calculations done based on discount rate and amount given

Finally, another derived column component is used to derive InsertDate and ModifiedDate and the output is sent to an OLE DB Destination which refers the FactEvents in the data warehouse where transactions will get inserted according to the mapping of the columns done.

### 5.5.2.7. Send Staging to DW ETL Success Email

Last Script Task in the Control Flow is used to send a successful email notification upon the successful execution of the staging-to-data warehouse ETL process. The Format of all email notifications will have the following format.

**Subject:** either success/failure with stating whether its EtL(source to staging) or ETL(staging-to-warehouse)
**Timestamp:** Succeeded/Failed Timestamp
**OS Version:** Operating System/Server OS Version
**PC Name:** Server or PC name where the EtL or ETL or both have implemented

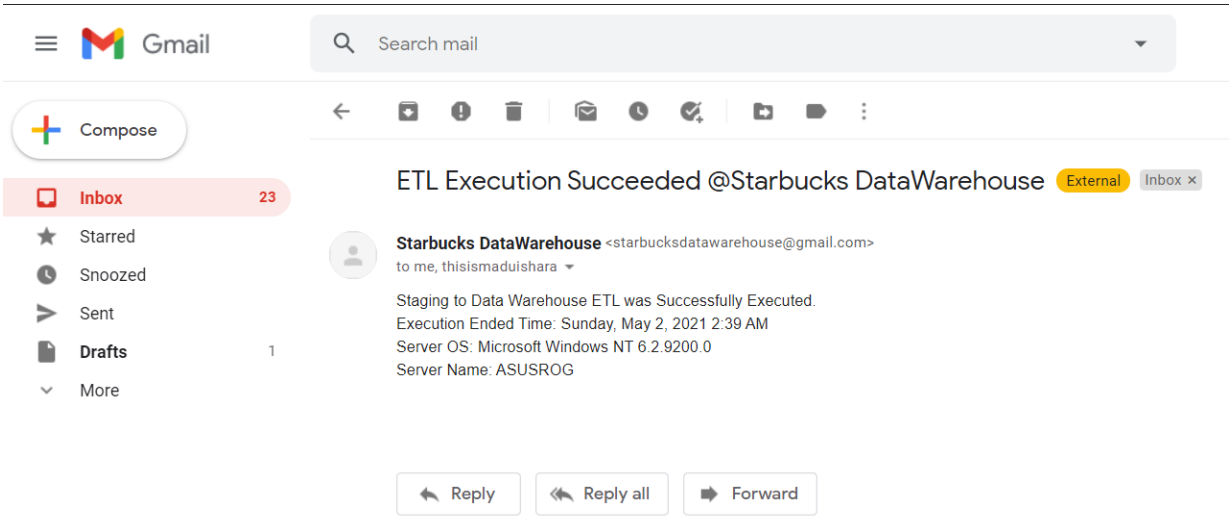



Figure 5.5.2.7. 1. A Received Emil indicating that the second ETL was a success that has been triggered by the ETL process script task.

# 6. Testing Integrated Packages

## Staging Database Table Counts Before sources-to-staging EtL was executed

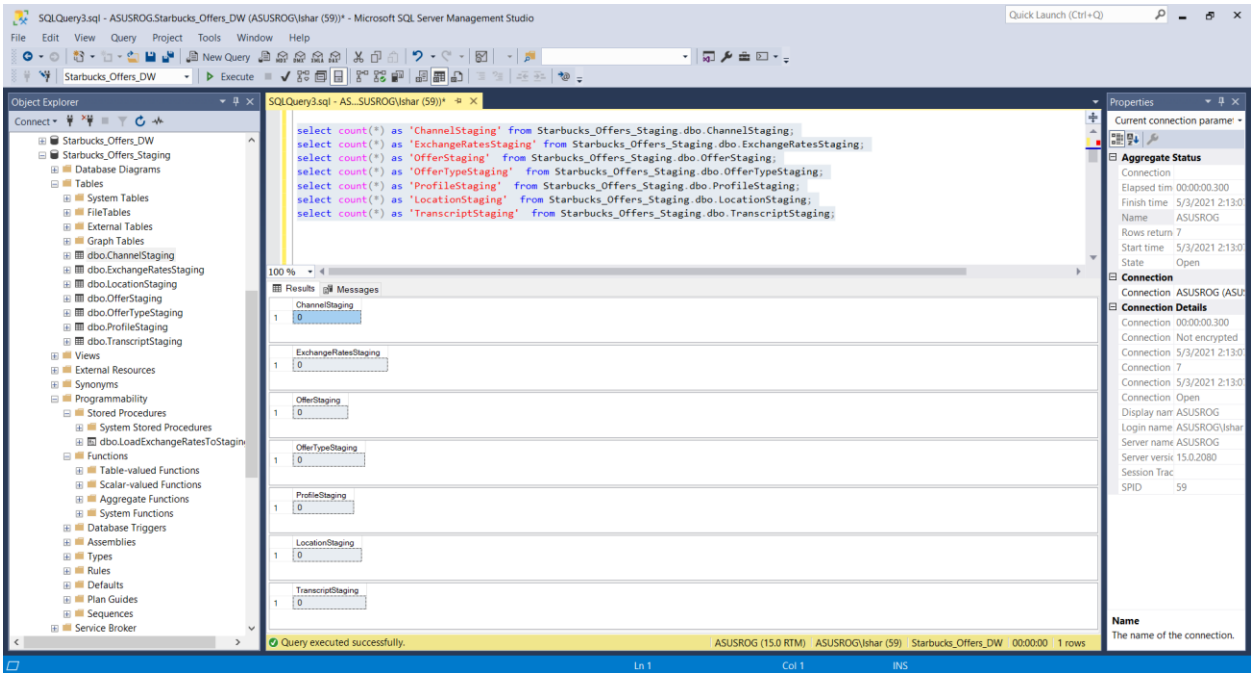



Figure 6. 1. Staging database tables before EtL was executed

## Data Warehouse Tables before staging-to-warehouse ETL package was executed

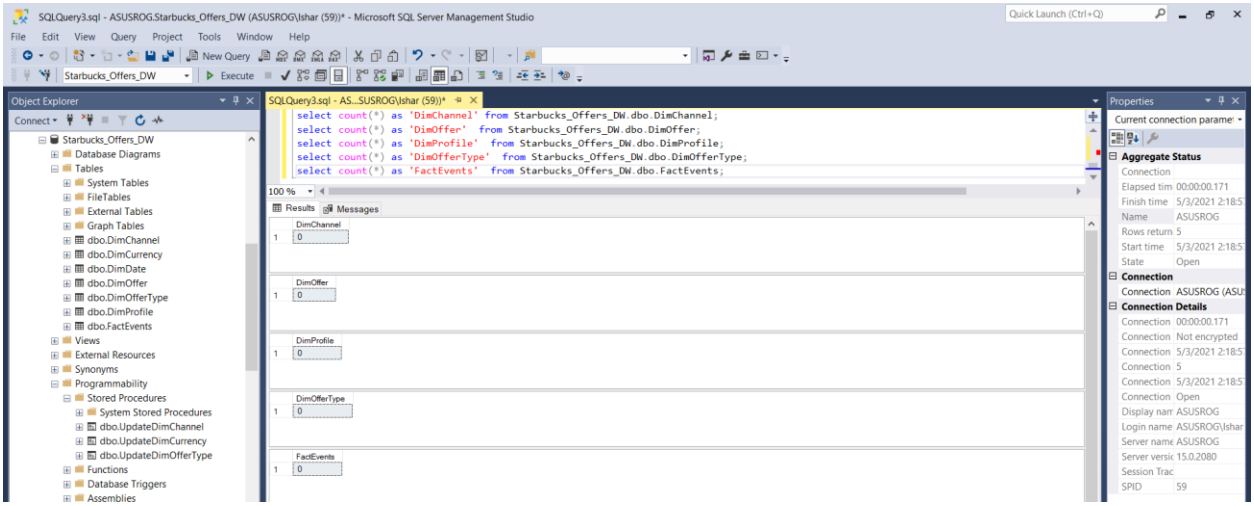



Figure 6. 2. Data warehouse tables before the ETL was executed. Note that DimDate and DimCurrency were statically filled, and real exchange rates have been updated up to the current date in DimCurrency and all future dates contain default values (Figure A-1.1 in Appendix A-1.)

**Starbucks_ETL_Staging.dtsx Package after execution was a success**

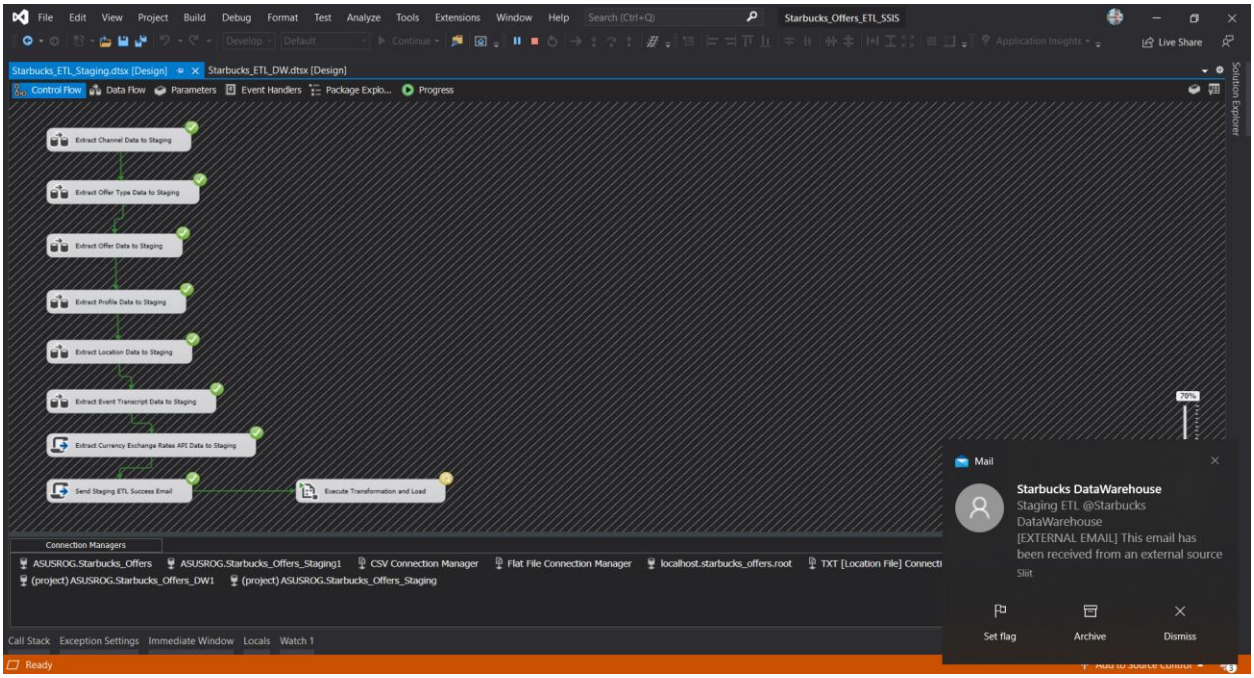



Figure 6. 3. After the first ETL was executed, an email will be sent, and the 2nd package execution is started

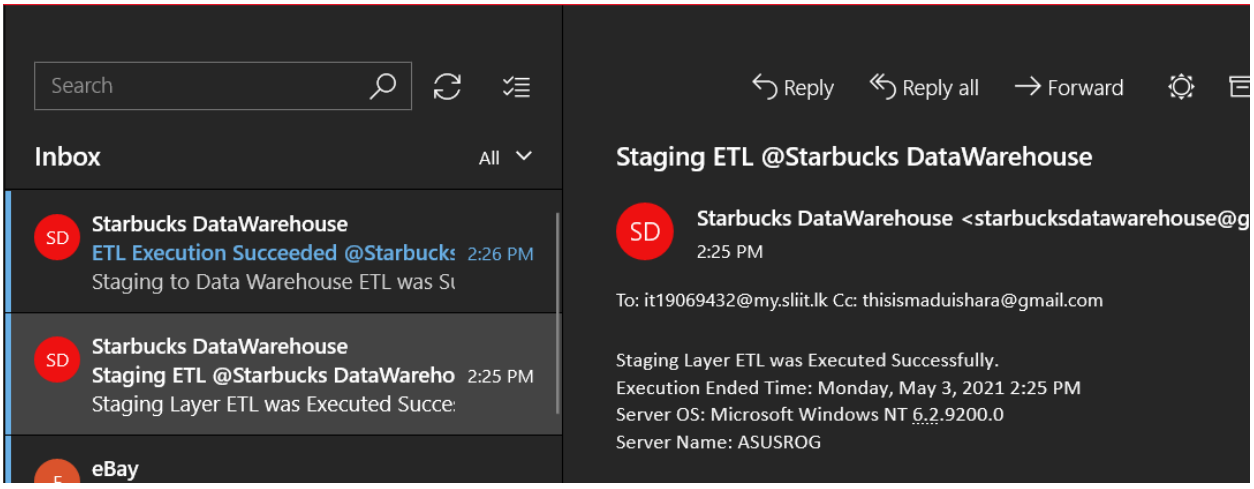



Figure 6. 4. Email Notification which received at 2:25 PM exactly after the sources-to-staging EtL was finished

**Starbucks_ETL_DW.dtsx Package after execution was a success**

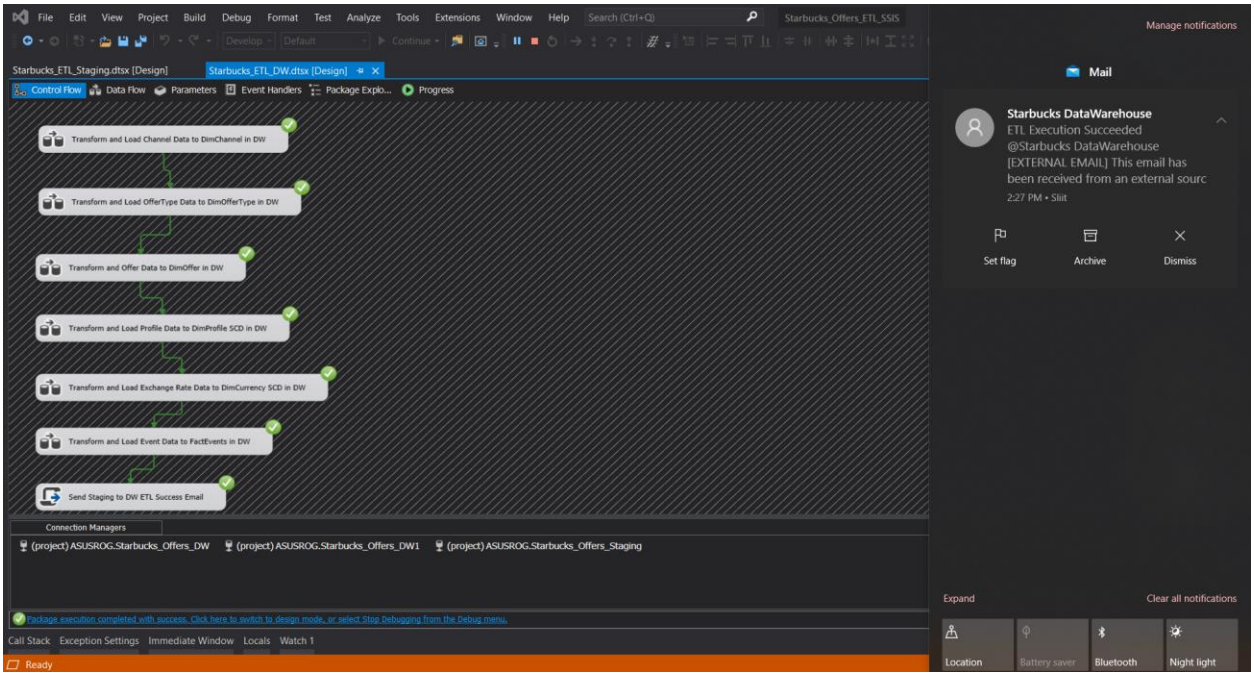



Figure 6. 5. After the 2nd package was executed successfully, which was triggered automatically by the 1st package

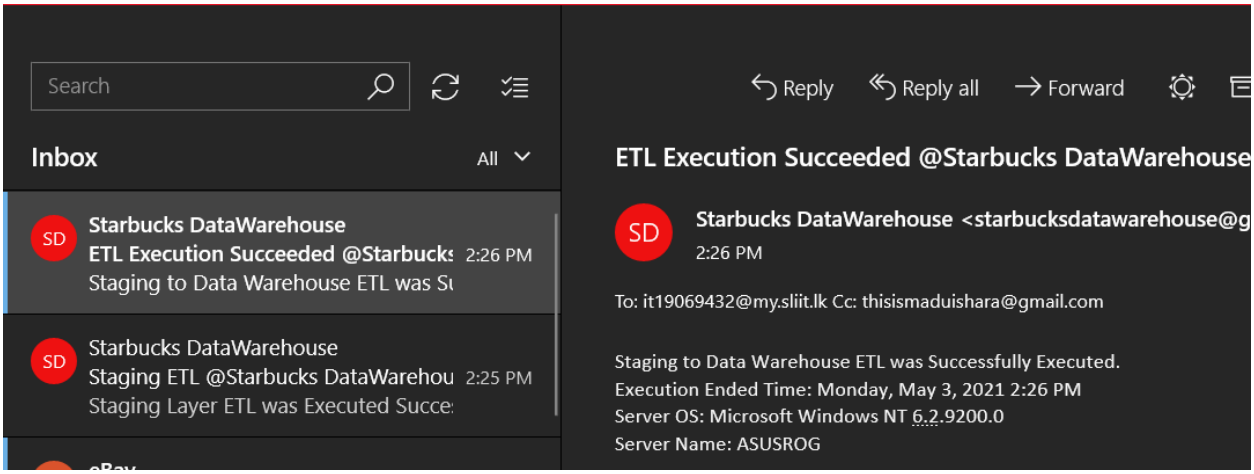



Figure 6. 6. Email Notification triggered by the 2nd ETL received exactly after the execution was a success. Received exactly at 2:26 PM

Note that the highest number of records are customer(profile), address(location) [17000 records each], and events (transcript) [306534 records]

## Staging Database Table Counts After sources-to-staging EtL was executed

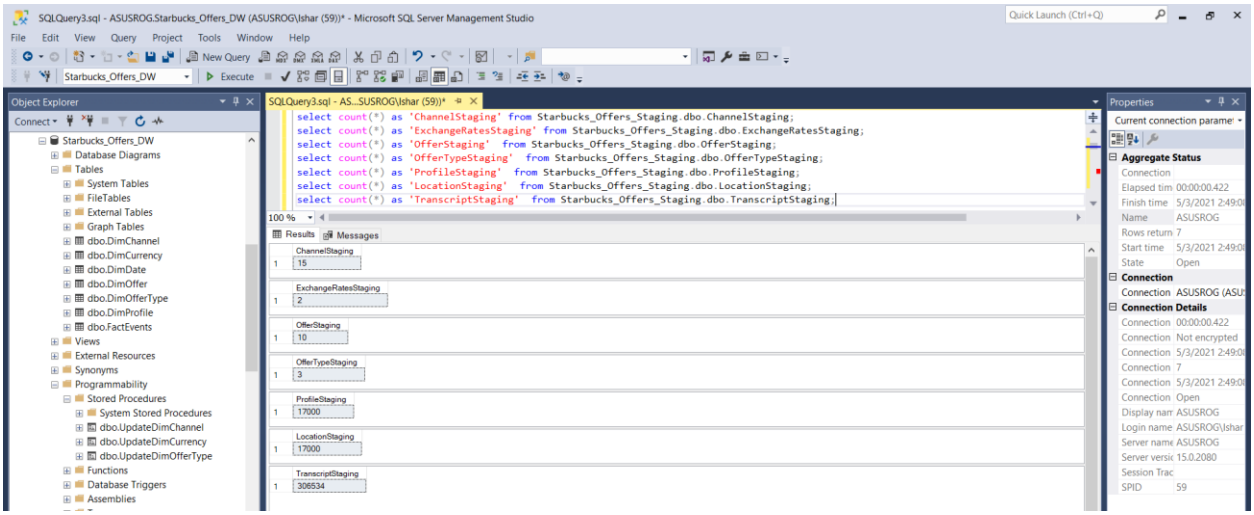



Figure 6. 7. Since a full load was done from sources to staging, records counts indicated here are the exact number of source records. Even if the package is executed again, the counts will remain unchanged unless the source files have been altered.

## Data Warehouse Tables before staging-to-warehouse ETL package was executed

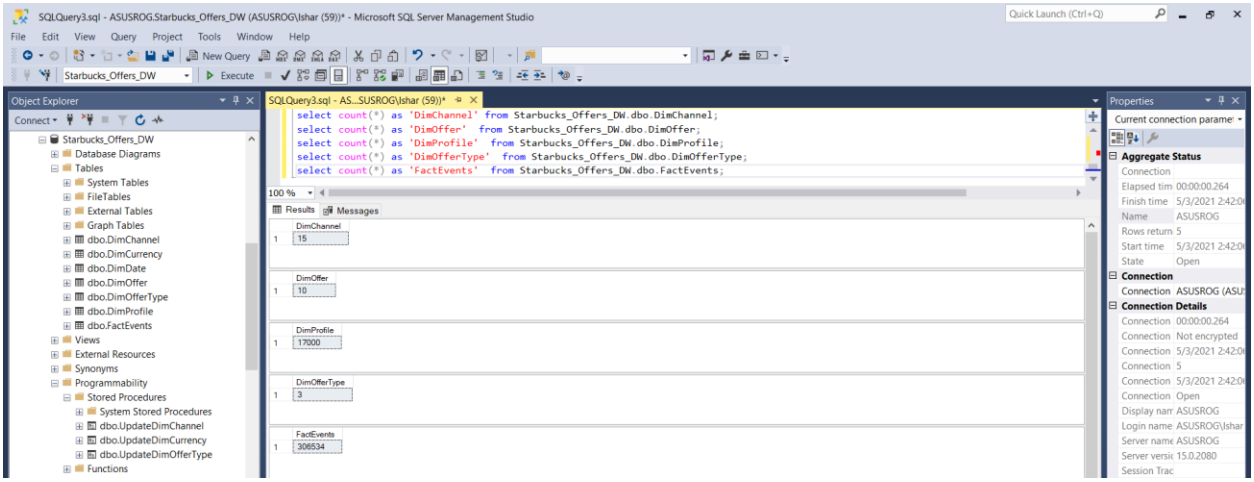



Figure 6. 8. Data warehouse tables after the ETL was executed. Note that real exchange rates have been updated for the current date in DimCurrency and all future dates contain default values (Figure A-1.3 in Appendix A-1.)

The counts will remain the same for DimChannel, DimOfferType, DimOffer, and DimProfile if staging tables have not received any updated records. However, the FactEvent table's record count will get increased if the ETL was executed again since it does not look for duplicates. It all depends on the staging layer's EtL logic and thus, the sources-to-staging EtL process must be altered only to obtain updated records from the source files/locations. Since in this context a full load is done each time, the FactEvent table record count will get doubled.

# 7. References


[1]. docs.microsoft.com, 'Database Dimensions - Create a Currency type Dimension', https://docs.microsoft.com/en-us/analysis-services/multidimensional-models/database-dimensions-create-a-currency-type-dimension?view=asallproducts-allversions [Accessed on: 28-Apr-2021]

[2]. medium.com, 'How to create a highly performant multi-currency solution in SSAS' https://medium.com/@roudgenoeg/how-to-create-a-highly-performant-multi-currency-solution-in-ssas-69e6c6cb81c0 [Accessed on: 28-Apr-2021]

[3]. c-sharpcorner.com, 'How To Consume Web API Through SSIS Package', https://www.c-sharpcorner.com/article/how-to-consume-web-api-through-ssis-package/ [Accessed on: 29-Apr-2021]

[4]. free.currencyconverterapi.com, 'The Free Currency Converter API', https://free.currencyconverterapi.com/ [Accessed on: 30-Apr-2021]

[5]. stackoverflow.com, 'Reading from JSON API with C# in SSIS', https://stackoverflow.com/questions/55224630/reading-from-json-api-with-c-sharp-in-ssis [Accessed on: 30-Apr-2021]

[6]. docs.microsoft.com, 'Connecting to Data Sources in the Script Task ', https://docs.microsoft.com/en-us/sql/integration-services/extending-packages-scripting/task/connecting-to-data-sources-in-the-script-task?view=sql-server-ver15 [Accessed on: 30-Apr-2021]

[7]. docs.microsoft.com, 'How to serialize and deserialize (marshal and unmarshal) JSON in .NET', https://docs.microsoft.com/en-us/dotnet/standard/serialization/system-text-json-how-to?pivots=dotnet-5-0 [Accessed on: 01-May-2021]

[8]. sharp-console-examples.com, 'Call a SQL Server Stored Procedure in C#', https://www.csharp-console-examples.com/winform/call-a-sql-server-stored-procedure-in-c/ [Accessed on: 25-Apr-2021]

[9]. stackoverflow.com, 'TRUNCATE table only if it exists (to avoid errors)', https://stackoverflow.com/questions/25394493/truncate-table-only-if-it-exists-to-avoid-errors [Accessed on: 26-Apr-2021]

[10]. tutorialgateway.org, 'SSIS Script Task Send Email', https://www.tutorialgateway.org/script-task-in-ssis/ [Accessed on: 01-Mar-2021]

# Appendix

## Appendix – A: Additional Diagrams

### A-1: Statically Filled Dimension Tables

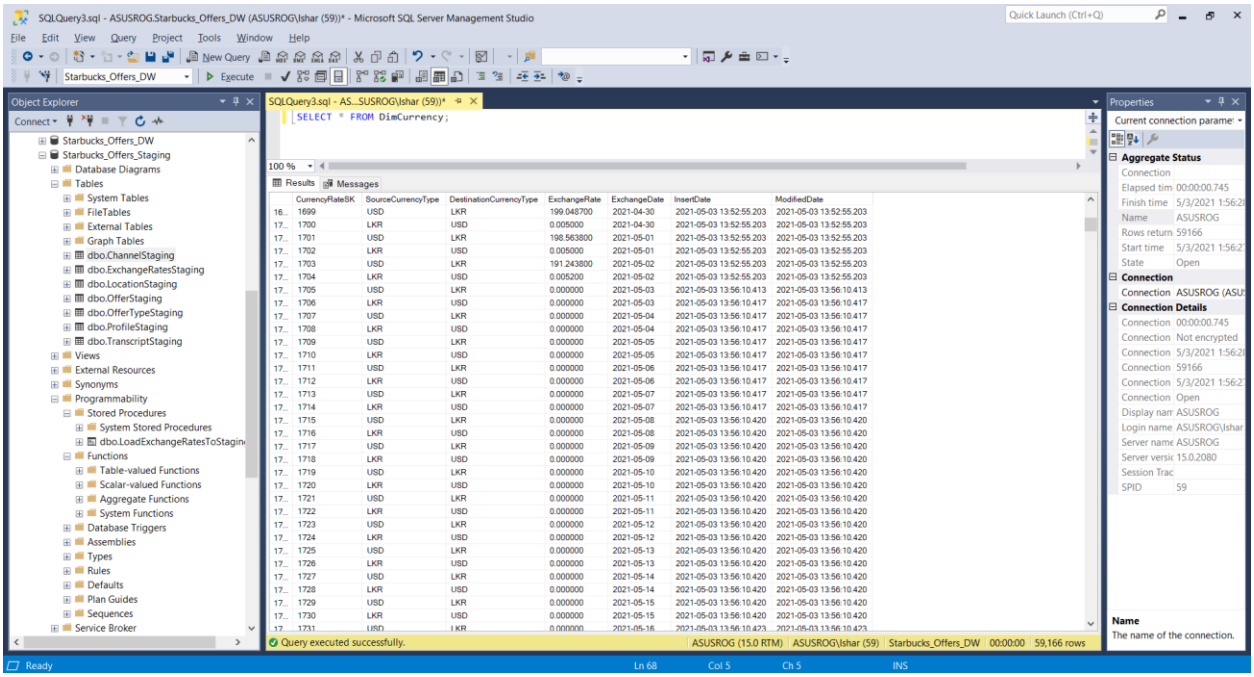



Figure A-1. 1. DimCurrency table was initially statically filled from 2019-01-02 until 2099-12-31 with exchange rate as 0.000000 which later will get replaced by the real exchange rate. Notice that real exchange rates have been updated until 2021-05-02 which was the day prior to the testing date (package execution)

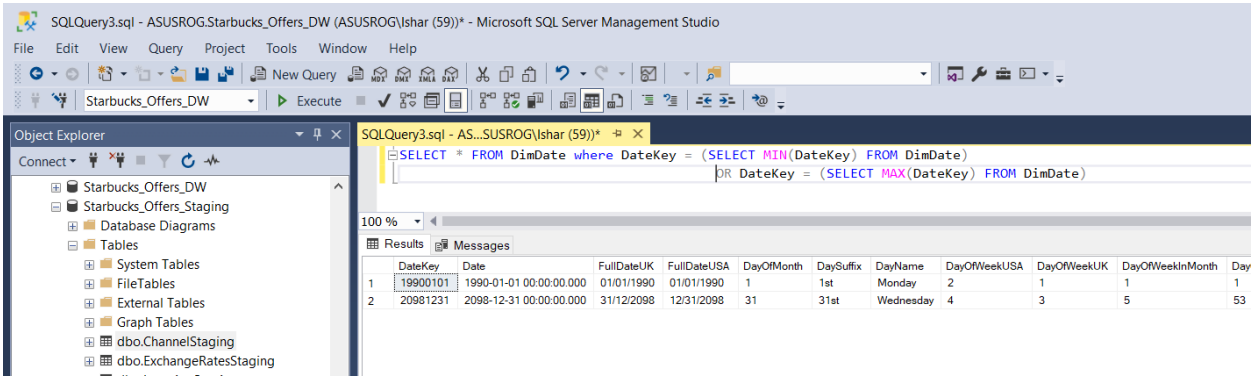



Figure A-1. 2. Min and Max dates in Statically filled DimDate

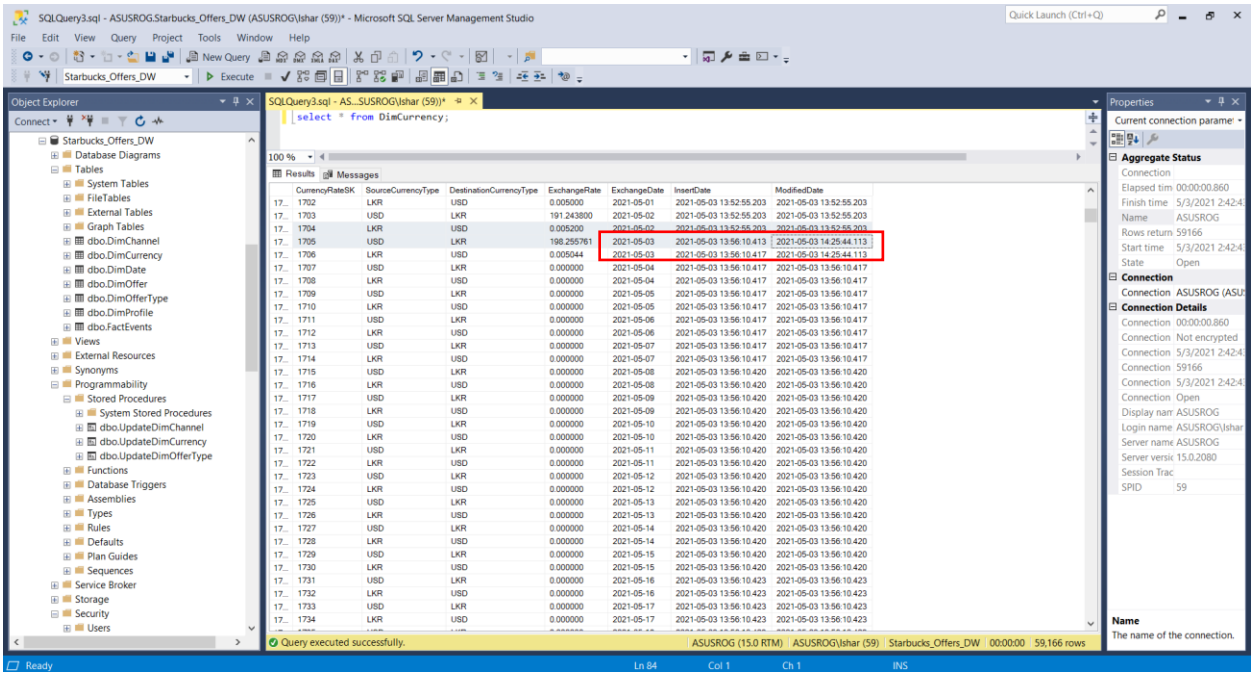



Figure A-1.3. DimCurrency table after both EtL and ETL processes were executed. Notice that current exchange rate values obtained from the API are updated on 2021-05-03 which was the testing date when the packages were executed.

## Appendix – B: Special Features and Code Listings

## B-1: Stored Procedures used in Staging Database

### B-1.1. LoadExchngeRatesToStaging

```
USE [Starbucks_Offers_Staging]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[LoadExchangeRatesToStaging]
@USDLKR DECIMAL(18,6),
@LKRUSD DECIMAL(18,6),
@CURDATE DATE AS
BEGIN
    -- CREATE TABLE IF NOT EXISTS
    if not exists (select * from sysobjects where name='ExchangeRatesStaging' and
xtype='U')
    create table ExchangeRatesStaging (
        source_currency nvarchar(3),
        destination_currency nvarchar(3),
        exchange_rate decimal(18,6),
        retrieved_date date
    )

       truncate table dbo.ExchangeRatesStaging

       -- INSERT USD TO LKR
    insert into dbo.ExchangeRatesStaging (source_currency, destination_currency,
exchange_rate, retrieved_date) values ('USD','LKR', @USDLKR, CONVERT(DATE, @CURDATE))

       --INSERT LKR TO USD
    insert into dbo.ExchangeRatesStaging (source_currency, destination_currency,
exchange_rate, retrieved_date) values ('LKR','USD', @LKRUSD, CONVERT(DATE, @CURDATE))


END;
```

Stored procedure above is called by the “Extract Currency Exchange Rates API Data to Staging” script task in the “Starbucks_ETL_Staging.dtsx” package containing sources-to-staging EtL. Since all the tables in the staging database are automatically created using tasks, the same principal was used in this scenario as well. If the table `'ExchangeRatesStaging'` is not found in the staging database, it will create a new table. The table will be truncated before loading extracted rates sent by the script task mentioned above which allows only newly extracted rows to be present in the table.

## B-2: Stored Procedures Used in Data Warehouse Database

### B-2.1. UpdateDimChannel

```
USE [Starbucks_Offers_DW]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateDimChannel]
@ChannelID nvarchar(6),
@ChannelName nvarchar(50) AS
BEGIN
    if not exists (select ChannelSK from dbo.DimChannel where AlternateChannelID =
@ChannelID)
    BEGIN
        insert into dbo.DimChannel (AlternateChannelID, ChannelName, InsertDate,
ModifiedDate) values (@ChannelID, @ChannelName, GETDATE(), GETDATE())
    END;
    if exists (select ChannelSK from dbo.DimChannel where AlternateChannelID =
@ChannelID)
    BEGIN
        update dbo.DimChannel set ChannelName = @ChannelName, ModifiedDate = GETDATE()
where AlternateChannelID = @ChannelID
    END;

END;
```

### B.-2.2. UpdateDimCurrency

```
USE [Starbucks_Offers_DW]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

ALTER PROCEDURE [dbo].[UpdateDimCurrency]
@Source nvarchar(3),
@Destination nvarchar(3),
@RetrievedDate date,
@InsertDate datetime,
@ModifiedDate datetime,
@ExchangeRate decimal(18,6) AS
BEGIN
    if not exists (select CurrencyRateSK from dbo.DimCurrency where SourceCurrencyType =
@Source AND DestinationCurrencyType = @Destination AND ExchangeDate = @RetrievedDate)
    BEGIN
        insert into dbo.DimCurrency (SourceCurrencyType, DestinationCurrencyType,
ExchangeRate, ExchangeDate, InsertDate, ModifiedDate) values (@Source, @Destination,
@ExchangeRate, @RetrievedDate, @InsertDate, @ModifiedDate)
    END;
    if exists (select CurrencyRateSK from dbo.DimCurrency where SourceCurrencyType =
@Source AND DestinationCurrencyType = @Destination AND ExchangeDate = @RetrievedDate)
    BEGIN
        update dbo.DimCurrency set ExchangeRate = @ExchangeRate, ModifiedDate =
@ModifiedDate where SourceCurrencyType = @Source AND DestinationCurrencyType =
@Destination AND ExchangeDate = @RetrievedDate
    END;

END;
```

### B-2.3. UpdateDimOfferType

```
USE [Starbucks_Offers_DW]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[UpdateDimOfferType]
@OfferTypeID nvarchar(6),
@TypeName nvarchar(15) AS
BEGIN
    if not exists (select OfferTypeSK from dbo.DimOfferType where AlternateOfferTypeID =
@OfferTypeID)
    BEGIN
        insert into dbo.DimOfferType (AlternateOfferTypeID, TypeName, InsertDate,
ModifiedDate) values (@OfferTypeID, @TypeName, GETDATE(), GETDATE())
    END;
    if exists (select OfferTypeSK from dbo.DimOfferType where AlternateOfferTypeID =
@OfferTypeID)
    BEGIN
        update dbo.DimOfferType set TypeName = @TypeName, ModifiedDate = GETDATE() where
AlternateOfferTypeID = @OfferTypeID
    END;

END;
```

## B-3: SQL Scripts to pre-populate DimCurrency Table

### B-3.1. Assigning Default values for a given period

```
USE Starbucks_Offers_DW;

DECLARE @MinDate DATE = '2019-01-02';
DECLARE @MaxDate DATE = '2099-12-31';

WHILE @MinDate < @MaxDate
       BEGIN
              DECLARE @USDLKR money =  0.00
              DECLARE @LKRUSD money = 0.00

              INSERT INTO DimCurrency
(SourceCurrencyType,DestinationCurrencyType,ExchangeRate,ExchangeDate,
InsertDate,ModifiedDate)
              VALUES('USD','LKR', @USDLKR,DATEADD(day, 1, @MinDate), GETDATE(),
GETDATE())
              INSERT INTO DimCurrency
(SourceCurrencyType,DestinationCurrencyType,ExchangeRate,ExchangeDate,
InsertDate,ModifiedDate)
              VALUES('LKR','USD', @LKRUSD,DATEADD(day, 1, @MinDate), GETDATE(),
GETDATE())
              SET @MinDate = DATEADD(day, 1, @MinDate)
       END;


SELECT * FROM DimCurrency;
```

**B-4: Script Task Code which retrieves API Exchange Rates**

Since this script task is configured directly in the first EtL process, it may not be possible to schedule it daily if the EtL process is not executed daily. However, because the Starbucks relies on daily exchange rate, a different package was created, named, "Starbucks_ETL_ExchangeRatesDaily.dtsx" that will make it possible for the package to be scheduled daily, which only performs the task of retrieving daily exchange rates and update the Data Warehouse DimCurrency table directly.

*Note that the Script Task in sources-to-staging EtL package was loading exchange rates to staging layer only.

```
#region Namespaces
using System;
using System.Data;
using Microsoft.SqlServer.Dts.Runtime;
using System.Windows.Forms;
using System.Net;
using System.Text.Json;
using System.Data.SqlClient;
#endregion

namespace ST_d20b80dd6f90445d815bea468810b77f
{
	[Microsoft.SqlServer.Dts.Tasks.ScriptTask.SSISScriptTaskEntryPointAttribute]
	public partial class ScriptMain :
Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
	{

	 public void Main()
		{
		var USD_LKR_url =
"https://free.currconv.com/api/v7/convert?q=USD_LKR&compact=ultra&apiKey=960591cbee556d44
e235";
		var LKR_USD_url =
"https://free.currconv.com/api/v7/convert?q=LKR_USD&compact=ultra&apiKey=960591cbee556d44
e235";
		var USD_LKR_Response = new WebClient().DownloadString(USD_LKR_url);
		var LKR_USD_Response = new WebClient().DownloadString(LKR_USD_url);
		//ExchangeRate exchangeRate = JsonSerializer.Deserialize<ExchangeRate>(json);


		var USD_LKR_JSON = JsonDocument.Parse(USD_LKR_Response);
		var LKR_USD_JSON = JsonDocument.Parse(LKR_USD_Response);
		var USD_LKR = USD_LKR_JSON.RootElement.GetProperty("USD_LKR").GetDecimal();
		var LKR_USD = LKR_USD_JSON.RootElement.GetProperty("LKR_USD").GetDecimal();
		string currentDate = DateTime.Now.ToString("yyyy-MM-dd");

		//Test
		//MessageBox.Show("USD to LKR: " + USD_LKR.ToString() + "\nLKR to USD: " +
LKR_USD.ToString());

		SqlConnection myADONETConnection =
```

```
Dts.Connections["ASUSROG.Starbucks_Offers_DW1"].AcquireConnection(Dts.Transaction)
                as SqlConnection;

            //Insert or Update using the SP in Data Warehouse directly
            SqlCommand sqlCommand;

            sqlCommand = new SqlCommand("UpdateDimCurrency", myADONETConnection);
            sqlCommand.CommandType = CommandType.StoredProcedure;
            sqlCommand.Parameters.Add("@Source", SqlDbType.NVarChar).Value = "LKR";
            sqlCommand.Parameters.Add("@Destination", SqlDbType.NVarChar).Value = "USD";
            sqlCommand.Parameters.Add("@ExchangeRate", SqlDbType.Decimal).Value =
LKR_USD;
            sqlCommand.Parameters.Add("@RetrievedDate", SqlDbType.Date).Value =
currentDate;
            sqlCommand.Parameters.Add("@InsertDate", SqlDbType.DateTime).Value =
DateTime.Now.ToString();
            sqlCommand.Parameters.Add("@ModifiedDate", SqlDbType.DateTime).Value =
DateTime.Now.ToString();
            sqlCommand.ExecuteNonQuery();

            sqlCommand = new SqlCommand("UpdateDimCurrency", myADONETConnection);
            sqlCommand.CommandType = CommandType.StoredProcedure;
            sqlCommand.Parameters.Add("@Source", SqlDbType.NVarChar).Value = "USD";
            sqlCommand.Parameters.Add("@Destination", SqlDbType.NVarChar).Value = "LKR";
            sqlCommand.Parameters.Add("@ExchangeRate", SqlDbType.Decimal).Value =
USD_LKR;
            sqlCommand.Parameters.Add("@RetrievedDate", SqlDbType.Date).Value =
currentDate;
            sqlCommand.Parameters.Add("@InsertDate", SqlDbType.DateTime).Value =
DateTime.Now.ToString();
            sqlCommand.Parameters.Add("@ModifiedDate", SqlDbType.DateTime).Value =
DateTime.Now.ToString();
            sqlCommand.ExecuteNonQuery();


            myADONETConnection.Close();


            Dts.TaskResult = (int)ScriptResults.Success;
              }

        #region ScriptResults declaration
        enum ScriptResults
        {
            Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
            Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
        };
        #endregion

        public class ExchangeRate
        {
            public String USD_LKR { get; set; }
            public String LKR_USD { get; set; }
        }
    }

}
```

### B-5: Script Task Code which retrieves API Exchange Rates in EtL

Unlike the script task presented in B-4 section of the appendix, the following script task only loads new exchange rates to staging database table. Note that the business relies on daily exchange rates and this task might not be running daily in case the package is scheduled to run in a different manner. In such scenarios, Package with the script task presented in B-4 section of the appendix can be scheduled to run daily and the issue will be solved.

```
#region Namespaces
using System;
using System.Data;
using Microsoft.SqlServer.Dts.Runtime;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Net;
using System.Text.Json;
#endregion

namespace ST_b4fb67a9f86a4aa8ad75a7f96239ff7d
{
       [Microsoft.SqlServer.Dts.Tasks.ScriptTask.SSISScriptTaskEntryPointAttribute]
       public partial class ScriptMain :
Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
       {
              public void Main()
              {
            var USD_LKR_url =
"https://free.currconv.com/api/v7/convert?q=USD_LKR&compact=ultra&apiKey=960591cbee556d44
e235";
            var LKR_USD_url =
"https://free.currconv.com/api/v7/convert?q=LKR_USD&compact=ultra&apiKey=960591cbee556d44
e235";
            var USD_LKR_Response = new WebClient().DownloadString(USD_LKR_url);
            var LKR_USD_Response = new WebClient().DownloadString(LKR_USD_url);
            //ExchangeRate exchangeRate = JsonSerializer.Deserialize<ExchangeRate>(json);


            var USD_LKR_JSON = JsonDocument.Parse(USD_LKR_Response);
            var LKR_USD_JSON = JsonDocument.Parse(LKR_USD_Response);
            var USD_LKR = USD_LKR_JSON.RootElement.GetProperty("USD_LKR").GetDecimal();
            var LKR_USD = LKR_USD_JSON.RootElement.GetProperty("LKR_USD").GetDecimal();
            string currentDate = DateTime.Now.ToString("yyyy-MM-dd");

            //Test
            //MessageBox.Show("USD to LKR: " + USD_LKR.ToString() + "\nLKR to USD: " +
LKR_USD.ToString());

            SqlConnection myADONETConnection =

Dts.Connections["ASUSROG.Starbucks_Offers_Staging1"].AcquireConnection(Dts.Transaction)
                as SqlConnection;

            //Insert or Update using a SP
            SqlCommand sqlCommand;
```

```
            sqlCommand = new SqlCommand("LoadExchangeRatesToStaging",
myADONETConnection);
            sqlCommand.CommandType = CommandType.StoredProcedure;
            sqlCommand.Parameters.Add("@USDLKR", SqlDbType.Decimal).Value = USD_LKR;
            sqlCommand.Parameters.Add("@LKRUSD", SqlDbType.Decimal).Value = LKR_USD;
            sqlCommand.Parameters.Add("@CURDATE", SqlDbType.Date).Value = currentDate;
            sqlCommand.ExecuteNonQuery();
            myADONETConnection.Close();


            Dts.TaskResult = (int)ScriptResults.Success;
        }

        #region ScriptResults declaration
        enum ScriptResults
        {
            Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
            Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
        };
        #endregion

        public class ExchangeRate
        {
            public String USD_LKR { get; set; }
            public String LKR_USD { get; set; }
        }
       }

}
```

### B-6: Script Task Code to Send Email Notifications

```
#region Namespaces
using System;
using System.Data;
using Microsoft.SqlServer.Dts.Runtime;
using System.Windows.Forms;
using System.Net.Mail;
using System.Net;
#endregion

namespace ST_76a8efc580384477a2d29b49f80a3fdf
{
       [Microsoft.SqlServer.Dts.Tasks.ScriptTask.SSISScriptTaskEntryPointAttribute]
       public partial class ScriptMain :
Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
       {
              public void Main()
              {
            String receiver = "it19069432@my.sliit.lk";
            String subject = "ETL Execution Succeeded @Starbucks DataWarehouse";
            String message = "Staging to Data Warehouse ETL was Successfully
Executed.\nExecution Ended Time: " + DateTime.Now.ToLongDateString() + " " +
DateTime.Now.ToShortTimeString() + "\nServer OS: " + Environment.OSVersion.ToString() +
"\nServer Name: " + Environment.MachineName.ToString();
            String smtp = "smtp.gmail.com";

            MailMessage mailMessage = new MailMessage("starbucksdatawarehouse@gmail.com",
receiver, subject, message);
            mailMessage.From = new MailAddress("starbucksdatawarehouse@gmail.com",
"Starbucks DataWarehouse");
            mailMessage.CC.Add("thisismaduishara@gmail.com");
            SmtpClient smtpClient = new SmtpClient(smtp, 587);
            smtpClient.EnableSsl = true;
            smtpClient.UseDefaultCredentials = false;
            NetworkCredential networkCredential = new
NetworkCredential("starbucksdatawarehouse@gmail.com", "******");
            smtpClient.Credentials = networkCredential;
            smtpClient.Send(mailMessage);

            Dts.TaskResult = (int)ScriptResults.Success;

            Dts.TaskResult = (int)ScriptResults.Success;
              }

        #region ScriptResults declaration
        enum ScriptResults
        {
            Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
            Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
        };
        #endregion

       }

}
```