



Sri Lanka Institute of Information Technology

Food Delivery Data Warehouse Solution

Assignment-1 Document

IT3021 - Data Warehousing and Business Intelligence
Assignment 1

Submitted by: IT20033828 – Ariyasinghe P.A.D.N.I.

Date of submission: 11/06/2022

Table of Contents

1. Dataset Selection.....	3
1.1. ER diagram	4
2. Preparation of Data sources - Description of the dataset.....	5
3. Solution Architecture	7
4. Data warehouse design and development	8
4.1. Dimensional Tables	9
4.2. Fact Tables	9
4.3. Identified Hierarchies	9
4.4. Derived Columns	9
5. ETL Development.....	10
5.1. Source Database to Staging Database.....	11
5.2. Staging Database to Datawarehouse.....	18
6. ETL Development – Accumulating fact table	35
7. References	40
Appendix.....	41
Appendix – A: SQL queries to create Dimension tables in Data Warehouse Database	41

1. Dataset selection

A Transactional Dataset about food delivery company was selected as the base dataset to build the Data Warehouse. The original source files can be found using the link provided below.

Link: <https://www.kaggle.com/datasets/nosbielcs/brazilian-delivery-center?select=hubs.csv>

With different operational hubs spread throughout Brazil, the Food Stores/ Delivery Centers are the places that takes food orders from the users and do deliveries in the region. Firstly, the users required to order foods and make necessary payments online through the system. After the food making process is finished, the paid order will be delivered to relevant places by the delivery partners (drivers) spread across all regions of the country.

There were originally seven source files provided in the CSV format, namely, “payments.csv”, “deliveries.csv”, “stores.csv”, “hubs.csv”, “orders.csv”, “channels.csv”, and “drivers.csv”. 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 few fields of some original source files contain missing values, thus there was a lot of data enrichment to be done in those data files.

Since these files had enough records and columns to be divided into several hierarchies such as [order created month, order created day, order created hour], [hub state, hub city] 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 ER diagram.

1.1. ER diagram:

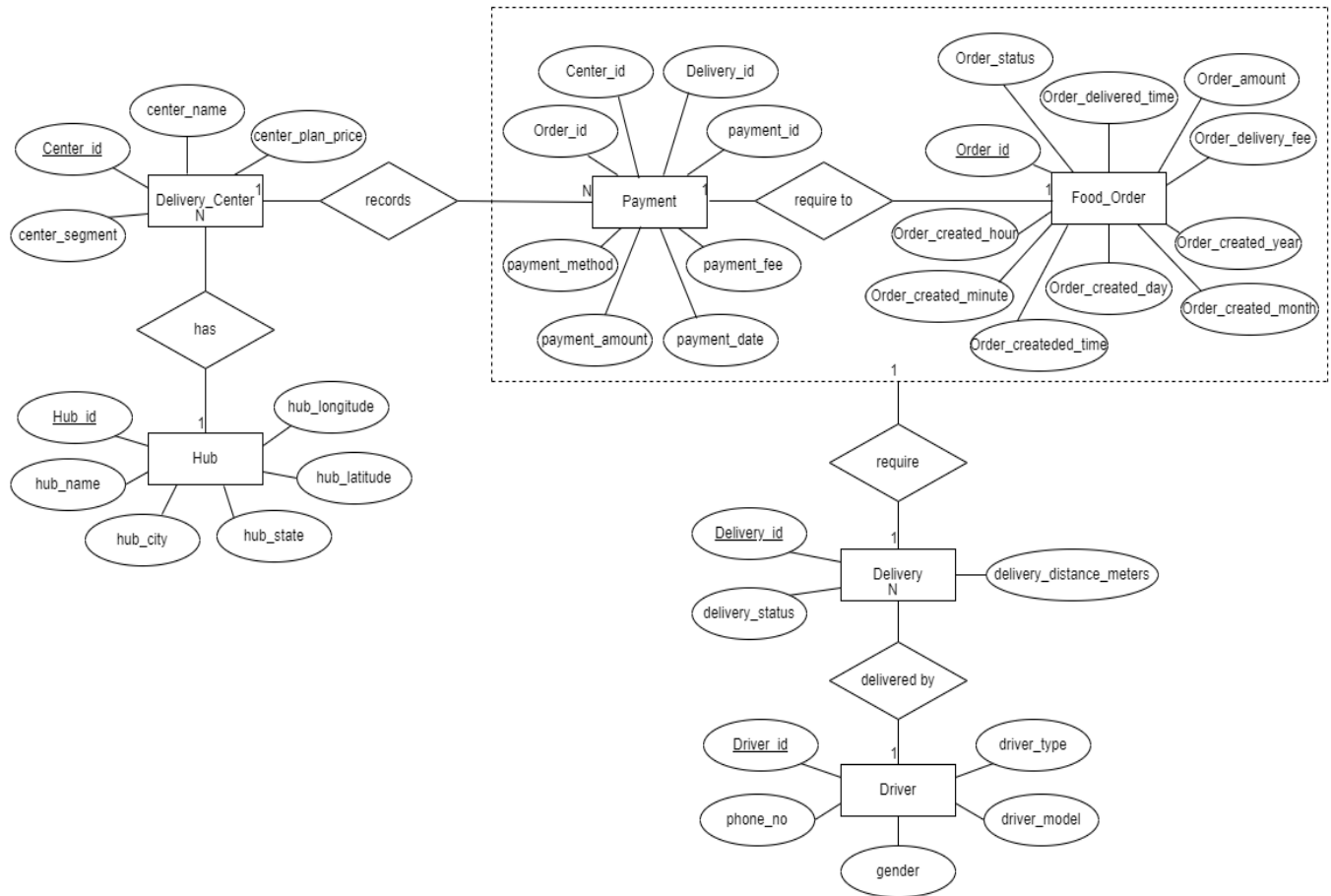


Figure 1.1.1. Er Diagram of the potential entities identified in the source files

As shown in the ER 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 such as Hub and Delivery Center.

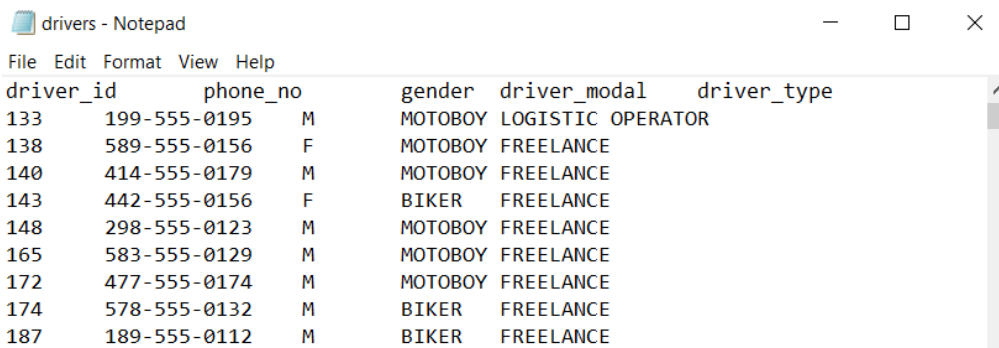
2. Preparation of Data sources - Description of the dataset

Table name	Column name	Data type	Description
Hubs	Hub_id	tinyint	Summary of the hubs
	Hub_name	nvarchar(50)	
	Hub_city	nvarchar(50)	
	Hub_state	nvarchar(50)	
	Hub_latitude	float	
	Hub_longitude	float	
Delivery Centers	Center_id	smallint	Summary of the delivery centers which are delivering food orders
	Hub_id	Tinyint	
	Center_name	nvarchar(50)	
	Center_segment	nvarchar(50)	
	Center_plan_price	float	
Deliveries	Delivery_id	int	Details of payments
	Driver_id	int	
	Delivery_distance	int	
	Delivery_status	nvarchar(50)	
Drivers	driver_id	int	Details of drivers who delivers food orders
	driver_modal	nvarchar(50)	
	driver_type	nvarchar(50)	
	phone_no	nvarchar(50)	
	gender	nvarchar(50)	
Payments	Order_id	Int	Details of the customer transactions
	Center_id	smallint	
	Delivery_id	int	
	Payment_id	int	
	Payment_amount	Float	
	Payment_fee	Float	
	Payment_method	Nvarchar(50)	
	Payment_date	datetime	
Food Orders	Order_id	Int	Details about the food orders made through the delivery center platform
	Order_status	Nvarchar(50)	
	Order_amount	Float	
	Order_delivery_fee	Float	
	Order_created_hour	Tinyint	
	Order_created_minute	Tinyint	
	Order_created_day	Tinyint	
	Order_created_year	smallint	

Complete_time_details	Order_created_time	Datetime2	Details about Transaction completion time.
	Order_delivered_time	Datetime2	
	fact_table_natural_key	Int	
	accm_txn_complete_time	Datetime	

- **Drivers.txt**

Since one of the requirements in the guidelines was to have multiples source types before data extraction, “drivers.csv” file was converted into “txt” file.

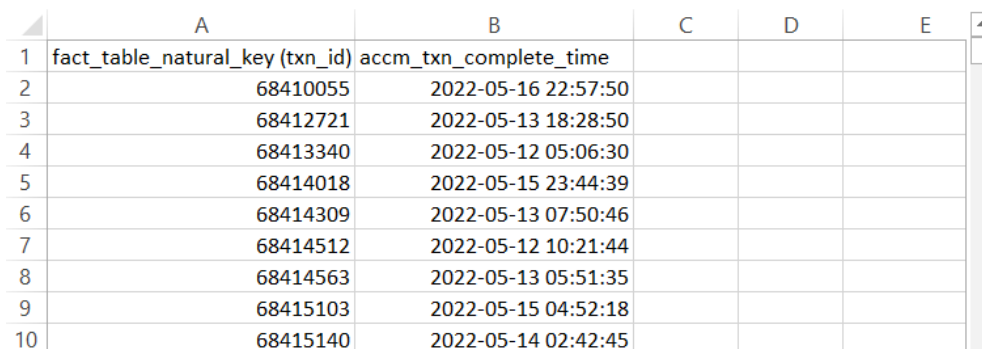


driver_id	phone_no	gender	driver_modal	driver_type
133	199-555-0195	M	MOTOBOY	LOGISTIC OPERATOR
138	589-555-0156	F	MOTOBOY	FREELANCE
140	414-555-0179	M	MOTOBOY	FREELANCE
143	442-555-0156	F	BIKER	FREELANCE
148	298-555-0123	M	MOTOBOY	FREELANCE
165	583-555-0129	M	MOTOBOY	FREELANCE
172	477-555-0174	M	MOTOBOY	FREELANCE
174	578-555-0132	M	BIKER	FREELANCE
187	189-555-0112	M	BIKER	FREELANCE

Figure 2.1. Drivers.txt file contents [Notepad]

- **Complete_time_details.xlsx**

For the final step in the assignment, which was to calculate the transaction completion time, a separate table was created named “Complete_time_details.xlsx” with following two columns named “fact_table_natural_key (txn_id)” By considering payment_id as a unique key in payments.csv and “accm_txn_complete_time” which is the time slot at transaction completion.



	A	B	C	D	E
1	fact_table_natural_key (txn_id)	accm_txn_complete_time			
2	68410055	2022-05-16 22:57:50			
3	68412721	2022-05-13 18:28:50			
4	68413340	2022-05-12 05:06:30			
5	68414018	2022-05-15 23:44:39			
6	68414309	2022-05-13 07:50:46			
7	68414512	2022-05-12 10:21:44			
8	68414563	2022-05-13 05:51:35			
9	68415103	2022-05-15 04:52:18			
10	68415140	2022-05-14 02:42:45			

Figure 2.2. Complete_time_details.xlsx [Excel]

3. Solution Architecture

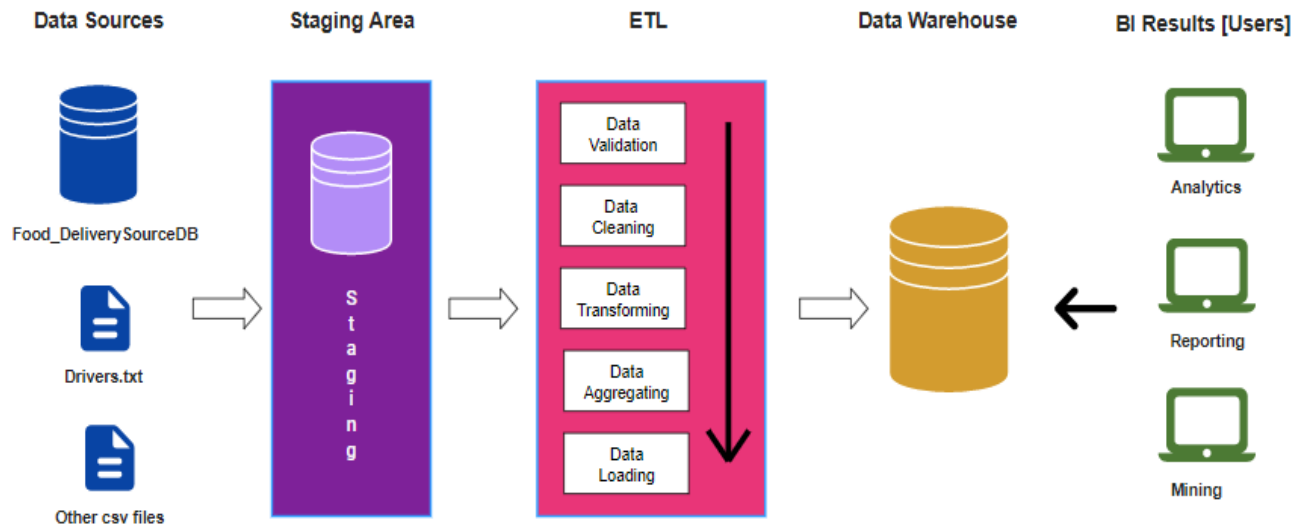


Figure 3. 1. The high-level architecture of the food Delivery Data Warehouse Solution.

After creating the *Food_DeliverySourceDB* with the help of other source files, the first step to do is staging the source data set. After the staging layer developed, the below mentioned staging tables were created: *StgHubs*, *StgDeliveryCenters*, *StgDeliveries*, *StgDrivers*, *StgFoodOrders* and *StgPayments*.

Then the staged tables were profiled, and aggregations were performed when necessary. The next step is data transforming and loading with the ETL process. After completing the described stages, the Datawarehouse is created. Names of data warehouse tables are: *DimHub*, *DimDeliveryCenters*, *DimDrivers*, *DimDeliveries*, *DimFoodOrders*, *DimPayments*, *DimDate*, *FactPayments*.

After the warehouse is created BI results such as OLAP analysis, Reports, Data visualization, Data mining can be obtained as results after further modifications.

4. Data warehouse design and development

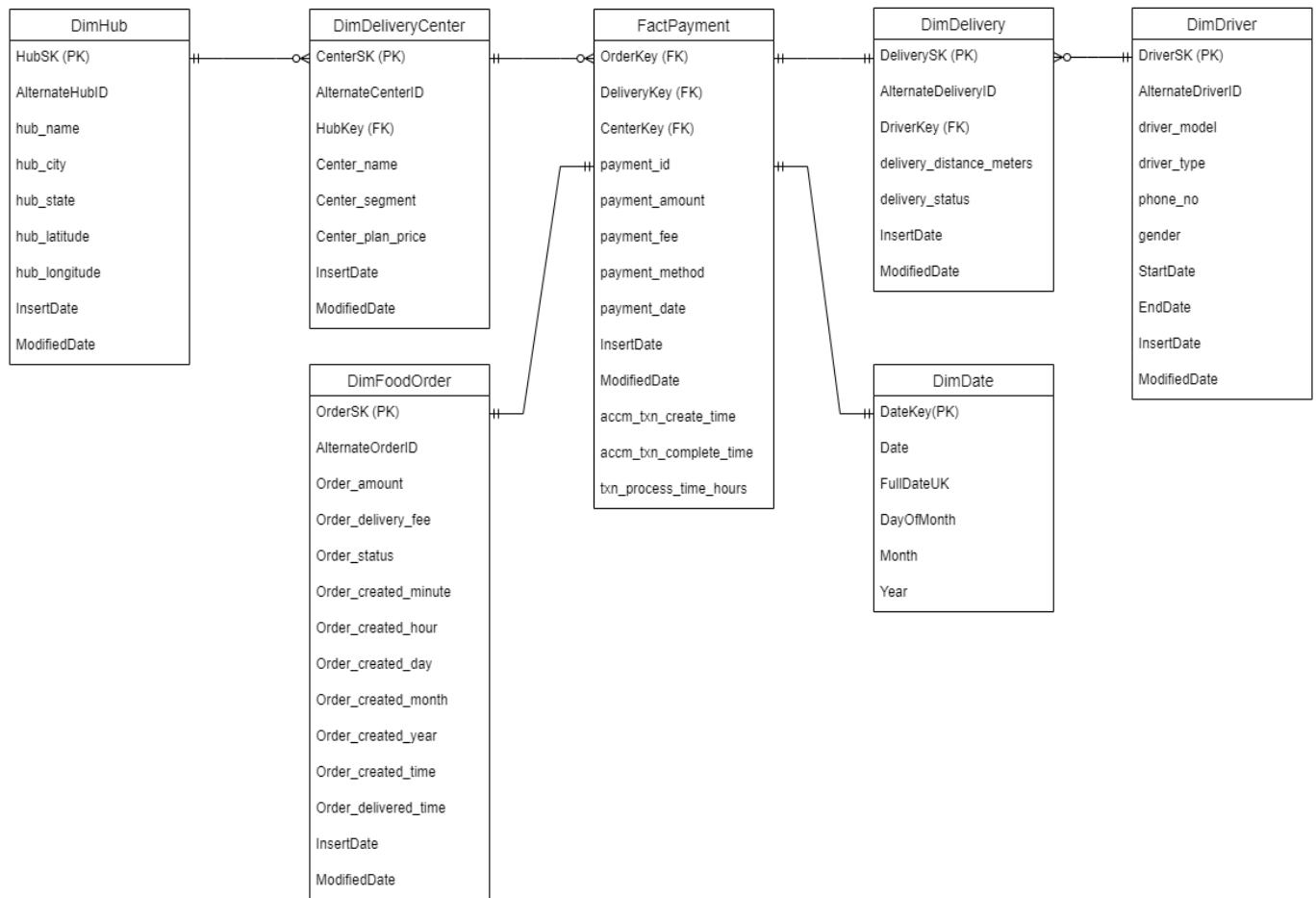


Figure 4. 1. Snowflake schema dimensional model used by Food Delivery Data Warehouse Solution

Snowflake schema is used to design the above dataset into a data warehouse. There is one fact table as Fact Payment which contains the transaction details of the food orders. All the others are dimensions. DimDrivers dimension is taken as a slowly changing dimension.

Assumptions:

Users must pay for food orders to confirm the order. If they cancel the order later, the paid money will be returned. Users can pay by the online system or voucher, debit card or paychecks. Online paid orders will be home delivered.

4.1. Dimensional Tables

- **Slowly Changing Dimensions (SCD)**

DimDrivers has treated as a **Slowly Changing Dimension**. It contains two derived fields to store start date and end date to maintain and indicate historical records. (**Type 2**).

- **Static Dimension tables**

Both DimHub and DimDeliveryCenter tables are treated as static dimension tables and the values have been pre-inserted values which will remain unchanged.

4.2. Fact Tables

In the data warehouse design, there is only one fact table, namely, “FactPayment” which is falling under “Transaction Fact Table” type since it keeps a record of every even/transaction that happens. The table “StgPayments” in the staging database was mapped to FactPayment table in the data warehouse. “Payment_amount” and “Payment_fee” are **Fully Additive Measures**.

4.3. Identified Hierarchies

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

- hub state > hub city [DimHub]
- order created month > order created day > order created hour [DimFoodOrder]

4.4. Derived Columns

- **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 SCD table to keep track of historical value changes

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.

A SSIS project has been created using Visual Studio 2017 and all the SQL Server databases related to the Staging and Data Warehouse has been created using SQL Server 2017 Developer edition. The pre-requisites needed to be fulfilled to do ETL processes have mentioned below.

Pre-requisites:

- Visual Studio 2017
- VSIX Packages of SSIS, SSAS, and SSRS compatible with Visual Studio 2017
- SQL Server 2017

After creating the project, three packages were created to do the ETL processes.

- [1]. Food_Delivery_Staging.dtsx - To extract data from Source database to Staging database.
- [2]. Food_Delivery_Load_DW.dtsx - To transform and load data from Staging database to Data Warehouse.
- [3]. Step_6_UpdateColumns.dtsx - To update the fact table with transaction completion time and the number of hours to complete transactions.

5.1. Source Database to Staging Database

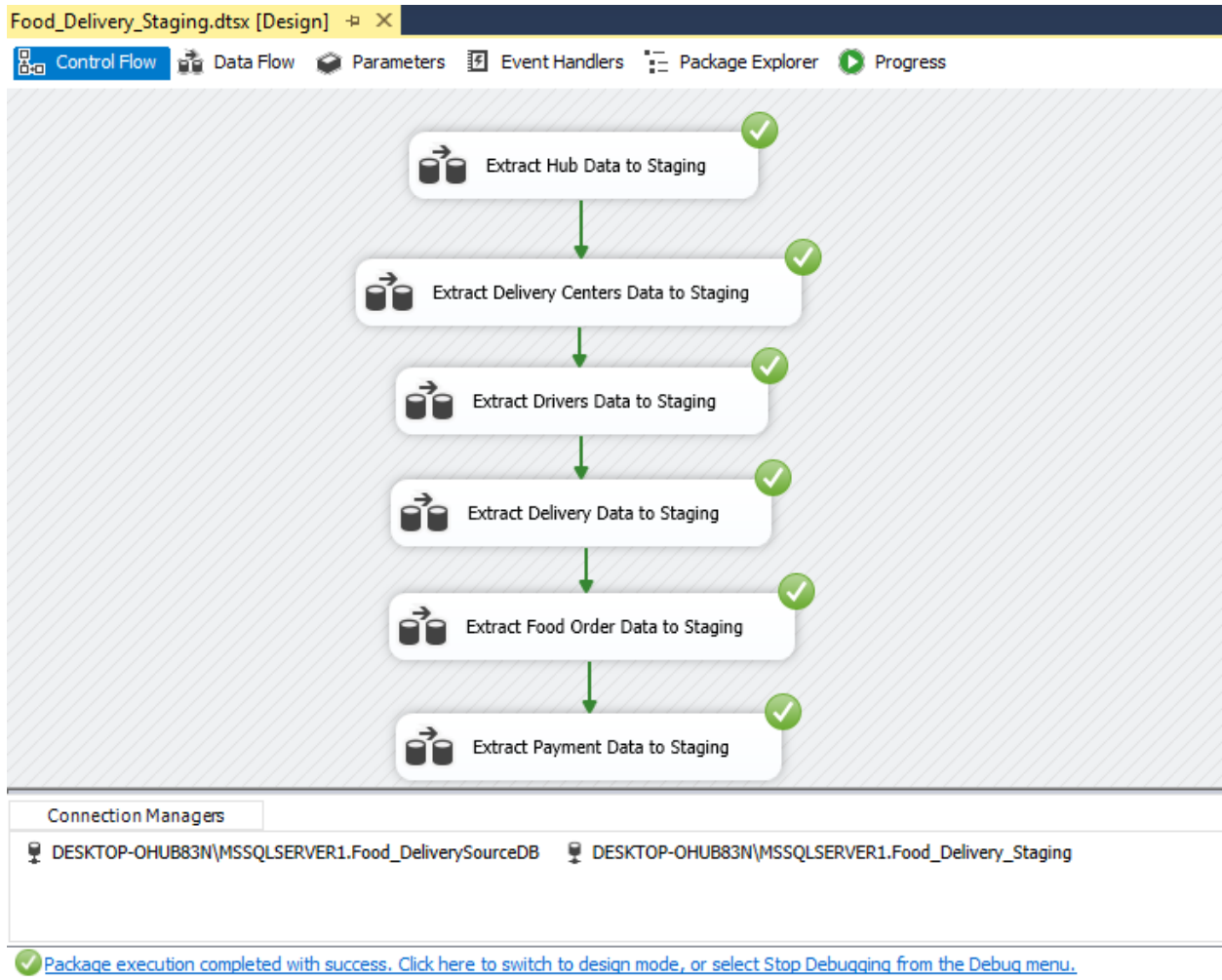


Figure 5.1. 1. Overall Control Flow of the Staging Package Tasks

OLE DB Source and OLE DB Destination components were used to extract data from tables in MSSQL source database “Food_DeliverySourceDB” to the staging tables of the MSSQL Staging database “Food_Delivery_Staging”. No transformations were done during the process.

An “Execute SQL Task” was used inside the relevant event handler of each data flow task on Pre-execution to truncate the staging tables beforehand the loading takes place to clear any previously staged data since they are simply outdated.

Screenshots of all the data sources that were staged, truncate details, and view of the staging tables after the data extraction are attached below:

5.1.1. Staging Hub Details:

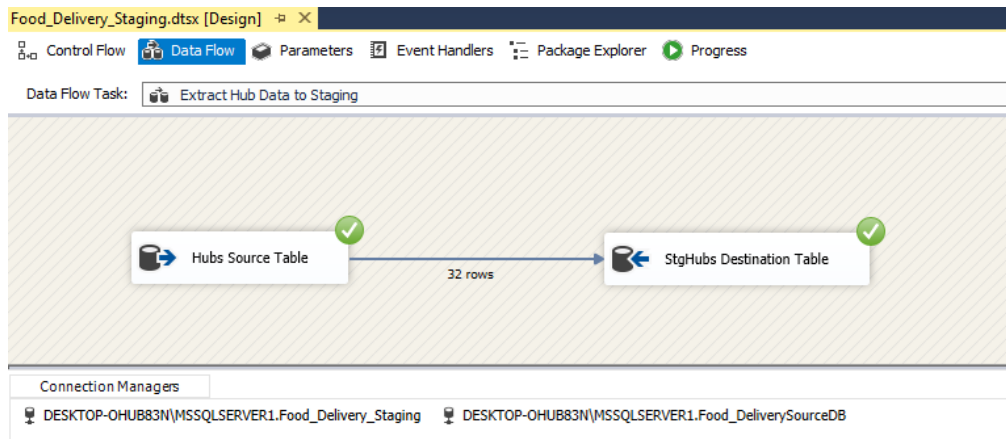


Figure 5.1.1.1. Data Flow of the “Extract Hub Data to Staging” Data Flow Task.

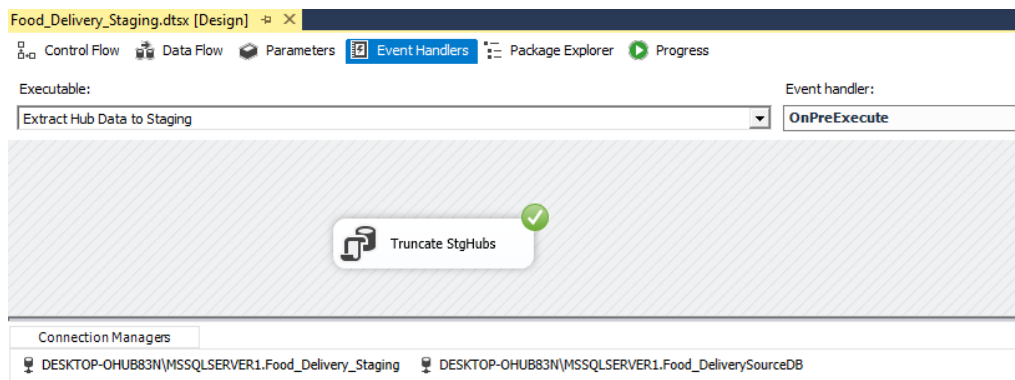


Figure 5.1.1. 1. The event handler which runs before the data flow gets executed, to truncate StgHubs table.

The screenshot shows the 'SQLQuery6.sql - DE...HUB83N\nesal (61))*' window. The query is a SELECT TOP (1000) statement. The results are displayed in a table with columns: hub_id, hub_name, hub_city, hub_state, hub_latitude, and hub_longitude. The results show two rows of data.

hub_id	hub_name	hub_city	hub_state	hub_latitude	hub_longitude
2	BLUE SHOPPING	PORTO ALEGRE	RS	-30.0474147796631	-51.213508605957
3	GREEN SHOPPING	PORTO ALEGRE	RS	-30.0374145507813	-51.2035217285156

Below the main table, there is a summary row with the column name '(No column name)' and a value of 32.

(No column name)
32

Figure 5.1.1.3. StgHubs table after data extraction.

5.1.2. Staging Delivery Center Details:

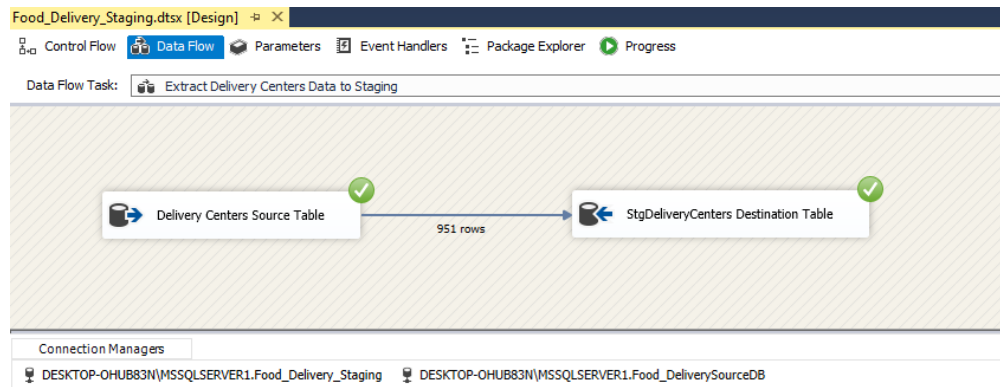


Figure 5.1. 2. 1. Data Flow of the “Extract Delivery Centers Data to Staging” Data Flow Task.

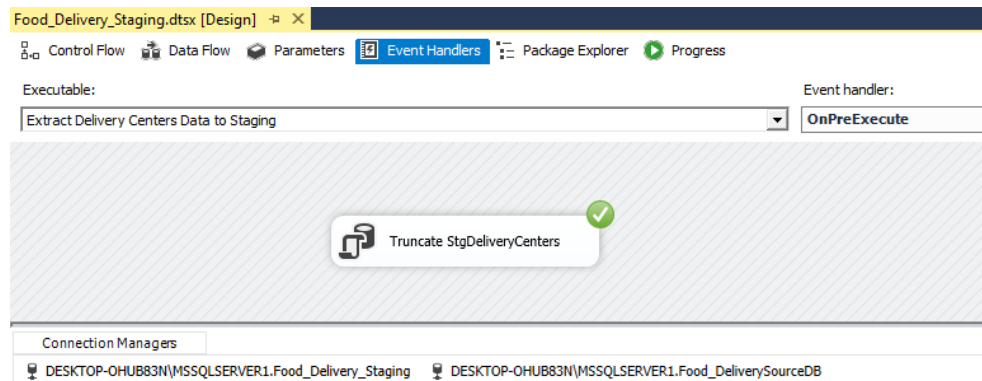


Figure 5.1.2. 2. The event handler which runs before the data flow gets executed, to truncate StgDeliveryCenters table.

The screenshot shows the results of a SQL query in the SQL Server Enterprise Manager. The query is a SELECT statement that retrieves the top 1000 rows from the StgDeliveryCenters table. The results are displayed in a table with 5 columns: center_id, hub_id, center_name, center_segment, and center_plan_price. The first 6 rows are shown, and a summary row indicates that there are 951 rows in total.

	center_id	hub_id	center_name	center_segment	center_plan_price
1	3	2	CUMIURI	FOOD	0
2	6	3	PIMGUCIS DA VIVA	FOOD	0
3	8	3	RASMUR S	FOOD	0
4	53	8	PAPA SUCIS	FOOD	0
5	54	8	VUZPI PAZZIS	FOOD	0
6	56	8	SUPSIO	FOOD	49
Summary					
	(No column name)				
1	951				

Figure 5.1.2. 3. StgDeliveryCenters table after data extraction.

5.1.3. Staging Driver Details:

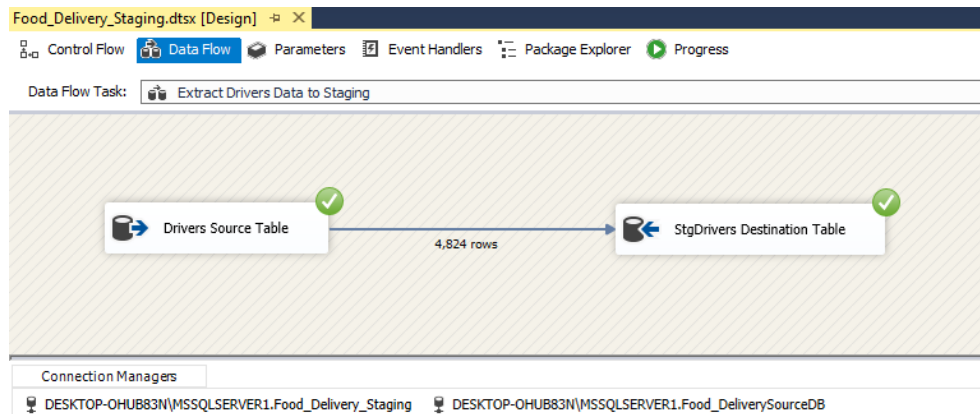


Figure 5.1.3.1. Data Flow of the “Extract Drivers Data to Staging” Data Flow Task

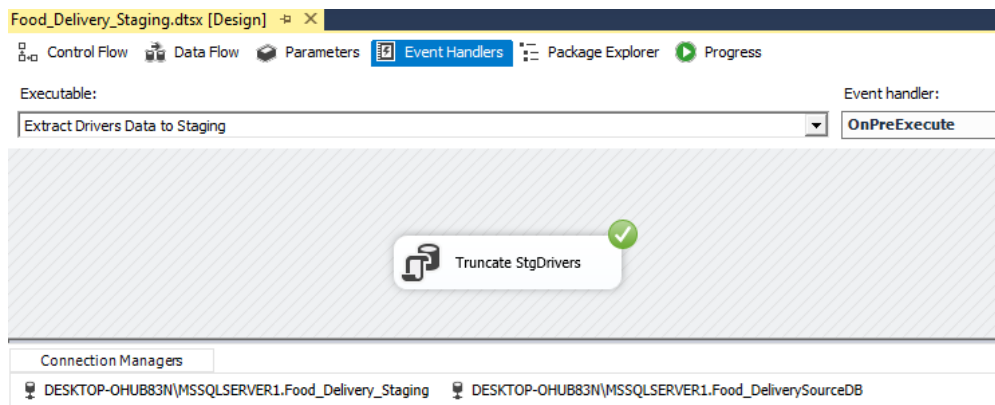


Figure 5.1.3. 3. The event handler which runs before the data flow gets executed, to truncate StgDrivers table.

```
SQLQuery3.sql - DE...HUB83N\nesal (61))*  SQLQuery2.sql - DE...HUB83N\nesal (60))*
/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [driver_id]
, [driver_modal]
, [driver_type]
, [phone_no]
, [gender]
FROM [Food_Delivery_Staging].[dbo].[StgDrivers]
select count(*) from [Food_Delivery_Staging].[dbo].[StgDrivers]
```

	driver_id	driver_modal	driver_type	phone_no	gender
1	38780	MOTOBOY	FREELANCE	732-555-0139	F
2	38803	MOTOBOY	FREELANCE	912-555-0186	F
3	38813	MOTOBOY	FREELANCE	1 (11) 500 555-0147	F

	(No column name)
1	4824

Figure 5.1.3. 3. StgDrivers table after data extraction.

5.1.4. Staging Delivery Details:

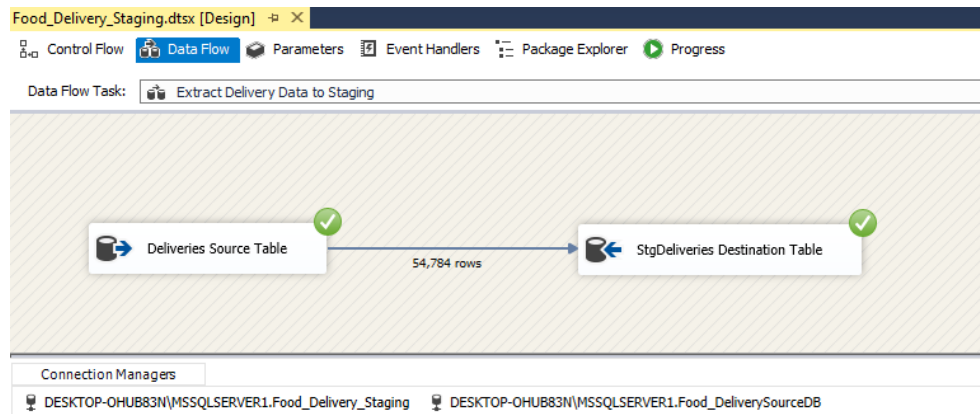


Figure 5.1.4.1. Data Flow of the “Extract Delivery Data to Staging” Data Flow Task.

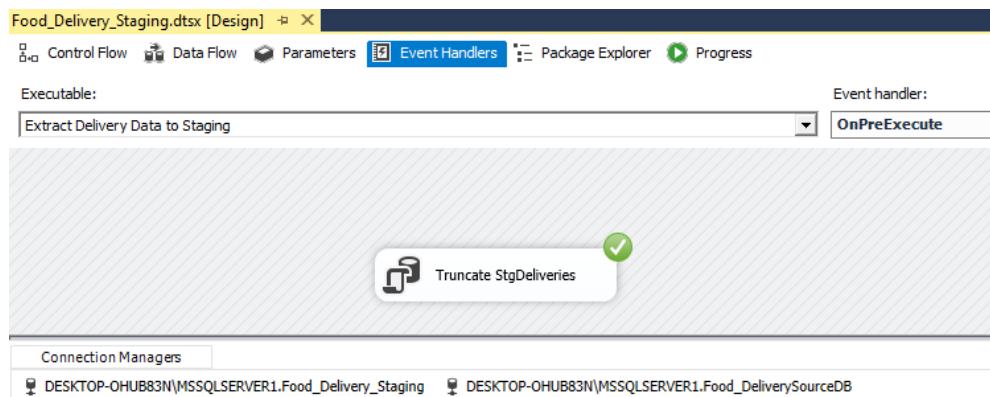


Figure 5.1.4. 4. The event handler which runs before the data flow gets executed, to truncate StgDeliveries table.

```
SQLQuery1.sql - DESKTOP-OHUB83N\mesal (58)) * X ~vsA9F9.sql - DESKTOP-OHUB83N\mesal (53))
/***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [delivery_id]
, [driver_id]
, [delivery_distance_meters]
, [delivery_status]
FROM [Food_Delivery_Staging].[dbo].[StgDeliveries]

select count(*) from [Food_Delivery_Staging].[dbo].[StgDeliveries]
```

	delivery_id	driver_id	delivery_distance_meters	delivery_status
1	71070440	298	959	DELIVERED
2	71072419	6714	1440	DELIVERED
3	71072406	16917	1429	DELIVERED
4	71071620	7549	2773	DELIVERED

	(No column name)
1	54784

Figure 5.1.4. 3. StgDeliveries table after data extraction.

5.1.5. Staging Food Order Details:

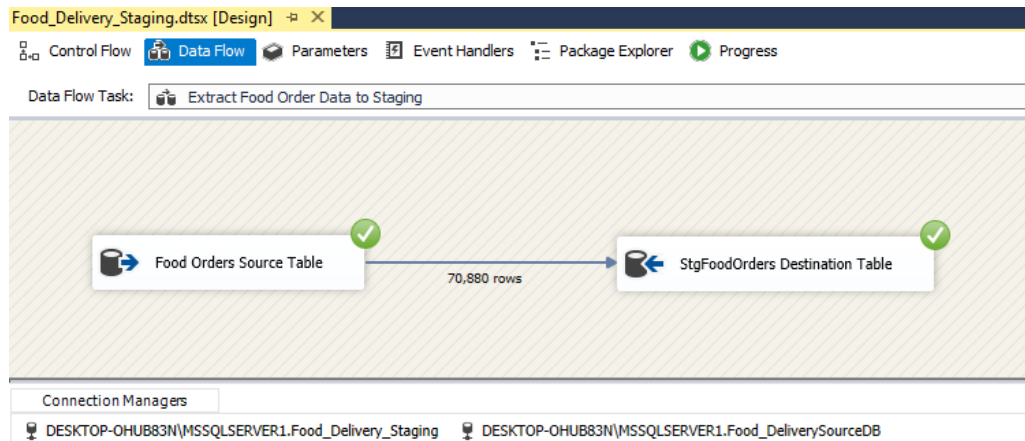


Figure 5.1.5.1. Data Flow of the “Extract Food Order Data to Staging” Data Flow Task.

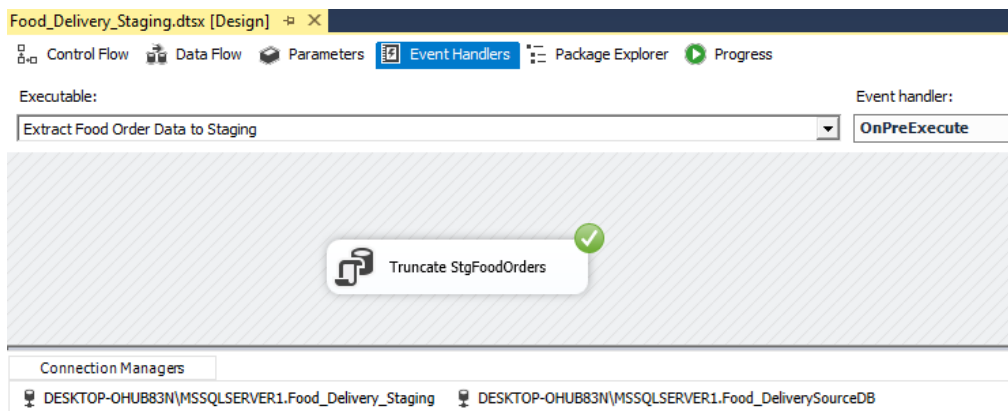


Figure 5.1.5. 5. The event handler which runs before the data flow gets executed, to truncate StgFoodOrders table.

order_id	order_status	order_amount	order_delivery_fee	order_created_hour	order_created_minute	order_created_day	order_created_month	order_created_year	order_created_time
68712450	FINISHED	74.9000015258789	0	21	34	2	1	2021	2021-01-02 21:34:00.00
68712506	FINISHED	154.100006103516	6.90000009536743	21	34	2	1	2021	2021-01-02 21:34:00.00

(No column name)

1 70880

Figure 5.1.5. 3. StgFoodOrders table after data extraction.

5.1.6. Staging Payment Details:

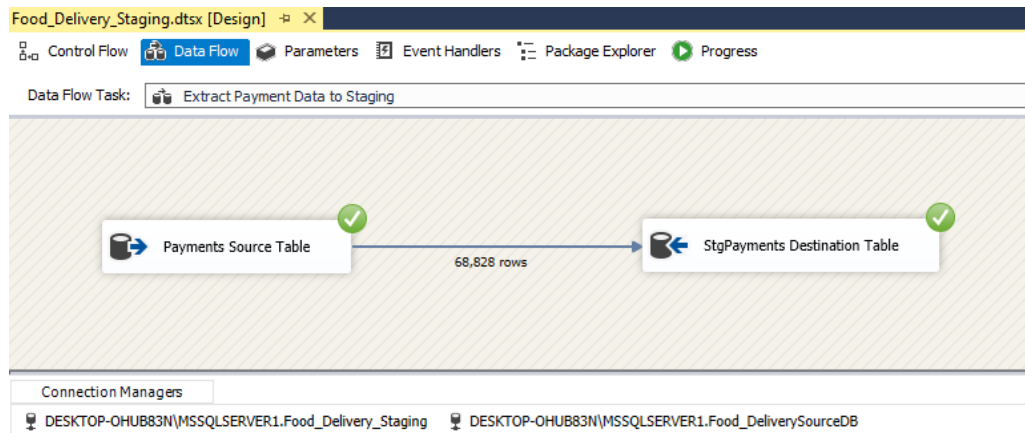


Figure 5.1.6.1. Data Flow of the “Extract Payment Data to Staging” Data Flow Task.

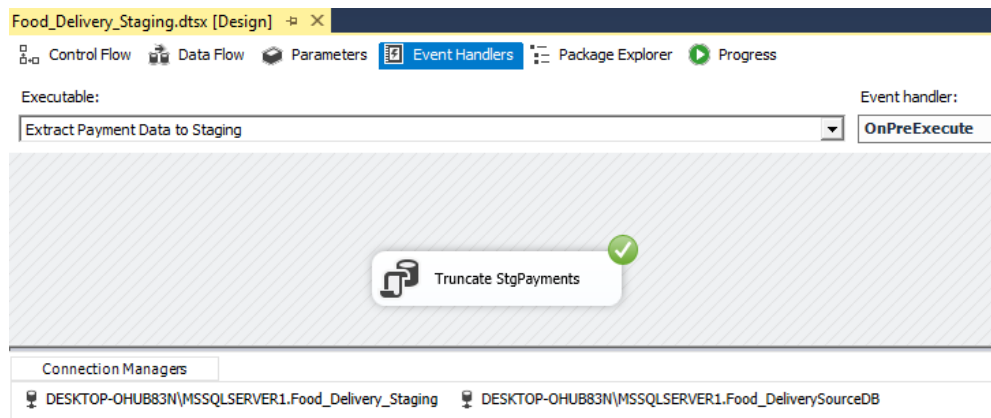


Figure 5.1.6. 6. The event handler which runs before the data flow gets executed, to truncate StgPayments table.

SQLQuery6.sql - DE...HUB83N\nesal (61)* X ~vsA9F9.sql - DESK...HUB83N\nesal (53) ~vs93A0.sql - DESK...HUB83N\nesal (52)

/****** Script for SelectTopNRows command from SSMS *****/

```
SELECT TOP (1000) [order_id]
, [center_id]
, [delivery_id]
, [payment_id]
, [payment_amount]
, [payment_fee]
, [payment_method]
, [payment_date]
FROM [Food_Delivery_Staging].[dbo].[StgPayments]
select count(*) from [Food_Delivery_Staging].[dbo].[StgPayments]
```

100 %

Results Messages

	order_id	center_id	delivery_id	payment_id	payment_amount	payment_fee	payment_method	payment_date
1	68410055	3512	68410055	68410055	394.809997558594	7.90000009536743	ONLINE	2021-01-01 00:04:00.0000000
2	68412721	3512	68412721	68412721	206.949996948242	5.59000015258789	ONLINE	2021-01-01 00:13:00.0000000

(No column name)

1	68828
---	-------

Figure 5.1.6. 3. StgPayments table after data extraction.

Later, to use the staging table data to analyze how the data looks like to determine what type of transformations we need to perform on the data, a data profiling task has been executed by selecting all tables option as shown in the figure 5.1.2.

A separate SSIS package named “Data_Profiling” was created for this task.

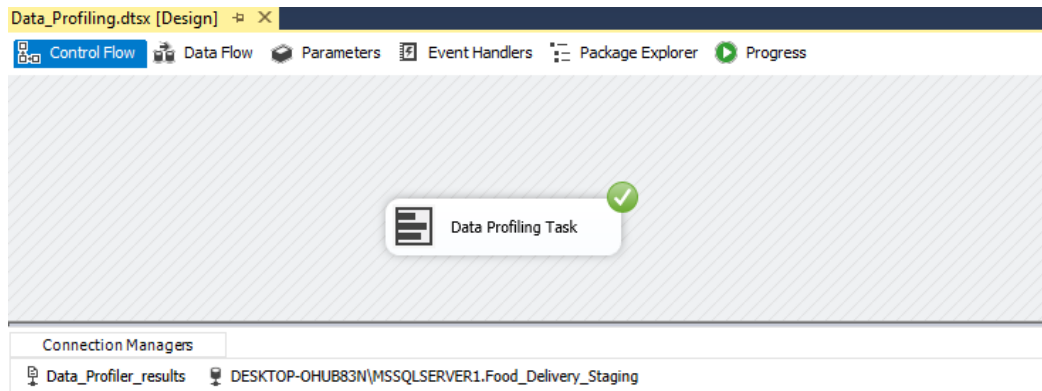


Figure 5.1.2. Data Profiling Task.

5.2. Staging Database to Data Warehouse

The second ETL process is to transfer data from Staging database to Data Warehouse. Unlike the first ETL process 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 such as data transformations, data cleaning, data enrichments. In the second ETL process the ETL task execution order is very most important. To preserve the links between dim and fact tables and to populate derived key columns with relevant surrogate keys, business keys and surrogate keys have been created.

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.

A separate SSIS package named “Food_Delivery_Load_DW.dtsx” has been created to transform and load data from staging database to data warehouse.

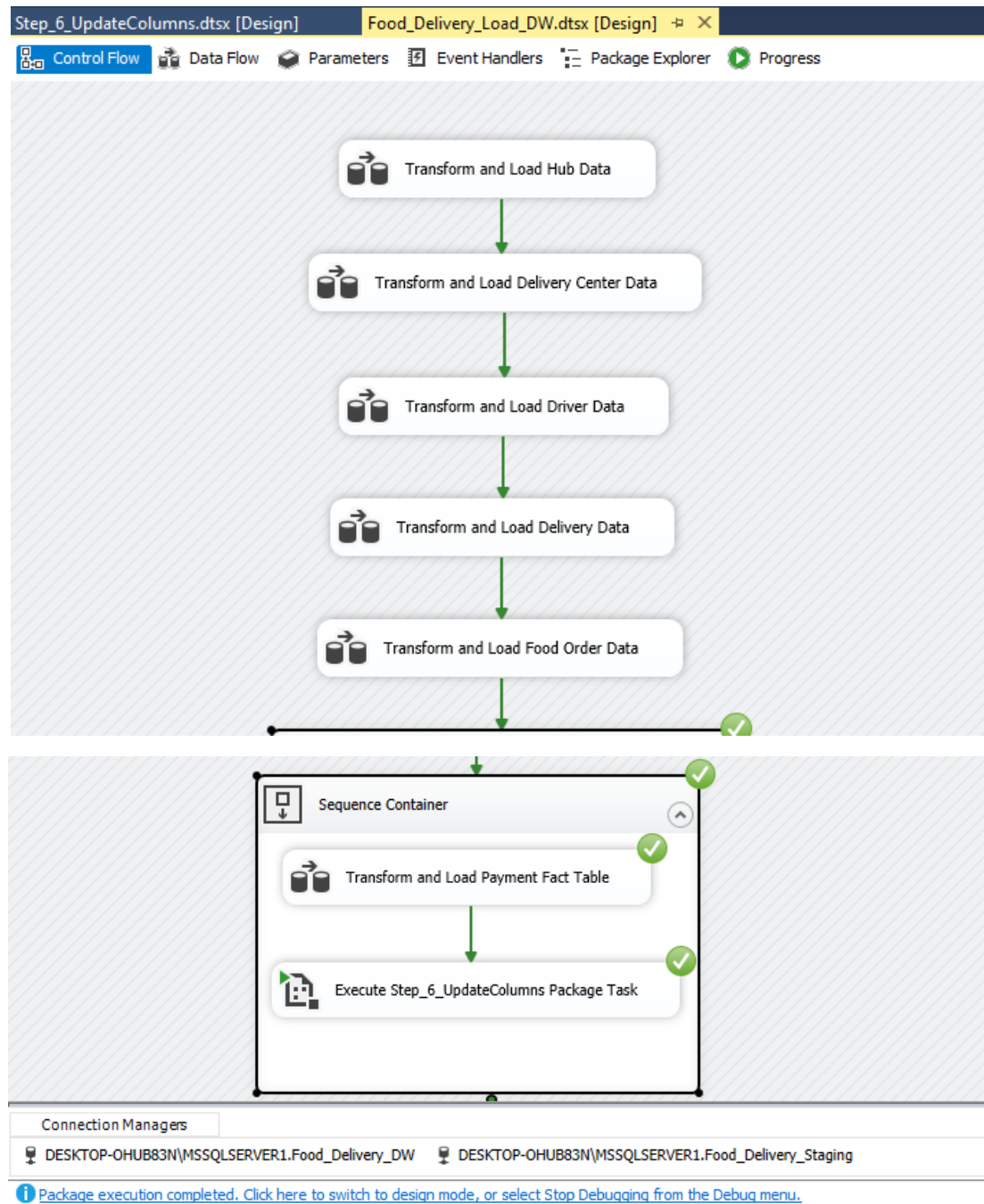


Figure 5.2. 1. Overall flow of ETL tasks in the Food_Delivery_Load_DW.dtsx package which is responsible for loading data to data warehouse after performing a set of complex transformations on the extracted data from the staging layer tables before loading.

Before performing any ETL task, fact and dimension tables were created in data warehouse. Then the ETL tasks were executed.

5.2.1. Transform and Load Hubs data to Data Warehouse:

There were no transformations to be done. So, 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 insert and update accordingly. The relevant stored procedure in this case is “**UpdateDimHub**”.

The stored procedure is used to figure out if the incoming data is already present in the data warehouse table “DimHub” using the business key provided. Based on the passed hub_id, which is the business key, 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.

Stored procedure call: **exec dbo.UpdateDimHub ?, ?, ?, ?, ?, ?**

```
SQLQuery1.sql - DE...HUB83N\nesal (54)) * X
CREATE PROCEDURE dbo.UpdateDimHub
    @hub_id tinyint,
    @hub_name nvarchar(50),
    @hub_city nvarchar(50),
    @hub_state nvarchar(50),
    @hub_latitude float,
    @hub_longitude float
AS
BEGIN
    if not exists (select HubSK
    from dbo.DimHub
    where AlternateHubID = @hub_id)
    BEGIN
        insert into dbo.DimHub
        (AlternateHubID, hub_name, hub_city, hub_state, hub_latitude, hub_longitude, InsertDate, ModifiedDate)
        values
        (@hub_id, @hub_name, @hub_city, @hub_state, @hub_latitude, @hub_longitude, GETDATE(), GETDATE())
    END;
    if exists (select HubSK
    from dbo.DimHub
    where AlternateHubID = @hub_id)
    BEGIN
        update dbo.DimHub
        set hub_name = @hub_name, hub_city = @hub_city, hub_state = @hub_state, hub_latitude = @hub_latitude, hub_longitude = @hub_longitude,
        ModifiedDate = GETDATE()
        where AlternateHubID = @hub_id
    END;
END;
```

100 %

Messages

Commands completed successfully.

100 %

Query executed successfully.

DESKTOP-OHUB83N\MSSQLSERVER... | DESKTOP-OHUB83N\esal... | Food_Delivery_DW

Figure 5.2.1. 1. Stored procedure to do insertions and updates: UpdateDimHub.

Before executing the task, it has checked whether the columns of the Staging table and the Dimension table were mapped correctly.

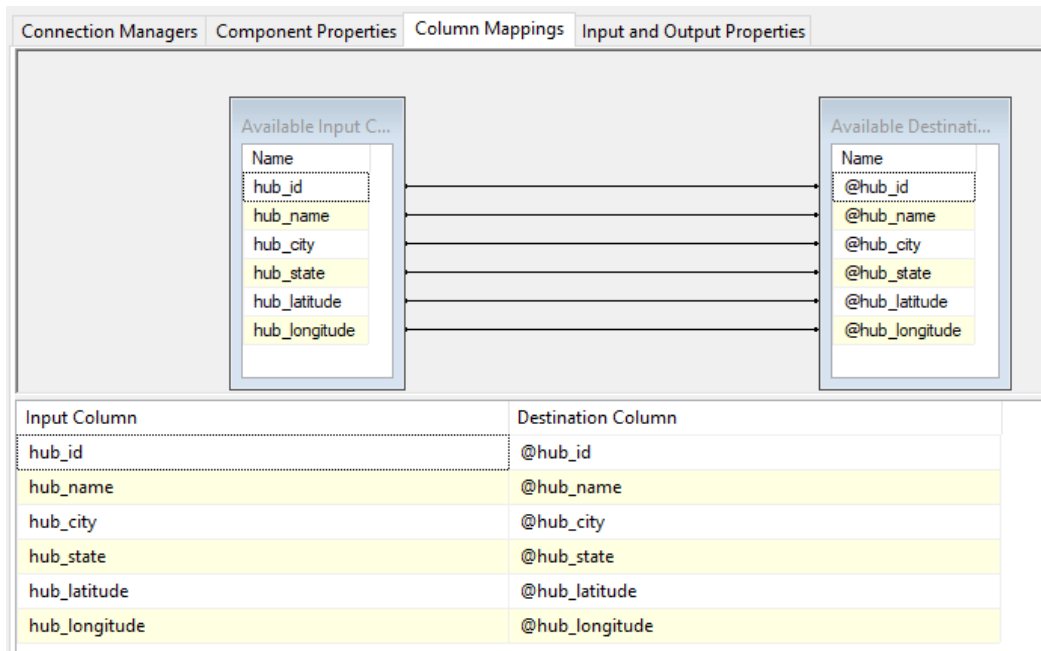


Figure 5.2.1. 2. Column mappings to load data.

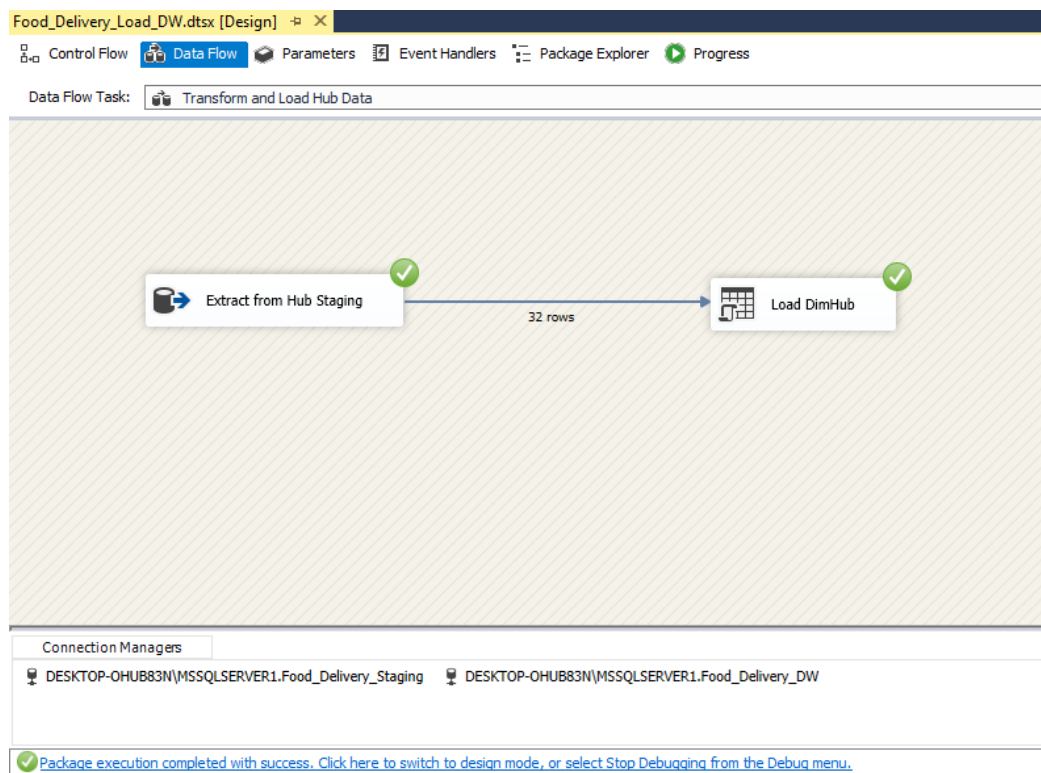


Figure 5.2.1. 3. Data Flow of the Data Flow Task "Transform and Load Hub Data" to DimHub in DW.

5.2.2. Transform and Load Delivery Centers data to Data Warehouse:

Since it has a foreign key reference to Hub dimension it was chosen secondly. The relevant stored procedure in this case is “UpdateDimDeliveryCenters”.

Stored procedure call: `exec dbo.UpdateDimDeliveryCenters ?, ?, ?, ?, ?`

```
SQLQuery4.sql - DE...HUB83N\nesal (51))* X SQLQuery1.sql - DE...HUB83N\nesal (54))* SQLQuery3.sql - DE...HUB83N\nesal (52))
CREATE PROCEDURE dbo.UpdateDimDeliveryCenters
    @center_id smallint,
    @HubKey tinyint,
    @center_name nvarchar(50),
    @center_segment nvarchar(50),
    @center_plan_price float
AS
BEGIN
    if not exists (select CenterSK
from dbo.DimDeliveryCenters
where AlternateCenterID = @center_id)
    BEGIN
        insert into dbo.DimDeliveryCenters
        (AlternateCenterID, HubKey, center_name, center_segment, center_plan_price,
        InsertDate, ModifiedDate)
        values
        (@center_id, @HubKey, @center_name, @center_segment, @center_plan_price,
        GETDATE(), GETDATE())
    END;
    if exists (select CenterSK
from dbo.DimDeliveryCenters
where AlternateCenterID = @center_id)
    BEGIN
        update dbo.DimDeliveryCenters
        set HubKey = @HubKey, center_name = @center_name, center_segment = @center_segment, center_plan_price = @center_plan_price,
        ModifiedDate = GETDATE()
        where AlternateCenterID = @center_id
    END;
END;
100 %
Messages
Commands completed successfully.
100 %
Activate Windows
Query executed successfully. | DESKTOP-OHUB83N\MSSQLSERVER... | DESKTOP-OHUB83N\esal... | Food_Delivery_DW
```

Figure 5.2.2. 2. Stored procedure to do insertions and updates: UpdateDimDeliveryCenters.

Since there is a foreign key reference from hubs table to delivery_centers table, both delivery_centers and hubs tables were needed to be joined to load and transfer data from delivery centers staging to the delivery center dimension table. To do this there was two ways, using lookups, or using sort + merge join. In this case, the second method which is **Sort + Merge Join** was used.

Steps to Transform and Load Delivery Centers Data to Data Warehouse:

- 1) Data were extracted from delivery centers staging table and the Hub dimension.
- 2) Delivery center details were sorted according to delivery id and hub details were sorted according to HubSK.
- 3) Two tables were joined using the “Left Outer Join”.
- 4) Finally, the data were loaded into DimDeliveryCenters after the specified mappings were done in the OLE DB Destination component.

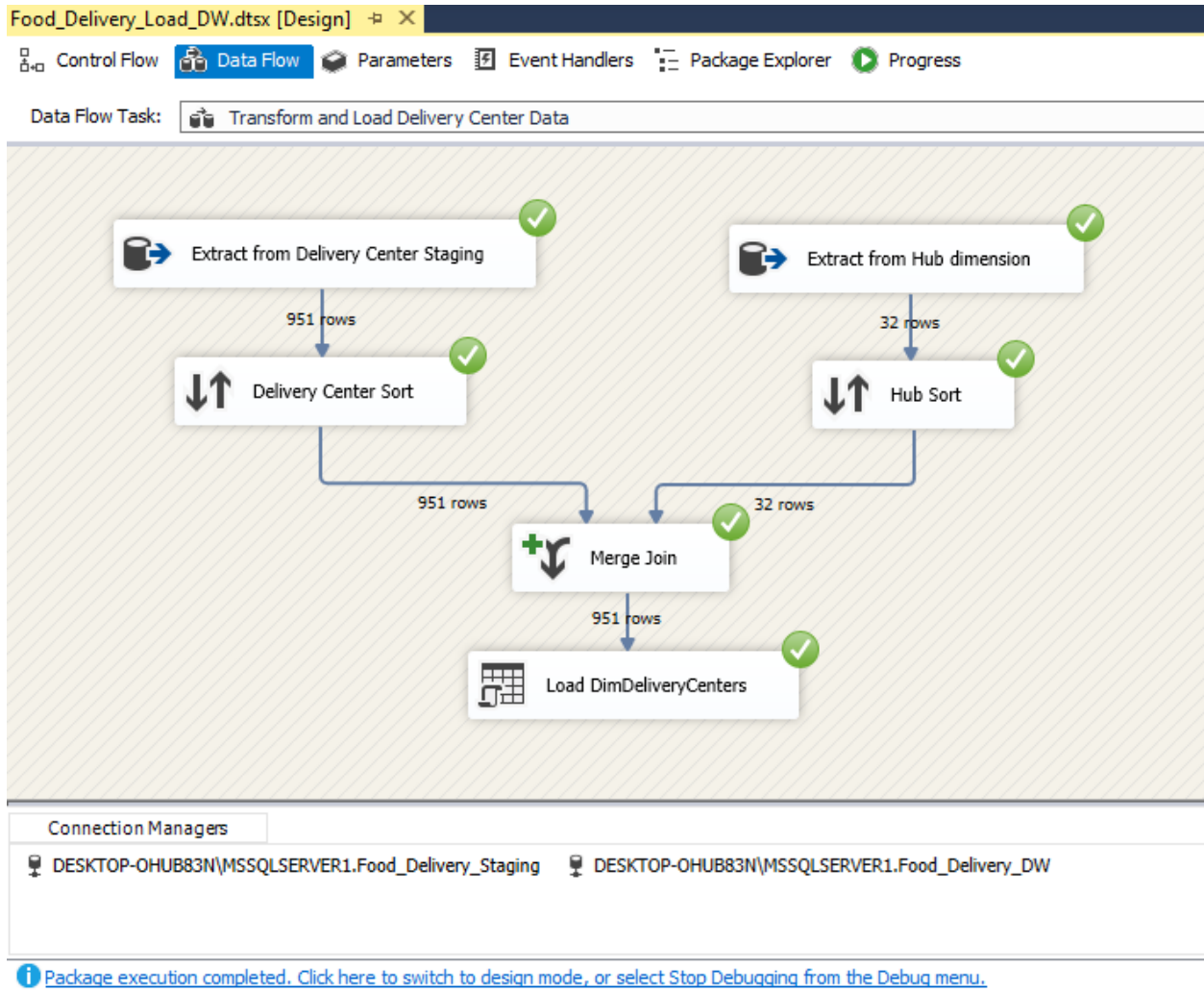


Figure 5.2.2. 2. Data Flow of the Data Flow Task "Transform and Load Delivery Center Data" to DimDeliveryCenters in DW.

5.2.3. Transform and Load Drivers data to Data Warehouse:

Drivers dimension (DimDrivers) was selected as the third dimension to be load into the data warehouse. It was considered as a Slowly Changing dimension as it was assumed that Food Delivery Company is interested in the types and models of the drivers who are doing delivery tasks and would like to perform analysis based on the driver types and models. Driver model and phone number 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. Driver type was set as Historical Column which means if it was updated in an existing row in DimDrivers, 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.

Steps to Transform and Load Data from Drivers Staging Table to Drivers Slowly Changing Dimension table:

- 1) Extract data from drivers staging table (StgDrivers).
- 2) Sorted records by driver id.
- 3) Then three derived columns were specified, InsertDate and ModifiedDate columns to get the current timestamp and phone_no column to replace null value with a default phone number.
- 4) Dimension was specified as a slowly changing dimension by selecting a change type for slowly changing dimension 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 (Figure 5.2.3. 1.).
- 5) Previously it was identified that the “phone_no” column of the Drivers table had assigned NULL values for drivers who have not provided contact numbers at the time of the registration, as a step of **data enrichment**, the phone number was set to company main delivery contact number where it was empty (Figure 5.2.3. 1.).

OLE DB Command Component

```
UPDATE [dbo].[DimDrivers] SET [EndDate] = ? , ModifiedDate = GETDATE()  
WHERE [AlternatedriverID] = ? AND [EndDate] IS NULL
```

This updates 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].[DimDrivers] SET [driver_modal] = ?, [phone_no] = ? , ModifiedDate =  
GETDATE() WHERE [AlternatedriverID] = ? 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.

- 6) Finally, the changes will get either inserted or updated in the DimDrivers dimension according to the specified mappings done in the OLE DB Destination component in the end (Figure 5.2.3. 1.).

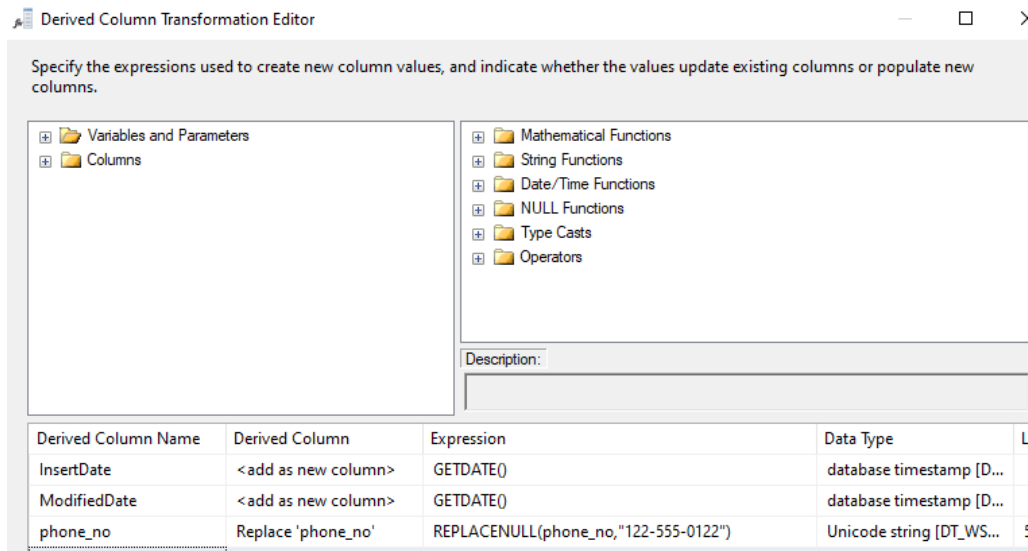


Figure 5.2.3. 1. new derived columns with current timestamp assigned

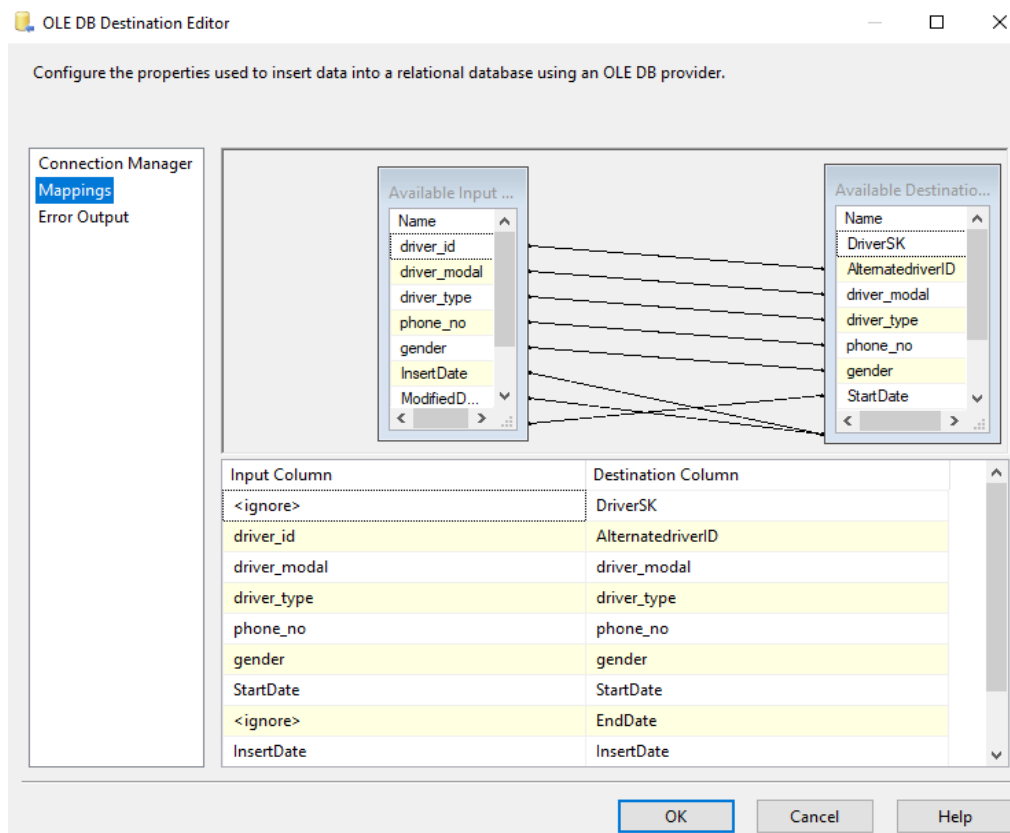


Figure 5.2.3. 2. mappings done in the OLE DB Destination component.

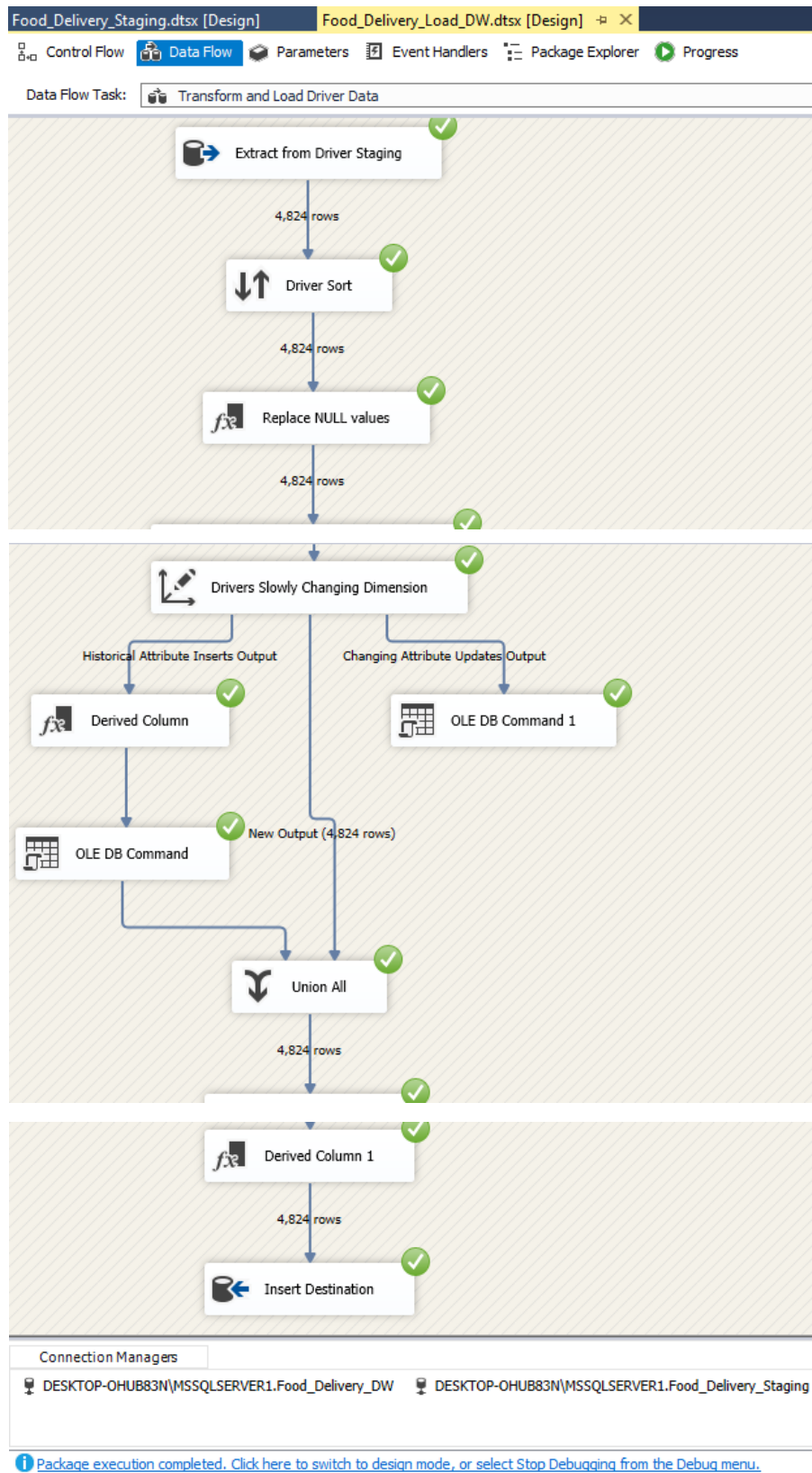


Figure 5.2.3. 3. Data Flow of the Data Flow Task "Transform and Load Driver Data" to DimDrivers in DW.

5.2.4. Transform and Load Deliveries data to Data Warehouse:

Since it has a foreign key reference to Drivers slowly changing dimension, it was chosen next. The relevant stored procedure in this case is “**UpdateDimDeliveries**”.

Stored procedure call: **exec dbo.UpdateDimDeliveries ?, ?, ?, ?**

```
SQLQuery16.sql - D:\HUB83N\nesal (58)) * SQLQuery14.sql - D:\HUB83N\nesal (53)) SQLQuery12.sql - D:\HUB83N\nesal (59))
CREATE PROCEDURE dbo.UpdateDimDeliveries
    @delivery_id int,
    @DriverKey int,
    @delivery_distance_meters int,
    @delivery_status nvarchar(50)
AS
BEGIN
    if not exists (select DeliverySK
from dbo.DimDeliveries
where AlternateDeliveryID = @delivery_id)
    BEGIN
        insert into dbo.DimDeliveries
        (AlternateDeliveryID, DriverKey, delivery_distance_meters, delivery_status,
        InsertDate, ModifiedDate)
        values
        (@delivery_id, @DriverKey, @delivery_distance_meters, @delivery_status,
        GETDATE(), GETDATE())
    END;
    if exists (select DeliverySK
from dbo.DimDeliveries
where AlternateDeliveryID = @delivery_id)
    BEGIN
        update dbo.DimDeliveries
        set DriverKey = @DriverKey, delivery_distance_meters = @delivery_distance_meters, delivery_status = @delivery_status,
        ModifiedDate = GETDATE()
        where AlternateDeliveryID = @delivery_id
    END;
END;
```

100 % Messages
Commands completed successfully.

100 % Activate
Query executed successfully. DESKTOP-OHUB83N\MSSQLSERVER... DESKTOP-OHUB83N\nesal... Fo

Figure 5.2.4. 3. Stored procedure to do insertions and updates: UpdateDimDeliveries.

The Loading process of the Deliveries Dimension is same as the delivery centers dimension since it has a foreign key reference. In this process **Lookups Method** was used instead of “sort + merge join” method to load and transfer data from deliveries staging to the deliveries dimension table.

In the lookup, full cache mode and OLE DB Connection Manager type was selected, and it was set to ignore failure when there are rows with no matching entries.

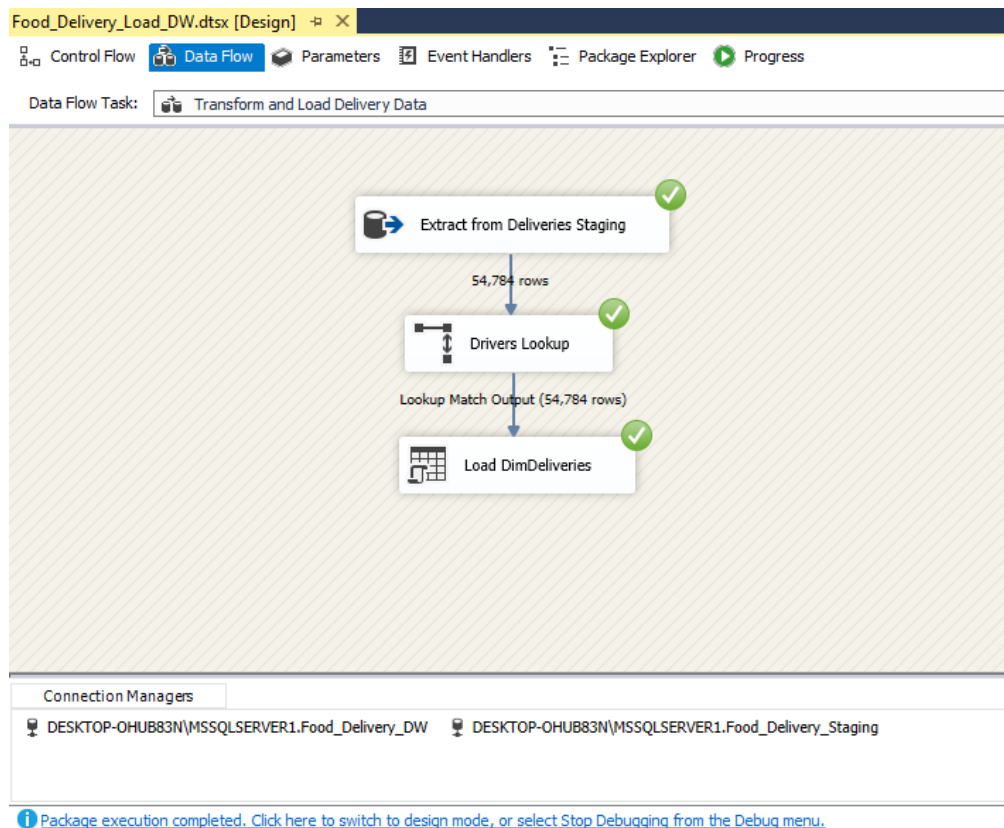


Figure 5.2.4. 2. Data Flow of the Data Flow Task "Transform and Load Delivery Data" to DimDeliveries in DW.

5.2.5. Transform and Load Food Orders data to Data Warehouse:

Since this was as same as Hubs table there were 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 insert and update accordingly. The relevant stored procedure in this case is "UpdateDimFoorOrders".

Stored procedure call: **exec dbo.UpdateDimFoodOrders ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?**

```

SQLQuery7.sql - DE...HUB83N\nesal (54)*  ~vsA9F9.sql - DESK...HUB83N\nesal (53)  ~vs93A0.sql - DESK...HUB83N\nesal (52)
CREATE PROCEDURE dbo.UpdateDimFoodOrders
    @order_id int,
    @order_status nvarchar(50),
    @order_amount float,
    @order_delivery_fee float,
    @order_created_hour tinyint,
    @order_created_minute tinyint,
    @order_created_day tinyint,
    @order_created_month tinyint,
    @order_created_year smallint,
    @order_created_time datetime,
    @order_delivered_time datetime
AS
BEGIN
    if not exists (select OrderSK from dbo.DimFoodOrders where AlternateOrderID = @order_id)
    BEGIN
        insert into dbo.DimFoodOrders
        (AlternateOrderID, order_status, order_amount, order_delivery_fee, order_created_hour, order_created_minute, order_created_day, order_created_month,
        order_created_year, order_created_time, order_delivered_time, InsertDate, ModifiedDate)
        values
        (@order_id, @order_status, @order_amount, @order_delivery_fee, @order_created_hour, @order_created_minute, @order_created_day, @order_created_month,
        @order_created_year, @order_created_time, @order_delivered_time, GETDATE(), GETDATE())
    END;
    if exists (select OrderSK from dbo.DimFoodOrders where AlternateOrderID = @order_id)
    BEGIN
        update dbo.DimFoodOrders
        set order_status = @order_status, order_amount = @order_amount, order_delivery_fee = @order_delivery_fee, order_created_hour = @order_created_hour,
        order_created_minute = @order_created_minute, order_created_day = @order_created_day, order_created_month = @order_created_month,
        order_created_year = @order_created_year, order_created_time = @order_created_time, order_delivered_time = @order_delivered_time,
        ModifiedDate = GETDATE()
        where AlternateOrderID = @order_id
    END;
END;

```

Figure 5.2.5. 4. Stored procedure to do insertions and updates: UpdateDimFoorOrders.

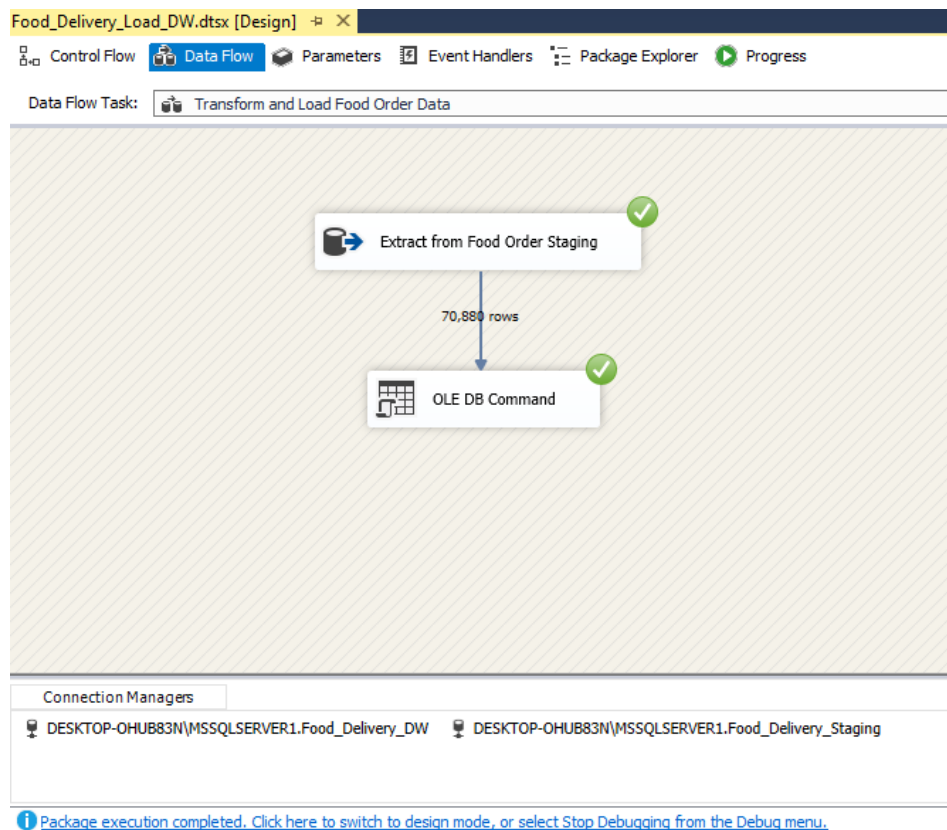


Figure 5.2.5. 2. Data Flow of the Data Flow Task "Transform and Load Food Order Data" to DimFoodOrders in DW.

5.2.6. Transform and Load Payments data to Data Warehouse: Accumulating Fact Table

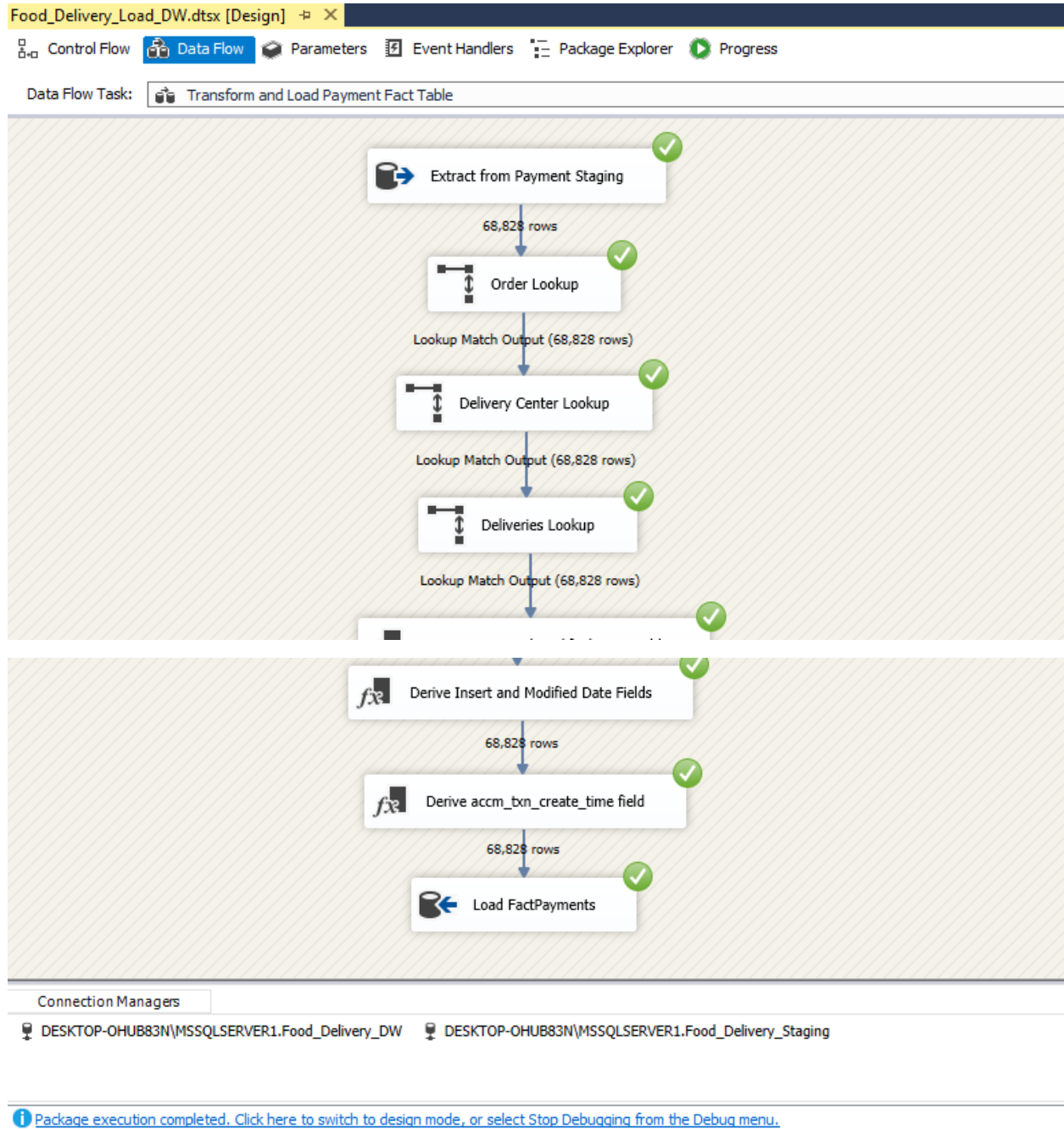


Figure 5.2.6. 1. Data Flow of the Data Flow Task "Transform and Load Payment Fact Table".

After loading to all the dimensions, lastly data was loaded into the fact table. Since the fact table refers a lot of dimension tables **lookups** were used (instead of using sort + merge join) to get the corresponding surrogate keys to establish table references.

Steps to Transform and Load Payments Data to Data Warehouse:

- 1). Firstly, the payment details were extracted from Payment staging table.
- 2). First Lookup is done to retrieve the OrderSK from DimFoodOrders by comparing the business key of an Order which is order_id (Figure 5.2.6. 2.).
- 3). Second Lookup is done to retrieve the CenterSK from DimDeliveryCenters by comparing the business key of a delivery center which is center_id (Figure 5.2.6. 3.).
- 4). Third Lookup is done to retrieve the DeliverySK from DimDeliveries by comparing the business key of a delivery which is delivery_id (Figure 5.2.6. 4.).
- 5). Then a derived column is used to assign insert date and modified date the current date (Figure 5.2.6. 5.).
- 6). Another derived column is used to assign accm_txn_create_time column the event time as same as the above (Figure 5.2.6. 6.).
- 7). Finally, the data was loaded into fact payment after matching columns (Figure 5.2.6. 7.).

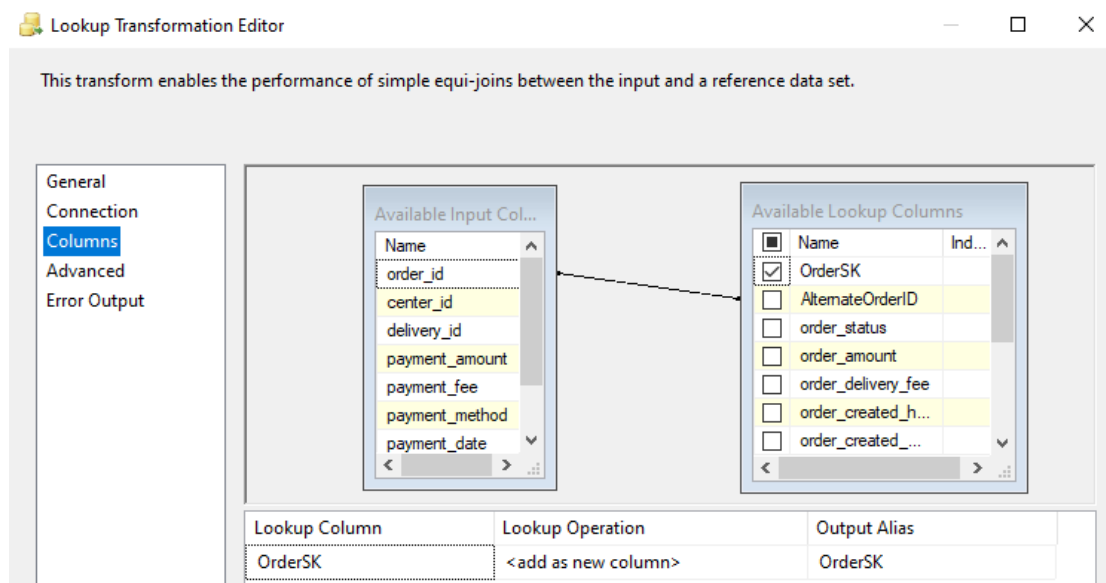


Figure 5.2.6. 2. Retrieving surrogate key from DimFoodOrders using its business key, order_id using the Lookup.

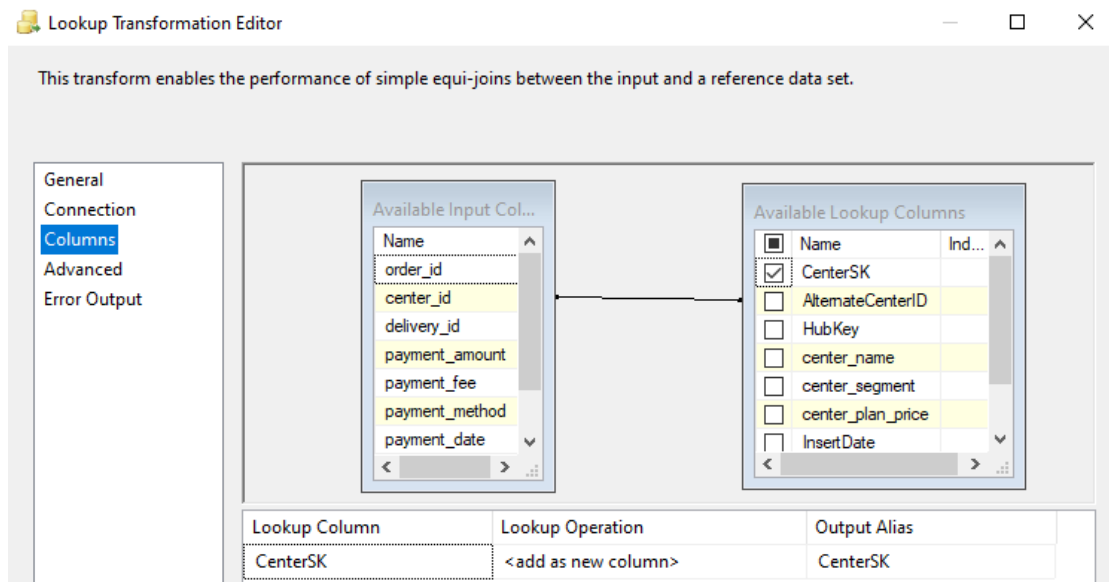


Figure 5.2.6. 3. Retrieving surrogate key from DimDeliveryCenters using its business key, center_id using the Lookup.

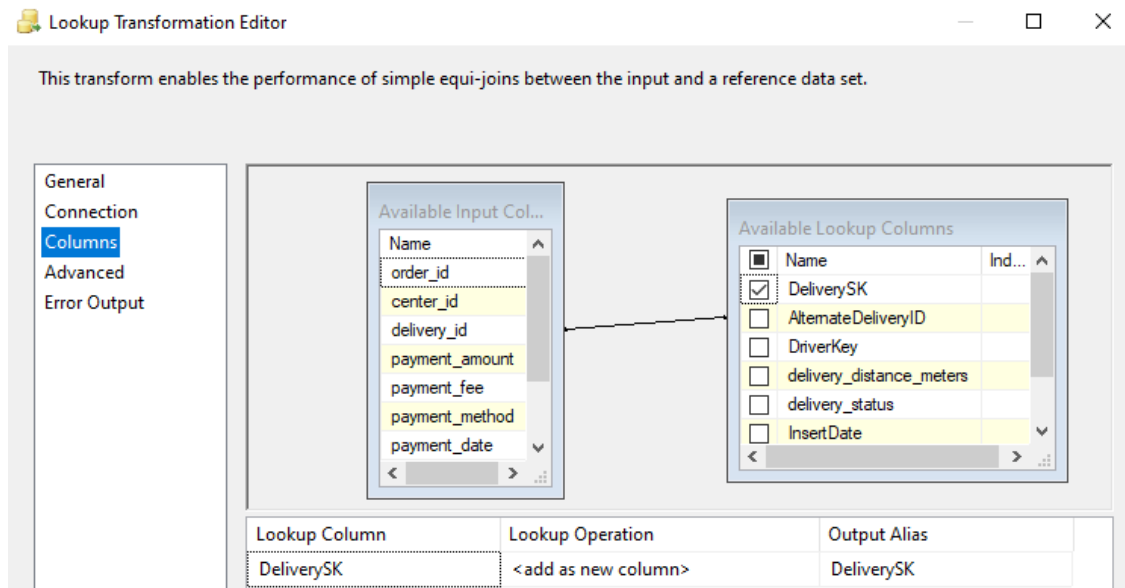


Figure 5.2.6. 4. Retrieving surrogate key from DimDeliveries using its business key, delivery_id using the Lookup.

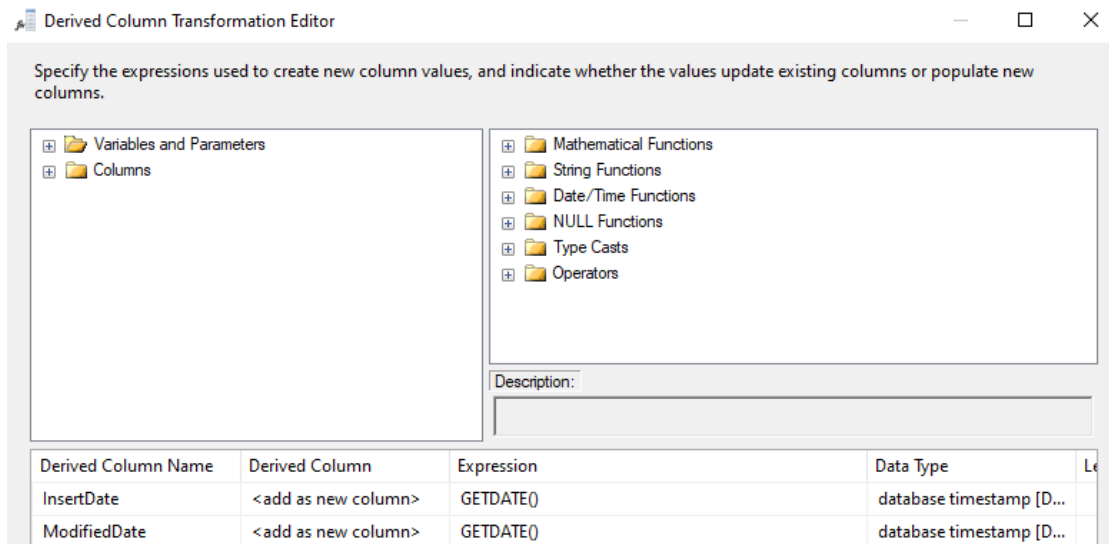


Figure 5.2.6. 4. Assign values to Derived Columns InsertDate and ModifiedDate.

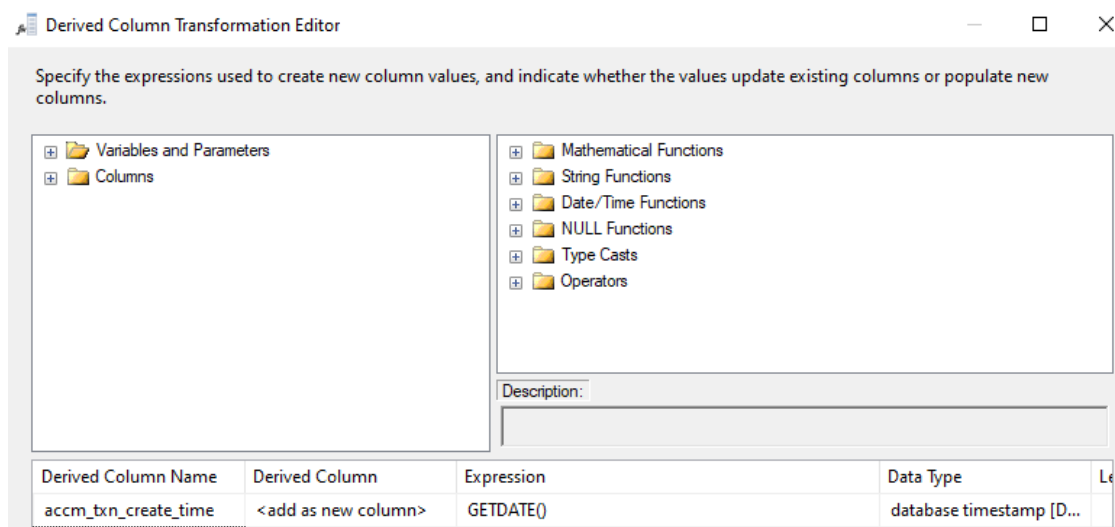


Figure 5.2.6. 4. Assign values to Derived Column accm_txn_create_time (Order created time).

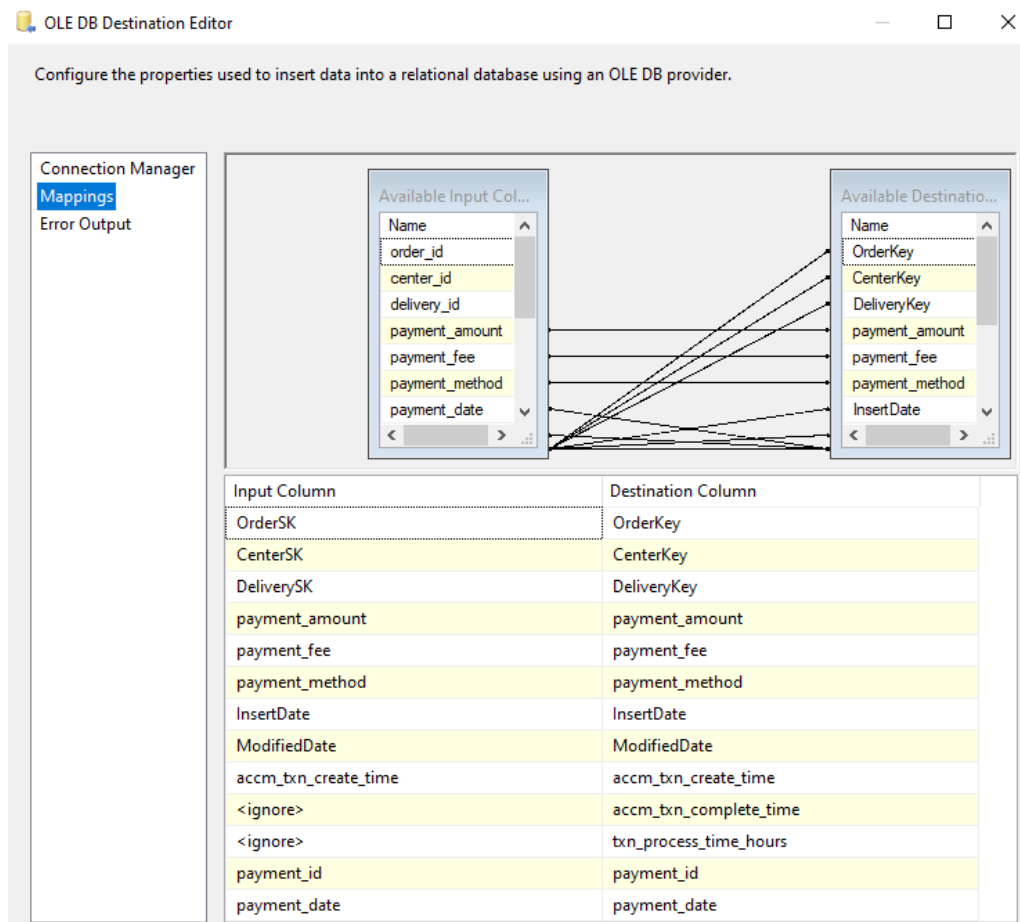
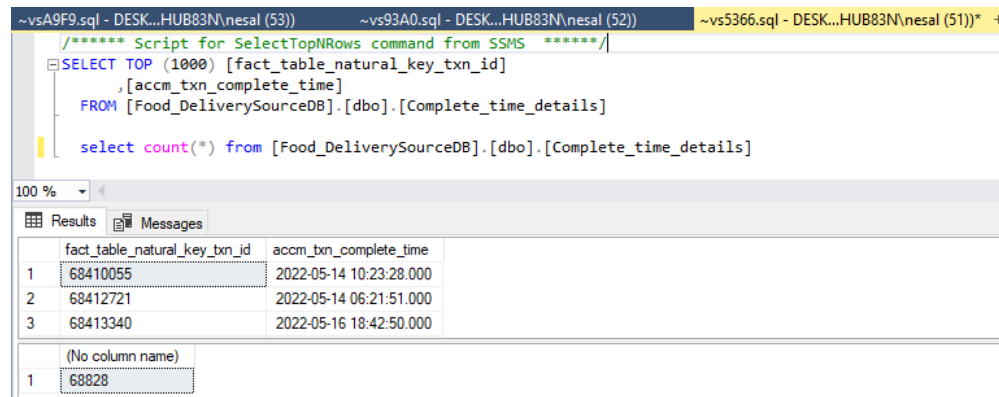


Figure 5.2.6. 4. Matching columns of the payments staging table and fact table to load data.

6. ETL Development – Accumulating fact table

Before the fact table was loaded into the data warehouse, 3 additional columns were added into the fact table named “accm_txn_create_time – To record the time of the payment”, “accm_txn_complete_time – To record the payment completion time (It was considered that to complete a transaction it may takes few hours or days)” and “txn_process_time_hours – Time to complete the payment”. The “accm_txn_create_time” column was assigned to be equal to the current system date when loading the fact table data. After executing the “Food_Delivery_Load_DW.dtsx” package all the columns were filled with transformed data except the accm_txn_complete_time and txn_process_time_hours columns which were assign to be null.

Then a separate XLSX table named “Complete_time_details” was added in the source database as shown below to fill the other two columns (It was exported as a flat file and added into data sources folder):



```
/****** Script for SelectTopNRows command from SSMS *****/
SELECT TOP (1000) [fact_table_natural_key_txn_id]
,[accm_txn_complete_time]
FROM [Food_DeliverySourceDB].[dbo].[Complete_time_details]

select count(*) from [Food_DeliverySourceDB].[dbo].[Complete_time_details]
```

	fact_table_natural_key_txn_id	accm_txn_complete_time
1	68410055	2022-05-14 10:23:28.000
2	68412721	2022-05-14 06:21:51.000
3	68413340	2022-05-16 18:42:50.000

	(No column name)
1	68828

Figure 6. 1. Details of “Complete_time_details.xlsx”.

Later, a separate ETL SSIS package named “Step_6_UpdateColumns.dtsx” was created which reads data from the above file (Figure 6.1.) and update the corresponding “accm_txn_complete_time” in the DW fact table and update “txn_process_time_hours” by taking the hours difference between “accm_txn_create_time” and “accm_txn_complete_time”.

Steps to the above process are mentioned below:

- 1). First, a data flow task named “Update accm_txn_create_time, accm_txn_complete_time” was added in the control flow (Figure 6. 2.).
- 2). Secondly, in the data flow, data from Payments Fact table in the data warehouse and Complete time details from the source database was extracted.
- 3). The data were sorted according to payment_id(unique) in the fact table and txn_id (primary key) in Complete time details.

- 4). Two tables were joined using the merge join component (Figure 6. 3.).
- 5). Then a formula to calculate the transaction process time (in hours) was assigned to the derived column named “txn_process_time_hours” (Figure 6. 4.).
- 6). At the end of data flow a destination component was added and mapped the columns to update two null columns (Figure 6. 5.).
- 7). Finally, an execute task was added at the end of the data warehouse SSIS package below the fact table data flow, inside a container to be execute after loading the fact table and executed it (Figure 6. 6. And Figure 6.7.).

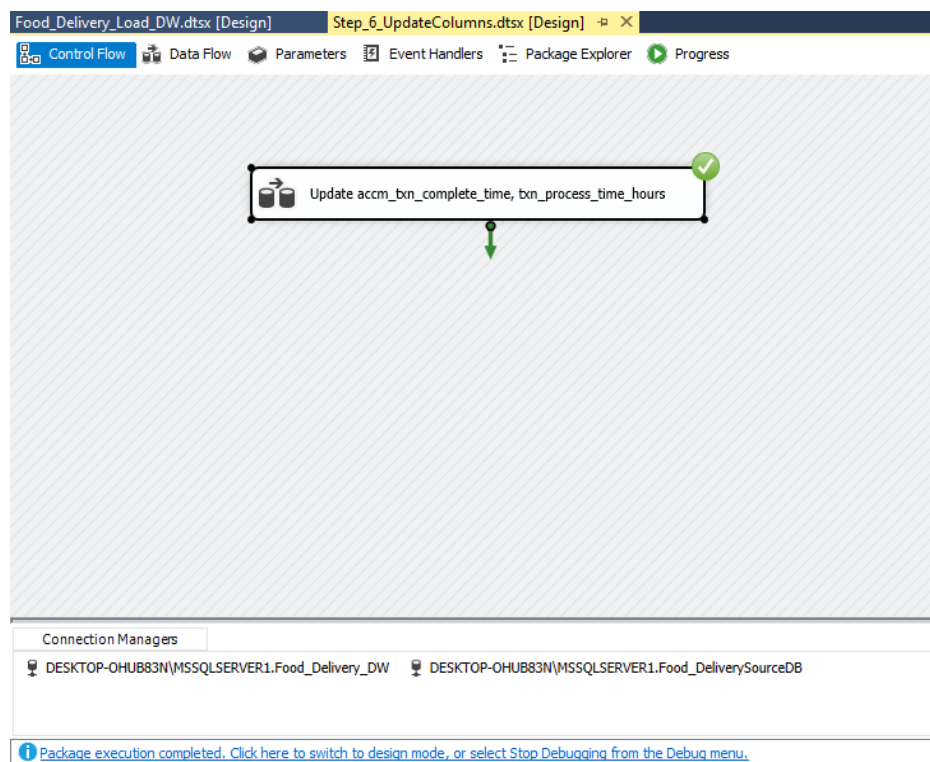


Figure 6. 2. Adding data flow task to the control flow to calculate the transaction process time.

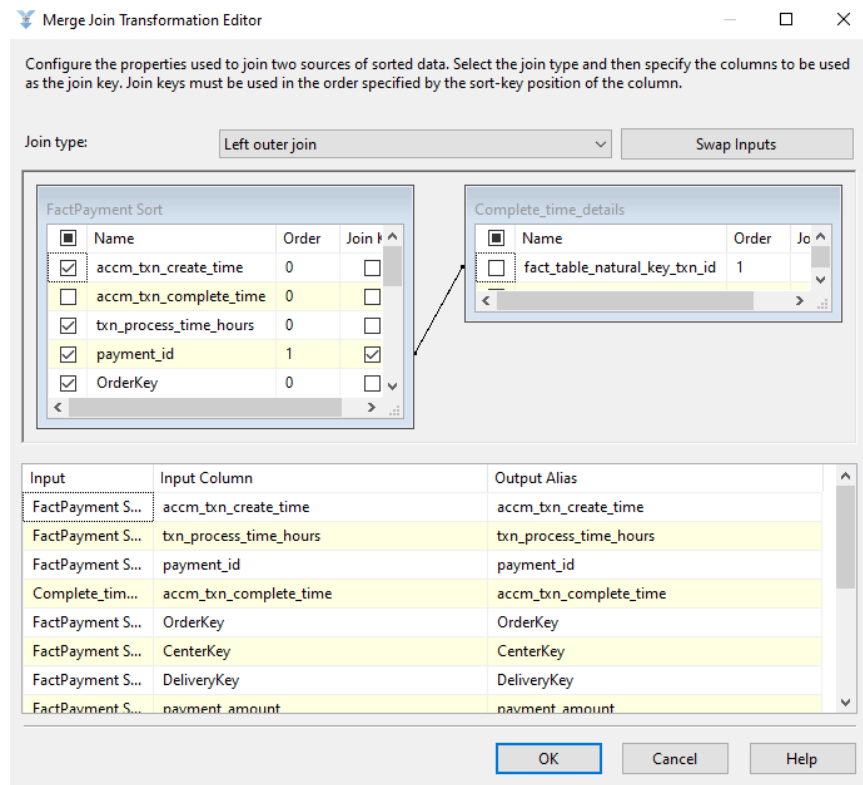


Figure 6. 3. Payment fact table and Complete time details table were joined using “Left Outer Join”.

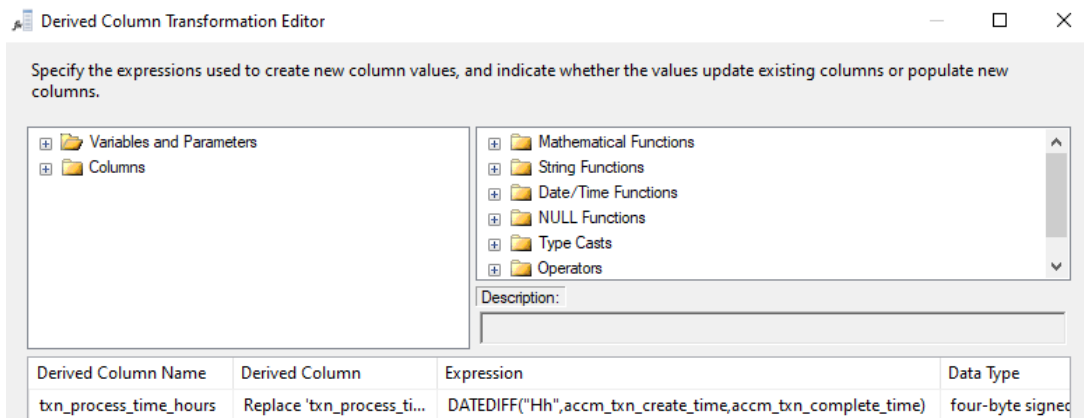


Figure 6. 4. Assign calculation expression to the derived column “txn_process_time_hours”.

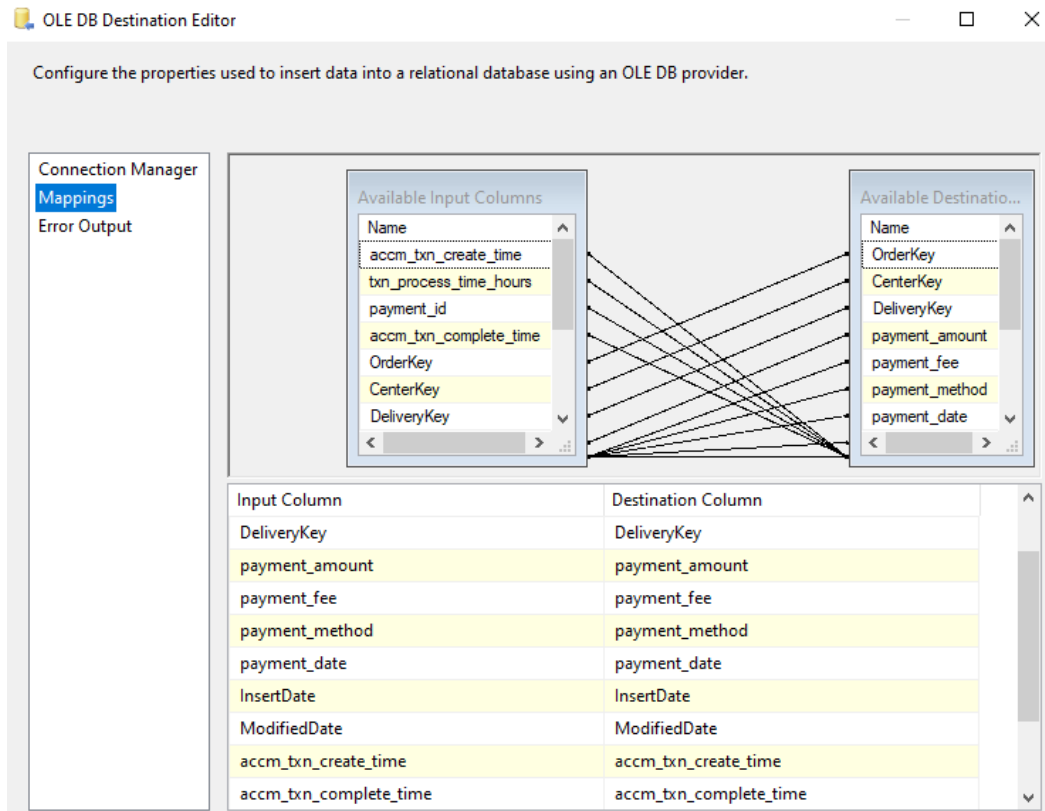


Figure 6. 5. Matching the columns of previously loaded fact table to the destination table before loading data to update “accm_txn_create_time” and “accm_txn_complete_time” columns.

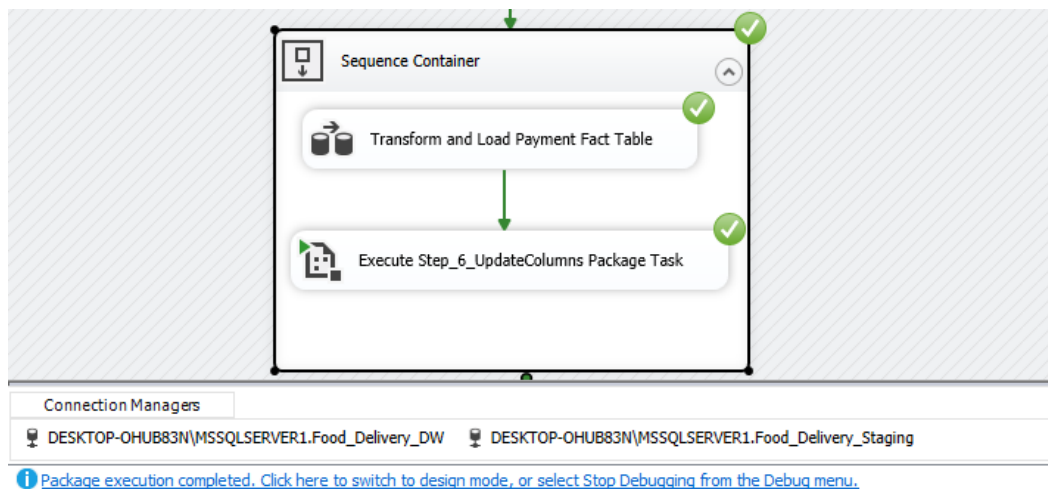


Figure 6. 6. Execution of the package “Step_6_UpdateColumns” – Control Flow View.

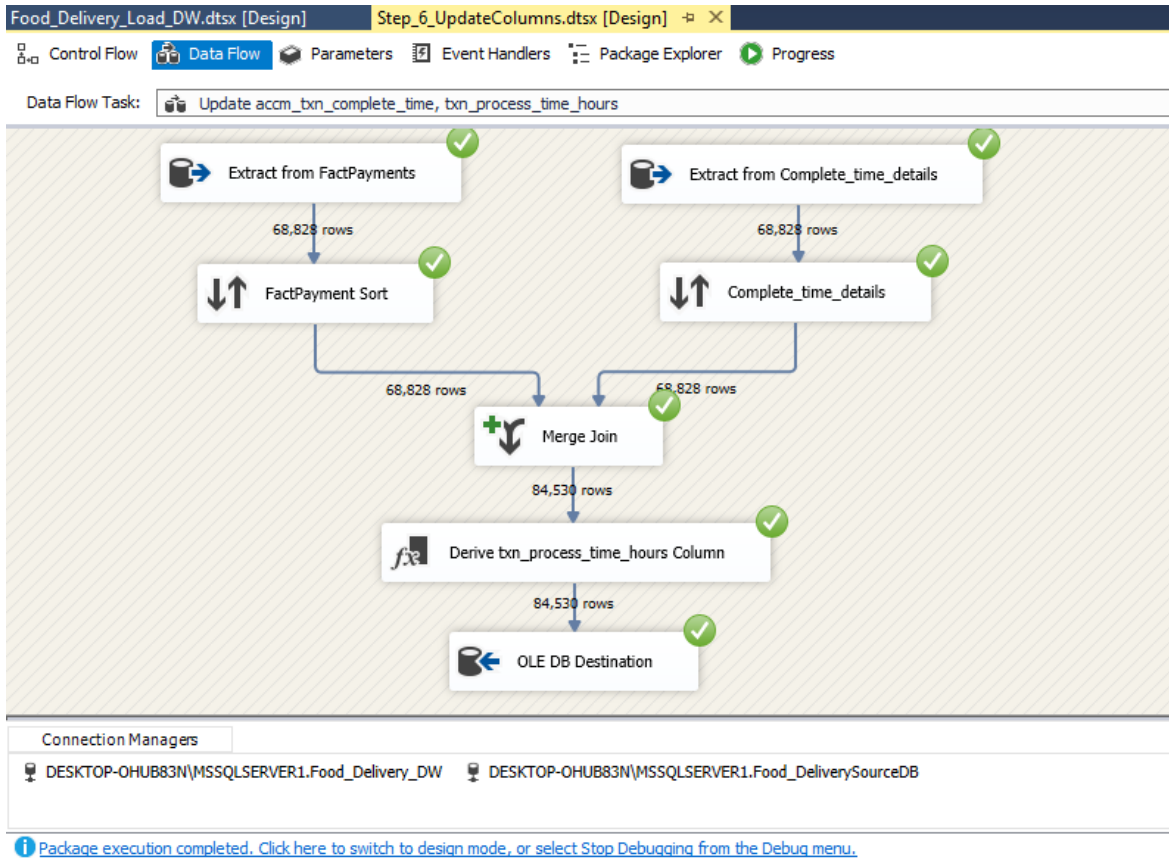


Figure 6. 7. Execution of the package “Step_6_UpdateColumns” – Data Flow View.

- The final fact table looks as shown below in the data warehouse:

SQLQuery2.sql - DE...HUB83N\mesal (60)* SQLQuery1.sql - DE...HUB83N\mesal (58)*

```

SELECT TOP (1000) [OrderKey]
, [CenterKey]
, [DeliveryKey]
, [payment_id]
, [payment_amount]
, [payment_fee]
, [payment_method]
, [payment_date]
, [InsertDate]
, [ModifiedDate]
, [accm_txn_create_time]
, [accm_txn_complete_time]
, [txn_process_time_hours]
FROM [Food_Delivery_DW].[dbo].[FactPayments]

select * from [Food_Delivery_DW].[dbo].[FactPayments]
select COUNT(*) from [Food_Delivery_DW].[dbo].[FactPayments]
  
```

Results

key	payment_id	payment_amount	payment_fee	payment_method	payment_date	InsertDate	ModifiedDate	accm_txn_create_time	accm_txn_complete_time	txn_process_time_
1	68410055	394.809997558594	7.90000009536743	ONLINE	2021-01-01 00:04:00.0000000	2022-05-11 18:01:38.060	2022-05-11 18:01:38.060	2022-05-11 18:01:38.077	2022-05-14 10:23:28.000	64
2	68412721	206.949996948242	5.59000015258789	ONLINE	2021-01-01 00:13:00.0000000	2022-05-11 18:01:38.060	2022-05-11 18:01:38.060	2022-05-11 18:01:38.077	2022-05-14 06:21:51.000	60
3	68413340	58.7999992370605	1.5900000333786	ONLINE	2021-01-01 00:19:00.0000000	2022-05-11 18:01:38.060	2022-05-11 18:01:38.060	2022-05-11 18:01:38.077	2022-05-14 18:42:50.000	120
4	68414018	45.7999992370605	0.92000016689301	ONLINE	2021-01-01 00:26:00.0000000	2022-05-11 18:01:38.060	2022-05-11 18:01:38.060	2022-05-11 18:01:38.077	2022-05-14 11:22:37.000	65
5	68414309	106.800003051758	2.88000011444092	ONLINE	2021-01-01 00:56:00.0000000	2022-05-11 18:01:38.060	2022-05-11 18:01:38.060	2022-05-11 18:01:38.077	2022-05-12 23:10:32.000	29
6	68414512	57.7999992370605	1.55999994277954	ONLINE	2021-01-01 00:56:00.0000000	2022-05-11 18:01:38.060	2022-05-11 18:01:38.060	2022-05-11 18:01:38.077	2022-05-12 03:59:53.000	9
7	68414563	76.8999996185303	0.400000015960454	ONLINE	2021-01-01 01:56:00.0000000	2022-05-11 18:01:38.060	2022-05-11 18:01:38.060	2022-05-11 18:01:38.077	2022-05-14 18:17:41.000	77

(No column name)
1 84530

Activate Windows

Figure 6. 8. Details of “FactPayments” after executing the package “Step_6_UpdateColumns”.

7. References

- [1].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-May-2022]
- [2].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-May-2022]
- [3].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-May-2022]
- [4].exceldemy.com, 'How to Generate Random Date and Time in Excel (3 Ways)',
<https://www.exceldemy.com/generate-random-date-and-time-in-excel/> [Accessed on: 01-May-2022]

Appendix

Appendix – A: Stored Procedures Used in Data Warehouse Database

A-1. SQL query to create the Hub Dimension: DimHub

```
SQLQuery7.sql - DE...HUB83N\nesal (54))* -p X ~vsA9F9.sql - DESK...HUB83N\nesal
drop table if exists DimHub;
create table DimHub(
    HubSK tinyint identity(1,1) primary key,
    AlternateHubID int,
    hub_name nvarchar(50),
    hub_city nvarchar(50),
    hub_state nvarchar(50),
    hub_latitude float,
    hub_longitude float,
    InsertDate datetime,
    ModifiedDate datetime
)
```

A-2. SQL query to create the Delivery Centers Dimension: DimDeliveryCenters

```
SQLQuery7.sql - DE...HUB83N\nesal (54))* -p X ~vsA9F9.sql - DESK...HUB83N\nesal (53))
drop table if exists DimDeliveryCenters;
create table DimDeliveryCenters(
    CenterSK smallint identity(1,1) primary key,
    AlternateCenterID int,
    HubKey tinyint foreign key references DimHub(HubSK),
    center_name nvarchar(50),
    center_segment nvarchar(50),
    center_plan_price nvarchar(50),
    InsertDate datetime,
    ModifiedDate datetime
)
```

A-3. SQL query to create the Drivers Dimension: DimDrivers

```
SQLQuery7.sql - DE...HUB83N\nesal (54))* -p X ~vsA9F9.sql - DESK...HUB83N\nesal (53))
drop table if exists DimDrivers;
create table DimDrivers(
    DriverSK int identity(1,1) primary key,
    AlternatedriverID int,
    driver_modal nvarchar(50),
    driver_type nvarchar(50),
    StartDate datetime,
    EndDate datetime,
    InsertDate datetime,
    ModifiedDate datetime
)
```

A-4. SQL query to create the Deliveries Dimension: DimDeliveries

```
SQLQuery7.sql - DE...HUB83N\nesal (54))* -> X ~vsA9F9.sql - DESK...HUB83N\nesal (53))
drop table if exists DimDeliveries;
create table DimDeliveries(
    DeliverySK int identity(1,1) primary key,
    AlternateDeliveryID int,
    DriverKey int foreign key references DimDrivers(DriverSK),
    delivery_distance_meters int,
    delivery_status nvarchar(50),
    InsertDate datetime,
    ModifiedDate datetime
)
```

A-5. SQL query to create the Food Orders Dimension: DimFoodOrders

```
SQLQuery7.sql - DE...HUB83N\nesal (54))* -> X ~vsA9F9.sql - DESK...HUB83N\nesal (53))
drop table if exists DimFoodOrders;
create table DimFoodOrders(
    OrderSK int identity(1,1) primary key,
    AlternateOrderID int,
    order_status nvarchar(50),
    order_amount float,
    order_delivery_fee float,
    order_created_hour tinyint,
    order_created_minute tinyint,
    order_created_day tinyint,
    order_created_month tinyint,
    order_created_year smallint,
    order_created_time datetime2,
    order_delivered_time datetime2,
    InsertDate datetime,
    ModifiedDate datetime
)
```

A-6. SQL query to create the Payments Fact Table: FactPayments

```
SQLQuery9.sql - DE...HUB83N\nesal (52))* -> X SQLQuery8.sql - DE...HUB83N\nesal (65))* SQLQuery7.sql - DE...HUB83N\nesal (57))
drop table if exists FactPayments;
create table FactPayments(
    OrderKey int foreign key references DimFoodOrders(OrderSK),
    CenterKey smallint foreign key references DimDeliveryCenters(CenterSK),
    DeliveryKey int foreign key references DimDeliveries(DeliverySK),
    payment_id int,
    payment_amount float,
    payment_fee float,
    payment_method nvarchar(50),
    payment_date datetime2(7),
    InsertDate datetime,
    ModifiedDate datetime,
    accm_txn_create_time datetime,
    accm_txn_complete_time datetime,
    txn_process_time_hours int
)
select * from FactPayments;
```

OrderKey	CenterKey	DeliveryKey	payment_id	payment_amount	payment_fee	payment_method	payment_date	InsertDate	ModifiedDate	accm_txn_create_time	accm_txn_complete_time
----------	-----------	-------------	------------	----------------	-------------	----------------	--------------	------------	--------------	----------------------	------------------------