Sri Lanka Institute of Information Technology

ASSIGNMENT 1

Data Warehouse and Business Intelligence

2021

Submitted by:

Heenkenda C.S.K.

IT19207414

Contents

1. Data set selection and Preparation

The link to the selected source data set is mentioned below:

https://www.kaggle.com/fivethirtyeight/uber-pickups-in-new-york-city

Modifications were done accordingly to the data set derived from the above data set reflects the uber pickups in USA. Customer details involved in uber trips, drivers and vehicle information are some of the key details included in the data set.

The 2 main sources are listed below.

SQL Database

Text file – Driver Address data

Also, the below CSV files were imported to the sql source database.

State

Street

Trips

Customer

Driver

Vehicle

Vehicle category

Vehicle Driver

Description of the data set:

Source	Source Type	Object Name	Scheme	Object Type	Description
Uber_Trip_USA	SQL Database	Customer	dbo	Table	Includes all customer information
		State	dbo	Table	Includes all state details
		Street	dbo	Table	Includes the street details with the state id
		VehicleCategory	dbo	Table	Includes the vehicle category which each and every vehicle belong to with the charges per hour
		Vehicle	dbo	Table	Includes all the vehicle details
		Driver	dbo	Table	Includes all the driver information
		VehicleDriver	dbo	Table	Includes the details of which driver assigned to which vehicle
		Trip	dbo	Table	Includes the information about the trips inclding the total amount
driverAddress	Text File				Includes the driver address details

Figure 1.1: Description of data set

Table Name	Column Name	Data Type	Link Table	Link Column	Description
Customer	ID	int			Unique ID.
	fname	Nvarchar(50)			First name of the customer
	lname	Nvarchar(50)			Last name of the customer
	email	Nvarchar(50)			Email address of the customer
	phoneNo	Nvarchar(50)			Phone number of the customer
	NIC	Nvarchar(50)			National Identification Number of the Customer
State	ID	int			Unique ID.
	Name	Nvarchar(50)			Name of the state
	PostalCode	int			Postal Code of the state
Street	ID	int			Unique ID.
	Name	Nvarchar(50)			Name of the street
	stateId	int	State	ID	stateId of the street
VehicleCategory	ID	int			Unique ID.
	Category	Nvarchar(50)			Category of the vehicle
	chargerPerKm	Money			Charges per Km
Vehicle	ID	int			Unique ID.
	vehicleNo	Nvarchar(50)			vehicleNo of the vehicle
	colour	Nvarchar(50)			colour of the vehicle
	vehicleCategoryId	int	VehicleCategory	ID	vehicleCategoryId of the vehicle
Driver	ID	int			Unique ID.
	fname	Nvarchar(50)			First name of the driver
	lname	Nvarchar(50)			Last name of the driver
	email	Nvarchar(50)			Email address of the driver
	phoneNo	Nvarchar(50)			Phone number of the driver
	NIC	Nvarchar(50)			National Idenitification Number of the driver
VehicleDriver	VehicleId	int	Vehicle	ID	VehicleId of the vehicle
	DriverId	int	Driver	ID	DriverId of the driver
Trip	ID	int			Unique ID.
	edate	DateTime			date of the trip
	pickUpStreetId	int	State	ID	street id of the pickup street
	dropStreetId	int	State	ID	street id of the drop street
	stime	DateTime			time of the trip started
	etime	DateTime			time of the trip ended
	customerId	int	Customer	ID	id of the customer

	vehicleId	int	VehicleDriver	ID	id of the vehicle
	DriverId	int	VehicleDriver	ID	id of the driver
	noHours	int			number of hours went for the trip
	chargerPerKm	Money			Charges per Km
	tax	Money			tax given for the trip
	totalAmount	Money			(totalCharge + tax) fo the trip
	PaymentType	Nvarchar(50)			Payment type of the trip (cash, creditcard ot paypal)
DriverAddress	driverId	int	Driver	ID	Unique ID.
	Address 1	Nvarchar(50)			Addressline 1 of the driver
	Address 2	Nvarchar(50)			Addressline 2 of the driver

Entity Relationship Diagram

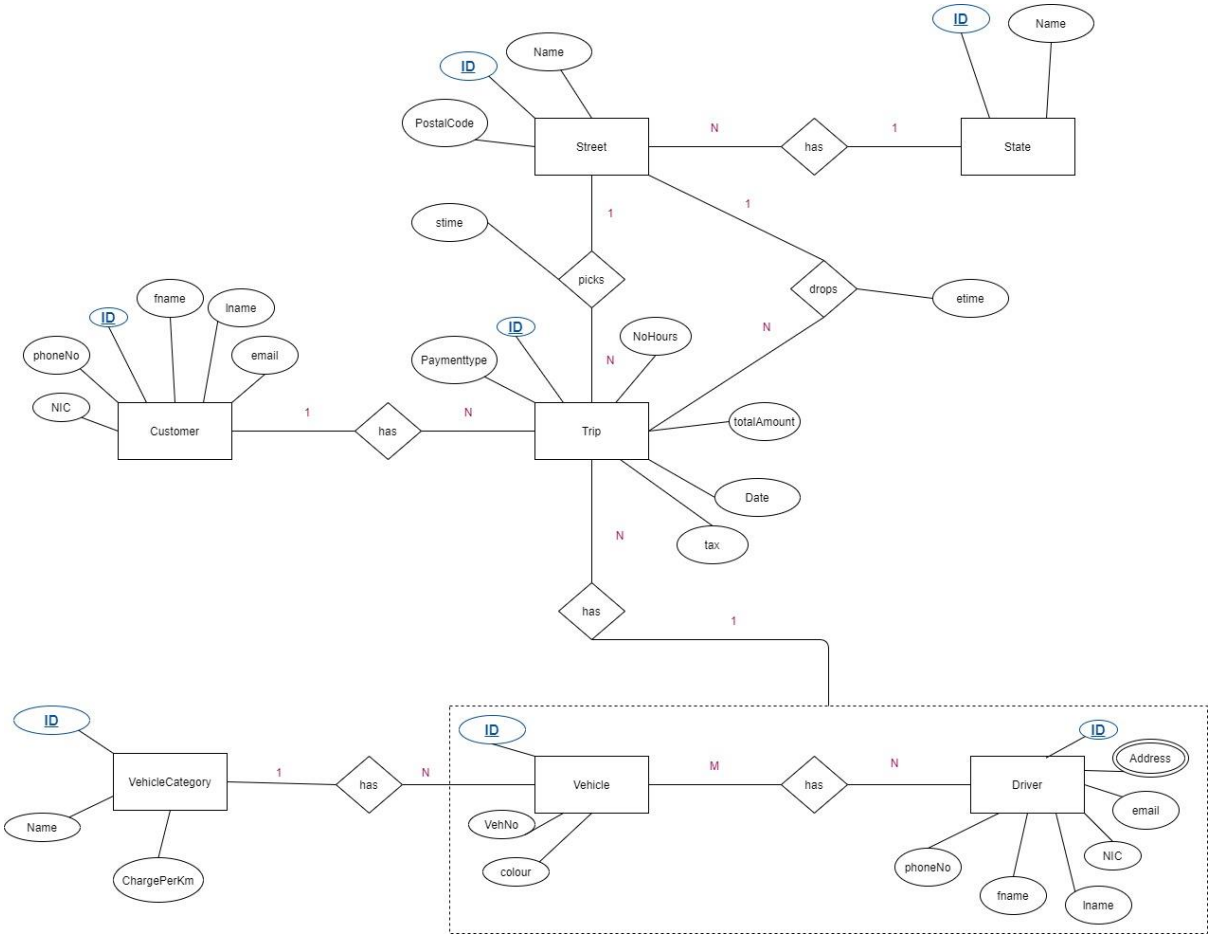


Figure 1.2: ER Diagram

This diagram shows the relationships between the entities in the data set and how the tables are created in the database.

2. Solution Architecture

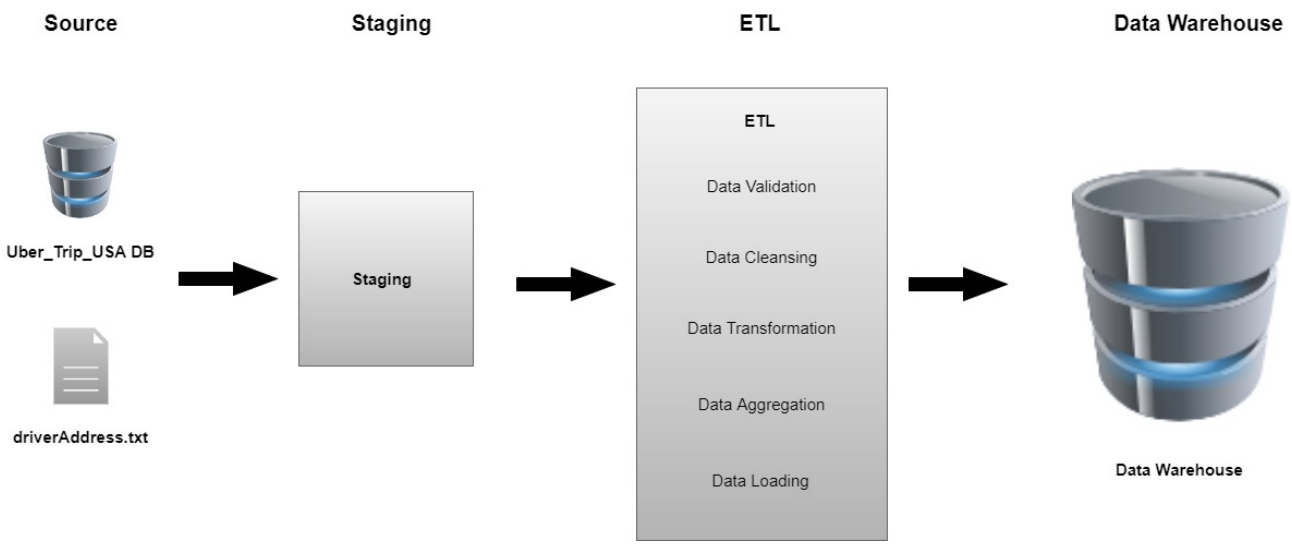


Figure 2.1: Solution Architecture

The first step in creating a stable DW architecture starts with the gathering of data from various data sources such as databases, files, or APIs, depending on the requirements and other resources.
In our scenario, the data sources are in 2 different data sources as stated before.

When data is collected through scattered systems, the subsequent stage proceeds in extracting data and loading it to a data warehouse. This process is called ETL (Extract-Transform-Load). This is done to guarantee data integration. ETL tool extracts the data from various data source systems, transforms it in the staging area and then finally, loads it into the Data Warehouse system.

Extract:

The initial step of the ETL process is extraction. In this step, data from various source systems is extracted which can be in various formats like relational databases, XML and flat files into the staging area. It is critical to extract the data from various source systems and store it into the staging area first and not directly into the data warehouse because the extracted data is in different formats and can be corrupted also. Consequently, loading it directly into the data warehouse may damage it and rollback will be subsequently more difficult. Therefore, this is one of the most important steps of ETL process.

In this scenario, I have implemented a separate database to store the Staging Tables. I have extracted the data from all these data sources to separate staging tables as shown below in Figure 2.2.

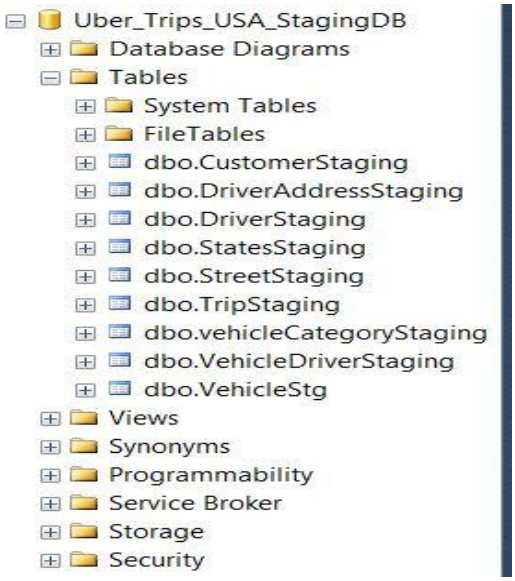


Figure 2.2: Staging Tables

Transformation:

The second step of the ETL process is transformation. The above extracted data sources are from multiple sources. Therefore, in this step, a set of rules or functions are applied on the extracted data to convert it into a single standard format.

It may involve following transformations:

- Cleaning – filling up the NULL values with some default values.
- Joining – joining multiple attributes into one.
- Derived Columns
- Data Conversion
- Sorting – sorting tuples based on some attribute (generally key-attribute)
- This improves the quality of the data.

Loading:

The third and final step of the ETL process is loading. In this step, the transformed data is finally loaded into the data warehouse. Here, data is loaded to the dimensions and fact tables. The dimension tables should first be loaded before the fact table.

Using dimensional modelling the data warehouse can now be created. Basically, there are two types of schema that can be used for the creation of a data warehouse. Here, snowflake schema is used.

3. Data Warehouse Design and Development

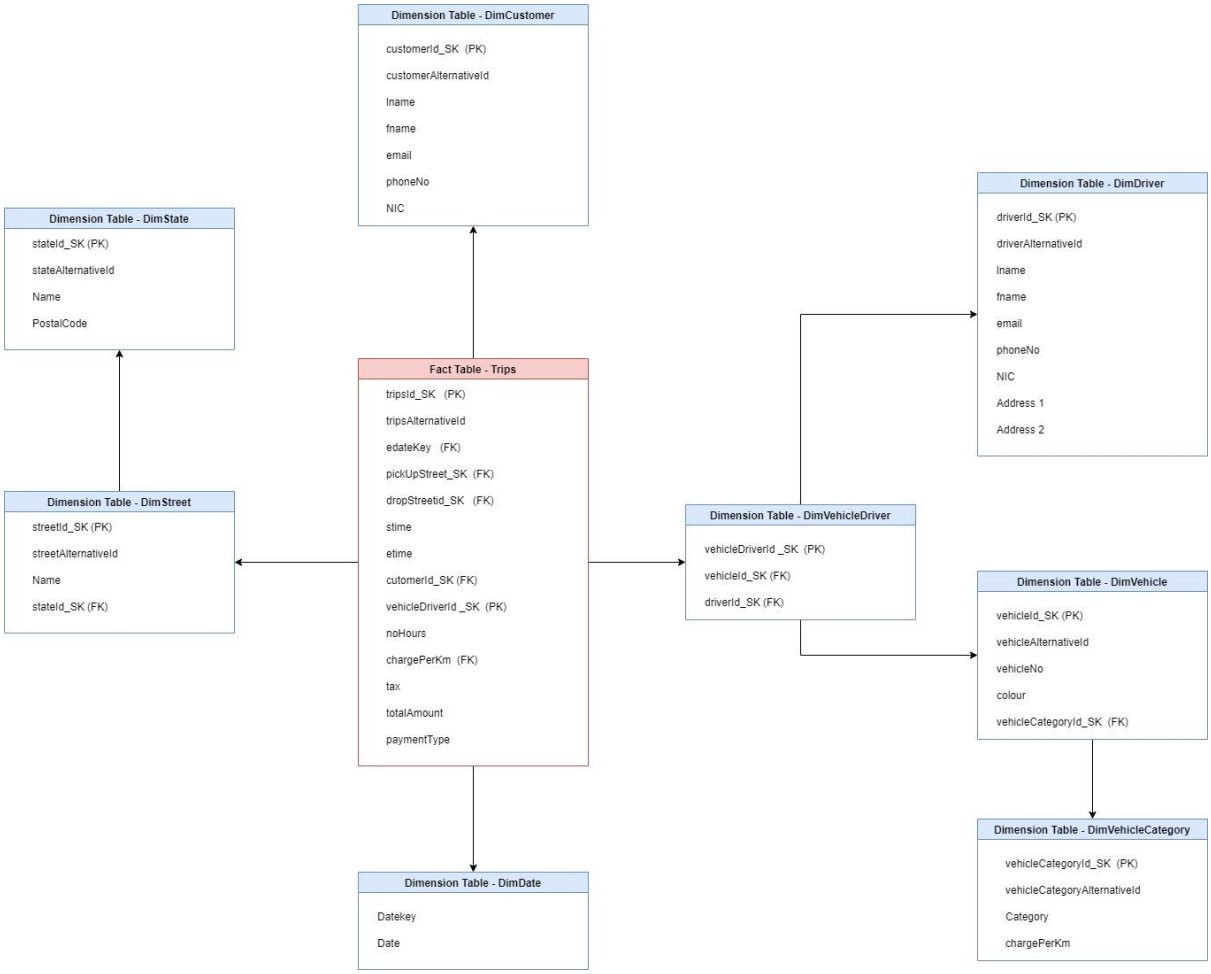


Figure 3.1: Snowflake schema

Snowflake schema is used to design the above data warehouse design. There is only a one fact table as trips and four dimensions.

Assumptions

Customer details and driver details were considered as slowly changing dimensions (Type II).

4. ETL Development

Data Extraction

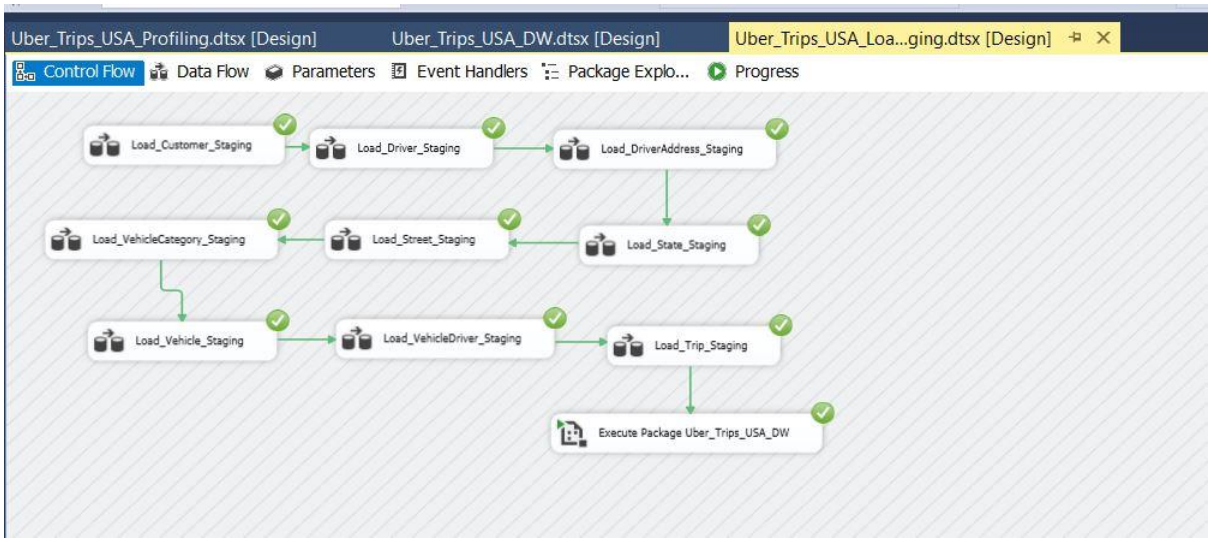


Figure 4.1

SSIS (SQL Server Integration Services) was used to extract data to the staging tables from the different data sources. SSIS is considered as the second-largest tool to perform Extraction, Transformation, and Load (ETL) operations.

Data Profiling

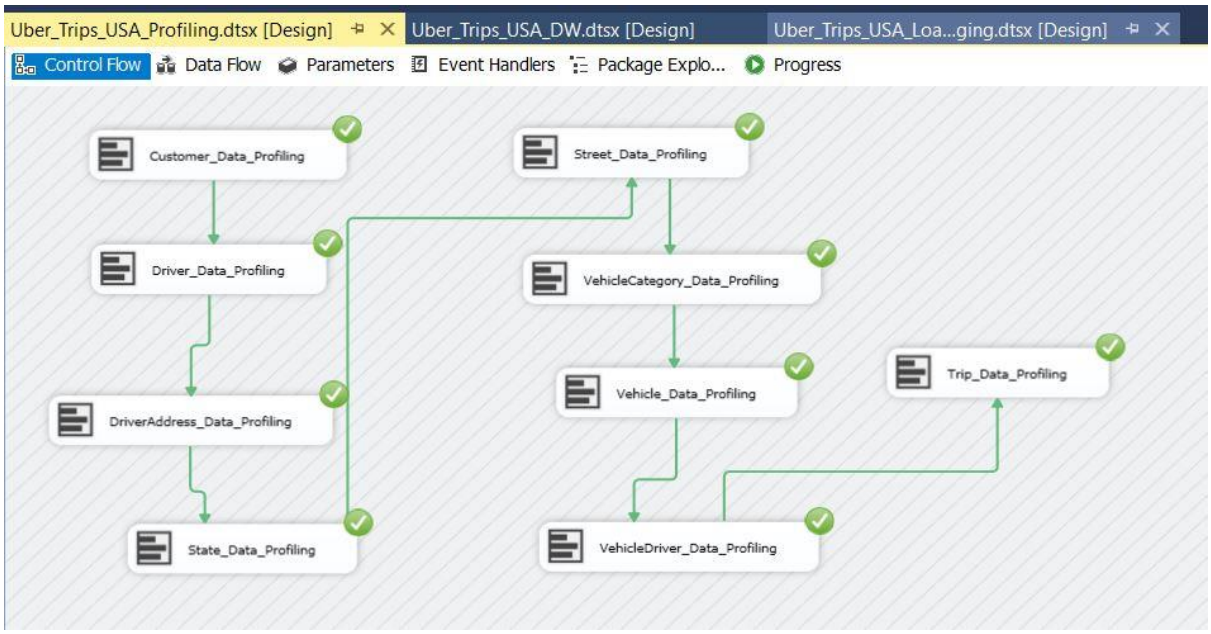


Figure 4.2

Data Profiling will provide an important amount of useful insight into the quality of your data. This knowledge is an essential component in the process of improving the health of your data.

Extract, Transform and Load State Data

First, the state data are loaded from the Uber_Trips_USA_SourceDB and StateStaging table is created in the Uber_Trips_USA_StagingDB.

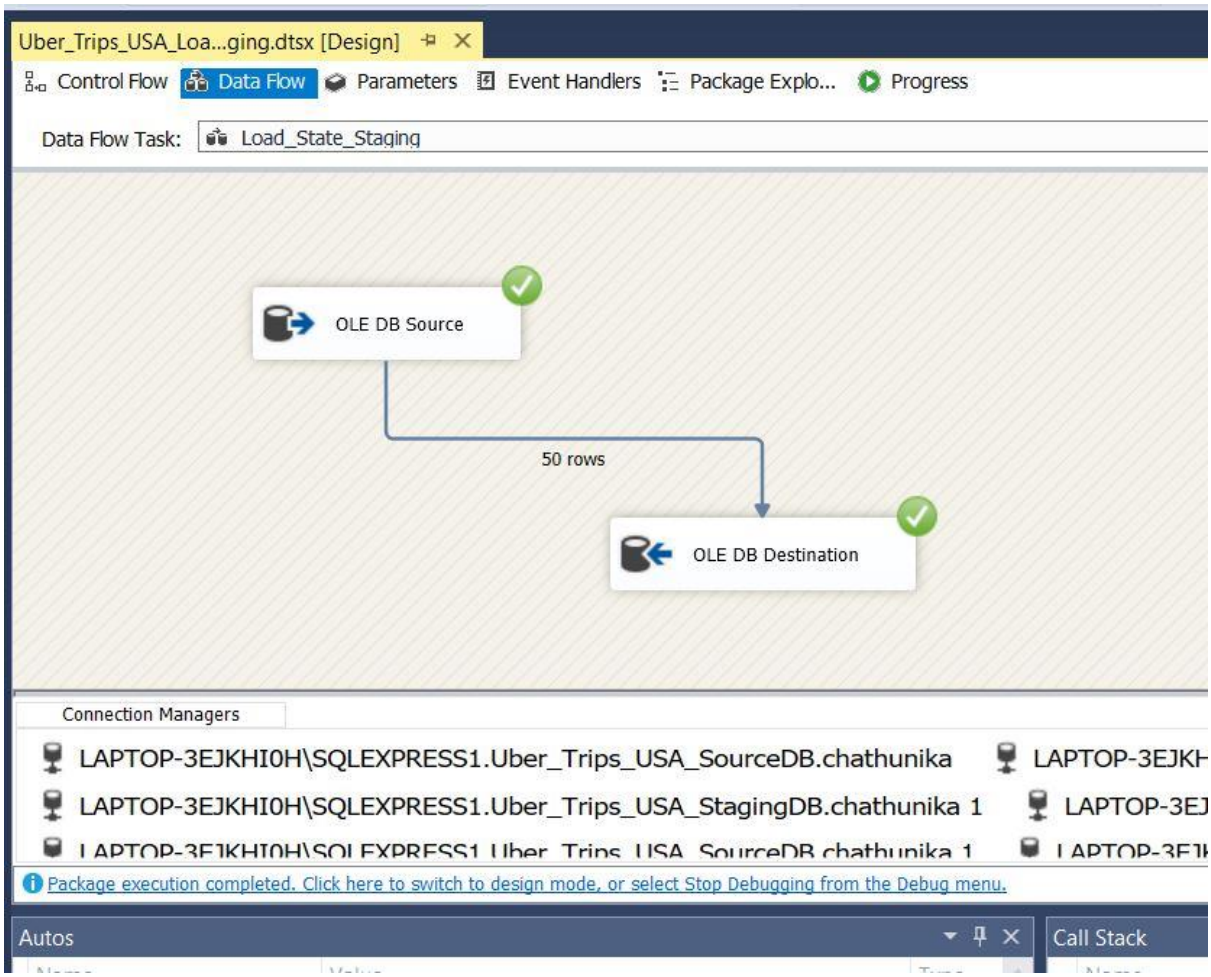


Figure 4.3

When the loading is used daily to prevent saving duplicate values following even handler is used.

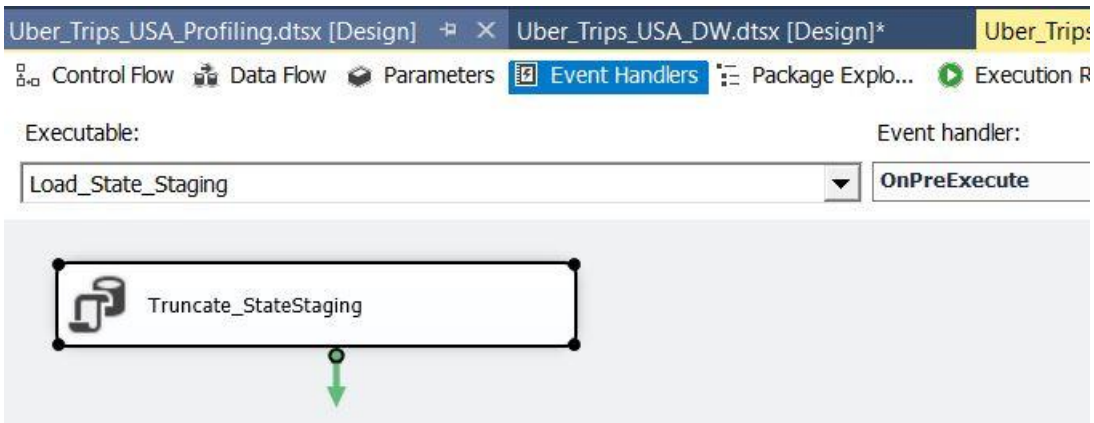


Figure 4.4

Next, using the state staging table from Uber_Trips_USA_StagingDB data were transformed and loaded into DimState table in Uber_Trips_USA_DWDB. When loading the data from the staging database to data warehouse database to check the existing values the following procedure is used.

```
CREATE PROCEDURE dbo.UpdateDimState
@stateId int,
@stateName varchar(50),
@postalCode int

AS
    BEGIN
    if not exists (select stateId_SK from dbo.DimState where stateAltervativeId = @stateId)
        BEGIN
            insert into dbo.DimState (stateAltervativeId, Name, PostalCode, insertDate, ModifiedDate)
            values (@stateId, @stateName, @postalCode, GETDATE(), GETDATE())
        END;
        if exists (select stateId_SK
            from dbo.DimState
            where stateAltervativeId = @stateId)
        BEGIN
                update dbo.DimState set Name = @stateName, PostalCode = @postalCode, ModifiedDate = GETDATE()
                where stateAltervativeId = @stateId
        END;
    END;
```

Figure 4.5

SqlCommand	exec dbo.UpdateDimState ?, ?, ?

Figure 4.6

Figure 4.7

Extract, Transform and Load Street Data

First, the street data are loaded from the Uber_Trips_USA_SourceDB and StreetStaging table is created in the Uber_Trips_USA_StagingDB.

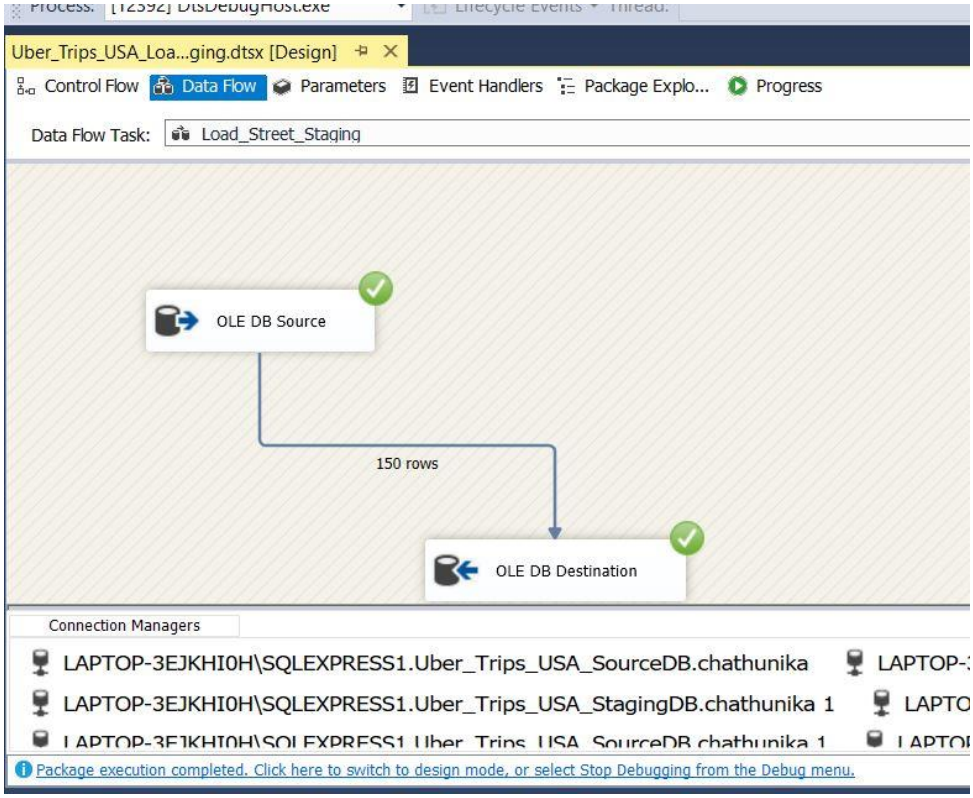


Figure 4.8

When the loading is used daily to prevent saving duplicate values following even handler is used.

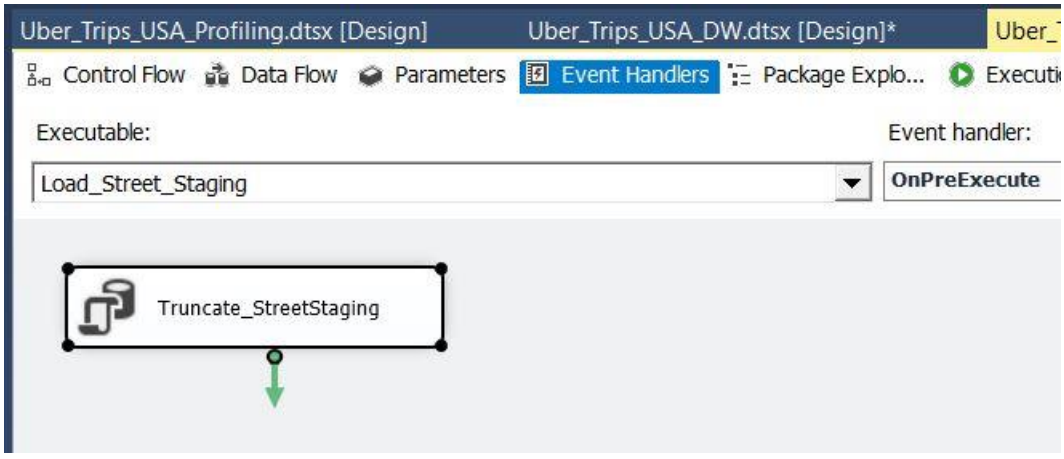


Figure 4.9

Next, using the street staging table from Uber_Trips_USA_StagingDB data were transformed and loaded into DimStreet table in Uber_Trips_USA_DWDB. When loading the data from the staging database to data warehouse database to check the existing values the following procedure is used.

```sql
CREATE PROCEDURE dbo.UpdateStreet
@streetId int,
@streetName varchar(50),
@stateId int
AS
BEGIN
    if not exists (select streetId_SK
    from dbo.DimStreet
    where streetAlternativeId = @streetId)

    BEGIN
        insert into dbo.DimStreet
        (streetAlternativeId, Name, stateId_SK, insertDate, ModifiedDate)
        values
        (@streetId, @streetName, @stateId, GETDATE(), GETDATE())
    END;
    if exists (select streetId_SK
    from dbo.DimStreet
    where streetAlternativeId = @streetId)

    BEGIN
        update dbo.DimStreet
        set Name = @streetName,
        stateId_SK = @stateId,
        ModifiedDate = GETDATE()
        where streetAlternativeId = @streetId
    END;
END;
```

Figure 4.10

SqlCommand	exec dbo.UpdateStreet ?,?,?

Figure 4.11

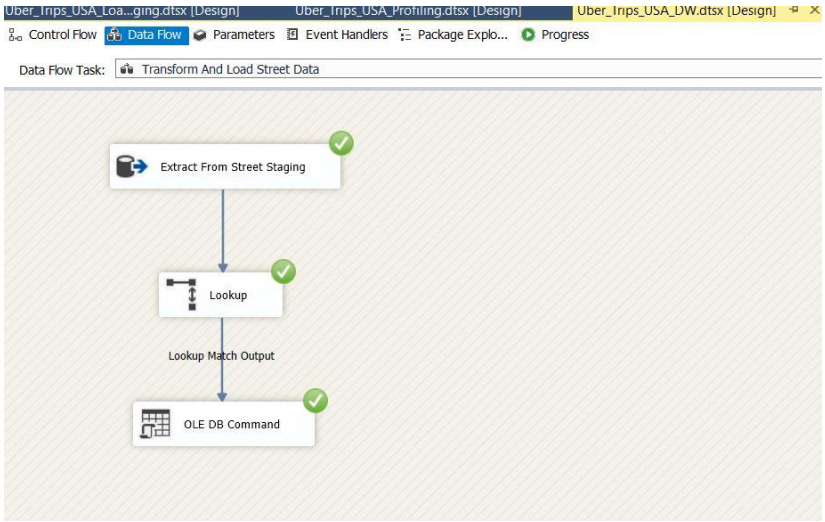


Figure 4.12

In the street table state id is inserted as the foreign key. Therefore, lookup is used to get the foreign key from DimState table.

Lookup Column	Lookup Operation	Output Alias
stateId_SK	<add as new column>	stateId_SK

Figure 4.13

Extract, Transform and Load Vehicle Category Data

The data is loaded to staging database like the street and state tables as above.

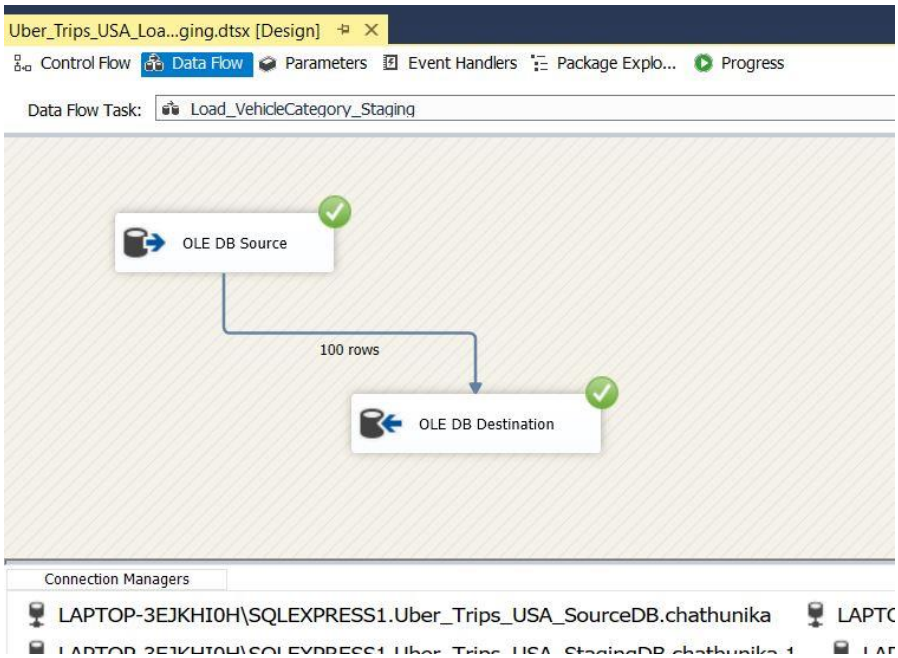


Figure 4.15

When the loading is used daily to prevent saving duplicate values following even handler is used.

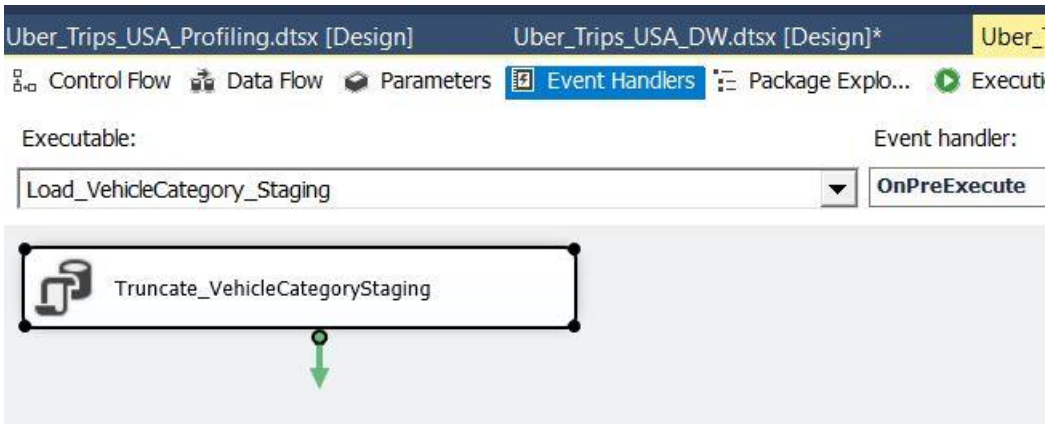


Figure 4.16

Then using the vehicle category staging table all the vehicle category data is extracted and loaded into vehicle category dimension table (DimVehicleCategory) in data warehouse database. When loading the data from the staging database to data warehouse database to check the existing values the following procedure is used.

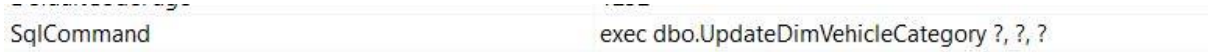


Figure 4.17

```sql
CREATE PROCEDURE dbo.UpdateDimVehicleCategory
@vehicleCategoryId int,
@CategoryName varchar(50),
@chargePerKm money

AS
    BEGIN
    if not exists (select vehicleCategoryId_SK from dbo.DimVehicleCategory where vehicleCategoryAlternativeId = @vehicleCategoryId)
        BEGIN
            insert into dbo.DimVehicleCategory (vehicleCategoryAlternativeId, Category, chargePerKm, insertDate, ModifiedDate)
            values (@vehicleCategoryId, @CategoryName, @chargePerKm, GETDATE(), GETDATE())
        END;
        if exists (select vehicleCategoryId_SK
            from dbo.DimVehicleCategory
            where vehicleCategoryAlternativeId = @vehicleCategoryId)
        BEGIN
                update dbo.DimVehicleCategory set Category = @CategoryName, chargePerKm = @chargePerKm, ModifiedDate = GETDATE()
                where vehicleCategoryAlternativeId = @vehicleCategoryId
        END;
    END;
```

Figure 4.18

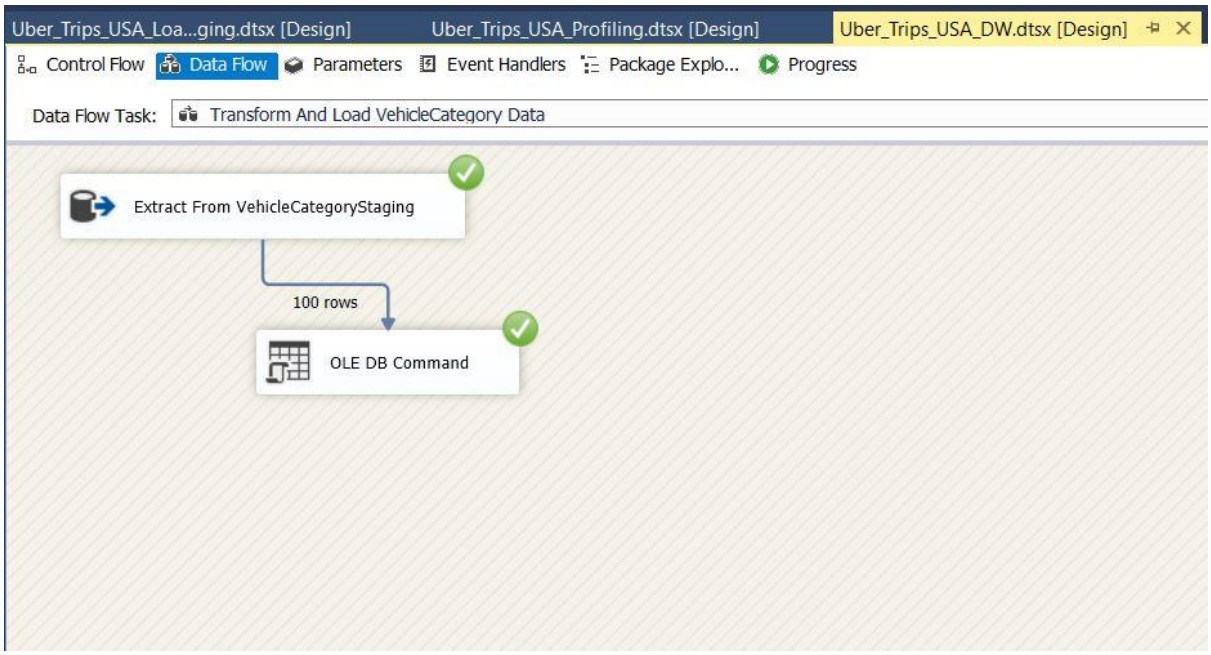


Figure 4.19

Extract, Transform and Load Vehicle Data

All the vehicle data is loaded to the staging database from source database as below.

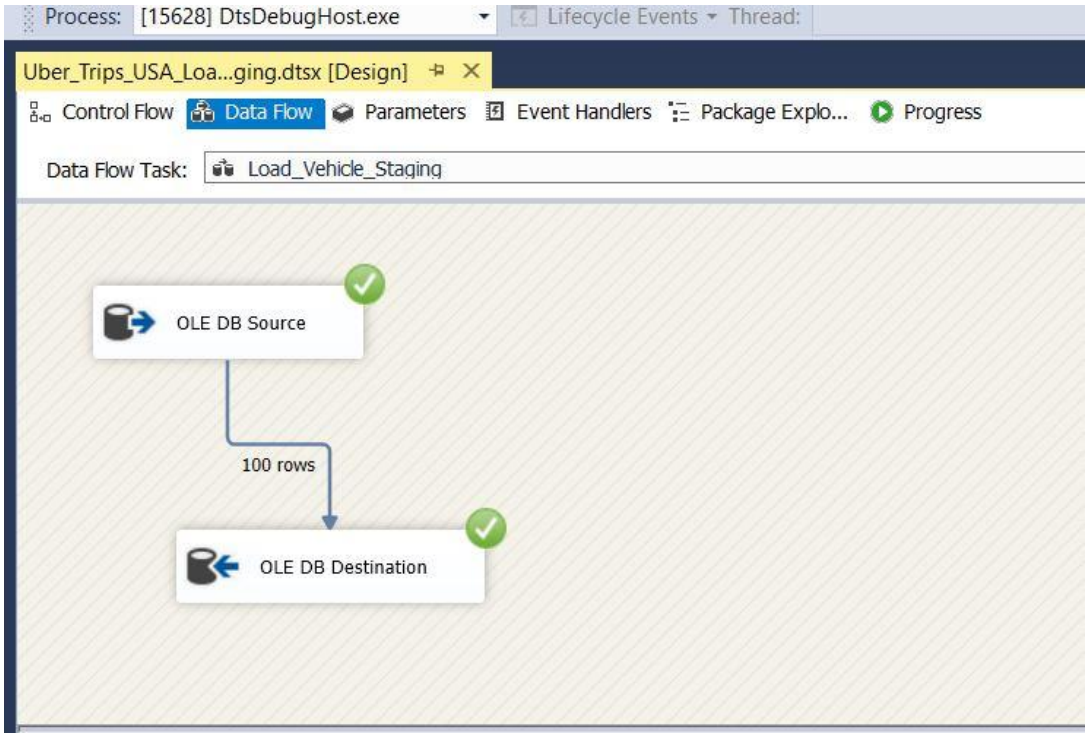


Figure 4.20

To avoid loading duplicate values following when using daily basis following event handler is used.

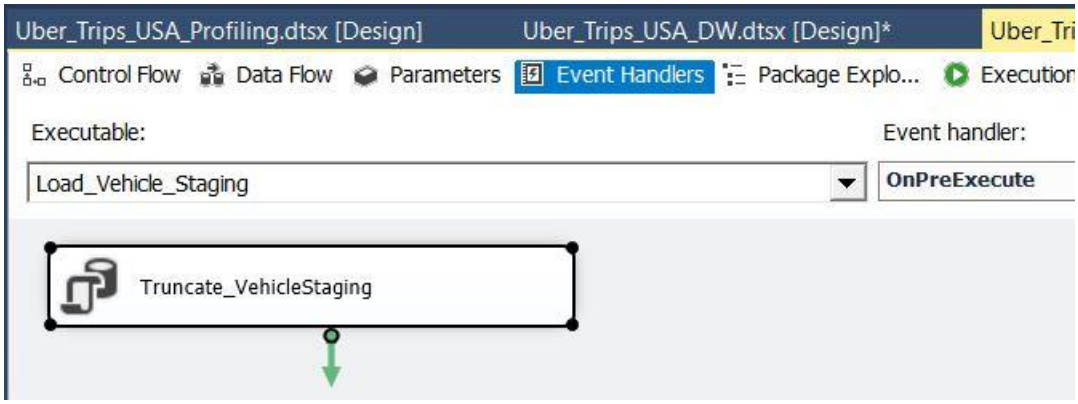


Figure 4.21

Vehicle data has the foreign key vehicle category id from the vehicle category table. So, when transforming data and adding data to the vehicle dimension table in data warehouse table the following lookup is used.

Lookup Column	Lookup Operation	Output Alias
vehicleCategoryId_SK	<add as new column>	vehicleCategoryId_SK

Figure 4.22

To check the existing data in the dimension table a procedure is used and implemented.

```
CREATE PROCEDURE dbo.UpdateDimVehicle
@vehicleId int,
@VehicleCategoryKey int,
@vehicleNo varchar(50),
@colourName varchar(50)
AS
BEGIN
    if not exists (select vehicleId_SK
    from dbo.DimVehicle
    where vehicleAlternativeId = @vehicleId)

    BEGIN
        insert into dbo.DimVehicle
        (vehicleAlternativeId, vehicleNo, colour, vehicleCategoryId_SK, insertDate, ModifiedDate)
        values
        (@vehicleId, @vehicleNo, @colourName, @VehicleCategoryKey, GETDATE(), GETDATE())
    END;

    if exists (select vehicleId_SK
    from dbo.DimVehicle
    where vehicleAlternativeId = @vehicleId)

    BEGIN
        update dbo.DimVehicle
        set vehicleNo = @vehicleNo,
        colour = @colourName,
        vehicleCategoryId_SK = @VehicleCategoryKey,
        ModifiedDate = GETDATE()
        where vehicleAlternativeId = @vehicleId
    END;
END;
```

Figure 4.23

SqlCommand	exec dbo.UpdateDimVehicle ?,?,?,?

Figure 4.24

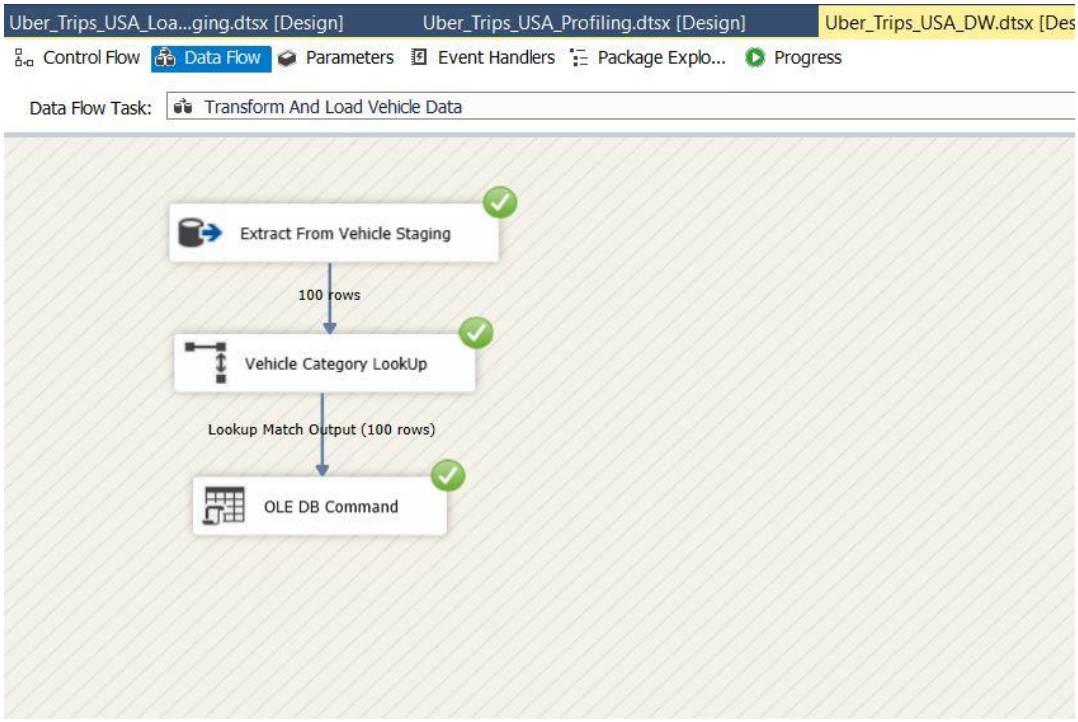


Figure 4.25

Extract, Transform and Load Driver and Driver Address Data

For this step data are loaded from 2 different sources.

- Driver from source database.
- Driver Address from the text file.

Firstly, the driver data is loaded from the source database to staging database by creating the driver staging table.

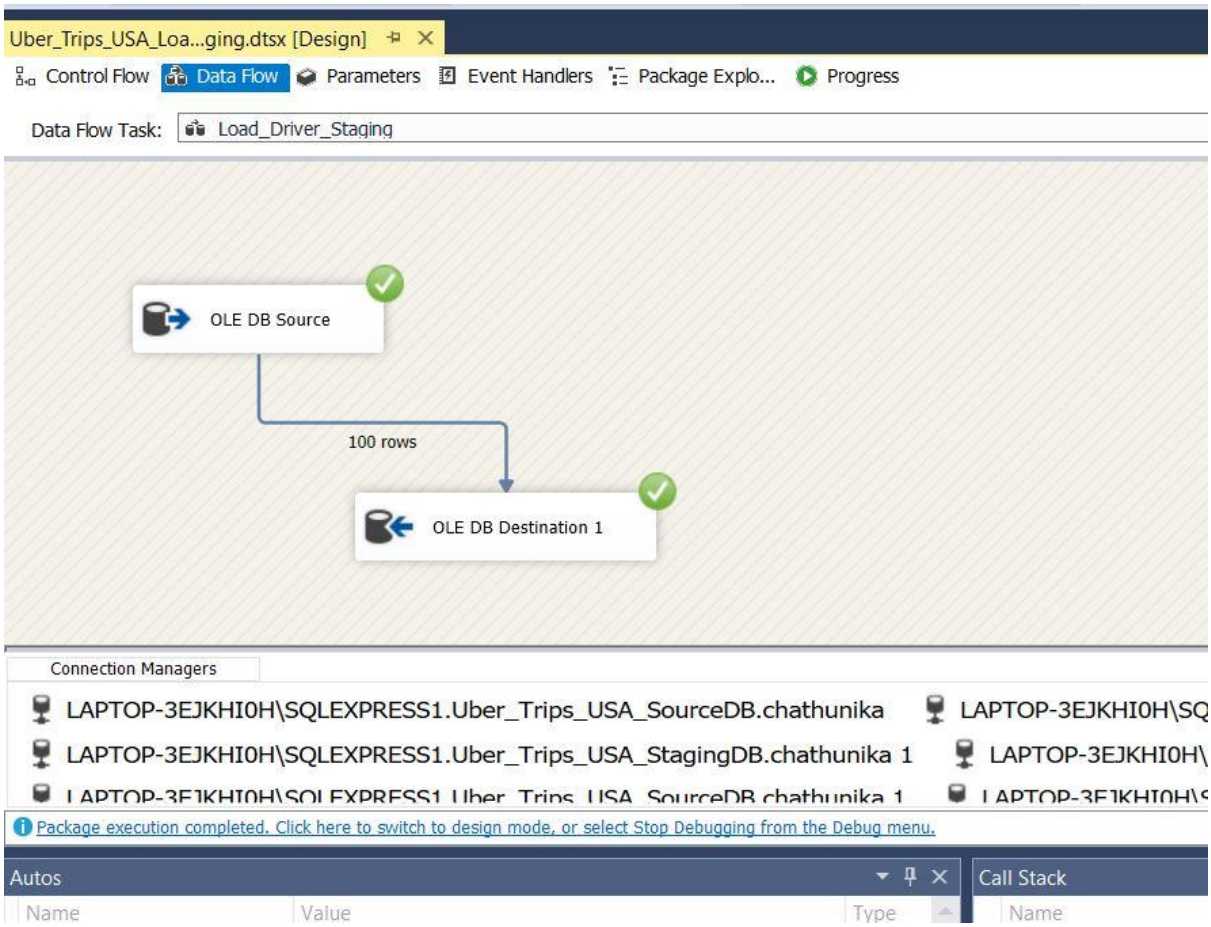


Figure 4.26

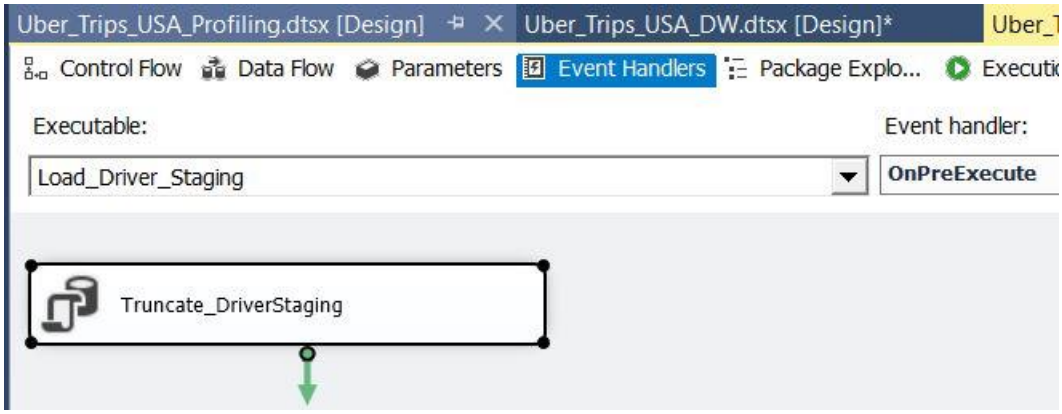


Figure 4.27

Then the driver address is loaded from the text file to staging database by creating the driver address staging table.

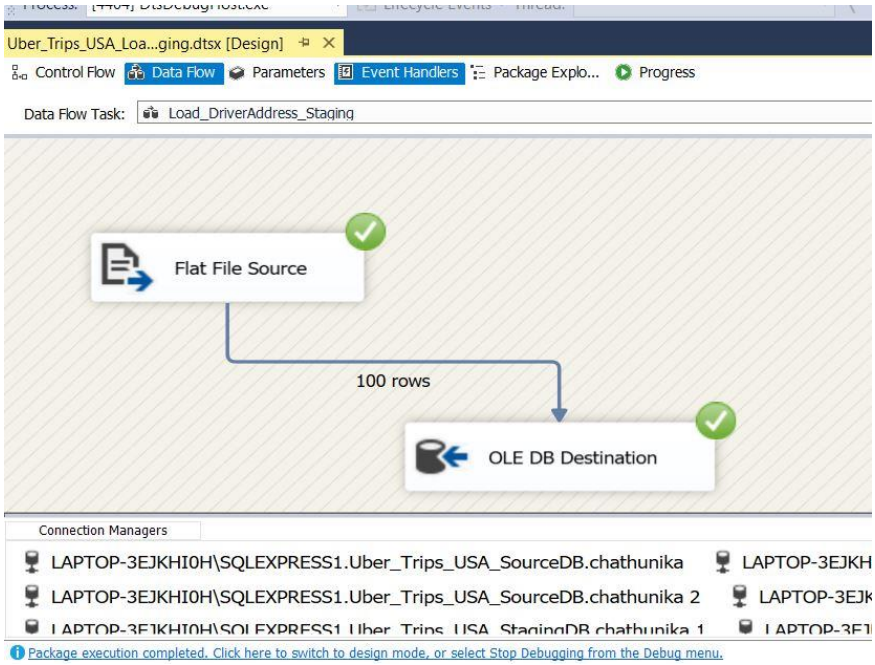


Figure 4.28

Figure 4.29

When creating a dimension, the combination of above 2 staging tables were used and transformed into the driver dimension table in data warehouse database.

For the above scenario following steps were used.

- First sorts were used for both driver and driver address.

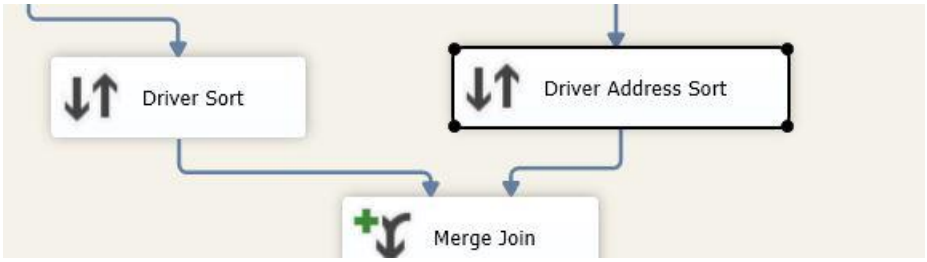
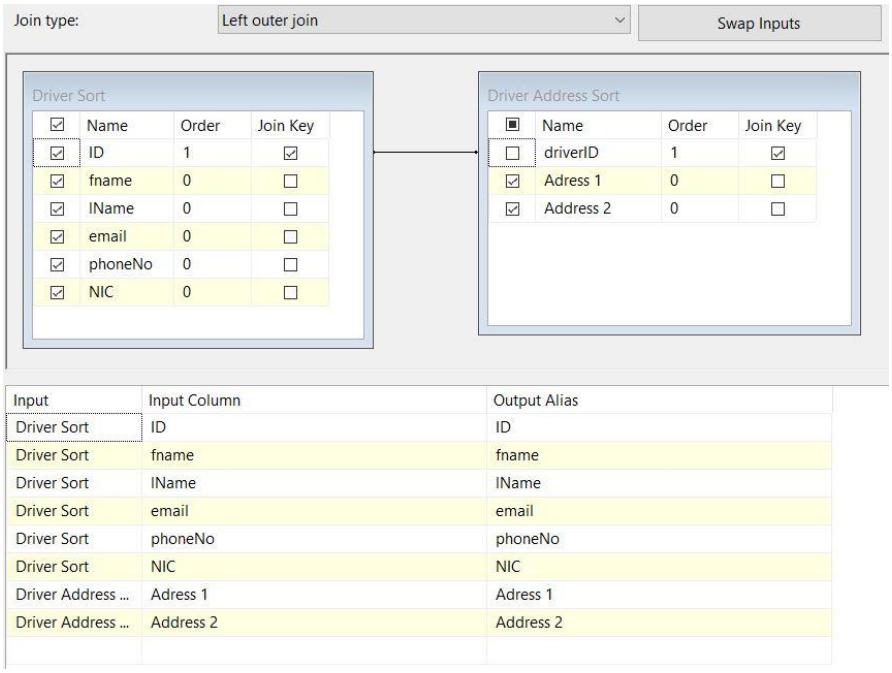


Figure 4.30

- The using the merge join all the necessary fields in both tables were merged.



Input	Input Column	Output Alias
Driver Sort	ID	ID
Driver Sort	fname	fname
Driver Sort	lName	lName
Driver Sort	email	email
Driver Sort	phoneNo	phoneNo
Driver Sort	NIC	NIC
Driver Address ...	Adress 1	Adress 1
Driver Address ...	Address 2	Address 2

Figure 4.31

- Then the null valued columns were implemented.

Derived Column Name	Derived Column	Expression	Data Type	Le
InsertDate	<add as new column>	GETDATE()	database timestamp [D...	
ModifiesDate	<add as new column>	GETDATE()	database timestamp [D...	
Address 2	Replace 'Address 2'	REPLACENULL([Address 2],"N")	string [DT_STR]	5(

Figure 4.32

- As Driver dimension is a slowly changing dimension slowly changing dimension is used and implemented columns as below.

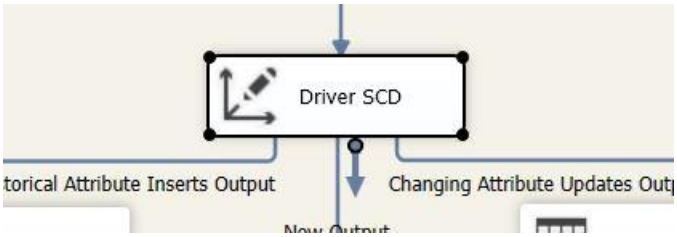


Figure 4.33

Dimension Columns	Change Type
AddressLine1	Historical a...
AddressLine2	Historical a...
email	Changing a...
fname	Fixed attrib...
lname	Fixed attrib...
NIC	Fixed attrib...
phoneNo	Changing a...

Figure 4.34

- Then the sdate of the driver dimension table is derived.

Derived Column Name	Derived Column	Expression	Data Type	Le
sdate	<add as new column>	(DT_DBTIMESTAMP)(@[System::StartTime])	database timestamp [D...	

Figure 4.35

The complete data flow of the driver dimension is as below.

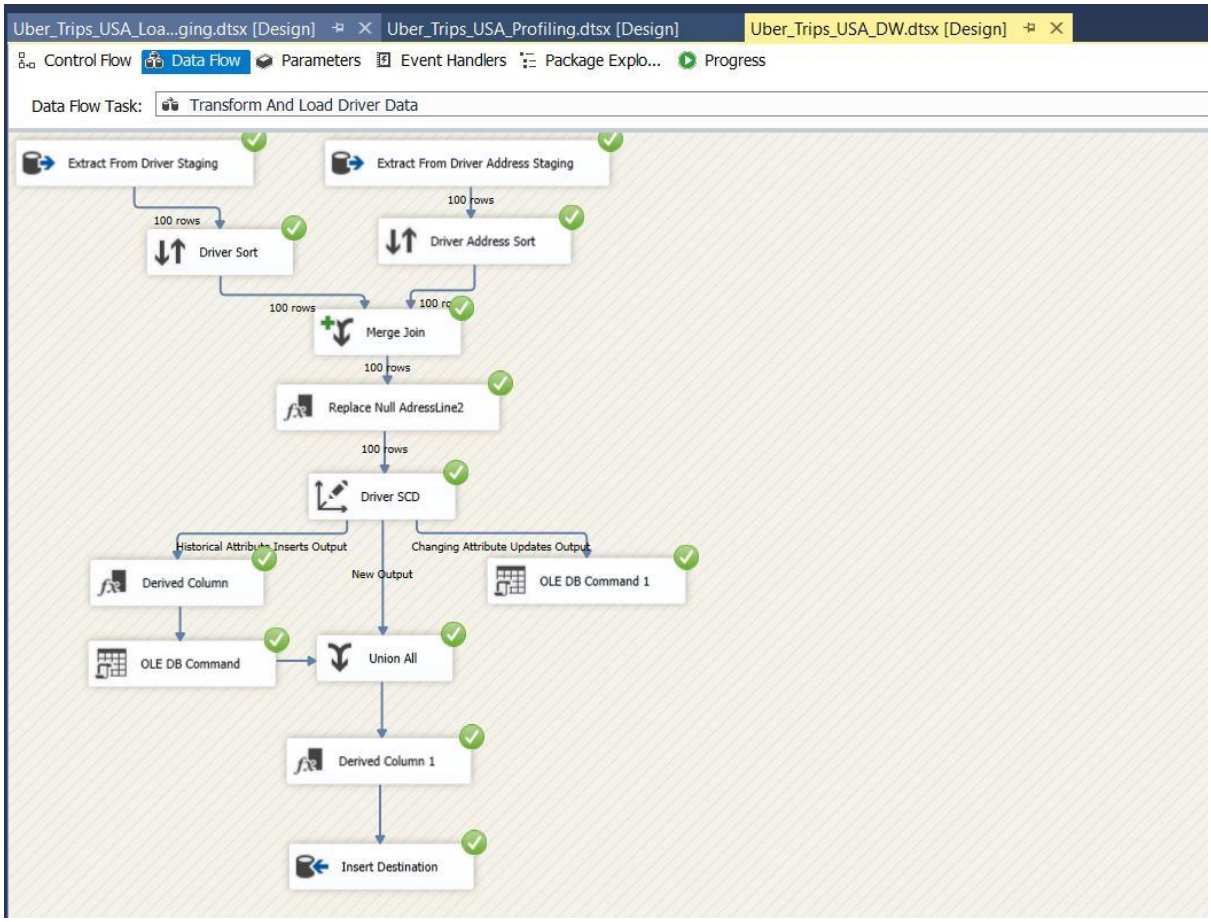


Figure 4.36

Extract, Transform and Load Driver Vehicle Data

Here also the first step was to load diver vehicle data from the source database to staging database by creating the vehicle driver staging table. To avoid from loading duplicate values an event handler is used.

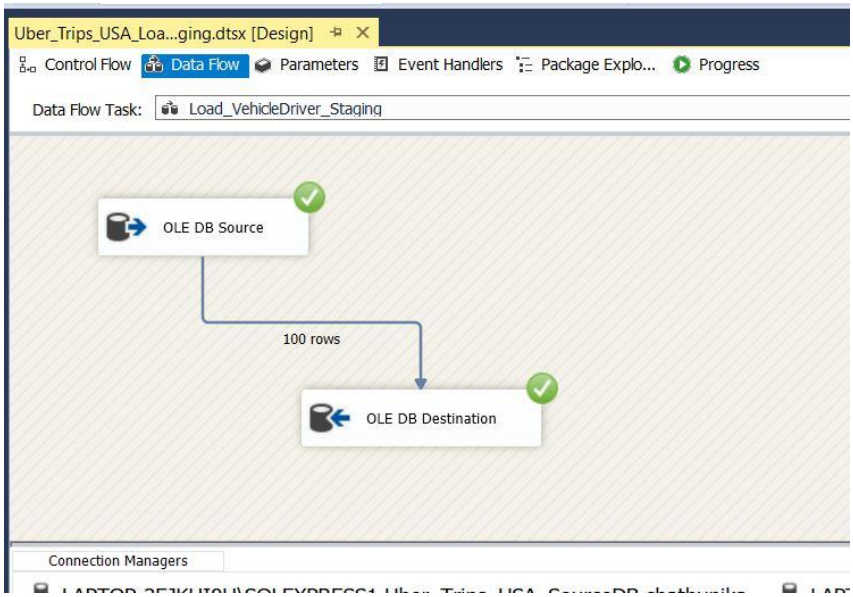


Figure 4.37

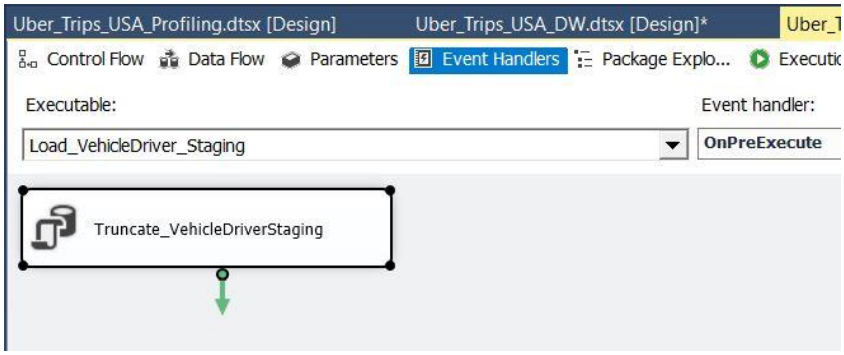


Figure 4.38

In the driver vehicle table both ids of the vehicle and driver is loaded from the vehicle and driver dimension tables respectively. Therefore, 2 lookups were used to load the ids into driver vehicle dimension and to transform data in the data warehouse database.

Lookup Column	Lookup Operation	Output Alias
vehicleId_SK	<add as new column>	vehicleId_SK

Figure 4.39

Lookup Column	Lookup Operation	Output Alias
driverId_SK	<add as new column>	driverId_SK

Figure 4.40

To check existing data in the driver vehicle dimension the following procedure is used.

```
CREATE PROCEDURE dbo.UpdateDriverVehicle
@vehicleId int,
@driverId int
AS
BEGIN
    if not exists (select driverVehicleId_SK
    from dbo.DimDriverVehicle
    where vehicleId_SK = @vehicleId)

    BEGIN
        insert into dbo.DimDriverVehicle
        (vehicleId_SK, driverId_SK, insertDate, ModifiedDate)
        values
        (@vehicleId, @driverId, GETDATE(), GETDATE())
    END;

    if exists (select driverVehicleId_SK
    from dbo.DimDriverVehicle
    where vehicleId_SK = @vehicleId)

    BEGIN
        update dbo.DimDriverVehicle
        set
        driverId_SK = @driverId,
        ModifiedDate = GETDATE()
        where vehicleId_SK = @vehicleId
    END;
END;
```

Figure 4.41

SqlCommand	exec dbo.UpdateDriverVehicle ?,?

Figure 4.42

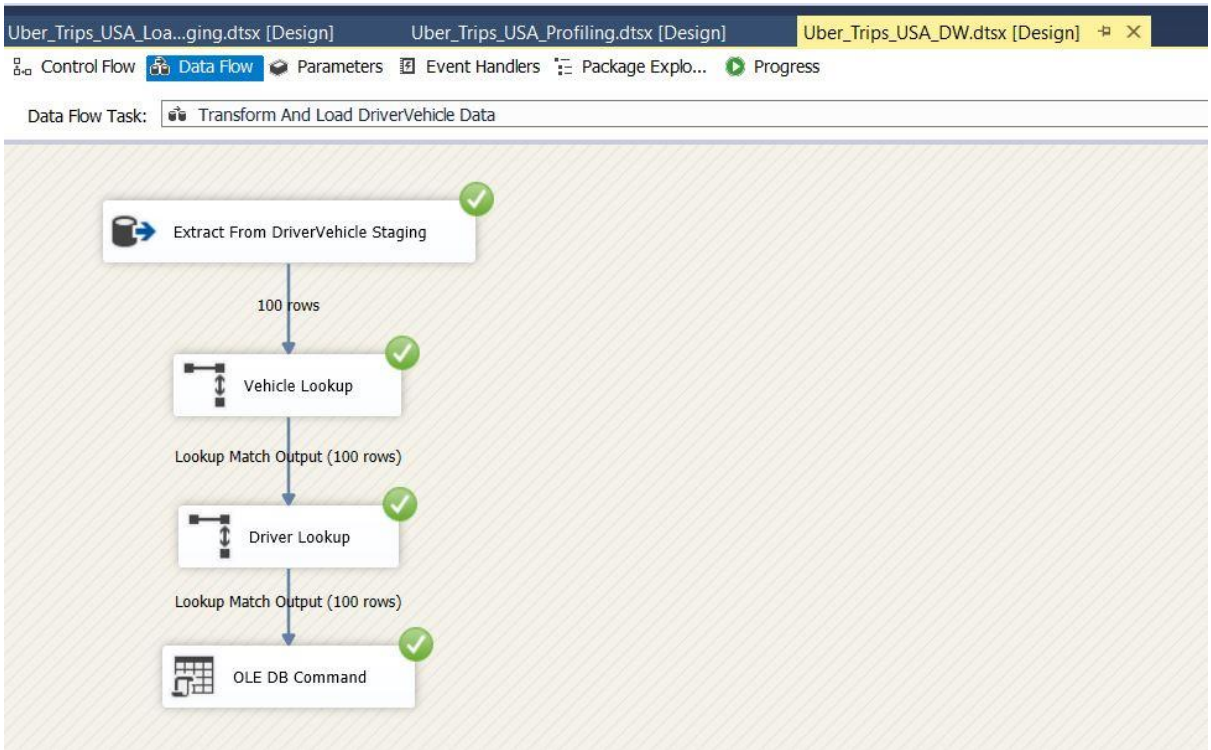


Figure 4.43

Extract, Transform and Load Customer Data

Initially, customer data is loaded from the source database to staging database by creating customer staging table. To avoid loading duplicate values an event handler is used.

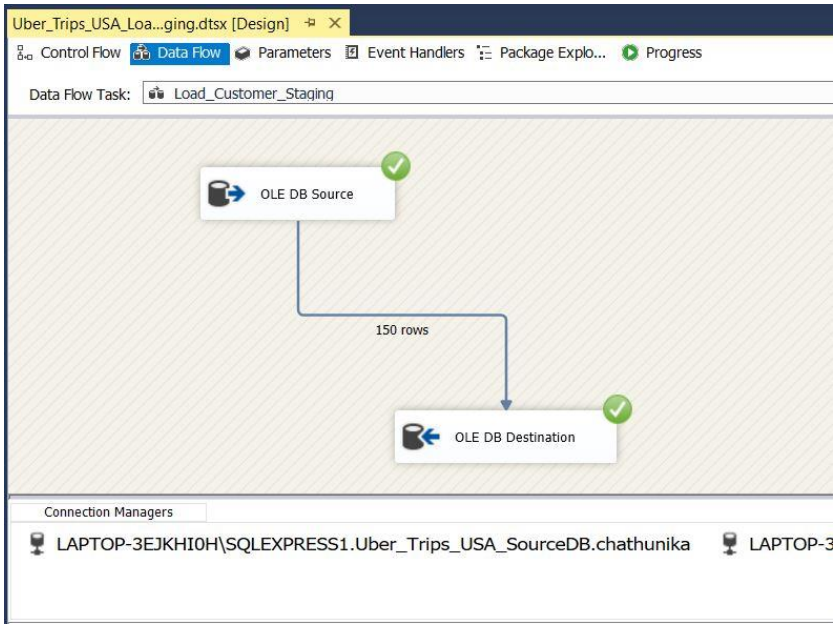


Figure 4.44

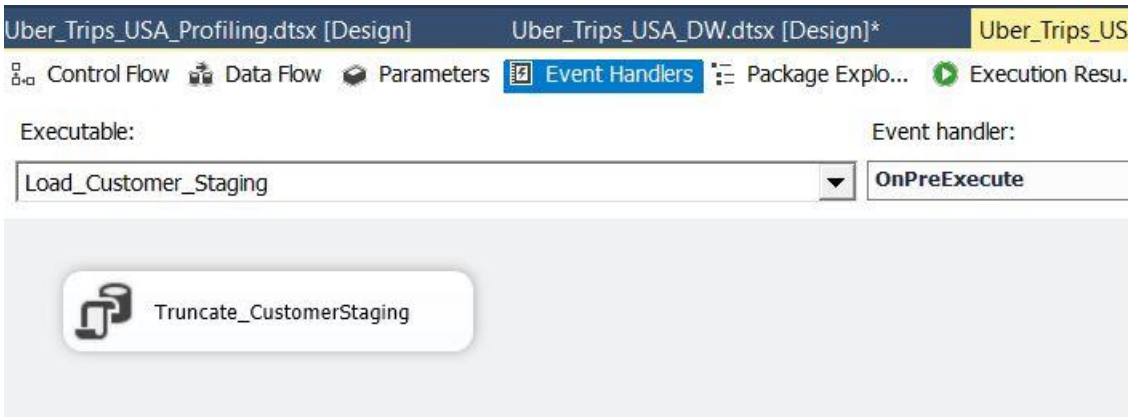


Figure 4.45

Customer is a slowly changing dimension in this dataset. Therefore, slowly changing dimension tool is used when developing the data flow.

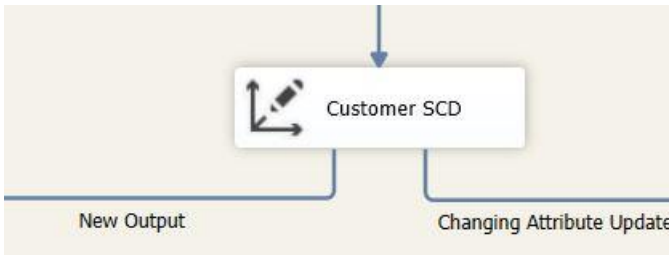


Figure 4.46

Dimension Columns	Change Type
email	Changing a...
fname	Fixed attrib...
lname	Fixed attrib...
NIC	Fixed attrib...
phoneNo	Changing a...

Figure 4.47

The full data flow of the customer dimension is as shown below.

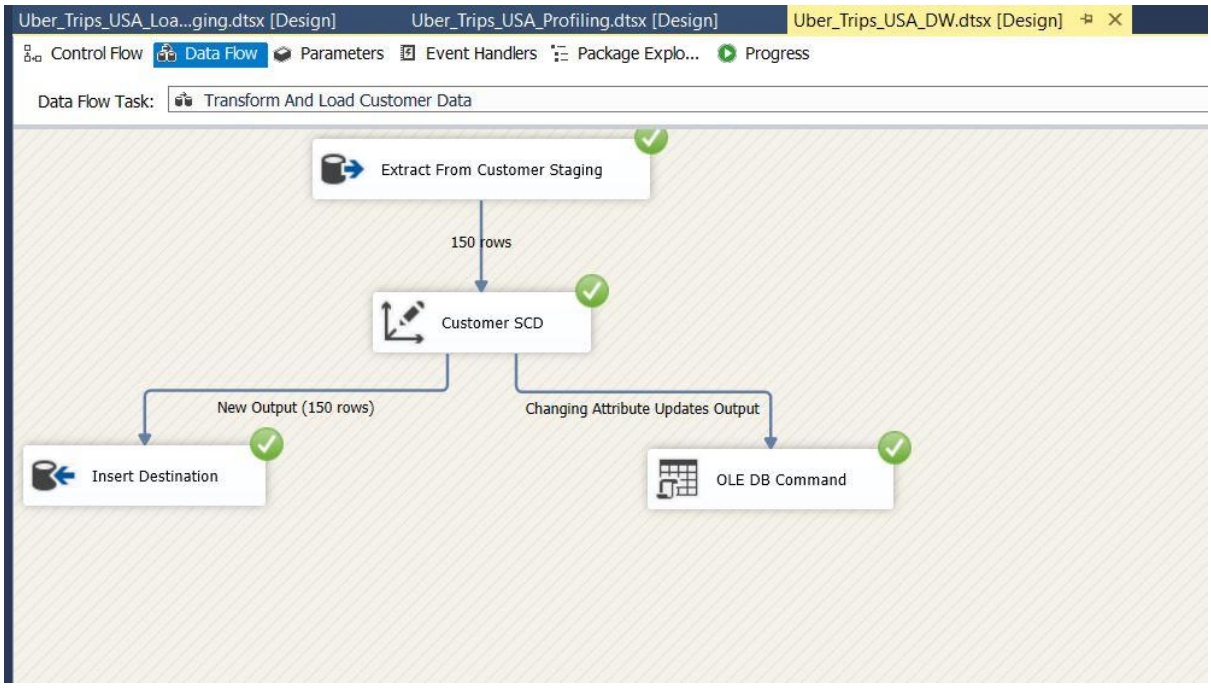


Figure 4.48

Creating Date Dimension

To create the date dimension following procedure is used and executed in the data warehouse database.

```
BEGIN TRY
    DROP TABLE [dbo].[DimDate]
END TRY

BEGIN CATCH
    /*No Action*/
END CATCH

/***********************************************************************************/

CREATE TABLE    [dbo].[DimDate]
    (   [DateKey] INT primary key,
        [Date] DATETIME,
        [FullDateUK] CHAR(10), -- Date in dd-MM-yyyy format
        [FullDateUSA] CHAR(10),-- Date in MM-dd-yyyy format
        [DayOfMonth] VARCHAR(2), -- Field will hold day number of Month
        [DaySuffix] VARCHAR(4), -- Apply suffix as 1st, 2nd ,3rd etc
        [DayName] VARCHAR(9), -- Contains name of the day, Sunday, Monday
        [DayOfWeekUSA] CHAR(1),-- First Day Sunday=1 and Saturday=7
        [DayOfWeekUK] CHAR(1),-- First Day Monday=1 and Sunday=7
        [DayOfWeekInMonth] VARCHAR(2), --1st Monday or 2nd Monday in Month
        [DayOfWeekInYear] VARCHAR(2),
        [DayOfQuarter] VARCHAR(3),
        [DayOfYear] VARCHAR(3),
        [WeekOfMonth] VARCHAR(1),-- Week Number of Month
        [WeekOfQuarter] VARCHAR(2), --Week Number of the Quarter
        [WeekOfYear] VARCHAR(2),--Week Number of the Year
        [Month] VARCHAR(2), --Number of the Month 1 to 12
        [MonthName] VARCHAR(9),--January, February etc
        [MonthOfQuarter] VARCHAR(2),-- Month Number belongs to Quarter
        [Quarter] CHAR(1),
        [QuarterName] VARCHAR(9),--First,Second..
        [Year] CHAR(4),-- Year value of Date stored in Row
        [YearName] CHAR(7), --CY 2012,CY 2013
        [MonthYear] CHAR(10), --Jan-2013,Feb-2013
        [MMYYYY] CHAR(6),
        [FirstDayOfMonth] DATE,
        [LastDayOfMonth] DATE,
        [FirstDayOfQuarter] DATE,
        [LastDayOfQuarter] DATE,
        [FirstDayOfYear] DATE,
        [LastDayOfYear] DATE,
        [IsHolidaySL] BIT,-- Flag 1=National Holiday, 0-No National Holiday
        [IsWeekday] BIT,-- 0=Week End ,1=Week Day
        [HolidaySL] VARCHAR(50),--Name of Holiday in US
        [isCurrentDay] int, -- Current day=1 else = 0
        [isDataAvailable] int, -- data available for the day = 1, no data available for the day = 0
        [isLatestDataAvailable] int
    )
GO
```

```
DECLARE @StartDate DATETIME = '01/01/1990' --Starting value of Date Range
DECLARE @EndDate DATETIME = '01/01/2099' --End Value of Date Range

--Temporary Variables To Hold the Values During Processing of Each Date of Year
DECLARE
    @DayOfWeekInMonth INT,
    @DayOfWeekInYear INT,
    @DayOfQuarter INT,
    @WeekOfMonth INT,
    @CurrentYear INT,
    @CurrentMonth INT,
    @CurrentQuarter INT

/*Table Data type to store the day of week count for the month and year*/
DECLARE @DayOfWeek TABLE (DOW INT, MonthCount INT, QuarterCount INT, YearCount INT)

INSERT INTO @DayOfWeek VALUES (1, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (2, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (3, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (4, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (5, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (6, 0, 0, 0)
INSERT INTO @DayOfWeek VALUES (7, 0, 0, 0)

--Extract and assign various parts of Values from Current Date to Variable

DECLARE @CurrentDate AS DATETIME = @StartDate
SET @CurrentMonth = DATEPART(MM, @CurrentDate)
SET @CurrentYear = DATEPART(YY, @CurrentDate)
SET @CurrentQuarter = DATEPART(QQ, @CurrentDate)
```

```
WHILE @CurrentDate < @EndDate
BEGIN

/*Begin day of week logic*/

        /*Check for Change in Month of the Current date if Month changed then
          Change variable value*/
    IF @CurrentMonth != DATEPART(MM, @CurrentDate)
    BEGIN
        UPDATE @DayOfWeek
        SET MonthCount = 0
        SET @CurrentMonth = DATEPART(MM, @CurrentDate)
    END

        /* Check for Change in Quarter of the Current date if Quarter changed then change
         Variable value*/

    IF @CurrentQuarter != DATEPART(QQ, @CurrentDate)
    BEGIN
        UPDATE @DayOfWeek
        SET QuarterCount = 0
        SET @CurrentQuarter = DATEPART(QQ, @CurrentDate)
    END

        /* Check for Change in Year of the Current date if Year changed then change
         Variable value*/


    IF @CurrentYear != DATEPART(YY, @CurrentDate)
    BEGIN
        UPDATE @DayOfWeek
        SET YearCount = 0
        SET @CurrentYear = DATEPART(YY, @CurrentDate)
    END

        -- Set values in table data type created above from variables

    UPDATE @DayOfWeek
    SET
        MonthCount = MonthCount + 1,
        QuarterCount = QuarterCount + 1,
        YearCount = YearCount + 1
    WHERE DOW = DATEPART(DW, @CurrentDate)

    SELECT
        @DayOfWeekInMonth = MonthCount,
        @DayOfQuarter = QuarterCount,
        @DayOfWeekInYear = YearCount
    FROM @DayOfWeek
    WHERE DOW = DATEPART(DW, @CurrentDate)
```

```
INSERT INTO [dbo].[DimDate]
SELECT

    CONVERT (char(8),@CurrentDate,112) as DateKey,
    @CurrentDate AS Date,
    CONVERT (char(10),@CurrentDate,103) as FullDateUK,
    CONVERT (char(10),@CurrentDate,101) as FullDateUSA,
    DATEPART(DD, @CurrentDate) AS DayOfMonth,
    --Apply Suffix values like 1st, 2nd 3rd etc..
    CASE
        WHEN DATEPART(DD,@CurrentDate) IN (11,12,13)
        THEN CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'th'
        WHEN RIGHT(DATEPART(DD,@CurrentDate),1) = 1
        THEN CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'st'
        WHEN RIGHT(DATEPART(DD,@CurrentDate),1) = 2
        THEN CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'nd'
        WHEN RIGHT(DATEPART(DD,@CurrentDate),1) = 3
        THEN CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'rd'
        ELSE CAST(DATEPART(DD,@CurrentDate) AS VARCHAR) + 'th'
        END AS DaySuffix,

    DATENAME(DW, @CurrentDate) AS DayName,
    DATEPART(DW, @CurrentDate) AS DayOfWeekUSA,

    -- check for day of week as Per US and change it as per UK format
    CASE DATEPART(DW, @CurrentDate)
        WHEN 1 THEN 7
        WHEN 2 THEN 1
        WHEN 3 THEN 2
        WHEN 4 THEN 3
        WHEN 5 THEN 4
        WHEN 6 THEN 5
        WHEN 7 THEN 6
        END
        AS DayOfWeekUK,

    @DayOfWeekInMonth AS DayOfWeekInMonth,
    @DayOfWeekInYear AS DayOfWeekInYear,
    @DayOfQuarter AS DayOfQuarter,
    DATEPART(DY, @CurrentDate) AS DayOfYear,
    DATEPART(WW, @CurrentDate) + 1 - DATEPART(WW, CONVERT(VARCHAR,
    DATEPART(MM, @CurrentDate)) + '/1/' + CONVERT(VARCHAR,
    DATEPART(YY, @CurrentDate))) AS WeekOfMonth,
    (DATEDIFF(DD, DATEADD(QQ, DATEDIFF(QQ, 0, @CurrentDate), 0),
    @CurrentDate) / 7) + 1 AS WeekOfQuarter,
    DATEPART(WW, @CurrentDate) AS WeekOfYear,
    DATEPART(MM, @CurrentDate) AS Month,
    DATENAME(MM, @CurrentDate) AS MonthName,
```

```
        CASE
            WHEN DATEPART(MM, @CurrentDate) IN (1, 4, 7, 10) THEN 1
            WHEN DATEPART(MM, @CurrentDate) IN (2, 5, 8, 11) THEN 2
            WHEN DATEPART(MM, @CurrentDate) IN (3, 6, 9, 12) THEN 3
            END AS MonthOfQuarter,
        DATEPART(QQ, @CurrentDate) AS Quarter,
        CASE DATEPART(QQ, @CurrentDate)
            WHEN 1 THEN 'First'
            WHEN 2 THEN 'Second'
            WHEN 3 THEN 'Third'
            WHEN 4 THEN 'Fourth'
            END AS QuarterName,
        DATEPART(YEAR, @CurrentDate) AS Year,
        'CY ' + CONVERT(VARCHAR, DATEPART(YEAR, @CurrentDate)) AS YearName,
        LEFT(DATENAME(MM, @CurrentDate), 3) + '-' + CONVERT(VARCHAR,
        DATEPART(YY, @CurrentDate)) AS MonthYear,
        RIGHT('0' + CONVERT(VARCHAR, DATEPART(MM, @CurrentDate)),2) +
        CONVERT(VARCHAR, DATEPART(YY, @CurrentDate)) AS MNYYYY,
        CONVERT(DATETIME, CONVERT(DATE, DATEADD(DD, - (DATEPART(DD,
        @CurrentDate) - 1), @CurrentDate))) AS FirstDayOfMonth,
        CONVERT(DATETIME, CONVERT(DATE, DATEADD(DD, - (DATEPART(DD,
        (DATEADD(MM, 1, @CurrentDate)))), DATEADD(MM, 1,
        @CurrentDate)))) AS LastDayOfMonth,
        DATEADD(QQ, DATEDIFF(QQ, 0, @CurrentDate), 0) AS FirstDayOfQuarter,
        DATEADD(QQ, DATEDIFF(QQ, -1, @CurrentDate), -1) AS LastDayOfQuarter,
        CONVERT(DATETIME, '01/01/' + CONVERT(VARCHAR, DATEPART(YY,
        @CurrentDate))) AS FirstDayOfYear,
        CONVERT(DATETIME, '12/31/' + CONVERT(VARCHAR, DATEPART(YY,
        @CurrentDate))) AS LastDayOfYear,
        NULL AS IsHolidaySL,
        CASE DATEPART(DW, @CurrentDate)
            WHEN 1 THEN 0
            WHEN 2 THEN 1
            WHEN 3 THEN 1
            WHEN 4 THEN 1
            WHEN 5 THEN 1
            WHEN 6 THEN 1
            WHEN 7 THEN 0
            END AS IsWeekday,
        NULL AS HolidaySL, (case when @CurrentDate = convert(date, sysdatetime()) then 1 else 0 end), 0, 0

    SET @CurrentDate = DATEADD(DD, 1, @CurrentDate)
END
```

Figure 4.49

Extract, Transform and Load Trip Data

Trip table is transactional table in this dataset. This table will be transformed as a fact table in the data warehouse.

As other tables this table data also loaded from the source database to staging database by creating trip staging table. An event handler is used also in this step.

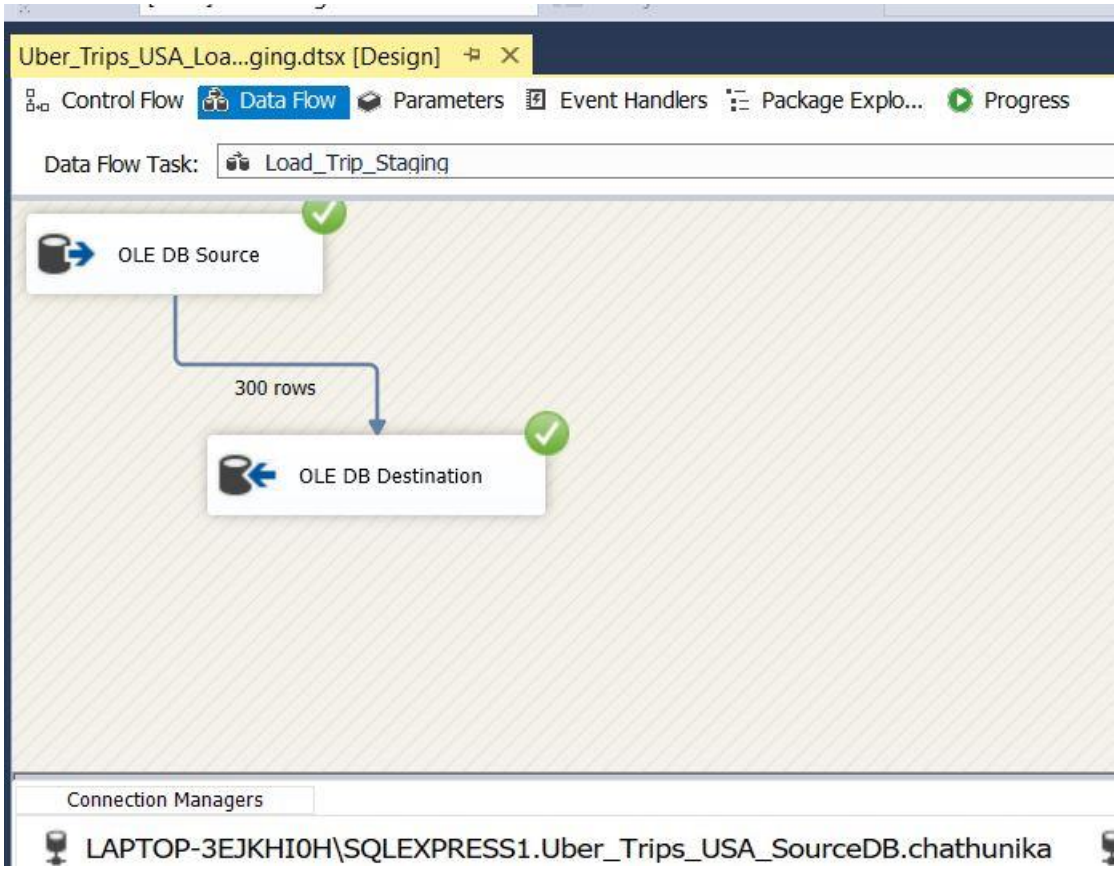


Figure 4.50

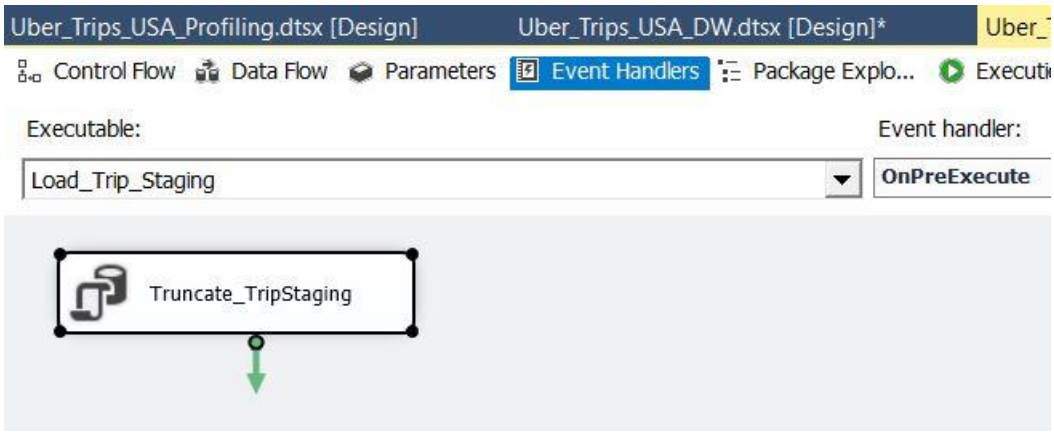


Figure 4.51

When extracting and transforming trip data to fact trip table in data warehouse table, there were so many foreign keys were included. For each and every foreign key the following lookups were used.

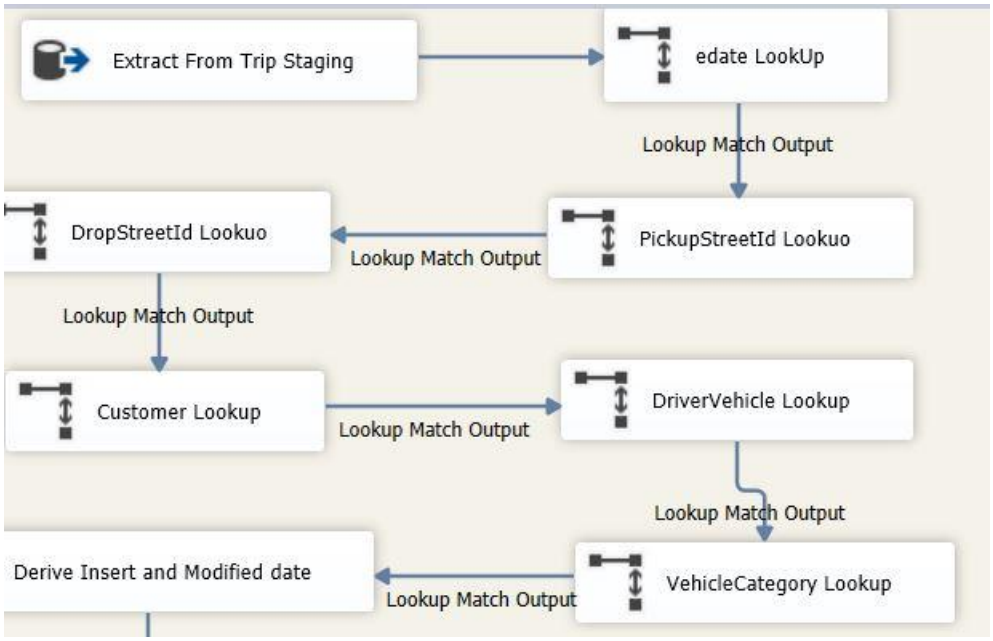


Figure 4.52

Lookup Column	Lookup Operation	Output Alias
DateKey	<add as new column>	DateKey

Figure 4.53

Lookup Column	Lookup Operation	Output Alias
streetId_SK	<add as new column>	streetId_SK

Figure 4.54

Lookup Column	Lookup Operation	Output Alias
streetId_SK	<add as new column>	streetId_SK

Figure 4.55

Lookup Column	Lookup Operation	Output Alias
customerId_SK	<add as new column>	customerId_SK

Figure 4.56

Lookup Column	Lookup Operation	Output Alias
driverVehicleId_SK	<add as new column>	driverVehicleId_SK

Figure 4.57

Lookup Column	Lookup Operation	Output Alias
chargePerKm	<add as new column>	chargePerKm

Figure 4.58

After the above steps insert date and modified date were derived.

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

Figure 4.59

Then the columns of trip staging table and columns of fact trip table were mapped.

Input Column	Destination Column
<ignore>	tripId_SK
ID	tripAltenativeId
DateKey	edateKey
pickupStreetID	pickUpStreetId_SK
dropStreetID	dropStreetId_SK
stime	stime
etime	etime
customerId_SK	customerId_SK
driverVehicleId_SK	driverVehicleId_SK
No Km	noKm
Extract From Trip Staging.chargePerKm	chargePerKm
tax	tax
<ignore>	totalAmount
PaymentType	PaymentType
InsertDate	insertDate
ModifiedDate	ModifiedDate

Figure 4.60

In above diagram tripId_SK columns is not mapped because it is set to auto increment when creating the fact trip.

Identity Specification	Yes
(Is Identity)	Yes
Identity Increment	1
Identity Seed	1

Figure 4.61

Also, the totalAmount column also derived when creating the fact trip table.

Computed Column Specification	([tax]+[noKm]*[chargePerKm])
(Formula)	([tax]+[noKm]*[chargePerKm])
Is Persisted	No
Condensed Data Type	

Figure 4.62

The whole data flow of the fact trip dimension is as below.

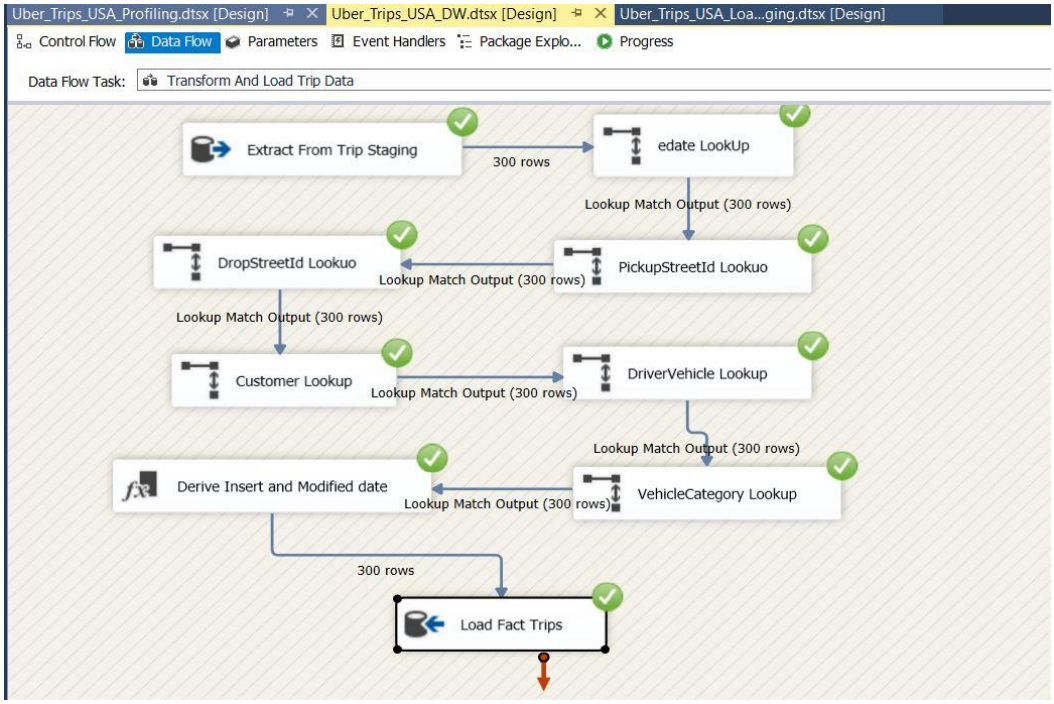


Figure 4.63

The whole control flow of extracting, loading and transforming dataset into data warehouse is as below.

Figure 4.64