CONTENTS

INTRODUCTION

AdventureWorks 2019 is an OLTP sample database originally published by Microsoft, which stores the data of an ecommerce retail company from multiple perspectives, such as sales, production and relative people. However, it is difficult for analysts of the company to effectively get insights and create business values, because it costs much time to integrate information from multiple tables in a database. Thus, to enhance the data quality and consistency, improve the decision-making process, and generate a higher return of interest based on business intelligence, it is necessary for us to maintain a data warehouse for OLAP.

In this project, our goals include:

- Build a data warehouse (dimensional model) of the AdventureWorks 2019 database - consolidate two fact tables *FactInternetSales* and *FactProductInventory* to present the data of internet sales and inventory in a clear manner
- Design and construct data pipeline via SSIS to automatically populate or backup data into our dimension and fact tables
- Feed data into an OLAP model for to improve efficiency of multi-dimensional analytical queries
- Integrate with BI tools to support underlying data reports refreshing and enable business intelligence

The data will be processed as shown in Figure 1 below.

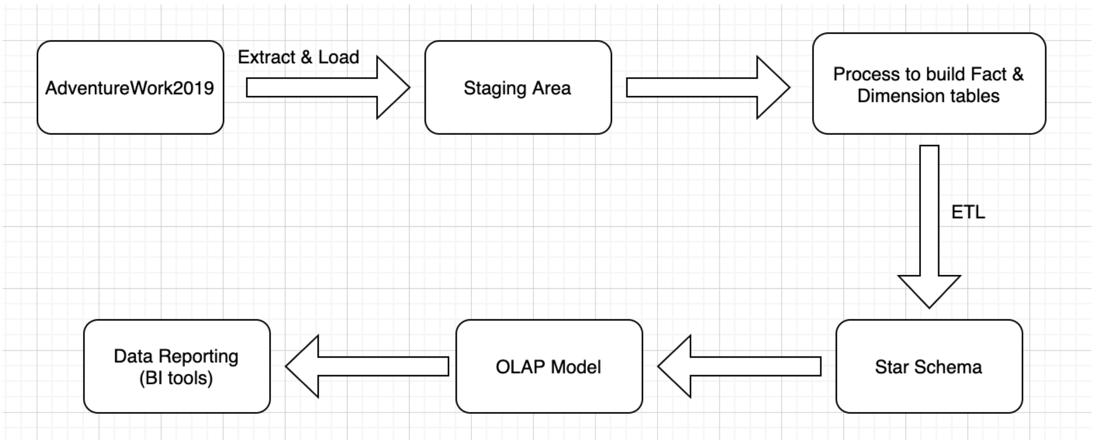


Figure 1: Data process diagram

DATA FLOW DIAGRAM

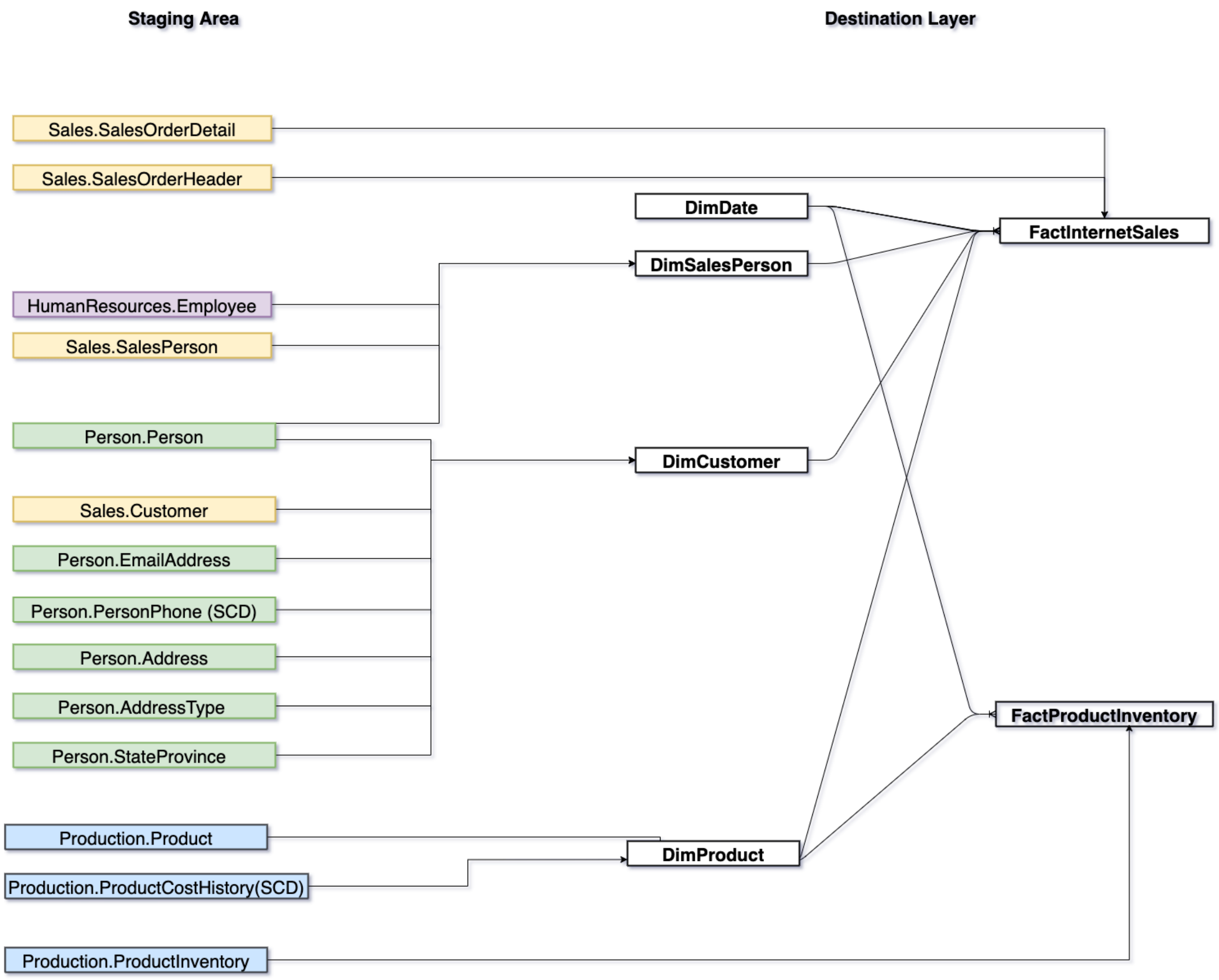


Figure 2: Data flow of *FactInternetSales* and *FactProductInventory*

Figure 2 shows the data flow of the dimensional model, which flows from source(left) to destination(right). There are two levels of data layer - staging area and destination (dim and fact) layer. (Link to a detailed data flow diagram: Data Flow Chart)

Staging Area:

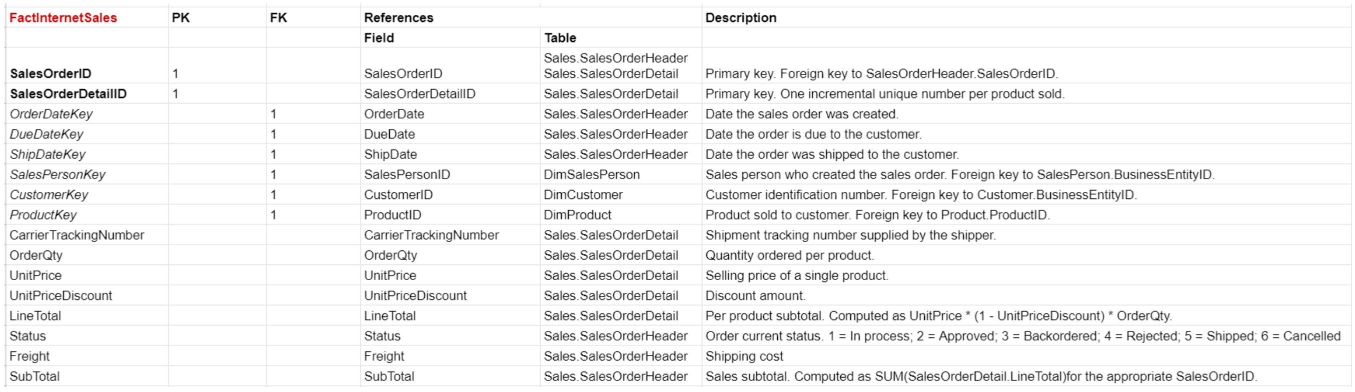
Staging area is the replica of the source tables in AdventureWorks 2019 database. Each source table will have a staging table in the data warehouse accordingly. For daily processing, we first load the current day's data of each table into the corresponding staging table. The staging table will work as the source of the ETL procedures to populate dim tables and fact tables on a daily basis.

Destination Design:

1. **Fact Tables**

FactInternetSales - the only source of truth of sales data, containing data of order details and related customer and salesperson.

- Granularity: order-product level (SalesOrderID + SalesOrderDetailID)
  There could be multiple products included in one single order. This table serves data at an order-product level, which means each record in this table identifies a unique order-product pair, providing information about the specific product in the order.
- Daily processing: based on OrderDateKey

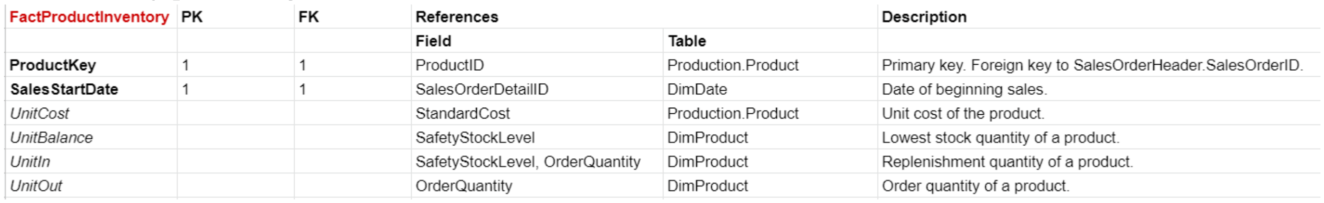


FactInternetSales	PK	FK	References		Description
			Field	Table	
SalesOrderID	1		SalesOrderID	Sales.SalesOrderHeader Sales.SalesOrderDetail	Primary key. Foreign key to SalesOrderHeader.SalesOrderID.
SalesOrderDetailID	1		SalesOrderDetailID	Sales.SalesOrderDetail	Primary key. One incremental unique number per product sold.
OrderDateKey		1	OrderDate	Sales.SalesOrderHeader	Date the sales order was created.
DueDateKey		1	DueDate	Sales.SalesOrderHeader	Date the order is due to the customer.
ShipDateKey		1	ShipDate	Sales.SalesOrderHeader	Date the order was shipped to the customer.
SalesPersonKey		1	SalesPersonID	DimSalesPerson	Sales person who created the sales order. Foreign key to SalesPerson.BusinessEntityID.
CustomerKey		1	CustomerID	DimCustomer	Customer identification number. Foreign key to Customer.BusinessEntityID.
ProductKey		1	ProductID	DimProduct	Product sold to customer. Foreign key to Product.ProductID.
CarrierTrackingNumber			CarrierTrackingNumber	Sales.SalesOrderDetail	Shipment tracking number supplied by the shipper.
OrderQty			OrderQty	Sales.SalesOrderDetail	Quantity ordered per product.
UnitPrice			UnitPrice	Sales.SalesOrderDetail	Selling price of a single product.
UnitPriceDiscount			UnitPriceDiscount	Sales.SalesOrderDetail	Discount amount.
LineTotal			LineTotal	Sales.SalesOrderDetail	Per product subtotal. Computed as UnitPrice * (1 - UnitPriceDiscount) * OrderQty.
Status			Status	Sales.SalesOrderHeader	Order current status. 1 = In process; 2 = Approved; 3 = Backordered; 4 = Rejected; 5 = Shipped; 6 = Cancelled
Freight			Freight	Sales.SalesOrderHeader	Shipping cost
SubTotal			SubTotal	Sales.SalesOrderHeader	Sales subtotal. Computed as SUM(SalesOrderDetail.LineTotal)for the appropriate SalesOrderID.

Figure 3: References and descriptions of attributes in *FactInternetSales* table

FactProductInventory - a daily snapshot table of products inventory data indicating the current inventory, daily in, daily out of products.

- Granularity: product + DailyDate
  The primary key of this table is the combination of ProductKey and DailyDate. Each record identifies the inventory of a certain product on a certain day. For each product, its inventory data will be recorded from the day it started to sell to the last day of the dataset.
- Daily processing: based on SellStartDate

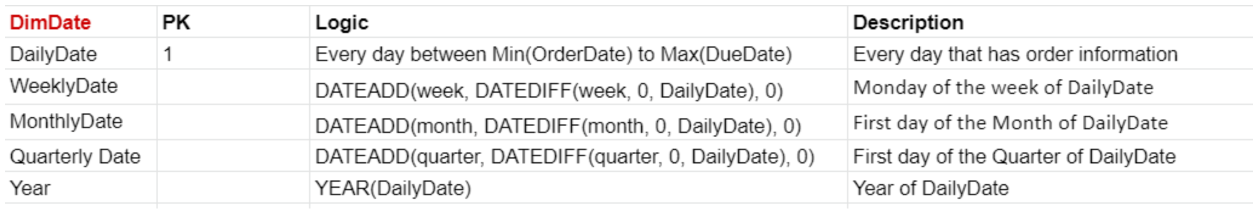


FactProductInventory	PK	FK	References		Description
			Field	Table	
ProductKey	1	1	ProductID	Production.Product	Primary key. Foreign key to SalesOrderHeader.SalesOrderID.
SalesStartDate	1	1	SalesOrderDetailID	DimDate	Date of beginning sales.
UnitCost			StandardCost	Production.Product	Unit cost of the product.
UnitBalance			SafetyStockLevel	DimProduct	Lowest stock quantity of a product.
UnitIn			SafetyStockLevel, OrderQuantity	DimProduct	Replenishment quantity of a product.
UnitOut			OrderQuantity	DimProduct	Order quantity of a product.

Figure 4: References and descriptions of attributes in *FactProductInventory* table

2. **Dimension Tables**

DimDate -
DimDate is a table of generated time data, containing every single day in the range of time that we care. We generate the time data in different intervals (daily, weekly, monthly, quarterly, year), join with *FactInternetSales* and *FactProductInventory* to query or aggregate sales and inventory data in a certain time period.

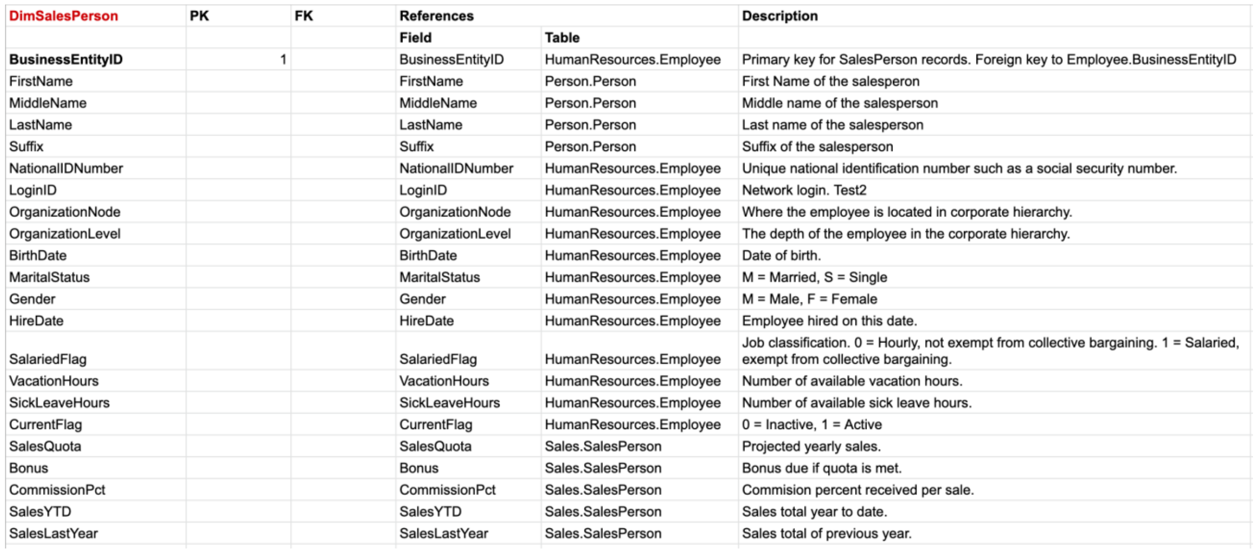


DimDate	PK	Logic	Description
DailyDate	1	Every day between Min(OrderDate) to Max(DueDate)	Every day that has order information
WeeklyDate		DATEADD(week, DATEDIFF(week, 0, DailyDate), 0)	Monday of the week of DailyDate
MonthlyDate		DATEADD(month, DATEDIFF(month, 0, DailyDate), 0)	First day of the Month of DailyDate
Quarterly Date		DATEADD(quarter, DATEDIFF(quarter, 0, DailyDate), 0)	First day of the Quarter of DailyDate
Year		YEAR(DailyDate)	Year of DailyDate

Figure 5: References and descriptions of attributes in *DimDate* table

DimSalesPerson -

The DimSalesPerson table stores data related to salesperson, in the perspective of demographic, employment and sales achievement. These attributes will provide salespersons information in the *FactInternetSales* table. Attributes stored in the [*Dim.SalesPerson*] table will be acquired by extracting data from joined tables including [*HumanResources.Employee*], [*Person.Person*] and [*Sales.SalesPerson*] with the same BusinessEntityID.



DimSalesPerson	PK	FK	References		Description
			Field	Table	
BusinessEntityID	1		BusinessEntityID	HumanResources.Employee	Primary key for SalesPerson records. Foreign key to Employee.BusinessEntityID
FirstName			FirstName	Person.Person	First Name of the salesperon
MiddleName			MiddleName	Person.Person	Middle name of the salesperson
LastName			LastName	Person.Person	Last name of the salesperson
Suffix			Suffix	Person.Person	Suffix of the salesperson
NationalIDNumber			NationalIDNumber	HumanResources.Employee	Unique national identification number such as a social security number.
LoginID			LoginID	HumanResources.Employee	Network login. Test2
OrganizationNode			OrganizationNode	HumanResources.Employee	Where the employee is located in corporate hierarchy.
OrganizationLevel			OrganizationLevel	HumanResources.Employee	The depth of the employee in the corporate hierarchy.
BirthDate			BirthDate	HumanResources.Employee	Date of birth.
MaritalStatus			MaritalStatus	HumanResources.Employee	M = Married, S = Single
Gender			Gender	HumanResources.Employee	M = Male, F = Female
HireDate			HireDate	HumanResources.Employee	Employee hired on this date.
SalariedFlag			SalariedFlag	HumanResources.Employee	Job classification. 0 = Hourly, not exempt from collective bargaining. 1 = Salaried, exempt from collective bargaining.
VacationHours			VacationHours	HumanResources.Employee	Number of available vacation hours.
SickLeaveHours			SickLeaveHours	HumanResources.Employee	Number of available sick leave hours.
CurrentFlag			CurrentFlag	HumanResources.Employee	0 = Inactive, 1 = Active
SalesQuota			SalesQuota	Sales.SalesPerson	Projected yearly sales.
Bonus			Bonus	Sales.SalesPerson	Bonus due if quota is met.
CommissionPct			CommissionPct	Sales.SalesPerson	Commision percent received per sale.
SalesYTD			SalesYTD	Sales.SalesPerson	Sales total year to date.
SalesLastYear			SalesLastYear	Sales.SalesPerson	Sales total of previous year.

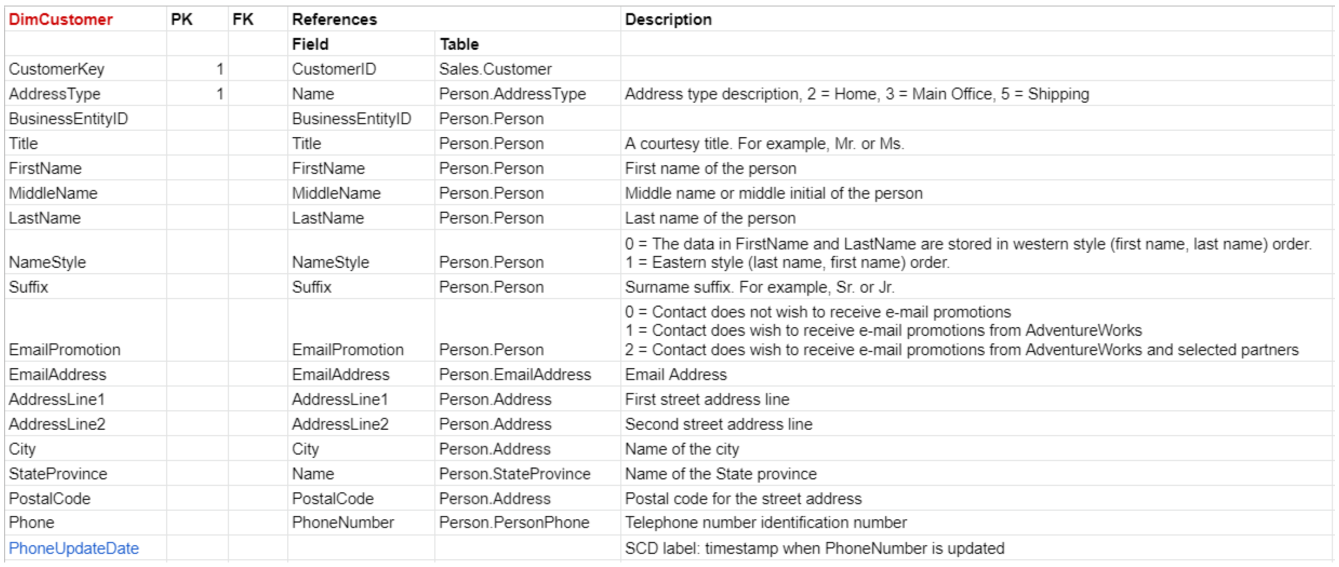
Figure 6: References and descriptions of attributes in *DimSalesPerson* table

DimCustomer (exists SCD) -

The DimCustomer table contains customer information including customer id, name, suffix, email address, customer address and phone number. These attributes will be used to keep track of customer purchase behavior. Customer's information of title, name, email address and phone number in [*Dim.Customer*] table will be collected by joining tables including [Sales.Customer], [Person.Person],[Person.AddressType],[Person.EmailAddress],[Person.SateProvince] and [Person.PersonPhone] using the relationship established by the BusinessEntityID key, AddressID, AddressTypeID and StateProvinceID.

This table could exist duplicate against CustomerKey. The reason behind is one customer could have multiple addresses which are in different address types (Shipping, Home, etc.). Thus, the primary key of this table is the combination of CustomerKey and AddressType.

There exists SCD in this dimension table. 'Phone number' will be overwritten when an update happens. In the meanwhile, the PhoneUpdateDate will also be updated to the current timestamp.



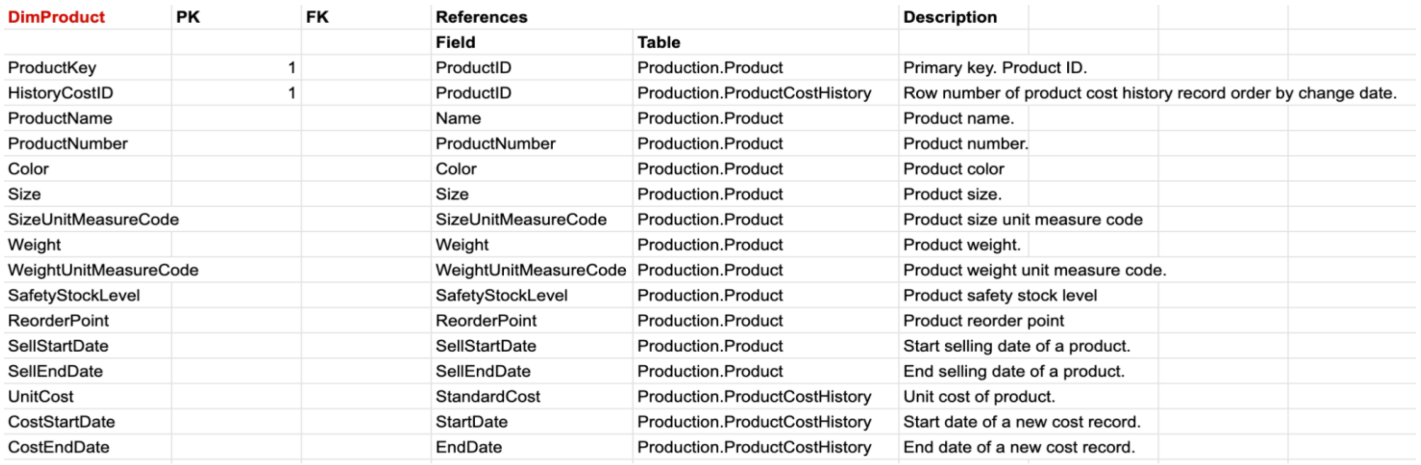
DimCustomer	PK	FK	References		Description
			Field	Table	
CustomerKey	1		CustomerID	Sales.Customer	
AddressType	1		Name	Person.AddressType	Address type description, 2 = Home, 3 = Main Office, 5 = Shipping
BusinessEntityID			BusinessEntityID	Person.Person	
Title			Title	Person.Person	A courtesy title. For example, Mr. or Ms.
FirstName			FirstName	Person.Person	First name of the person
MiddleName			MiddleName	Person.Person	Middle name or middle initial of the person
LastName			LastName	Person.Person	Last name of the person
NameStyle			NameStyle	Person.Person	0 = The data in FirstName and LastName are stored in western style (first name, last name) order. 1 = Eastern style (last name, first name) order.
Suffix			Suffix	Person.Person	Surname suffix. For example, Sr. or Jr.
EmailPromotion			EmailPromotion	Person.Person	0 = Contact does not wish to receive e-mail promotions 1 = Contact does wish to receive e-mail promotions from AdventureWorks 2 = Contact does wish to receive e-mail promotions from AdventureWorks and selected partners
EmailAddress			EmailAddress	Person.EmailAddress	Email Address
AddressLine1			AddressLine1	Person.Address	First street address line
AddressLine2			AddressLine2	Person.Address	Second street address line
City			City	Person.Address	Name of the city
StateProvince			Name	Person.StateProvince	Name of the State province
PostalCode			PostalCode	Person.Address	Postal code for the street address
Phone			PhoneNumber	Person.PersonPhone	Telephone number identification number
PhoneUpdateDate					SCD label: timestamp when PhoneNumber is updated

Figure 7: References and descriptions of attributes in *DimCustomer* table

DimProduct (exists SCD) -
The DimProduct table contains product id, product name, product number, start date, end date, standard cost, list price, product description, safety stock level, order quantity, and large photo of the product. These attributes will be used to maintain the information of product inventory. [*Dim.Product*] table consists of product information including product name, identify number, measure code etc will be gathered by joining tables involving [*Production.Product*], [*Production.ProductCostHistory*], [*Production.ProductListPriceHistory*], [*Production.ProductInventory*], [*Production.WordOrder*], [*Production.ProductDescription*] and [*Production.ProductLargePhoto*] using same ProductID.

There will be duplicate product keys since we may have the same product with multiple updated records. Therefore, a combination of ProductKey and HistoryCostID will be used as the primary key.

There exists SCD in this dimension table. The DimProduct table retains the full history of product cost values. When the value of ProductUnitCost changes, the current record is closed.



DimProduct	PK	FK	References		Description
			Field	Table	
ProductKey	1		ProductID	Production.Product	Primary key. Product ID.
HistoryCostID	1		ProductID	Production.ProductCostHistory	Row number of product cost history record order by change date.
ProductName			Name	Production.Product	Product name.
ProductNumber			ProductNumber	Production.Product	Product number.
Color			Color	Production.Product	Product color
Size			Size	Production.Product	Product size.
SizeUnitMeasureCode			SizeUnitMeasureCode	Production.Product	Product size unit measure code
Weight			Weight	Production.Product	Product weight.
WeightUnitMeasureCode			WeightUnitMeasureCode	Production.Product	Product weight unit measure code.
SafetyStockLevel			SafetyStockLevel	Production.Product	Product safety stock level
ReorderPoint			ReorderPoint	Production.Product	Product reorder point
SellStartDate			SellStartDate	Production.Product	Start selling date of a product.
SellEndDate			SellEndDate	Production.Product	End selling date of a product.
UnitCost			StandardCost	Production.ProductCostHistory	Unit cost of product.
CostStartDate			StartDate	Production.ProductCostHistory	Start date of a new cost record.
CostEndDate			EndDate	Production.ProductCostHistory	End date of a new cost record.

Figure 8: References and descriptions of attributes in *DimProduct* table

ETL PROCESS

1. Initial Loading: load data before 2014

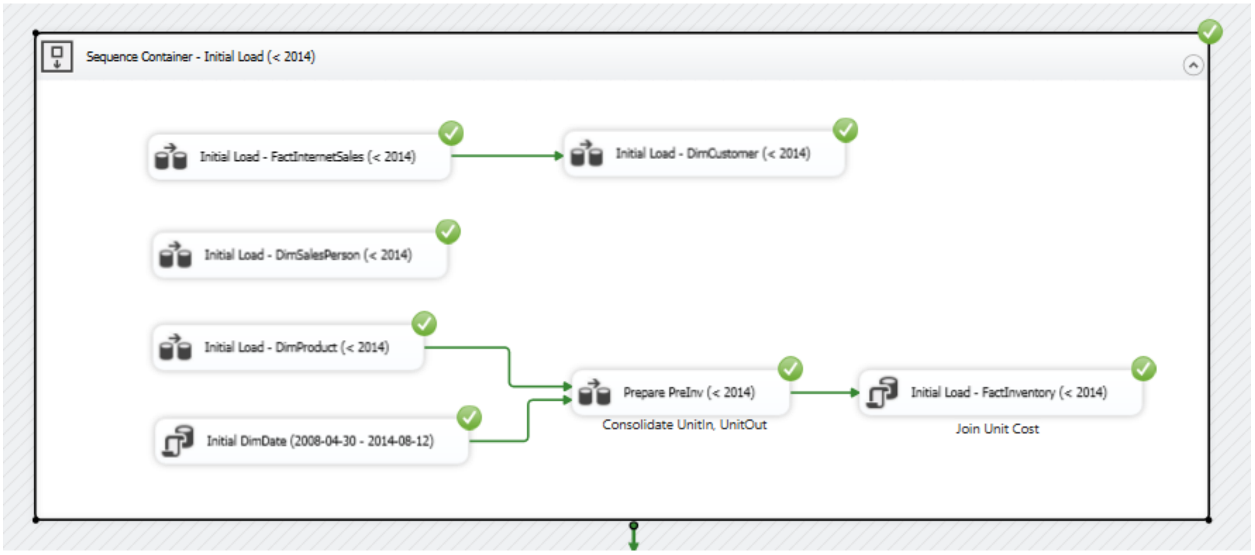
With an initial load, we retrieve the sales data with OrderDate before 2014-01-01 and load into *FactInternetSales.* Customer table doesn't have an appropriate timestamp for identifying data prior to 2014. To mimic the state of *DimCustomer* at '2013-12-31', we obtain all the related CustomerKey from *FactInternetSales*, and pull data only for those records to populate the *DimCustomer* table.

In the initial load for *DimSalesPerson*, we populated for the salespersons hired before '2014-01-01'. For the initial load of DimProduct, we populated the products that started selling before '2014-01-01'. Since DimDate is a list of dates that we want to backfill the data, we wouldn't make changes to it after initialization. It contains dates from '2008-04-30' to '2014-08-12'.

As for the initial load of *FactProductInvenotry*, we consolidated the UnitIn and UnitOut data for each product between its SellStartDate and '2013-12-31'. In the case that the product ended selling before '2013-12-31', we will only include inventory data to its SellEndDate.

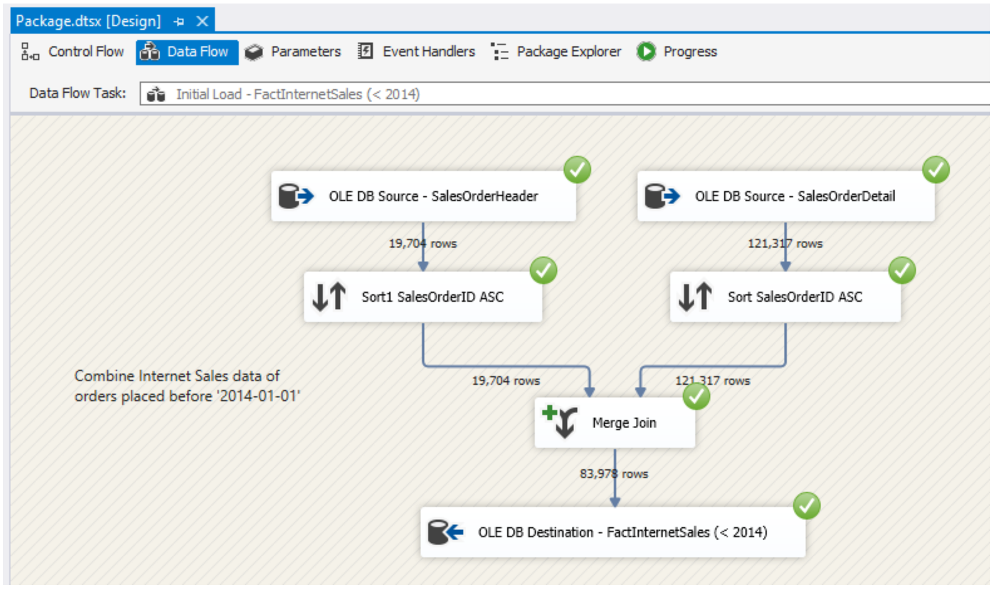
SSIS Implementation:

- Initial Load Control Flow


Sequence Container - Initial Load (< 2014)
Initial Load - FactInternetSales (< 2014)
Initial Load - DimCustomer (< 2014)
Initial Load - DimSalesPerson (< 2014)
Initial Load - DimProduct (< 2014)
Initial DimDate (2008-04-30 - 2014-08-12)
Prepare PreInv (< 2014)
Consolidate UnitIn, UnitOut
Initial Load - FactInventory (< 2014)
Join Unit Cost

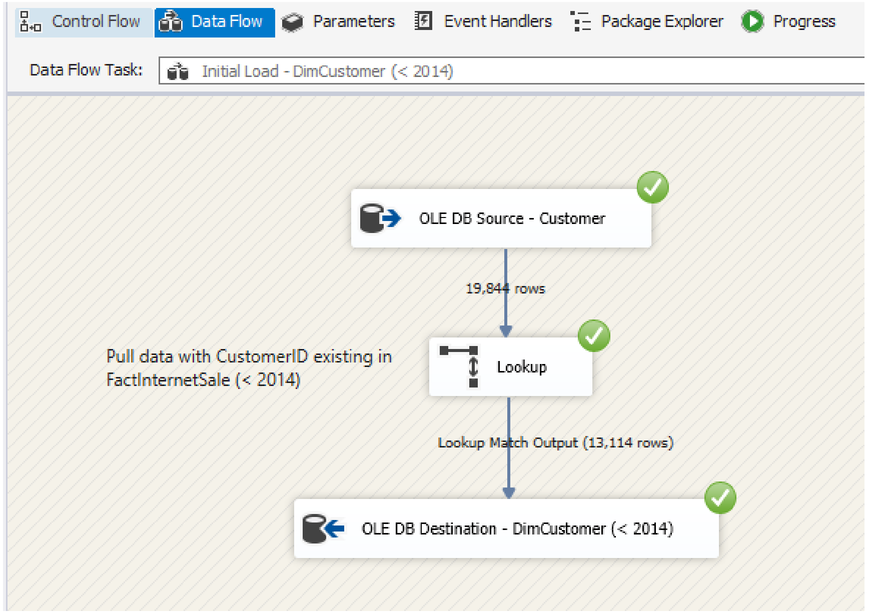
- Initial Load - FactInternetSales (< 2014)

Pull Internet sales data prior to 2014-01-01 (OrderDate < ‘2014-01-01’), and populate FactInternetSales table.


Package.dtsx [Design]
Control Flow
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
Data Flow Task:
Initial Load - FactInternetSales (< 2014)
OLE DB Source - SalesOrderHeader
OLE DB Source - SalesOrderDetail
19,704 rows
121,317 rows
Sort1 SalesOrderID ASC
Sort SalesOrderID ASC
Combine Internet Sales data of orders placed before '2014-01-01'
19,704 rows
121,317 rows
Merge Join
83,978 rows
OLE DB Destination - FactInternetSales (< 2014)

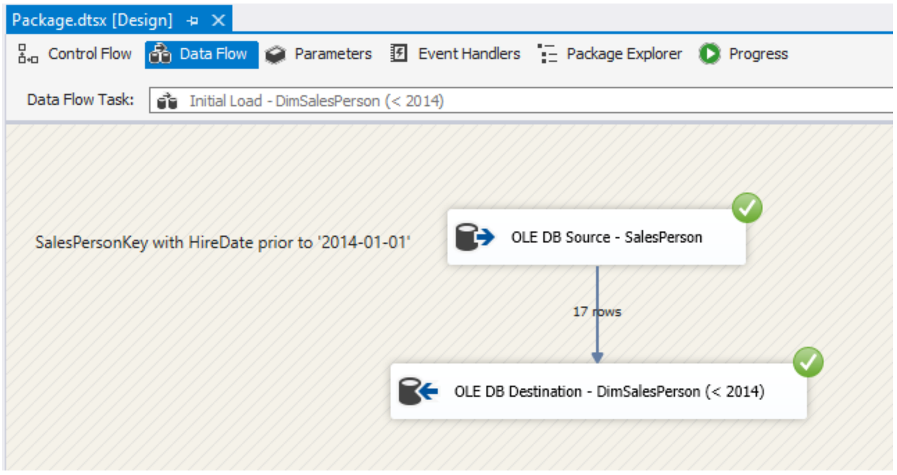
- Initial load - DimCustomer (< 2014)

First, consolidate customer data from the source tables in AdventureWork 2019. Then, build a Lookup task to identify those CustomerKey existing in FactInternetSales by the beginning of 2014. Last, populate customer data for those CustomerKey into the DimCustomer table.


Control Flow
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
Data Flow Task:
Initial Load - DimCustomer (< 2014)
OLE DB Source - Customer
19,844 rows
Pull data with CustomerID existing in FactInternetSale (< 2014)
Lookup
Lookup Match Output (13,114 rows)
OLE DB Destination - DimCustomer (< 2014)

- Initial load - DimSalesPerson (< 2014)

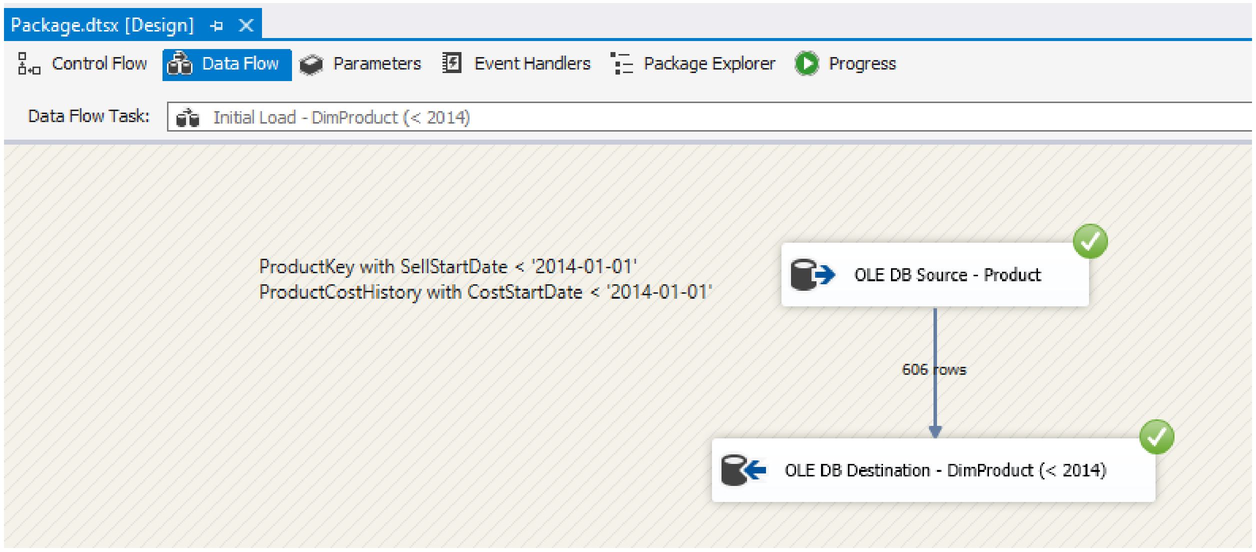
Populate SalePerson data for the SalesPerson that hired prior to ‘2014-01-01’.



SalesPersonKey with HireDate prior to '2014-01-01'

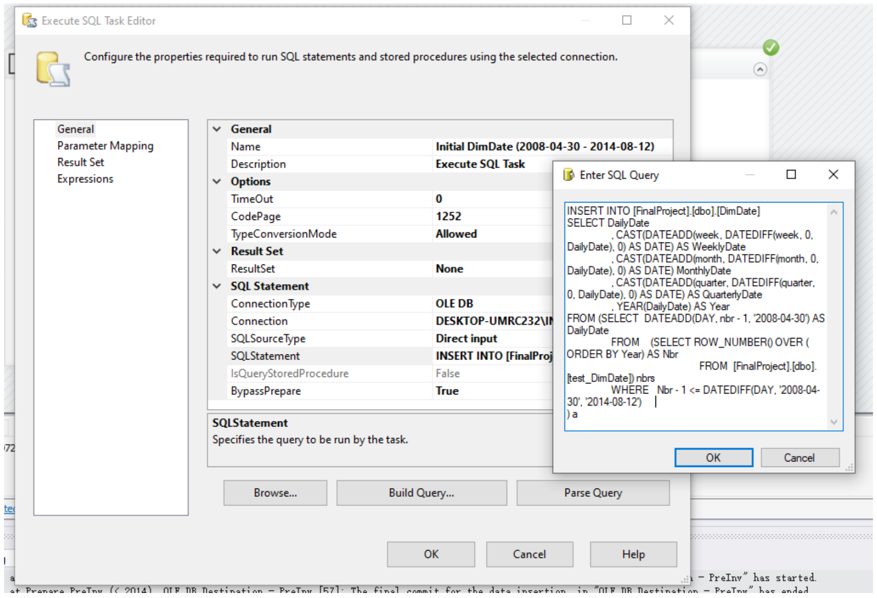
- Initial load - DimProduct (< 2014)

Consolidate product data for the products started selling before 2014. For the product's unit cost history, we only populate the records prior to ‘2014-01-01’.



ProductKey with SellStartDate < '2014-01-01'
ProductCostHistory with CostStartDate < '2014-01-01'

- Initial load - DimDate


Execute SQL Task Editor

Configure the properties required to run SQL statements and stored procedures using the selected connection.

General	
Name	**Initial DimDate (2008-04-30 - 2014-08-12)**
Description	**Execute SQL Task**
Options	
TimeOut	**0**
CodePage	**1252**
TypeConversionMode	**Allowed**
Result Set	
ResultSet	**None**
SQL Statement	
ConnectionType	**OLE DB**
Connection	**DESKTOP-UMRC232\IN**
SQLSourceType	**Direct input**
SQLStatement	**INSERT INTO [FinalProj**
IsQueryStoredProcedure	False
BypassPrepare	**True**

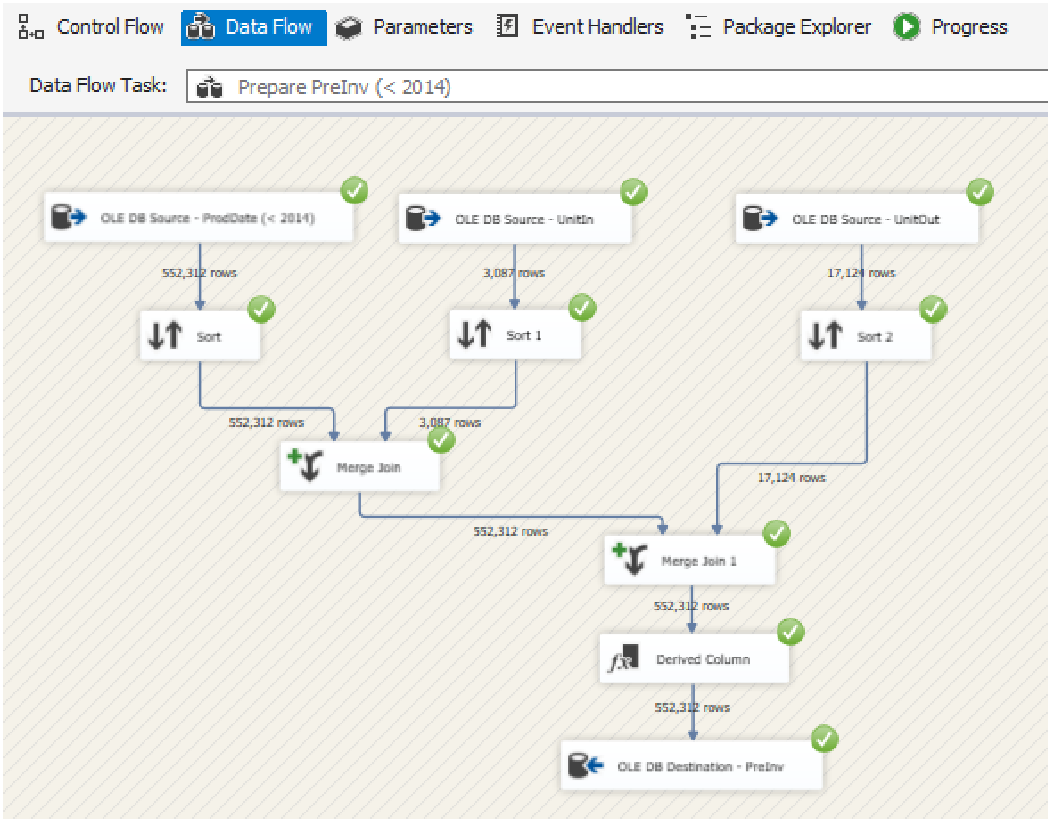
SQLStatement
Specifies the query to be run by the task.

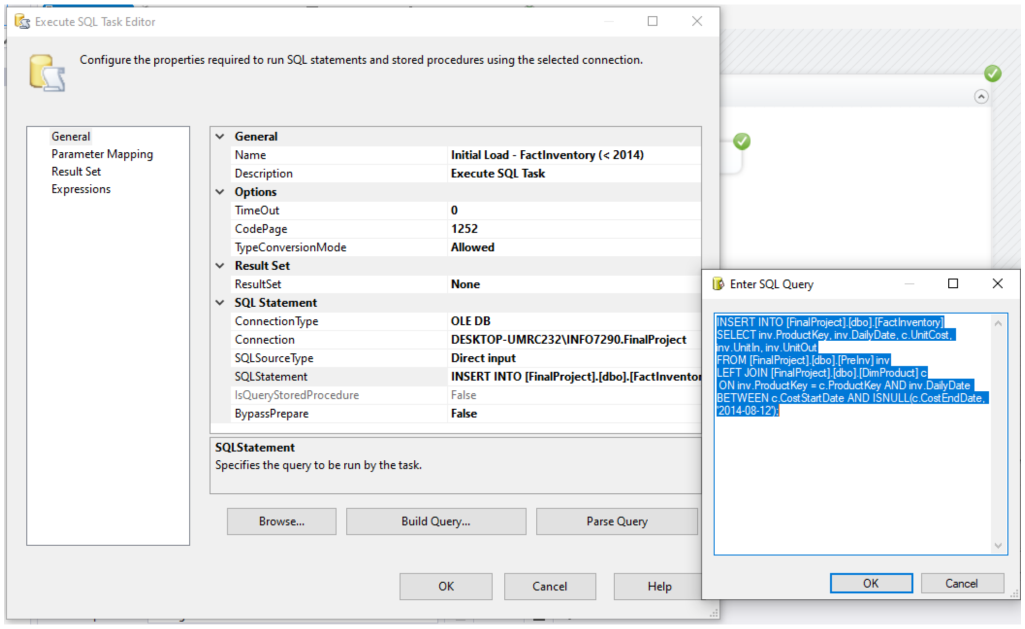
Enter SQL Query

```
INSERT INTO [FinalProject].[dbo].[DimDate]
SELECT DailyDate
          , CAST(DATEADD(week, DATEDIFF(week, 0,
DailyDate), 0) AS DATE) AS WeeklyDate
          , CAST(DATEADD(month, DATEDIFF(month, 0,
DailyDate), 0) AS DATE) MonthlyDate
          , CAST(DATEADD(quarter, DATEDIFF(quarter,
0, DailyDate), 0) AS DATE) AS QuarterlyDate
          , YEAR(DailyDate) AS Year
FROM (SELECT  DATEADD(DAY, nbr - 1, '2008-04-30') AS
DailyDate
          FROM    (SELECT ROW_NUMBER() OVER (
ORDER BY Year) AS Nbr
                              FROM  [FinalProject].[dbo].
[test_DimDate]) nbrs
          WHERE   Nbr - 1 <= DATEDIFF(DAY, '2008-04-
30', '2014-08-12')
) a
```

- Initial load - FactInventory

First, we prepared UnitsIn and UnitsOut data prior to ‘2014-01-01’, and then we included UnitCost historical data prior to ‘2014-01-01’.





Configure the properties required to run SQL statements and stored procedures using the selected connection.

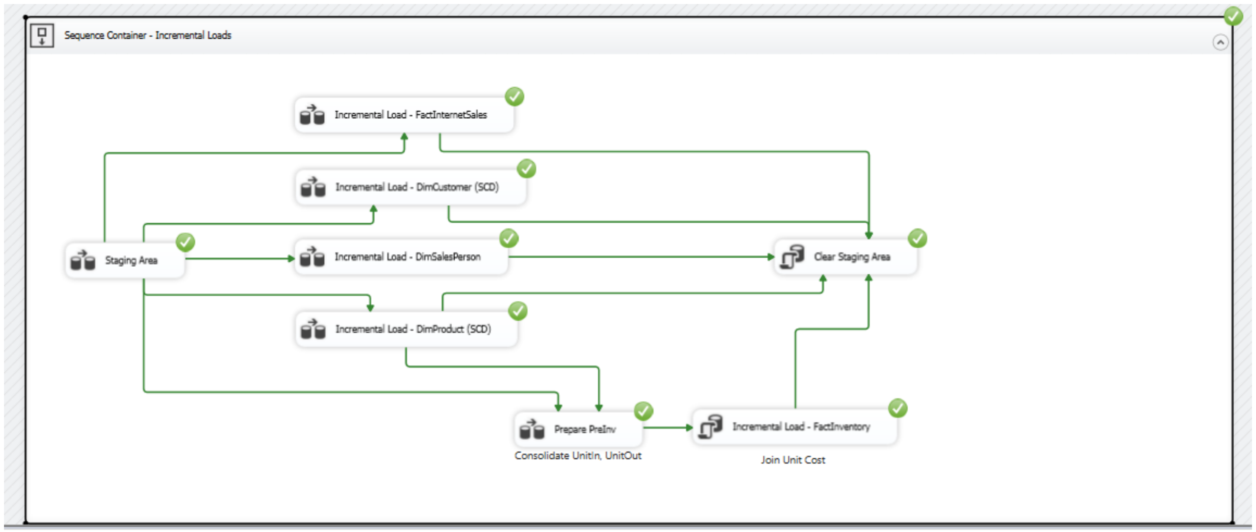
General	
Name	Initial Load - FactInventory (< 2014)
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1252
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	DESKTOP-UMRC232\INFO7290.FinalProject
SQLSourceType	Direct input
SQLStatement	INSERT INTO [FinalProject].[dbo].[FactInventor
IsQueryStoredProcedure	False
BypassPrepare	False

SQLStatement
Specifies the query to be run by the task.

```
INSERT INTO [FinalProject].[dbo].[FactInventory]
SELECT inv.ProductKey, inv.DailyDate, c.UnitCost,
inv.UnitIn, inv.UnitOut
FROM [FinalProject].[dbo].[PreInv] inv
LEFT JOIN [FinalProject].[dbo].[DimProduct] c
 ON inv.ProductKey = c.ProductKey AND inv.DailyDate
BETWEEN c.CostStartDate AND ISNULL(c.CostEndDate,
'2014-08-12');
```

After the initial load, we will be able to keep track of the records of DimCustomer which happened before 2014-01-01. To be specific, we have access to the information of customers who placed orders by querying the date when the event occurred. The data of Salesperson hired before 2014 and orders placed before 2014 are also available. We can also track the information of products stated selling before 2014-01-01 in DimProduct and get their corresponding inventory.

2. Incremental Loading: daily processing after 2014


Sequence Container - Incremental Loads
Incremental Load - FactInternetSales
Incremental Load - DimCustomer (SCD)
Staging Area
Incremental Load - DimSalesPerson
Clear Staging Area
Incremental Load - DimProduct (SCD)
Prepare PreInv
Consolidate UnitIn, UnitOut
Incremental Load - FactInventory
Join Unit Cost

For efficiency consideration, we plan to load data based on **month**. We performed 8 batches to load all data from ‘2014-01-01’ to ‘2014-08-12’.

The below table shows the tracking of data amount changes:

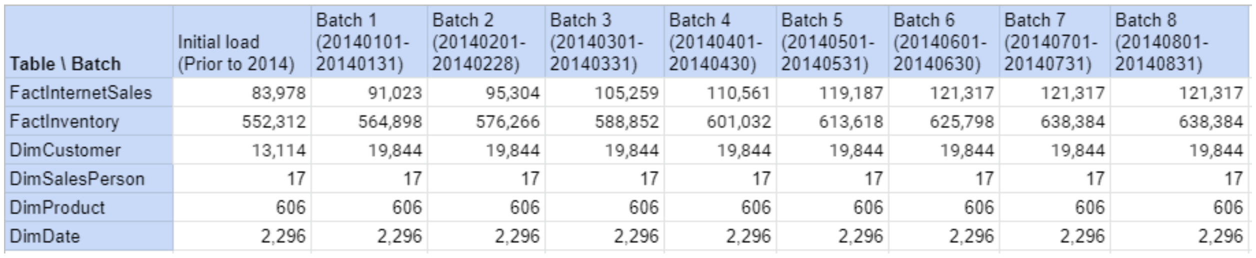
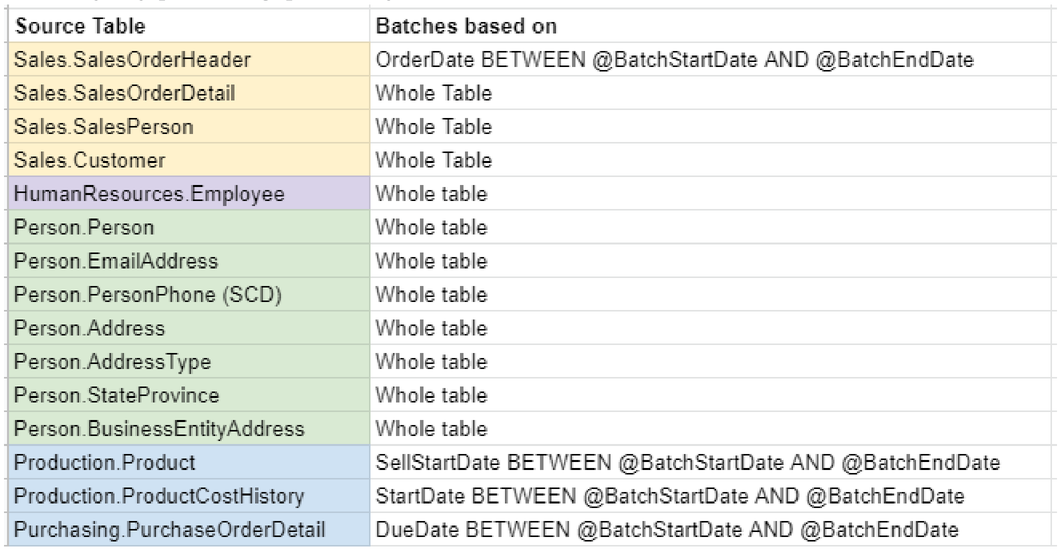


Table \ Batch	Initial load (Prior to 2014)	Batch 1 (20140101-20140131)	Batch 2 (20140201-20140228)	Batch 3 (20140301-20140331)	Batch 4 (20140401-20140430)	Batch 5 (20140501-20140531)	Batch 6 (20140601-20140630)	Batch 7 (20140701-20140731)	Batch 8 (20140801-20140831)
FactInternetSales	83,978	91,023	95,304	105,259	110,561	119,187	121,317	121,317	121,317
FactInventory	552,312	564,898	576,266	588,852	601,032	613,618	625,798	638,384	638,384
DimCustomer	13,114	19,844	19,844	19,844	19,844	19,844	19,844	19,844	19,844
DimSalesPerson	17	17	17	17	17	17	17	17	17
DimProduct	606	606	606	606	606	606	606	606	606
DimDate	2,296	2,296	2,296	2,296	2,296	2,296	2,296	2,296	2,296

1) Load current batch's Source Data into Staging Area

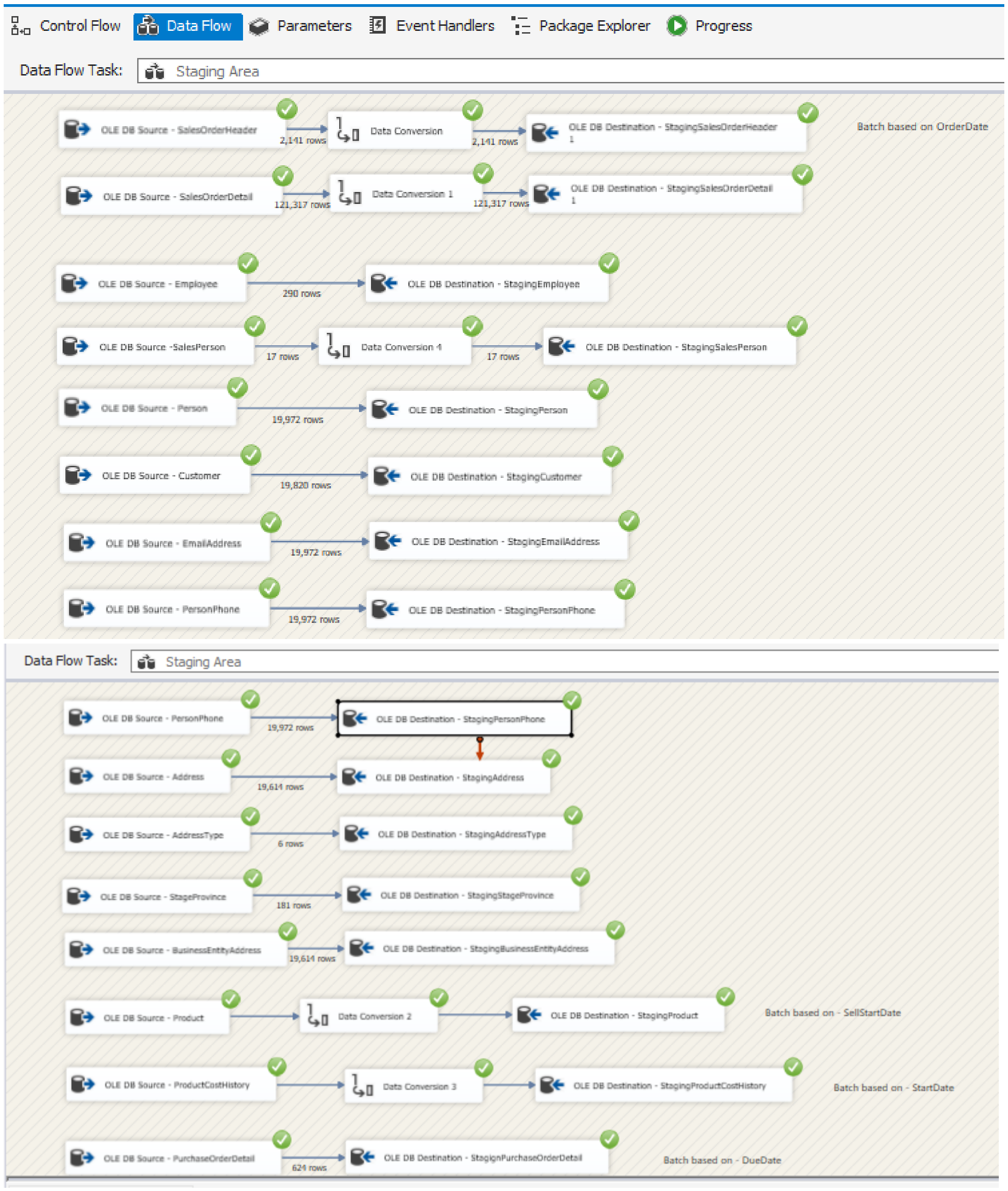
To achieve the incremental loads, we need to find one timestamp for each source table to distinguish data from different dates, in other words, to separate data into different batches. For some transactional source tables, such as SalesOrderHeader, we can easily find a column (OrderDate in this case) using for separate batches. For every day's processing, we will retrieve the current day's orders from the source table and load them into the staging area. However, there still exists some tables (e.g Customer) that we cannot find any useful timestamp, since they have maintained the latest data while didn't persist historical data. In this case, we will load the whole table on that day into the staging area and use it to update the downstream tables.

The following tables show how we handle the loading of staging area for each source table:
For every day processing, pass **DailyDate** as the current date.



Source Table	Batches based on
Sales.SalesOrderHeader	OrderDate BETWEEN @BatchStartDate AND @BatchEndDate
Sales.SalesOrderDetail	Whole Table
Sales.SalesPerson	Whole Table
Sales.Customer	Whole Table
HumanResources.Employee	Whole table
Person.Person	Whole table
Person.EmailAddress	Whole table
Person.PersonPhone (SCD)	Whole table
Person.Address	Whole table
Person.AddressType	Whole table
Person.StateProvince	Whole table
Person.BusinessEntityAddress	Whole table
Production.Product	SellStartDate BETWEEN @BatchStartDate AND @BatchEndDate
Production.ProductCostHistory	StartDate BETWEEN @BatchStartDate AND @BatchEndDate
Purchasing.PurchaseOrderDetail	DueDate BETWEEN @BatchStartDate AND @BatchEndDate

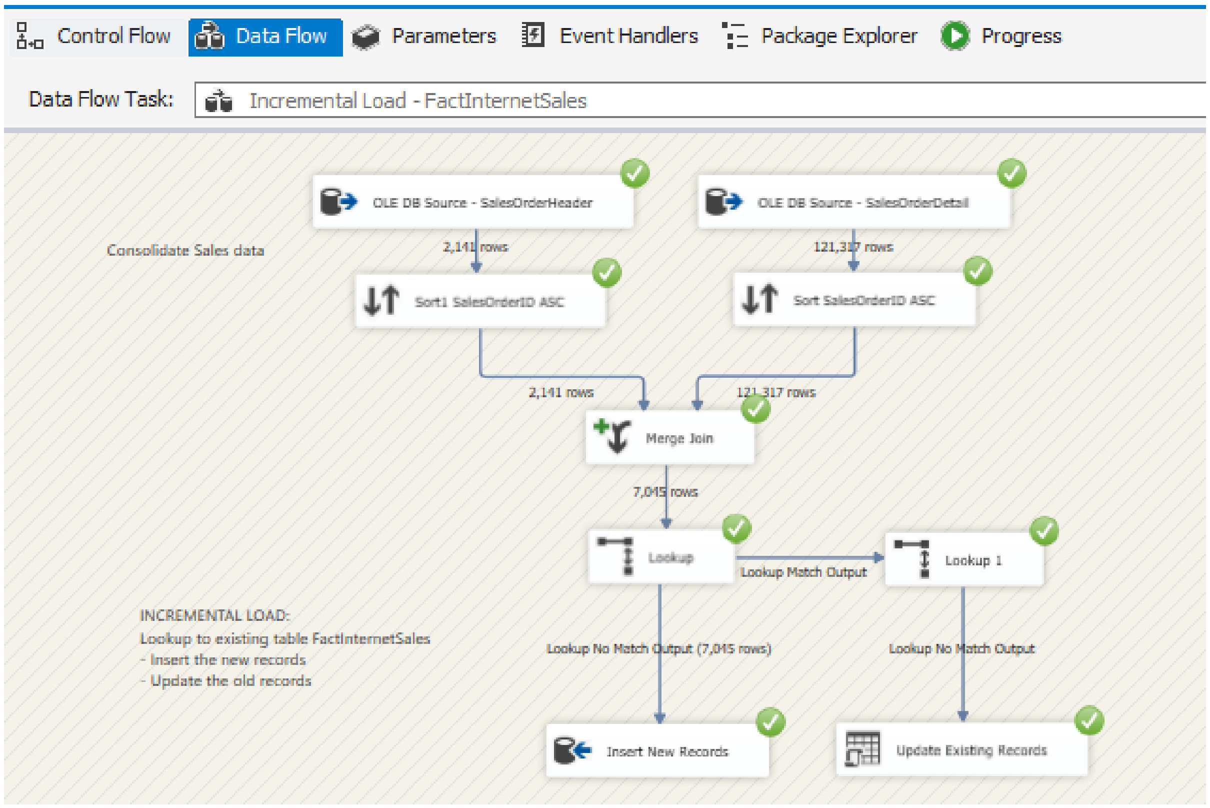
SSIS Implementation:


Control Flow
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
Data Flow Task:
Staging Area
OLE DB Source - SalesOrderHeader
2,141 rows
Data Conversion
2,141 rows
OLE DB Destination - StagingSalesOrderHeader 1
Batch based on OrderDate
OLE DB Source - SalesOrderDetail
121,317 rows
Data Conversion 1
121,317 rows
OLE DB Destination - StagingSalesOrderDetail 1
OLE DB Source - Employee
290 rows
OLE DB Destination - StagingEmployee
OLE DB Source -SalesPerson
17 rows
Data Conversion 4
17 rows
OLE DB Destination - StagingSalesPerson
OLE DB Source - Person
19,972 rows
OLE DB Destination - StagingPerson
OLE DB Source - Customer
19,820 rows
OLE DB Destination - StagingCustomer
OLE DB Source - EmailAddress
19,972 rows
OLE DB Destination - StagingEmailAddress
OLE DB Source - PersonPhone
19,972 rows
OLE DB Destination - StagingPersonPhone
Data Flow Task:
Staging Area
OLE DB Source - PersonPhone
19,972 rows
OLE DB Destination - StagingPersonPhone
OLE DB Source - Address
19,614 rows
OLE DB Destination - StagingAddress
OLE DB Source - AddressType
6 rows
OLE DB Destination - StagingAddressType
OLE DB Source - StageProvince
181 rows
OLE DB Destination - StagingStageProvince
OLE DB Source - BusinessEntityAddress
19,614 rows
OLE DB Destination - StagingBusinessEntityAddress
OLE DB Source - Product
Data Conversion 2
OLE DB Destination - StagingProduct
Batch based on - SellStartDate
OLE DB Source - ProductCostHistory
Data Conversion 3
OLE DB Destination - StagingProductCostHistory
Batch based on - StartDate
OLE DB Source - PurchaseOrderDetail
624 rows
OLE DB Destination - StagignPurchaseOrderDetail
Batch based on - DueDate

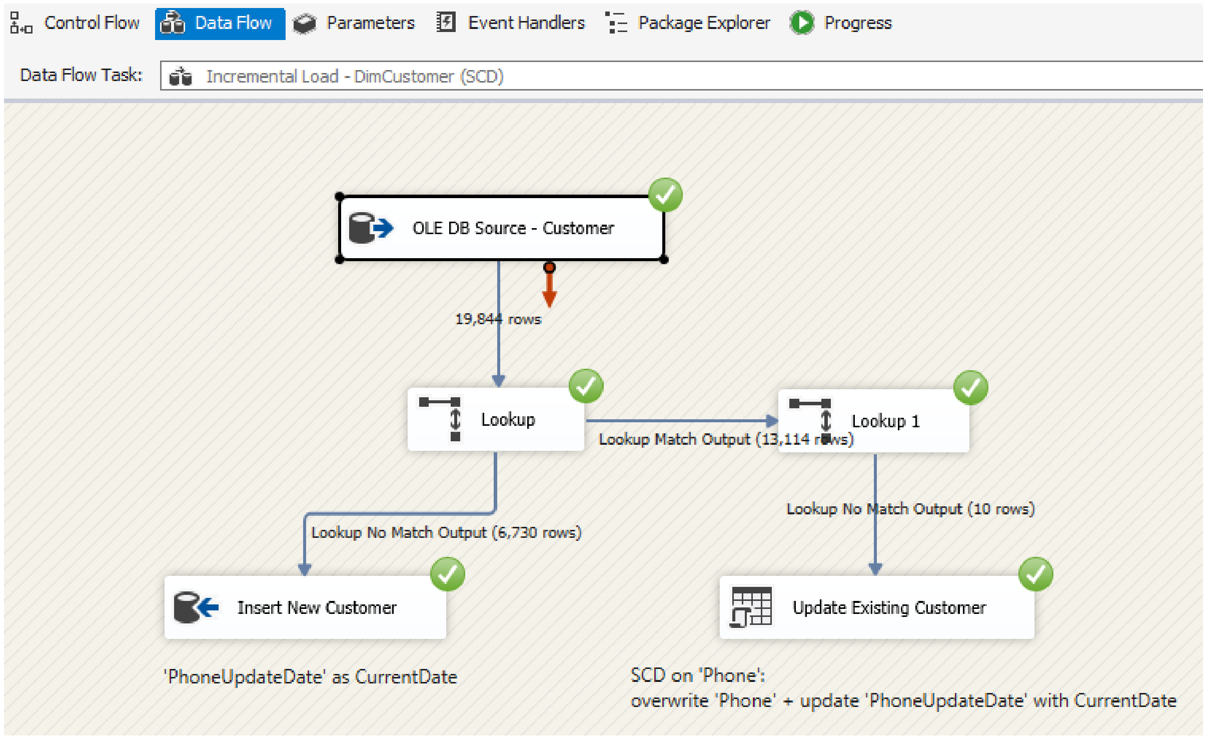
2) Populate Destination tables

FactInternetSales -

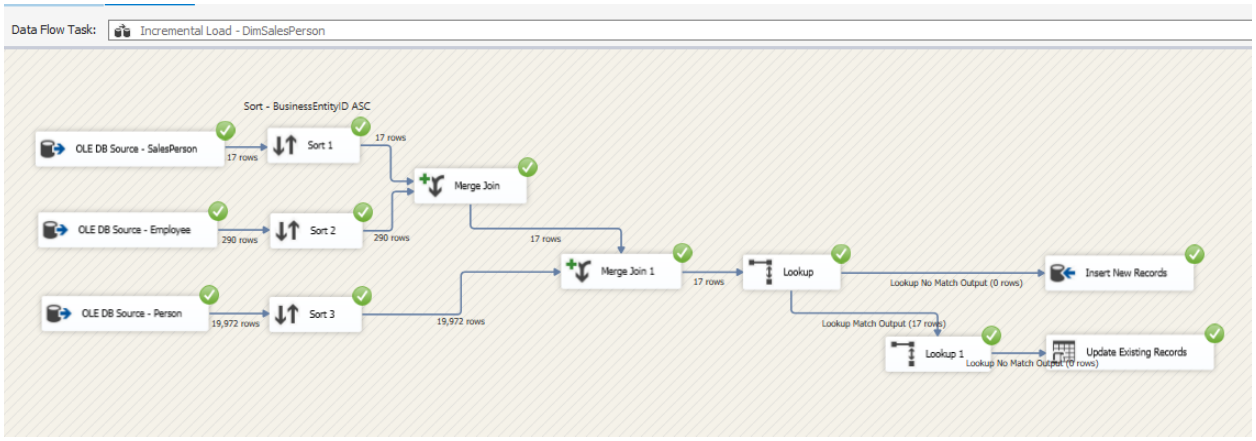
- Consolidate the data in two staging tables (SalesOrderDetail, SalesOrderHeader) to obtain the sales data on the current batch.
- Compare with the existing *FactInternetSales*, insert the new records and update the changing old records.



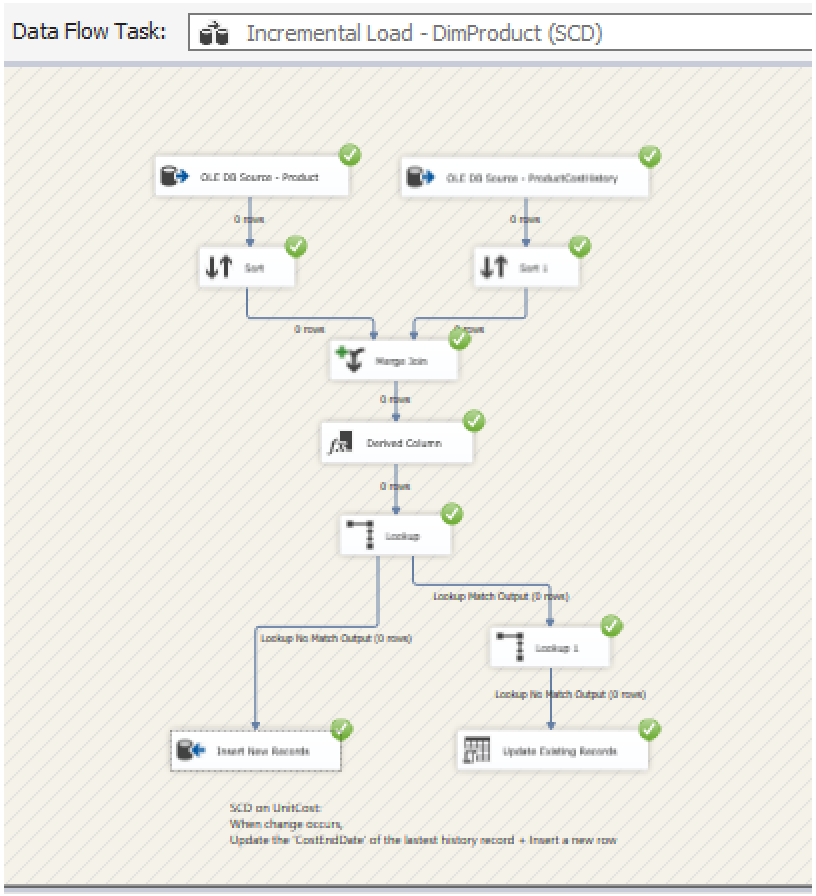
DimCustomer -
Extract data from staging tables (Customer, Person, EmailAddress, PersonPhone, BusinessEntityAddress, Address, AddressType, SateProvince and) to update or insert records into *DimCustomer*.


Control Flow
Data Flow
Parameters
Event Handlers
Package Explorer
Progress
Data Flow Task:
Incremental Load - DimCustomer (SCD)
OLE DB Source - Customer
19,844 rows
Lookup
Lookup Match Output (13,114 rows)
Lookup 1
Lookup No Match Output (10 rows)
Lookup No Match Output (6,730 rows)
Insert New Customer
Update Existing Customer
'PhoneUpdateDate' as CurrentDate
SCD on 'Phone':
overwrite 'Phone' + update 'PhoneUpdateDate' with CurrentDate

DimSalesPerson -
Use the data in three staging tables (Employee, SalesPerson and Person) to update or insert records into DimSalesPerson.



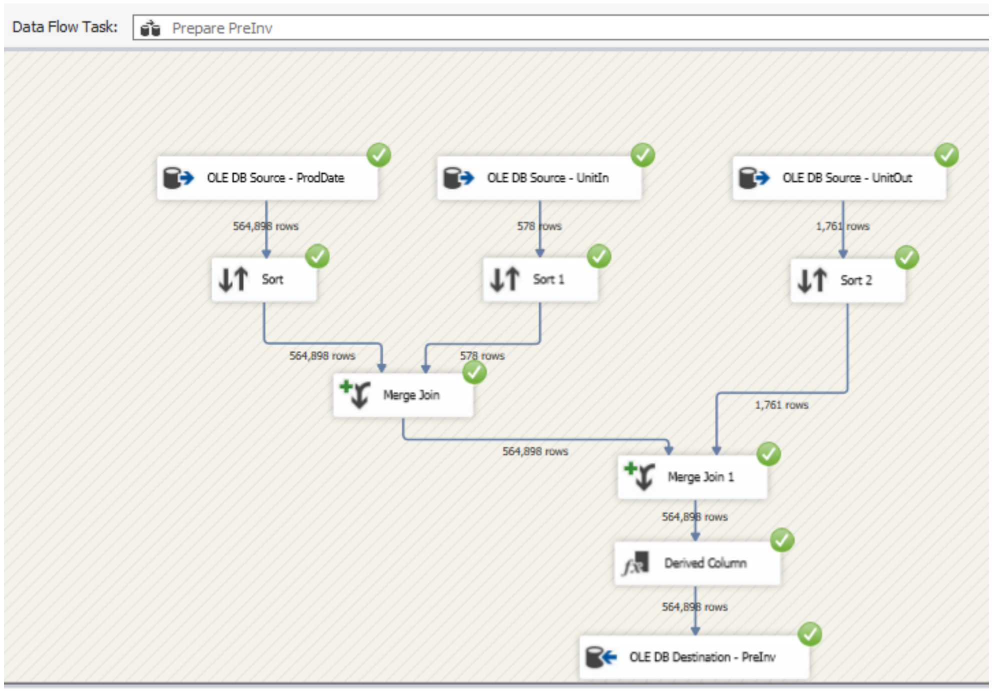
DimProduct -
Use the data in seven staging tables (Product, ProductCostHistory) to update or insert records into *DimProduct*.



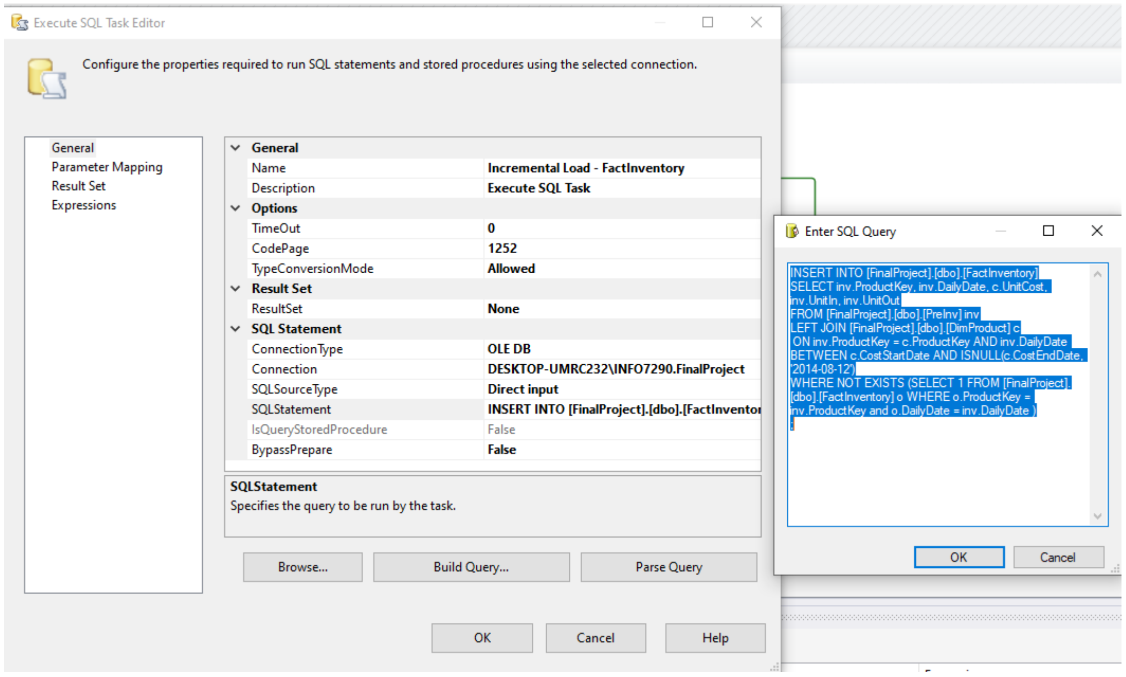
SCD on UnitCost:
When change occurs,
Update the 'CostEndDate' of the lastest history record + Insert a new row

FactProductInventory -

Step 1: Prepare PreInv - consolidated UnitIn and UnitOut data



Step 2: Join UnitCost data to update or insert records into *FactProductInventory*



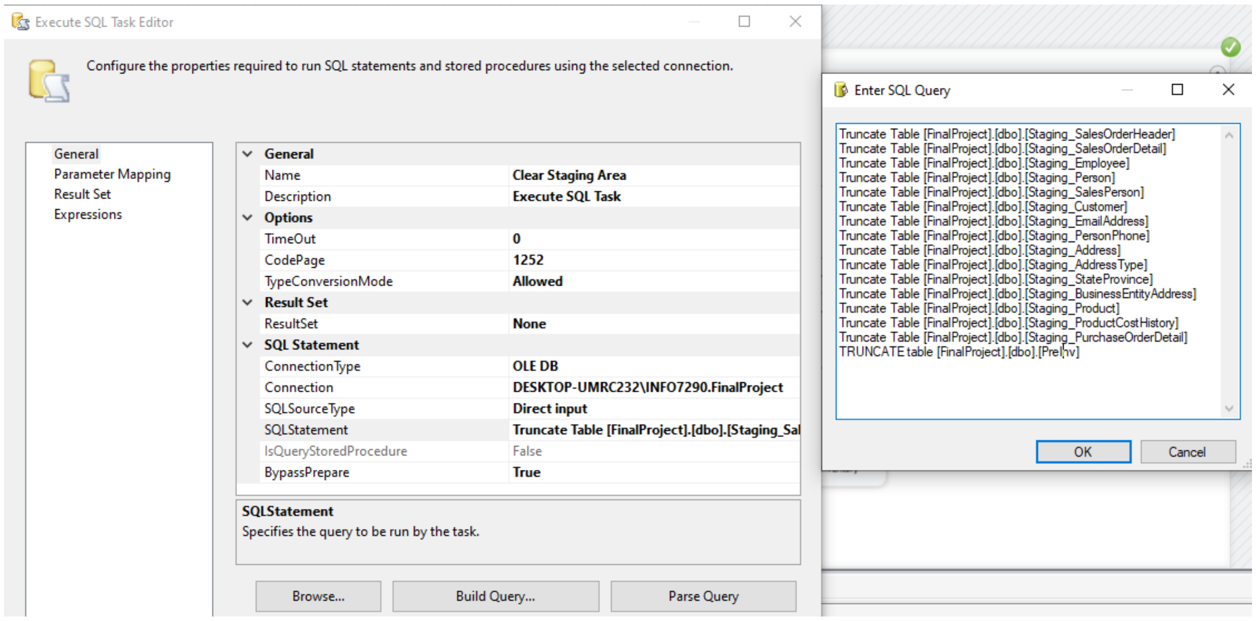
General	
Name	Incremental Load - FactInventory
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1252
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	DESKTOP-UMRC232\INFO7290.FinalProject
SQLSourceType	Direct input
SQLStatement	INSERT INTO [FinalProject].[dbo].[FactInventor
IsQueryStoredProcedure	False
BypassPrepare	False

SQLStatement
Specifies the query to be run by the task.

```
INSERT INTO [FinalProject].[dbo].[FactInventory]
SELECT inv.ProductKey, inv.DailyDate, c.UnitCost,
inv.UnitIn, inv.UnitOut
FROM [FinalProject].[dbo].[PreInv] inv
LEFT JOIN [FinalProject].[dbo].[DimProduct] c
 ON inv.ProductKey = c.ProductKey AND inv.DailyDate
BETWEEN c.CostStartDate AND ISNULL(c.CostEndDate,
'2014-08-12')
WHERE NOT EXISTS (SELECT 1 FROM [FinalProject].
[dbo].[FactInventory] o WHERE o.ProductKey =
inv.ProductKey and o.DailyDate = inv.DailyDate )
;
```

3) Clear Staging Tables

After the processing of one batch finished, we needed to clear all of the staging tables before the next round of processing.



Configure the properties required to run SQL statements and stored procedures using the selected connection.

General	
Name	**Clear Staging Area**
Description	**Execute SQL Task**
Options	
TimeOut	**0**
CodePage	**1252**
TypeConversionMode	**Allowed**
Result Set	
ResultSet	**None**
SQL Statement	
ConnectionType	**OLE DB**
Connection	**DESKTOP-UMRC232\INFO7290.FinalProject**
SQLSourceType	**Direct input**
SQLStatement	**Truncate Table [FinalProject].[dbo].[Staging_Sal**
IsQueryStoredProcedure	False
BypassPrepare	**True**

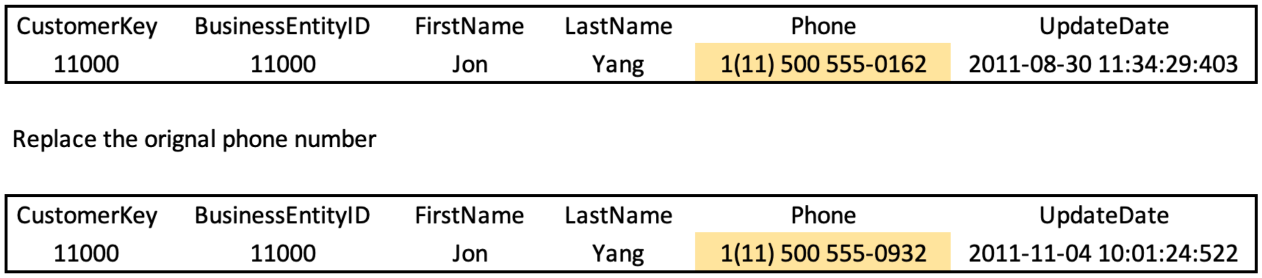
SQLStatement
Specifies the query to be run by the task.

```
Truncate Table [FinalProject].[dbo].[Staging_SalesOrderHeader]
Truncate Table [FinalProject].[dbo].[Staging_SalesOrderDetail]
Truncate Table [FinalProject].[dbo].[Staging_Employee]
Truncate Table [FinalProject].[dbo].[Staging_Person]
Truncate Table [FinalProject].[dbo].[Staging_SalesPerson]
Truncate Table [FinalProject].[dbo].[Staging_Customer]
Truncate Table [FinalProject].[dbo].[Staging_EmailAddress]
Truncate Table [FinalProject].[dbo].[Staging_PersonPhone]
Truncate Table [FinalProject].[dbo].[Staging_Address]
Truncate Table [FinalProject].[dbo].[Staging_AddressType]
Truncate Table [FinalProject].[dbo].[Staging_StateProvince]
Truncate Table [FinalProject].[dbo].[Staging_BusinessEntityAddress]
Truncate Table [FinalProject].[dbo].[Staging_Product]
Truncate Table [FinalProject].[dbo].[Staging_ProductCostHistory]
Truncate Table [FinalProject].[dbo].[Staging_PurchaseOrderDetail]
TRUNCATE table [FinalProject].[dbo].[PreInv]
```

DESIGN OF SCD

1. Customer phone number - carry an UpdateDate

In the customer dimension table [*DimCustomer*], the new phone number overwrites the existing data. This eases dimension updates and limits the change of the dimension table to only new records. The date of updating phone numbers will be also stored in the table to keep track of the changes. The historical value of the phone number is lost as it is not storing anywhere. Therefore, the dimension table will always contain only the current value since it is pointless to contact a customer with an obsolete phone number. An example is shown in the figure below.



CustomerKey	BusinessEntityID	FirstName	LastName	Phone	UpdateDate
11000	11000	Jon	Yang	1(11) 500 555-0162	2011-08-30 11:34:29:403

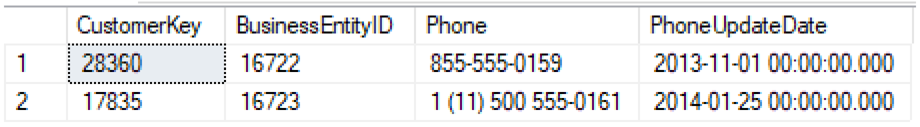
Replace the orignal phone number

CustomerKey	BusinessEntityID	FirstName	LastName	Phone	UpdateDate
11000	11000	Jon	Yang	1(11) 500 555-0932	2011-11-04 10:01:24:522

Figure 9: SCD of *DimCustomer* table

SSIS Test Case:

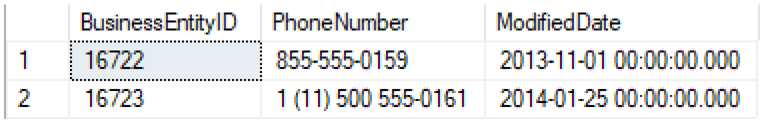
- Original records in *DimCuster*



	CustomerKey	BusinessEntityID	Phone	PhoneUpdateDate
1	28360	16722	855-555-0159	2013-11-01 00:00:00.000
2	17835	16723	1 (11) 500 555-0161	2014-01-25 00:00:00.000

- Update phone number in *Staging_PersonPhone*

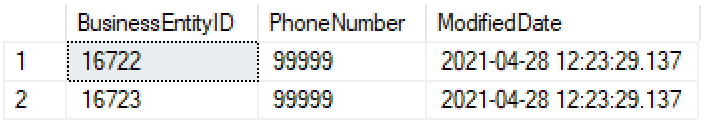
Original:



	BusinessEntityID	PhoneNumber	ModifiedDate
1	16722	855-555-0159	2013-11-01 00:00:00.000
2	16723	1 (11) 500 555-0161	2014-01-25 00:00:00.000

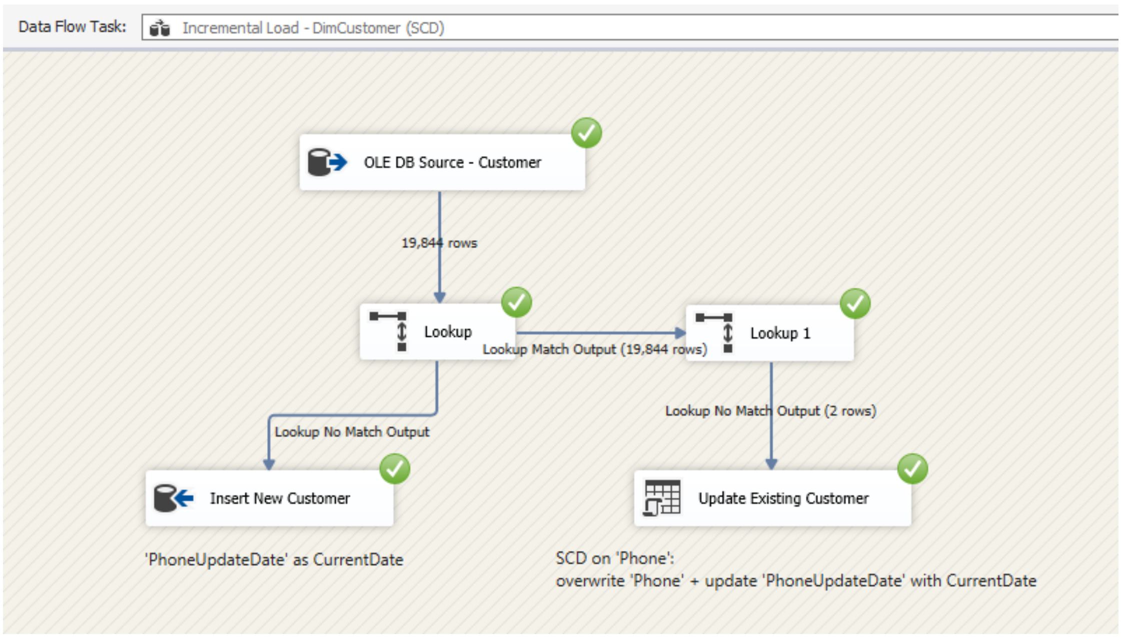
Update PhoneNumber + ModifiedDate:

```
UPDATE [FinalProject].[dbo].[Staging_PersonPhone]
SET PhoneNumber = '99999', ModifiedDate = GETDATE()
WHERE BusinessEntityID = 16722 OR BusinessEntityID = 16723
```

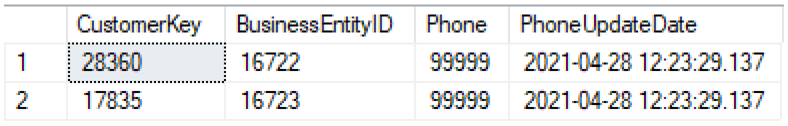


	BusinessEntityID	PhoneNumber	ModifiedDate
1	16722	99999	2021-04-28 12:23:29.137
2	16723	99999	2021-04-28 12:23:29.137

- Run SSIS Task ‘Incremental load - DimCustomer’


Data Flow Task:
Incremental Load - DimCustomer (SCD)
OLE DB Source - Customer
19,844 rows
Lookup
Lookup Match Output (19,844 rows)
Lookup 1
Lookup No Match Output
Lookup No Match Output (2 rows)
Insert New Customer
Update Existing Customer
'PhoneUpdateDate' as CurrentDate
SCD on 'Phone':
overwrite 'Phone' + update 'PhoneUpdateDate' with CurrentDate

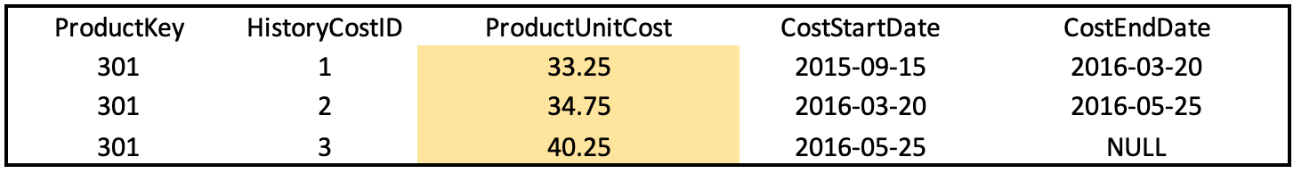
- Check the new records in *DimCustomer*



	CustomerKey	BusinessEntityID	Phone	PhoneUpdateDate
1	28360	16722	99999	2021-04-28 12:23:29.137
2	17835	16723	99999	2021-04-28 12:23:29.137

2. Product unit cost - add new rows in DimProduct table to store all history changes

In the product dimension table [*DimProuduct*], the new unit cost will be added to the table within a new row. The table retains the full history of product unit cost values: the current record is closed then the value of ProductUnitCost changes. An example is shown in the figure below: the CostStartDate of the second row is equal to the CostEndDate of the previous row. The null CostEndDate in row three indicated the current record version. The HistoryCostID is a surrogate key to show different version numbers. Since we regard the fluctuations in the unit cost as a significant indicator, it is necessary to keep all the records in a historical table in case of tracing back.

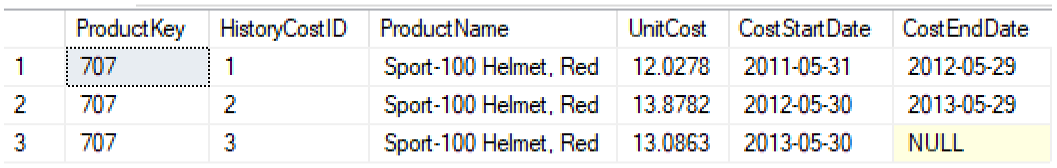


ProductKey	HistoryCostID	ProductUnitCost	CostStartDate	CostEndDate
301	1	33.25	2015-09-15	2016-03-20
301	2	34.75	2016-03-20	2016-05-25
301	3	40.25	2016-05-25	NULL

Figure 10: SCD of *DimProductInventory* table

SSIS Test Case:

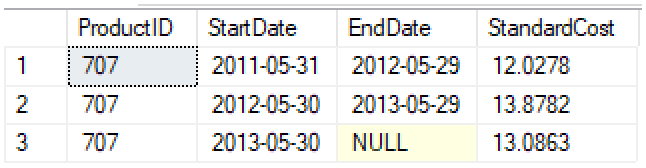
- Original records in *DimProduct*



	ProductKey	HistoryCostID	ProductName	UnitCost	CostStartDate	CostEndDate
1	707	1	Sport-100 Helmet, Red	12.0278	2011-05-31	2012-05-29
2	707	2	Sport-100 Helmet, Red	13.8782	2012-05-30	2013-05-29
3	707	3	Sport-100 Helmet, Red	13.0863	2013-05-30	NULL

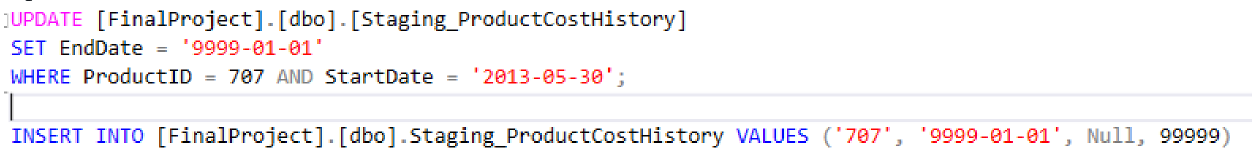
- Add a new record in *Staging_ProductCostHistory*

Original:



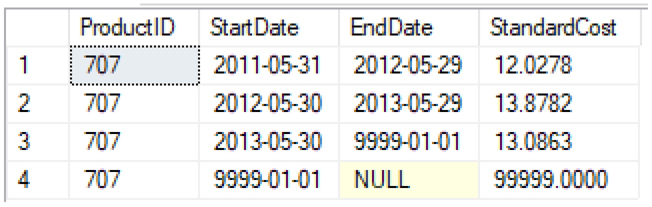
	ProductID	StartDate	EndDate	StandardCost
1	707	2011-05-31	2012-05-29	12.0278
2	707	2012-05-30	2013-05-29	13.8782
3	707	2013-05-30	NULL	13.0863

Update PhoneNumber + ModifiedDate:



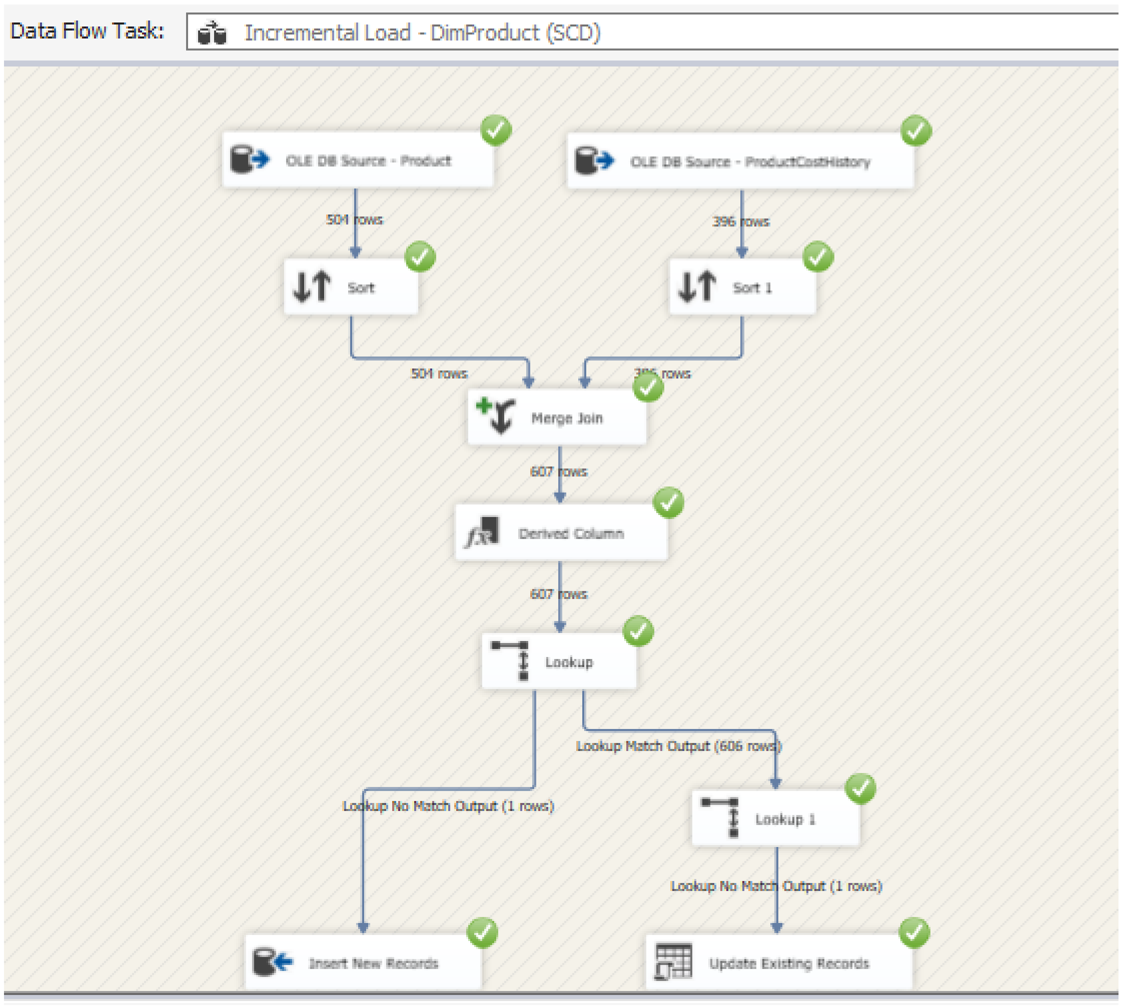
```
UPDATE [FinalProject].[dbo].[Staging_ProductCostHistory]
SET EndDate = '9999-01-01'
WHERE ProductID = 707 AND StartDate = '2013-05-30';

INSERT INTO [FinalProject].[dbo].Staging_ProductCostHistory VALUES ('707', '9999-01-01', Null, 99999)
```

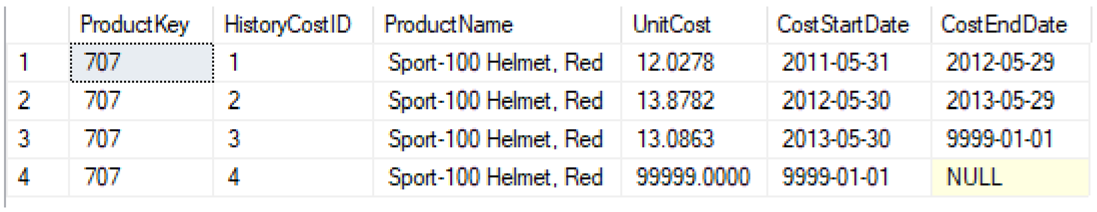


	ProductID	StartDate	EndDate	StandardCost
1	707	2011-05-31	2012-05-29	12.0278
2	707	2012-05-30	2013-05-29	13.8782
3	707	2013-05-30	9999-01-01	13.0863
4	707	9999-01-01	NULL	99999.0000

- Run SSIS Task ‘Incremental load - DimCustomer’



- Check the new records in *DimProduct*



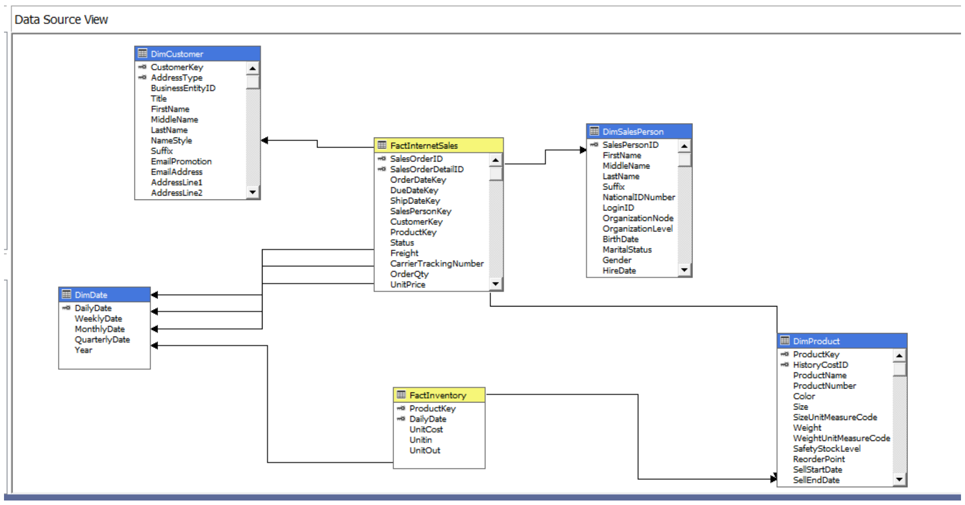
	ProductKey	HistoryCostID	ProductName	UnitCost	CostStartDate	CostEndDate
1	707	1	Sport-100 Helmet, Red	12.0278	2011-05-31	2012-05-29
2	707	2	Sport-100 Helmet, Red	13.8782	2012-05-30	2013-05-29
3	707	3	Sport-100 Helmet, Red	13.0863	2013-05-30	9999-01-01
4	707	4	Sport-100 Helmet, Red	99999.0000	9999-01-01	NULL

DESIGN OF DIMDATE

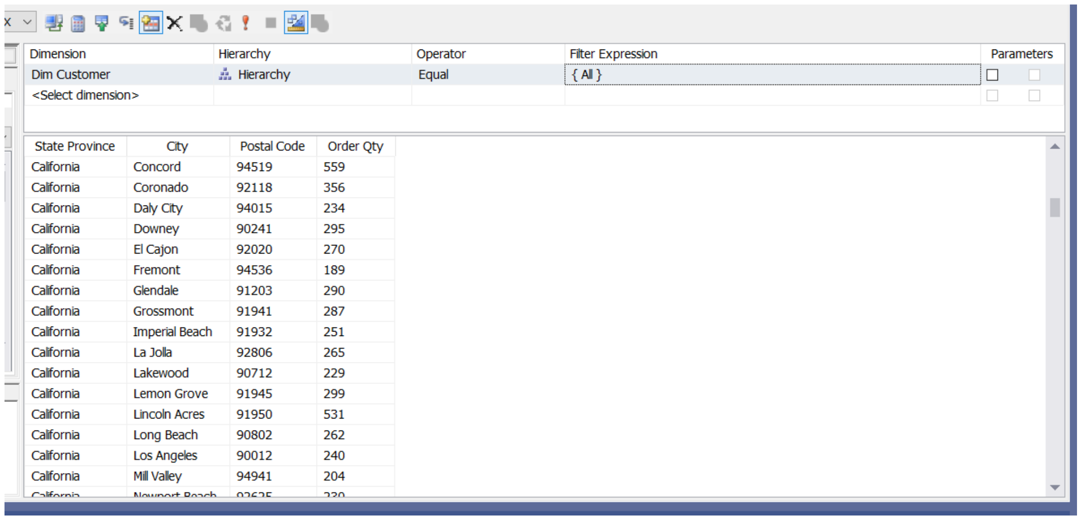
1. Find the date range in the Database AdventureWorks2019
2. Create a DimDate table which contains every consecutive day in the date range and let the date column be the primary key of DimDate table
3. Generate weekly date (first day of each week), monthly date (first day of each month), quarterly date (first day of each quarter), and year columns according to the date column, making it easy to generate weekly, monthly, quarterly or yearly reports later on.

OLAP MODEL

An olap model was constructed using sql server analysis service. A cube was created and deployed with 4 dimension tables and 2 fact tables. In the dimension tables, several hierarchies were built for better analysis of sales performance and product inventory. For example, calendar date hierarchy, goes from year to a quarter, to month, and ends with day, which enables analyzing sales in different time ranges.



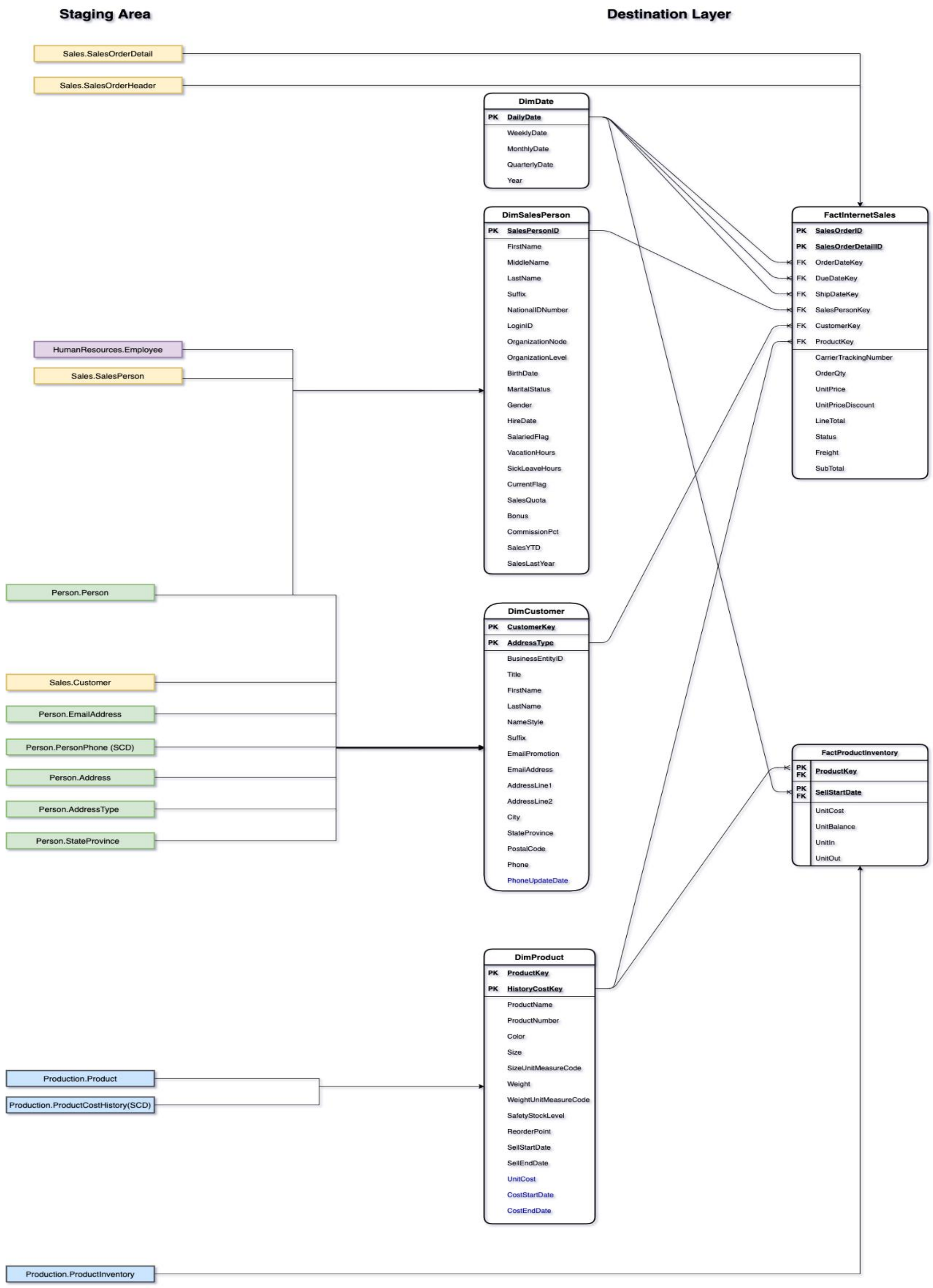
Geography hierarchy, which goes from state to city, and ending with postal code. As the figure illustrates, data is aggregated from the most detailed level of data to a higher level.



Dimension	Hierarchy	Operator	Filter Expression	Parameters
Dim Customer	Hierarchy	Equal	{ All }	☐
<Select dimension>				

State Province	City	Postal Code	Order Qty
California	Concord	94519	559
California	Coronado	92118	356
California	Daly City	94015	234
California	Downey	90241	295
California	El Cajon	92020	270
California	Fremont	94536	189
California	Glendale	91203	290
California	Grossmont	91941	287
California	Imperial Beach	91932	251
California	La Jolla	92806	265
California	Lakewood	90712	229
California	Lemon Grove	91945	299
California	Lincoln Acres	91950	531
California	Long Beach	90802	262
California	Los Angeles	90012	240
California	Mill Valley	94941	204
California	Newport Beach	92625	230

APPENDIX A: DATA FLOW CHART (DATA FLOW CHART)



Staging Area
Destination Layer
Sales.SalesOrderDetail
Sales.SalesOrderHeader
DimDate
PK DailyDate
WeeklyDate
MonthlyDate
QuarterlyDate
Year
DimSalesPerson
PK SalesPersonID
FirstName
MiddleName
LastName
Suffix
NationalIDNumber
LoginID
OrganizationNode
OrganizationLevel
BirthDate
MaritalStatus
Gender
HireDate
SalariedFlag
VacationHours
SickLeaveHours
CurrentFlag
SalesQuota
Bonus
CommissionPct
SalesYTD
SalesLastYear
FactInternetSales
PK SalesOrderID
PK SalesOrderDetailID
FK OrderDateKey
FK DueDateKey
FK ShipDateKey
FK SalesPersonKey
FK CustomerKey
FK ProductKey
CarrierTrackingNumber
OrderQty
UnitPrice
UnitPriceDiscount
LineTotal
Status
Freight
SubTotal
HumanResources.Employee
Sales.SalesPerson
Person.Person
Sales.Customer
Person.EmailAddress
Person.PersonPhone (SCD)
Person.Address
Person.AddressType
Person.StateProvince
DimCustomer
PK CustomerKey
PK AddressType
BusinessEntityID
Title
FirstName
LastName
NameStyle
Suffix
EmailPromotion
EmailAddress
AddressLine1
AddressLine2
City
StateProvince
PostalCode
Phone
PhoneUpdateDate
FactProductInventory
PK FK ProductKey
PK FK SellStartDate
UnitCost
UnitBalance
UnitIn
UnitOut
DimProduct
PK ProductKey
PK HistoryCostKey
ProductName
ProductNumber
Color
Size
SizeUnitMeasureCode
Weight
WeightUnitMeasureCode
SafetyStockLevel
ReorderPoint
SellStartDate
SellEndDate
UnitCost
CostStartDate
CostEndDate
Production.Product
Production.ProductCostHistory(SCD)
Production.ProductInventory