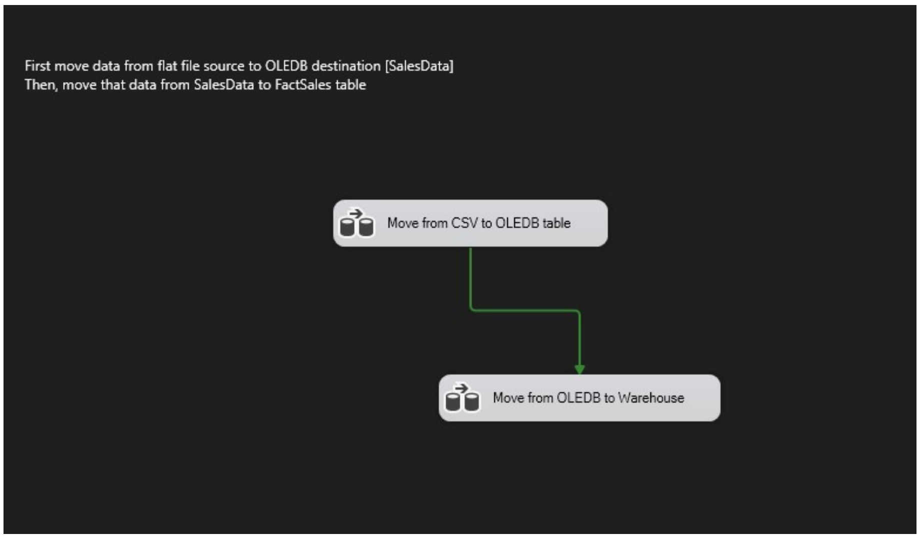
Assignment 9 - SSIS

Task1: Integration with ETL warehouse

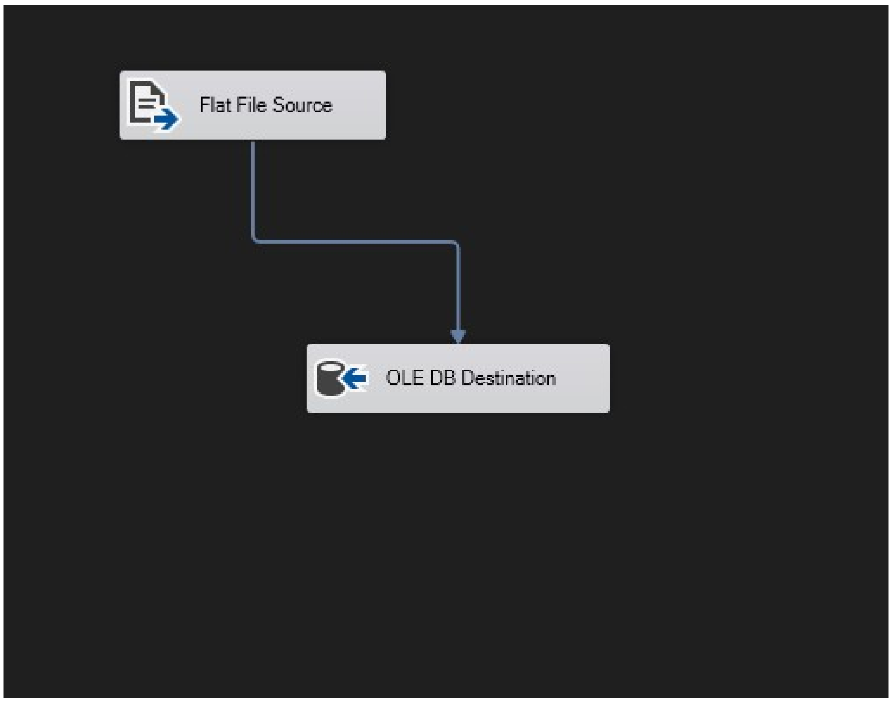
a. main control flow task



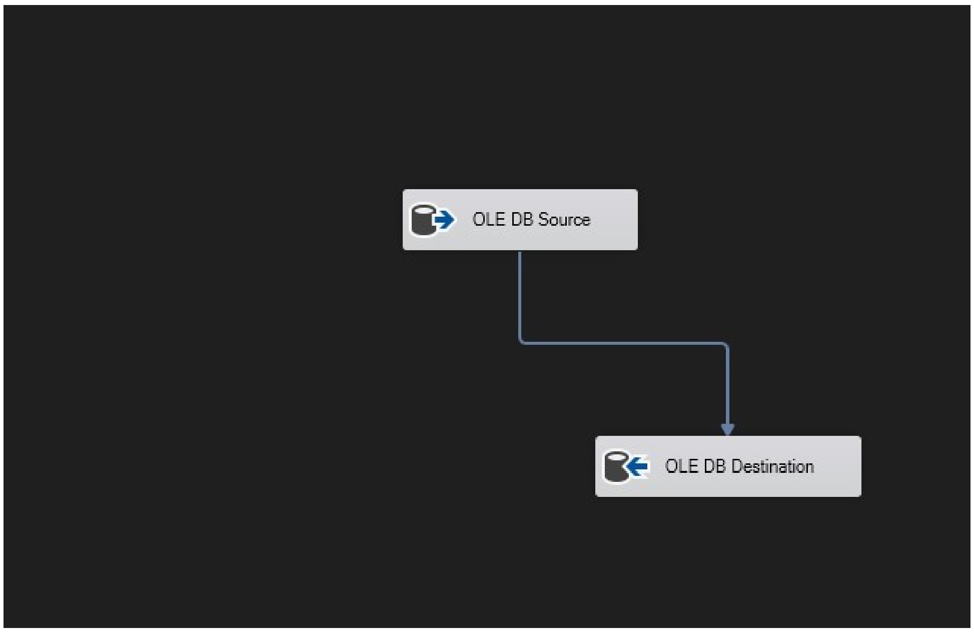
First move data from flat file source to OLEDB destination [SalesData]
Then, move that data from SalesData to FactSales table

b. data flow task to move data from a flatfile source to initial OLEDB destination [SalesData table]

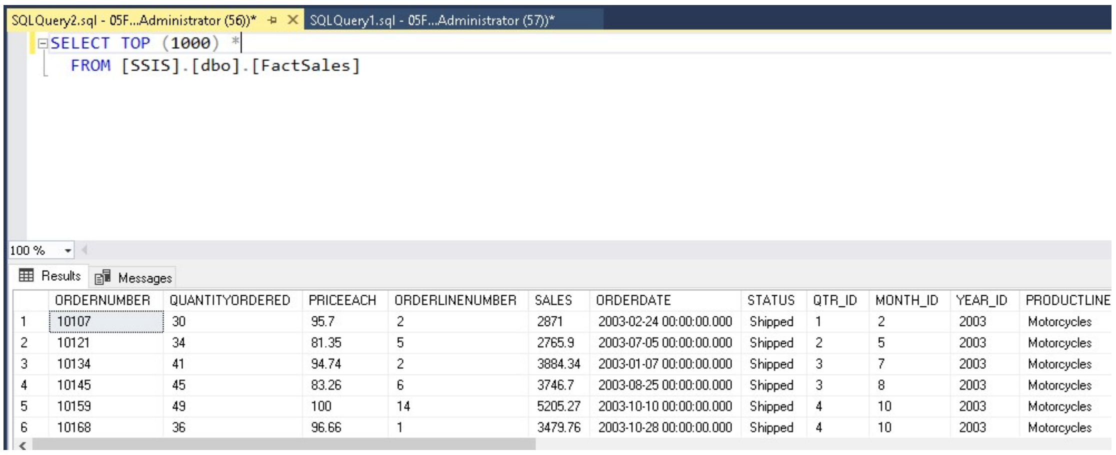
Datatype conversion is done from the Advanced section of Connection manager for Flat File Source



c. data flow task to move data from SalesData table to another OLEDB destination [FactSales table]



d. Final data in FactSales table

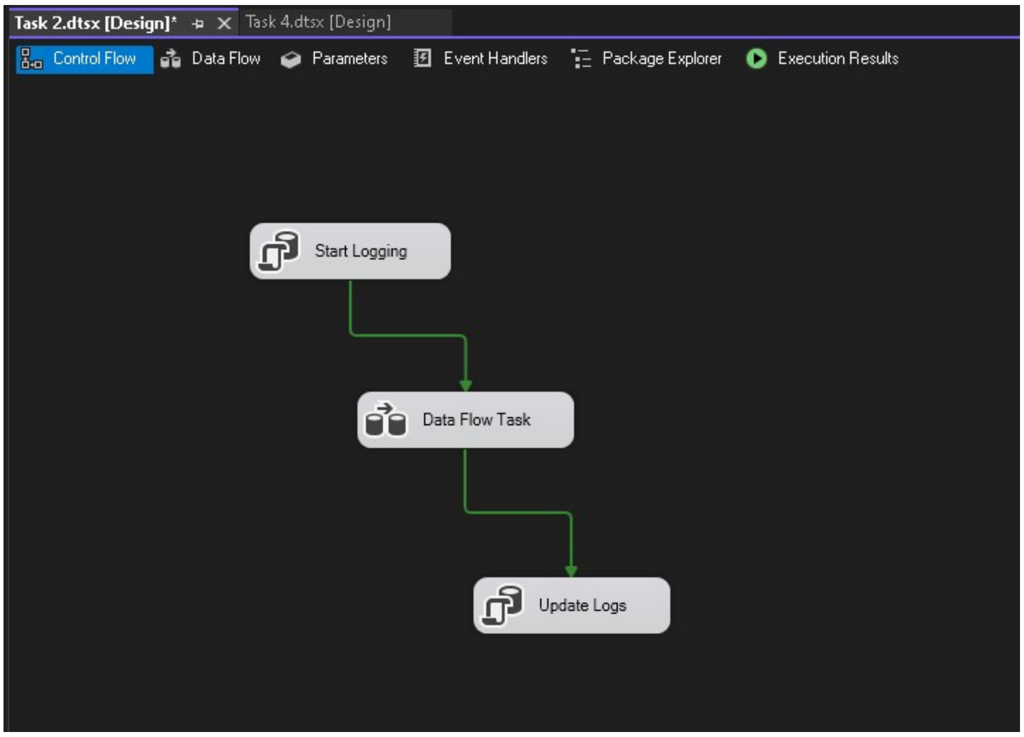

SQLQuery2.sql - 05F...Administrator (56))* SQLQuery1.sql - 05F...Administrator (57))*

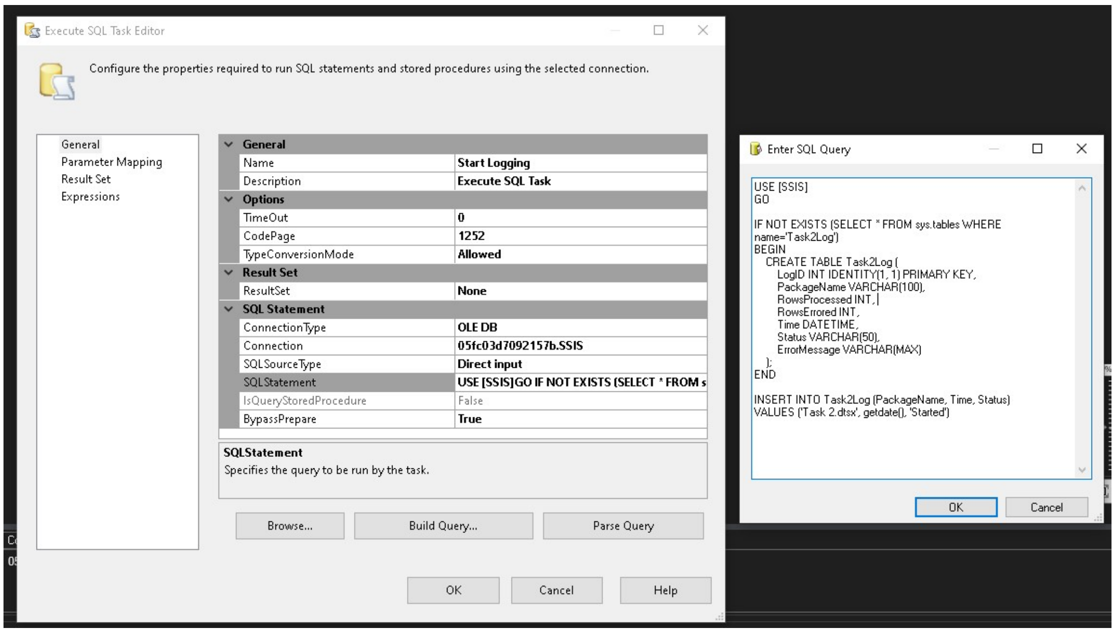
```
SELECT TOP (1000) *
  FROM [SSIS].[dbo].[FactSales]
```

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	STATUS	QTR_ID	MONTH_ID	YEAR_ID	PRODUCTLINE
1	10107	30	95.7	2	2871	2003-02-24 00:00:00.000	Shipped	1	2	2003	Motorcycles
2	10121	34	81.35	5	2765.9	2003-07-05 00:00:00.000	Shipped	2	5	2003	Motorcycles
3	10134	41	94.74	2	3884.34	2003-01-07 00:00:00.000	Shipped	3	7	2003	Motorcycles
4	10145	45	83.26	6	3746.7	2003-08-25 00:00:00.000	Shipped	3	8	2003	Motorcycles
5	10159	49	100	14	5205.27	2003-10-10 00:00:00.000	Shipped	4	10	2003	Motorcycles
6	10168	36	96.66	1	3479.76	2003-10-28 00:00:00.000	Shipped	4	10	2003	Motorcycles

Task2: Data warehouse migration

a. dataflow of the task. Start logging is an SQL script that creates the log table and iniitalizes a record whenever the control flow is run.





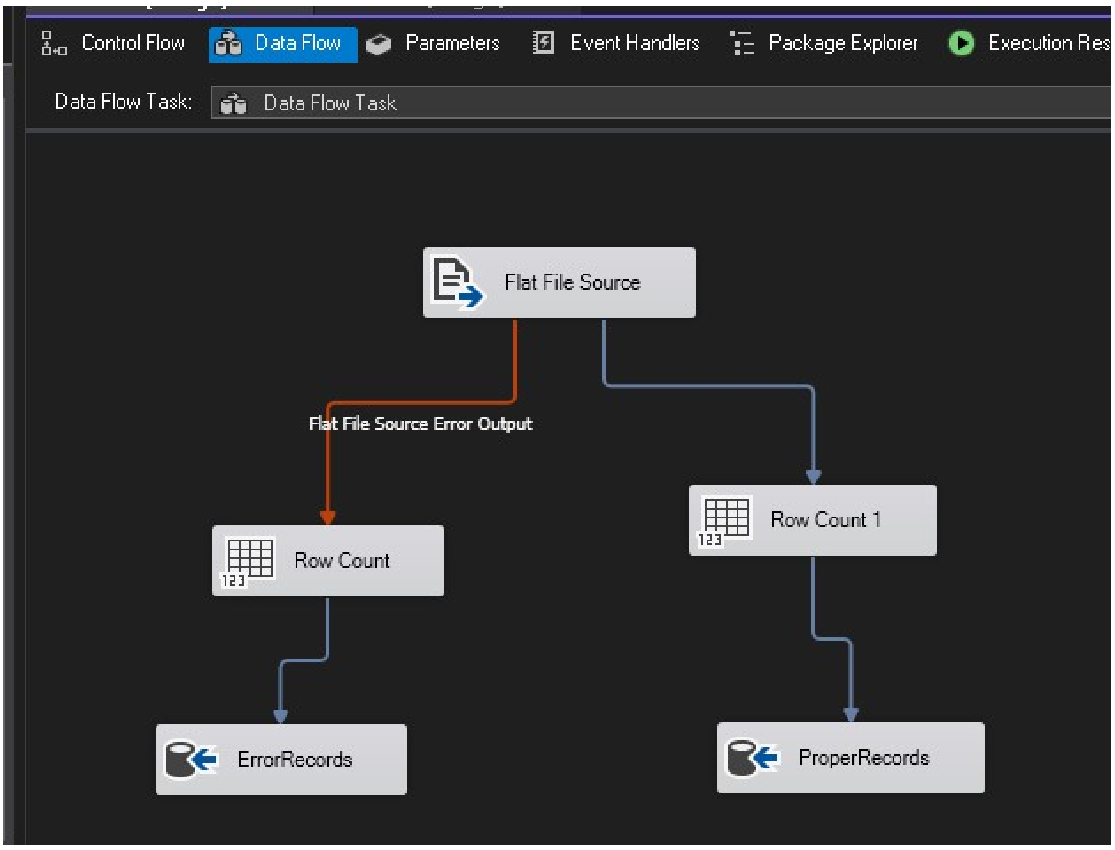
Configure the properties required to run SQL statements and stored procedures using the selected connection.

General	
Name	**Start Logging**
Description	**Execute SQL Task**
Options	
TimeOut	**0**
CodePage	**1252**
TypeConversionMode	**Allowed**
Result Set	
ResultSet	**None**
SQL Statement	
ConnectionType	**OLE DB**
Connection	**05fc03d7092157b.SSIS**
SQLSourceType	**Direct input**
SQLStatement	**USE [SSIS]GO IF NOT EXISTS (SELECT * FROM s**
IsQueryStoredProcedure	False
BypassPrepare	**True**

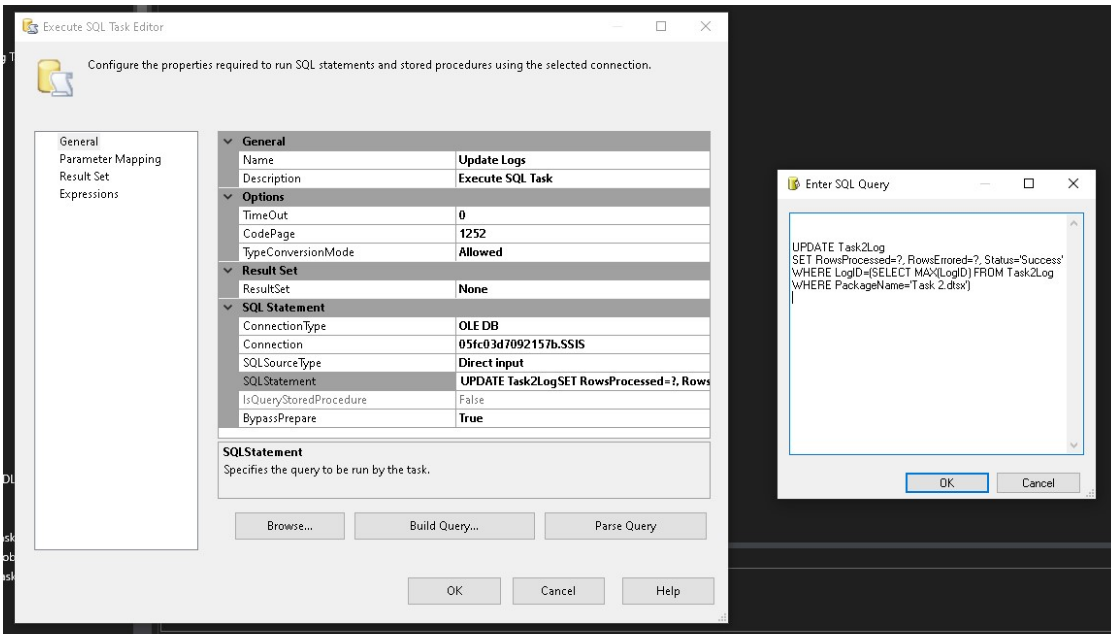
SQLStatement
Specifies the query to be run by the task.

```
USE [SSIS]
GO

IF NOT EXISTS (SELECT * FROM sys.tables WHERE
name='Task2Log')
BEGIN
    CREATE TABLE Task2Log (
        LogID INT IDENTITY(1, 1) PRIMARY KEY,
        PackageName VARCHAR(100),
        RowsProcessed INT,
        RowsErrored INT,
        Time DATETIME,
        Status VARCHAR(50),
        ErrorMessage VARCHAR(MAX)
    );
END

INSERT INTO Task2Log (PackageName, Time, Status)
VALUES ('Task 2.dtsx', getdate(), 'Started')
```

b. control flow of the task - rows are read from the CSV file and proceed along right arm if there are no issues - row count is taken. if there are any issues they are directed into the Error Records section which logs the errros.



c. Update logs is an SQL script that will save the logs into an SQL table



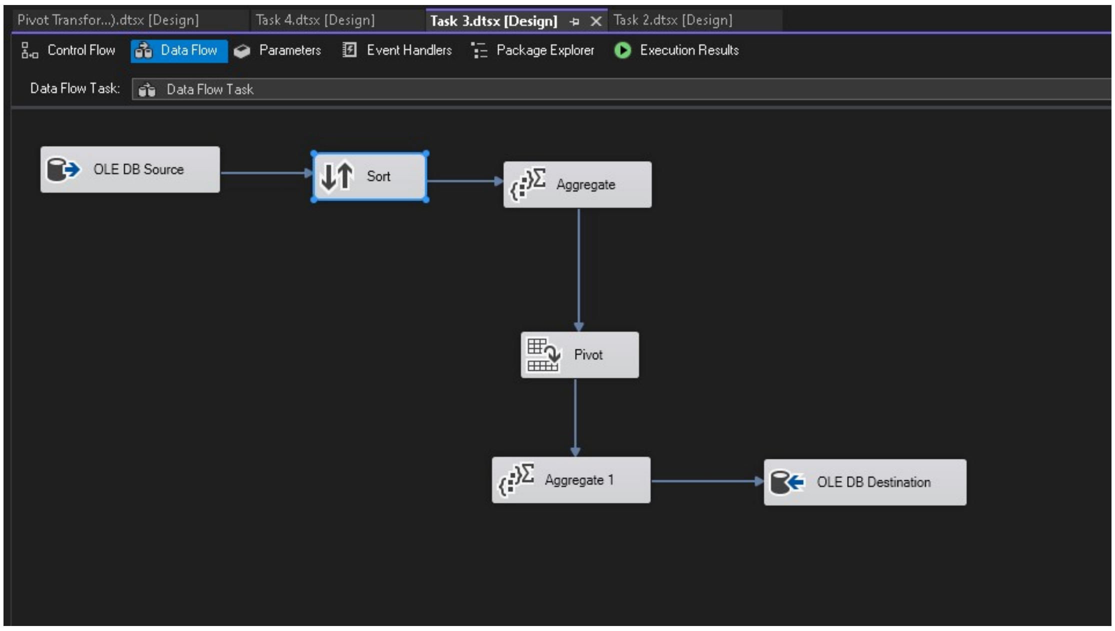
General	
Name	Update Logs
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1252
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	05fc03d7092157b.SSIS
SQLSourceType	Direct input
SQLStatement	UPDATE Task2LogSET RowsProcessed=?, Rows
IsQueryStoredProcedure	False
BypassPrepare	True

SQLStatement
Specifies the query to be run by the task.

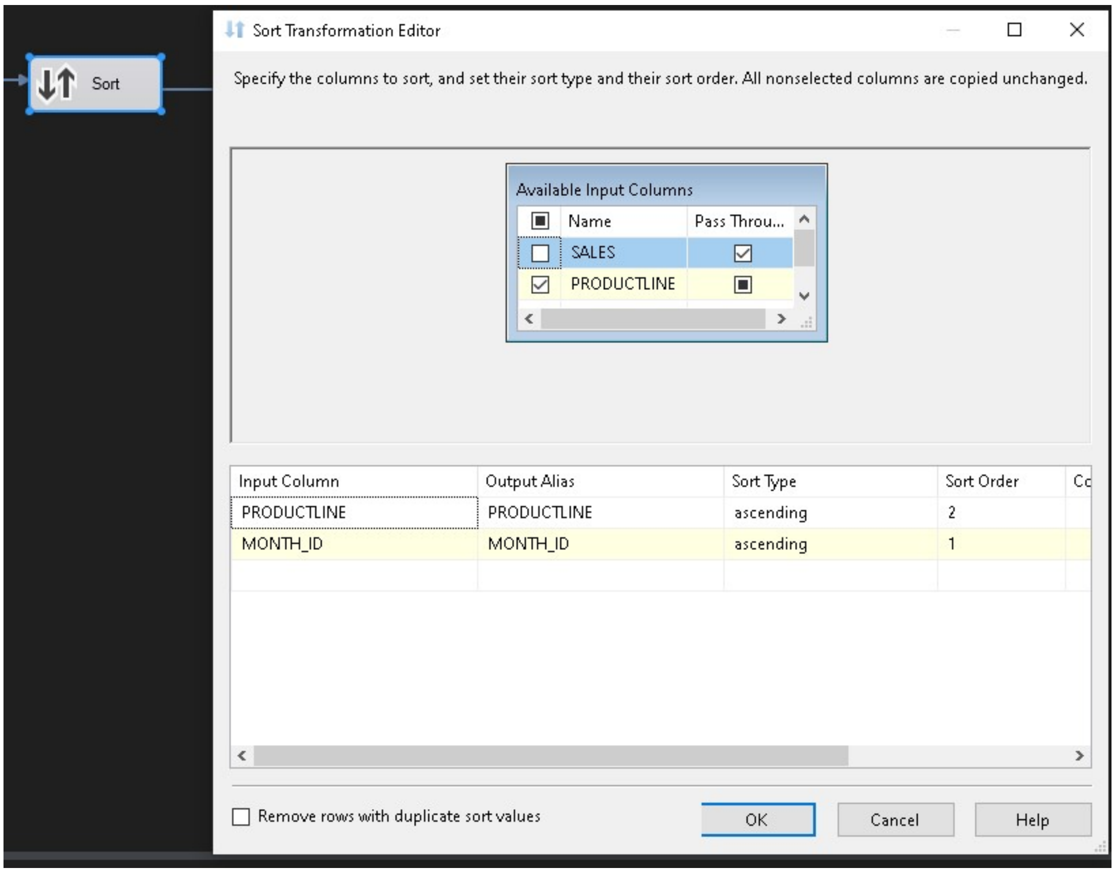
```
UPDATE Task2Log
SET RowsProcessed=?, RowsErrored=?, Status='Success'
WHERE LogID=(SELECT MAX(LogID) FROM Task2Log
WHERE PackageName='Task 2.dtsx')
```

Task3: Implementing pivot Transformation

a. dataflow of the task, source is SalesData table from which PRODUCTLINE, MONTH_ID and SALES columns are selected



b. data is sorted by MONTH_ID, PRODUCTLINE both in ascending order


Sort Transformation Editor

Specify the columns to sort, and set their sort type and their sort order. All nonselected columns are copied unchanged.

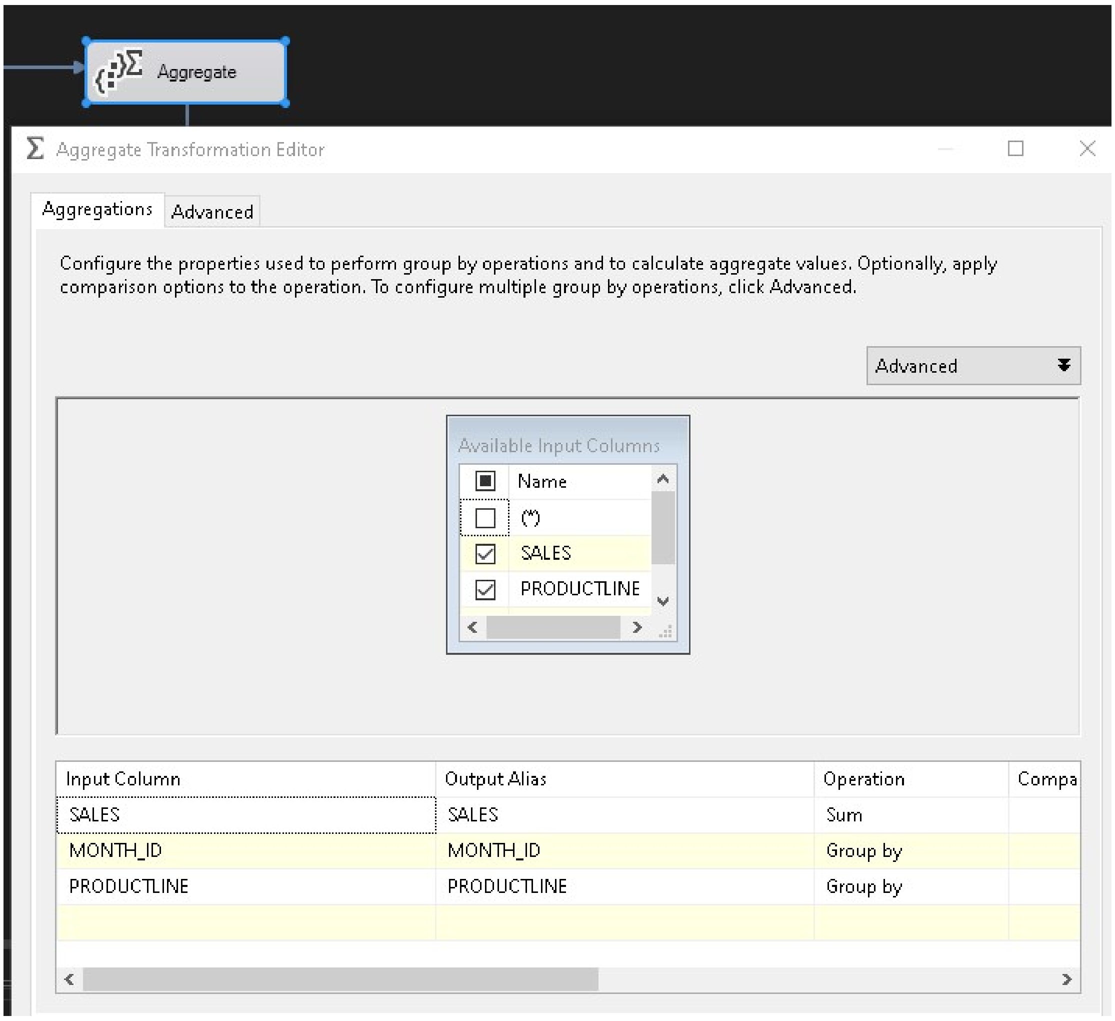
Available Input Columns

	Name	Pass Throu...
☐	SALES	✓
✓	PRODUCTLINE	■

Input Column	Output Alias	Sort Type	Sort Order	Cc
PRODUCTLINE	PRODUCTLINE	ascending	2	
MONTH_ID	MONTH_ID	ascending	1	

☐ Remove rows with duplicate sort values

c. aggregation is done to group rows by MONTH_ID and PRODUCTLINE, and find sum of SALES for grouped data



Configure the properties used to perform group by operations and to calculate aggregate values. Optionally, apply comparison options to the operation. To configure multiple group by operations, click Advanced.

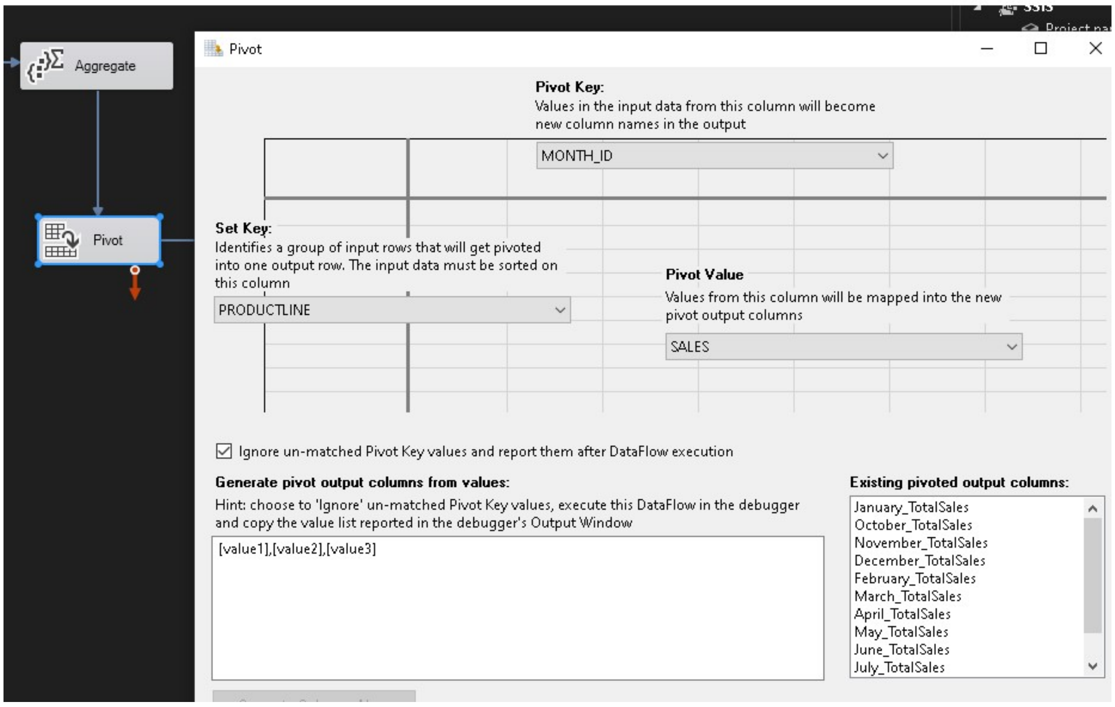
Advanced

Available Input Columns

	Name
☐	(*)
☑	SALES
☑	PRODUCTLINE

Input Column	Output Alias	Operation	Compa
SALES	SALES	Sum	
MONTH_ID	MONTH_ID	Group by	
PRODUCTLINE	PRODUCTLINE	Group by	

d. pivot transformation is done with MONTH_ID as pivot key and PRODUCTLINE as set key



Pivot

Pivot Key:
Values in the input data from this column will become new column names in the output

MONTH_ID

Set Key:
Identifies a group of input rows that will get pivoted into one output row. The input data must be sorted on this column

PRODUCTLINE

Pivot Value
Values from this column will be mapped into the new pivot output columns

SALES

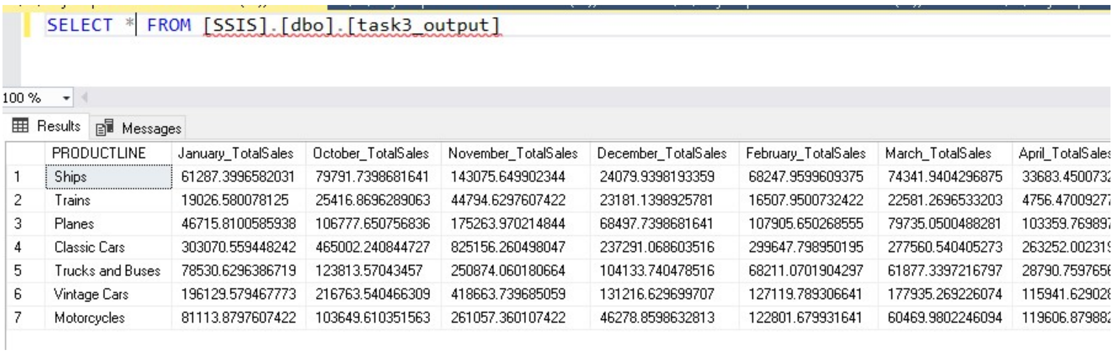
☑ Ignore un-matched Pivot Key values and report them after DataFlow execution

Generate pivot output columns from values:
Hint: choose to 'Ignore' un-matched Pivot Key values, execute this DataFlow in the debugger and copy the value list reported in the debugger's Output Window

[value1],[value2],[value3]

Existing pivoted output columns:
January_TotalSales
October_TotalSales
November_TotalSales
December_TotalSales
February_TotalSales
March_TotalSales
April_TotalSales
May_TotalSales
June_TotalSales
July_TotalSales

e. pivoted data saved to an SQL table



```
SELECT * FROM [SSIS].[dbo].[task3_output]
```

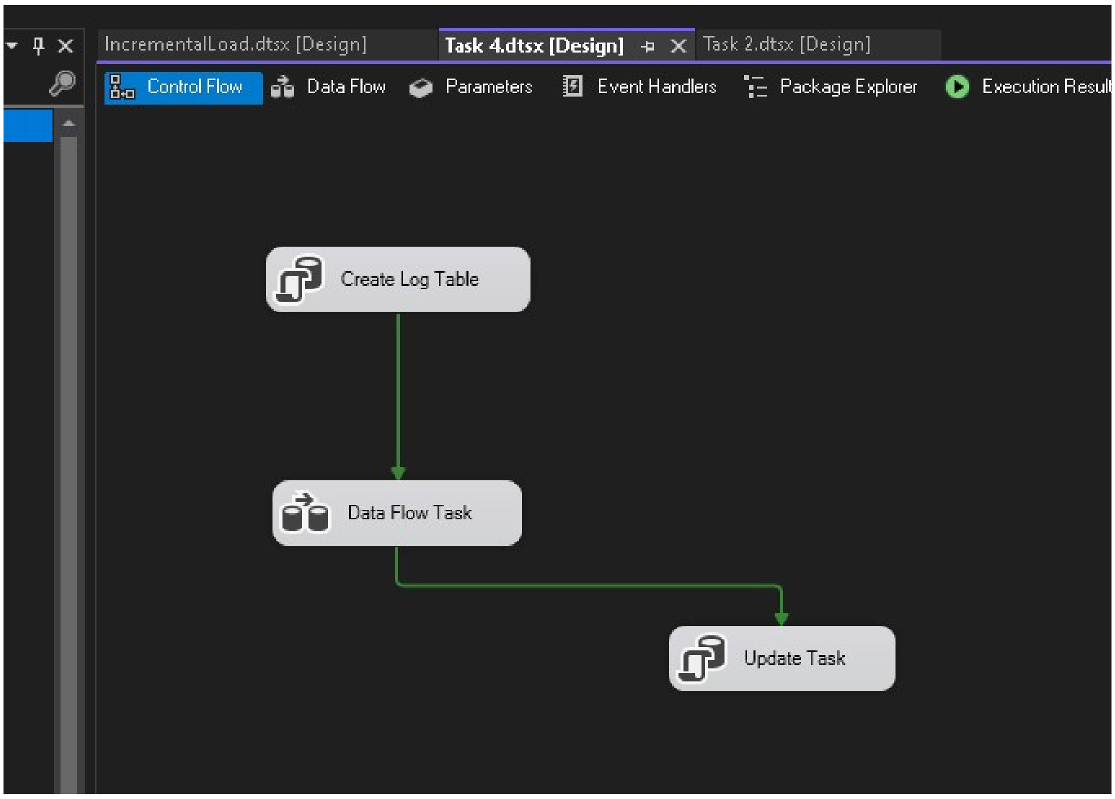
100 %

Results | Messages

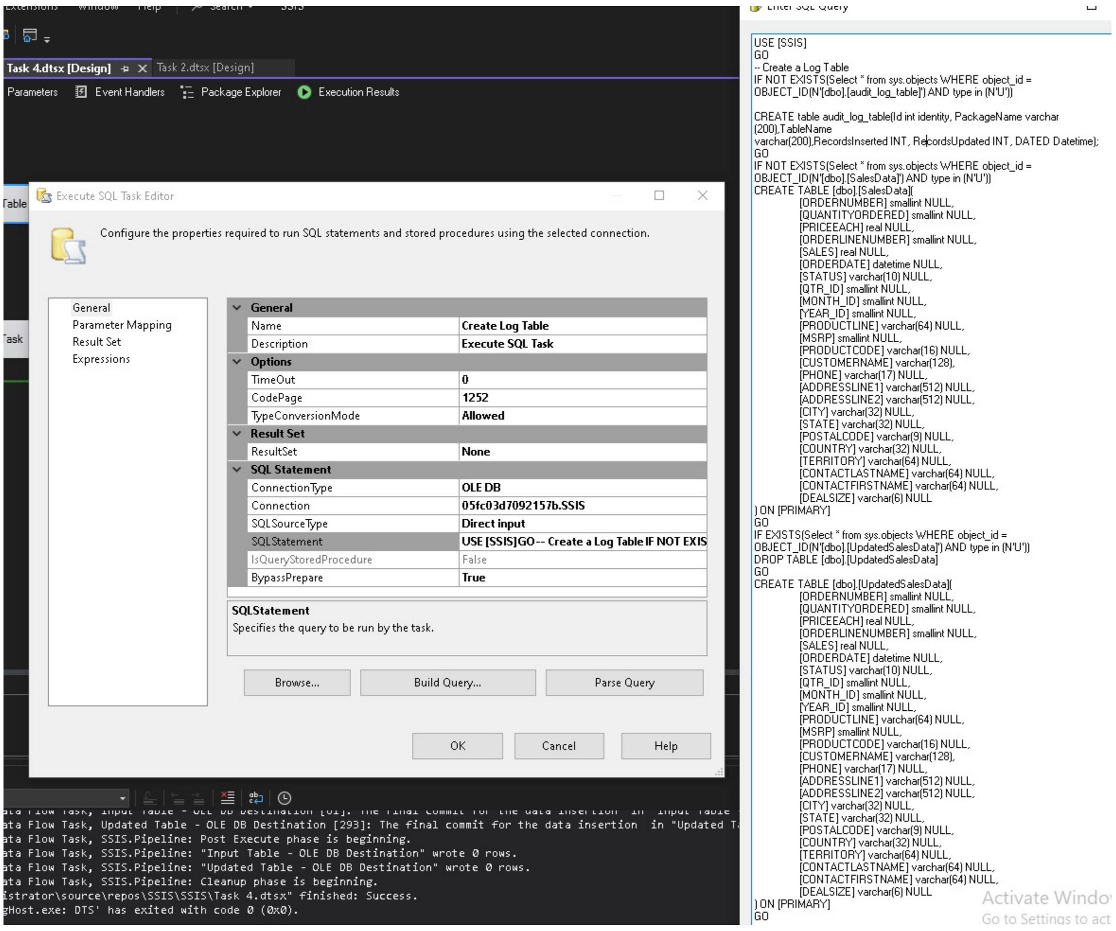
	PRODUCTLINE	January_TotalSales	October_TotalSales	November_TotalSales	December_TotalSales	February_TotalSales	March_TotalSales	April_TotalSales
1	Ships	61287.3996582031	79791.7398681641	143075.649902344	24079.9398193359	68247.9599609375	74341.9404296875	33683.450073
2	Trains	19026.580078125	25416.8696289063	44794.6297607422	23181.1398925781	16507.9500732422	22581.2696533203	4756.4700927
3	Planes	46715.8100585938	106777.650756836	175263.970214844	68497.7398681641	107905.650268555	79735.0500488281	103359.76989
4	Classic Cars	303070.559448242	465002.240844727	825156.260498047	237291.068603516	299647.798950195	277560.540405273	263252.00231
5	Trucks and Buses	78530.6296386719	123813.57043457	250874.060180664	104133.740478516	68211.0701904297	61877.3397216797	28790.759765
6	Vintage Cars	196129.579467773	216763.540466309	418663.739685059	131216.629699707	127119.789306641	177935.269226074	115941.62902
7	Motorcycles	81113.8797607422	103649.610351563	261057.360107422	46278.8598632813	122801.679931641	60469.9802246094	119606.87988

Task4: Incremental load

a. main control flow of incremental load process



b. sql task and its query used for creating a log tablewhich keeps track when records are updated



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

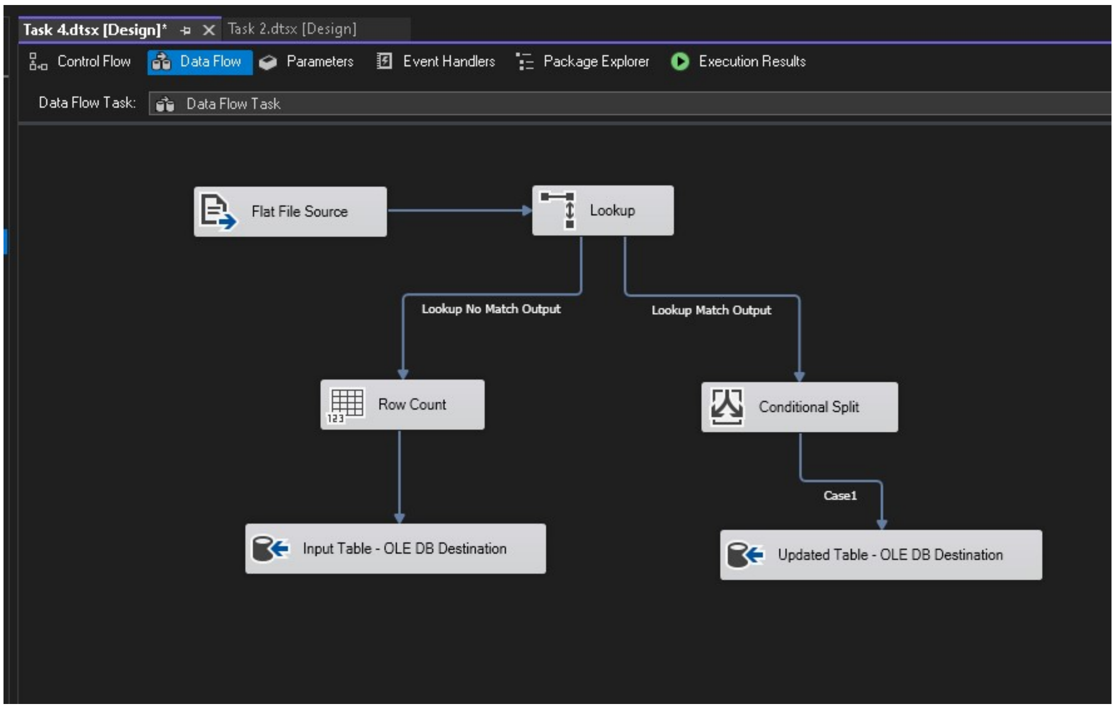
General	
Name	Create Log Table
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1252
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	05fc03d7092157b.SSIS
SQLSourceType	Direct input
SQLStatement	USE [SSIS]GO-- Create a Log Table IF NOT EXIS
IsQueryStoredProcedure	False
BypassPrepare	True

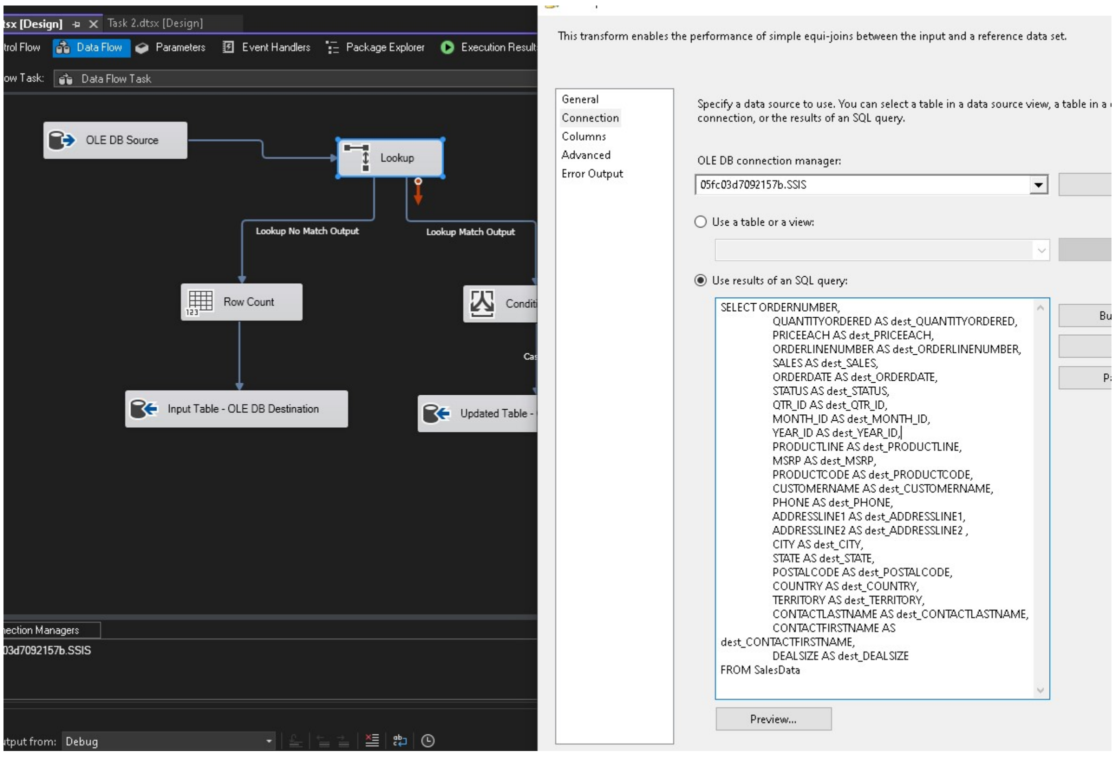
SQLStatement
Specifies the query to be run by the task.

```
ata Flow Task, Updated Table - OLE DB Destination [293]: The final commit for the data insertion  in "Updated Ta
ata Flow Task, SSIS.Pipeline: Post Execute phase is beginning.
ata Flow Task, SSIS.Pipeline: "Input Table - OLE DB Destination" wrote 0 rows.
ata Flow Task, SSIS.Pipeline: "Updated Table - OLE DB Destination" wrote 0 rows.
ata Flow Task, SSIS.Pipeline: Cleanup phase is beginning.
istrator\source\repos\SSIS\SSIS\Task 4.dtsx" finished: Success.
gHost.exe: DTS' has exited with code 0 (0x0).
```

```
USE [SSIS]
GO
-- Create a Log Table
IF NOT EXISTS(Select * from sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[audit_log_table]') AND type in (N'U'))

CREATE table audit_log_table(Id int identity, PackageName varchar
(200),TableName
varchar(200),RecordsInserted INT, RecordsUpdated INT, DATED Datetime);
GO
IF NOT EXISTS(Select * from sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[SalesData]') AND type in (N'U'))
CREATE TABLE [dbo].[SalesData](
        [ORDERNUMBER] smallint NULL,
        [QUANTITYORDERED] smallint NULL,
        [PRICEEACH] real NULL,
        [ORDERLINENUMBER] smallint NULL,
        [SALES] real NULL,
        [ORDERDATE] datetime NULL,
        [STATUS] varchar(10) NULL,
        [QTR_ID] smallint NULL,
        [MONTH_ID] smallint NULL,
        [YEAR_ID] smallint NULL,
        [PRODUCTLINE] varchar(64) NULL,
        [MSRP] smallint NULL,
        [PRODUCTCODE] varchar(16) NULL,
        [CUSTOMERNAME] varchar(128),
        [PHONE] varchar(17) NULL,
        [ADDRESSLINE1] varchar(512) NULL,
        [ADDRESSLINE2] varchar(512) NULL,
        [CITY] varchar(32) NULL,
        [STATE] varchar(32) NULL,
        [POSTALCODE] varchar(9) NULL,
        [COUNTRY] varchar(32) NULL,
        [TERRITORY] varchar(64) NULL,
        [CONTACTLASTNAME] varchar(64) NULL,
        [CONTACTFIRSTNAME] varchar(64) NULL,
        [DEALSIZE] varchar(6) NULL
) ON [PRIMARY]
GO
IF EXISTS(Select * from sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[UpdatedSalesData]') AND type in (N'U'))
DROP TABLE [dbo].[UpdatedSalesData]
GO
CREATE TABLE [dbo].[UpdatedSalesData](
        [ORDERNUMBER] smallint NULL,
        [QUANTITYORDERED] smallint NULL,
        [PRICEEACH] real NULL,
        [ORDERLINENUMBER] smallint NULL,
        [SALES] real NULL,
        [ORDERDATE] datetime NULL,
        [STATUS] varchar(10) NULL,
        [QTR_ID] smallint NULL,
        [MONTH_ID] smallint NULL,
        [YEAR_ID] smallint NULL,
        [PRODUCTLINE] varchar(64) NULL,
        [MSRP] smallint NULL,
        [PRODUCTCODE] varchar(16) NULL,
        [CUSTOMERNAME] varchar(128),
        [PHONE] varchar(17) NULL,
        [ADDRESSLINE1] varchar(512) NULL,
        [ADDRESSLINE2] varchar(512) NULL,
        [CITY] varchar(32) NULL,
        [STATE] varchar(32) NULL,
        [POSTALCODE] varchar(9) NULL,
        [COUNTRY] varchar(32) NULL,
        [TERRITORY] varchar(64) NULL,
        [CONTACTLASTNAME] varchar(64) NULL,
        [CONTACTFIRSTNAME] varchar(64) NULL,
        [DEALSIZE] varchar(6) NULL
) ON [PRIMARY]
GO
```

c. dataflow involves first taking data from FlatFile source and giving it to lookup transformation which selects rows [except ORDERNUMBER as it is used as Primary Key] and compare with reference table [SalesData] to find matching records with same ORDERNUMBER parameter.





This transform enables the performance of simple equi-joins between the input and a reference data set.

Specify a data source to use. You can select a table in a data source view, a table in a connection, or the results of an SQL query.

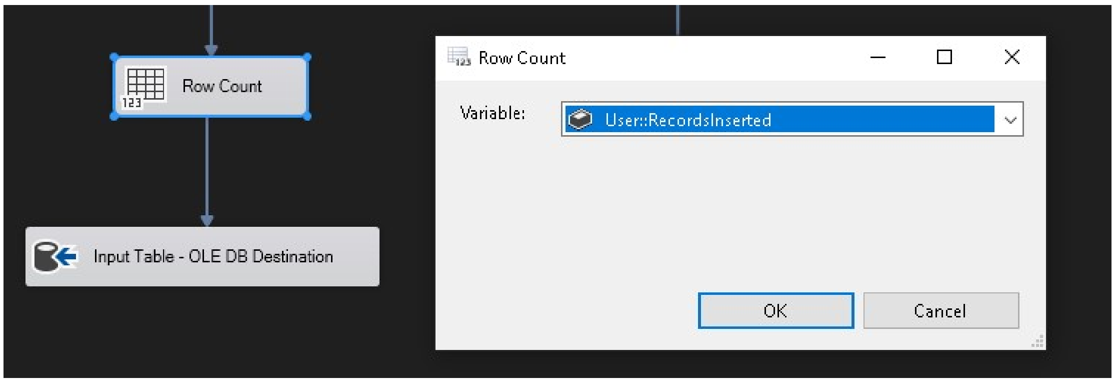
OLE DB connection manager: 05fc03d7092157b.SSIS

Use a table or a view:

Use results of an SQL query:

```
SELECT ORDERNUMBER,
          QUANTITYORDERED AS dest_QUANTITYORDERED,
          PRICEEACH AS dest_PRICEEACH,
          ORDERLINENUMBER AS dest_ORDERLINENUMBER,
          SALES AS dest_SALES,
          ORDERDATE AS dest_ORDERDATE,
          STATUS AS dest_STATUS,
          QTR_ID AS dest_QTR_ID,
          MONTH_ID AS dest_MONTH_ID,
          YEAR_ID AS dest_YEAR_ID,
          PRODUCTLINE AS dest_PRODUCTLINE,
          MSRP AS dest_MSRP,
          PRODUCTCODE AS dest_PRODUCTCODE,
          CUSTOMERNAME AS dest_CUSTOMERNAME,
          PHONE AS dest_PHONE,
          ADDRESSLINE1 AS dest_ADDRESSLINE1,
          ADDRESSLINE2 AS dest_ADDRESSLINE2 ,
          CITY AS dest_CITY,
          STATE AS dest_STATE,
          POSTALCODE AS dest_POSTALCODE,
          COUNTRY AS dest_COUNTRY,
          TERRITORY AS dest_TERRITORY,
          CONTACTLASTNAME AS dest_CONTACTLASTNAME,
          CONTACTFIRSTNAME AS
dest_CONTACTFIRSTNAME,
          DEALSIZE AS dest_DEALSIZE
FROM SalesData
```

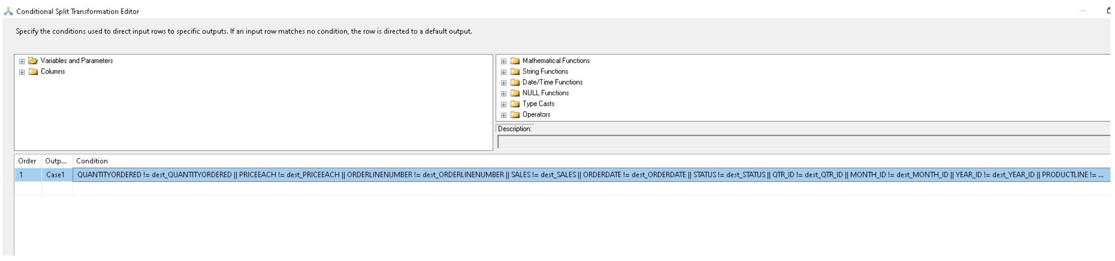
d. data having no match in lookup means they are newly included in the FlatFile source thus have to be inserted into SalesData table. Rowcount is used to find count and result is stored into a variable to be saved to audit log. The records are then inserted into SalesData table.



Variable: User::RecordsInserted

e. for records that do match, each filed is individually checked to find any updates. if updates are found, those records are to be updated and saved.

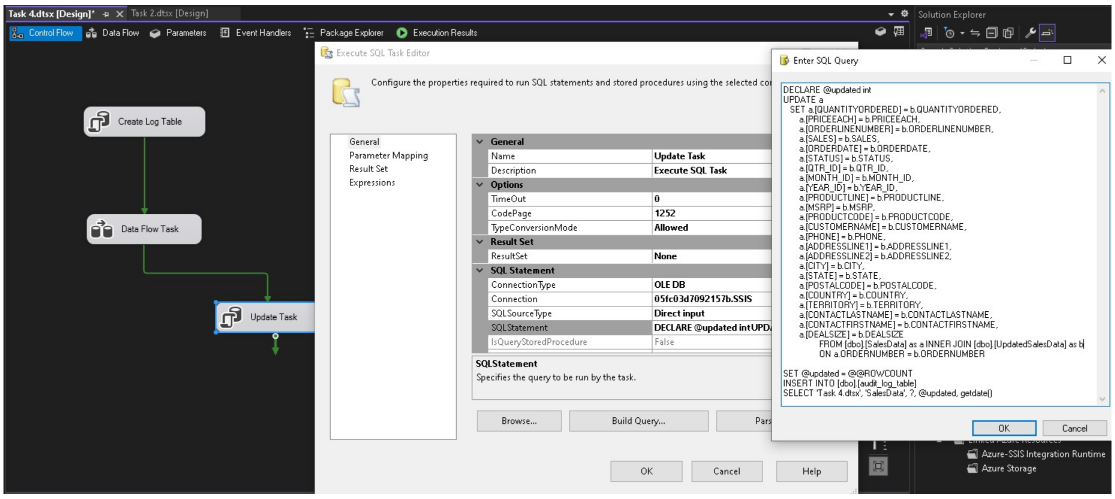
This is done by using a conditional split statement.



Specify the conditions used to direct input rows to specific outputs. If an input row matches no condition, the row is directed to a default output.

Order	Outp...	Condition
1	Case1	QUANTITYORDERED != dest_QUANTITYORDERED || PRICEEACH != dest_PRICEEACH || ORDERLINENUMBER != dest_ORDERLINENUMBER || SALES != dest_SALES || ORDERDATE != dest_ORDERDATE || STATUS != dest_STATUS || QTR_ID != dest_QTR_ID || MONTH_ID != dest_MONTH_ID || YEAR_ID != dest_YEAR_ID || PRODUCTLINE != ...

f. The final SQL task named 'Update Task' is used to write update logs to the table



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

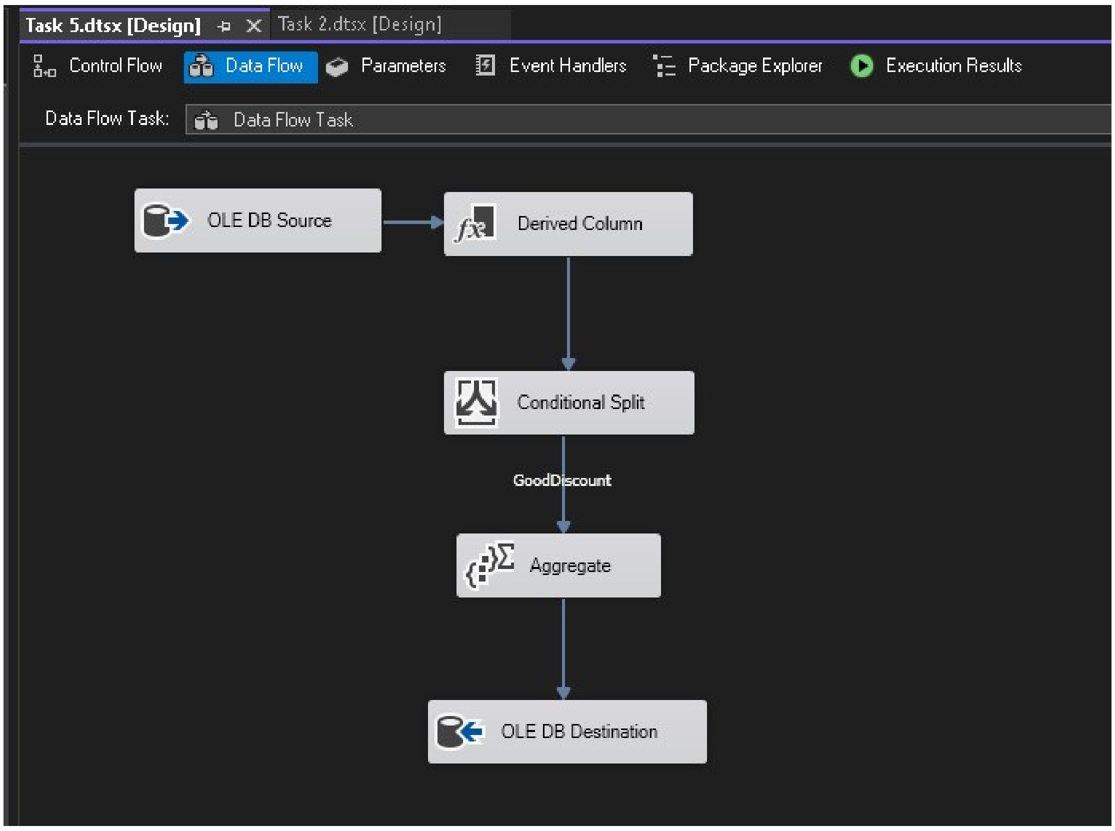
General	
Name	Update Task
Description	Execute SQL Task
Options	
TimeOut	0
CodePage	1252
TypeConversionMode	Allowed
Result Set	
ResultSet	None
SQL Statement	
ConnectionType	OLE DB
Connection	05fc03d7092157b.SSIS
SQLSourceType	Direct input
SQLStatement	DECLARE @updated int UPD
IsQueryStoredProcedure	False

SQLStatement
Specifies the query to be run by the task.

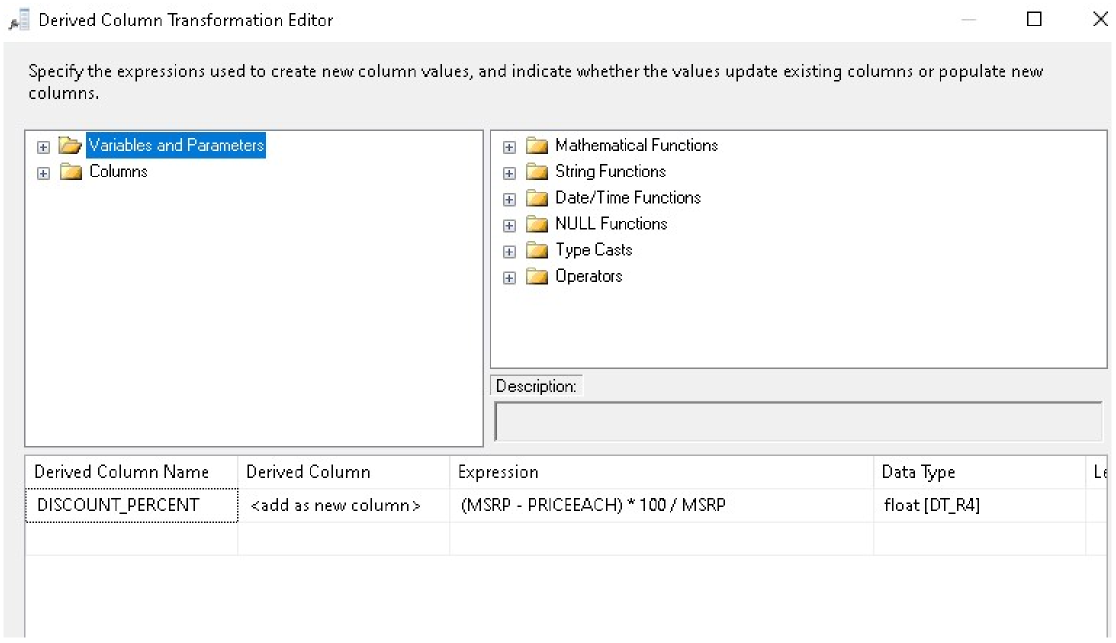
```
DECLARE @updated int
UPDATE a
  SET a.[QUANTITYORDERED] = b.QUANTITYORDERED,
     a.[PRICEEACH] = b.PRICEEACH,
     a.[ORDERLINENUMBER] = b.ORDERLINENUMBER,
     a.[SALES] = b.SALES,
     a.[ORDERDATE] = b.ORDERDATE,
     a.[STATUS] = b.STATUS,
     a.[QTR_ID] = b.QTR_ID,
     a.[MONTH_ID] = b.MONTH_ID,
     a.[YEAR_ID] = b.YEAR_ID,
     a.[PRODUCTLINE] = b.PRODUCTLINE,
     a.[MSRP] = b.MSRP,
     a.[PRODUCTCODE] = b.PRODUCTCODE,
     a.[CUSTOMERNAME] = b.CUSTOMERNAME,
     a.[PHONE] = b.PHONE,
     a.[ADDRESSLINE1] = b.ADDRESSLINE1,
     a.[ADDRESSLINE2] = b.ADDRESSLINE2,
     a.[CITY] = b.CITY,
     a.[STATE] = b.STATE,
     a.[POSTALCODE] = b.POSTALCODE,
     a.[COUNTRY] = b.COUNTRY,
     a.[TERRITORY] = b.TERRITORY,
     a.[CONTACTLASTNAME] = b.CONTACTLASTNAME,
     a.[CONTACTFIRSTNAME] = b.CONTACTFIRSTNAME,
     a.[DEALSIZE] = b.DEALSIZE
         FROM [dbo].[SalesData] as a INNER JOIN [dbo].[UpdatedSalesData] as b
         ON a.ORDERNUMBER = b.ORDERNUMBER

SET @updated = @@ROWCOUNT
INSERT INTO [dbo].[audit_log_table]
SELECT 'Task 4.dtsx', 'SalesData', ?, @updated, getdate()
```

Task5: Transformations

a. main dataflow, aim here is to find average price and average discount on each sale in each month of every year where the average discount is greater than 35% and stroe to an SQL table.



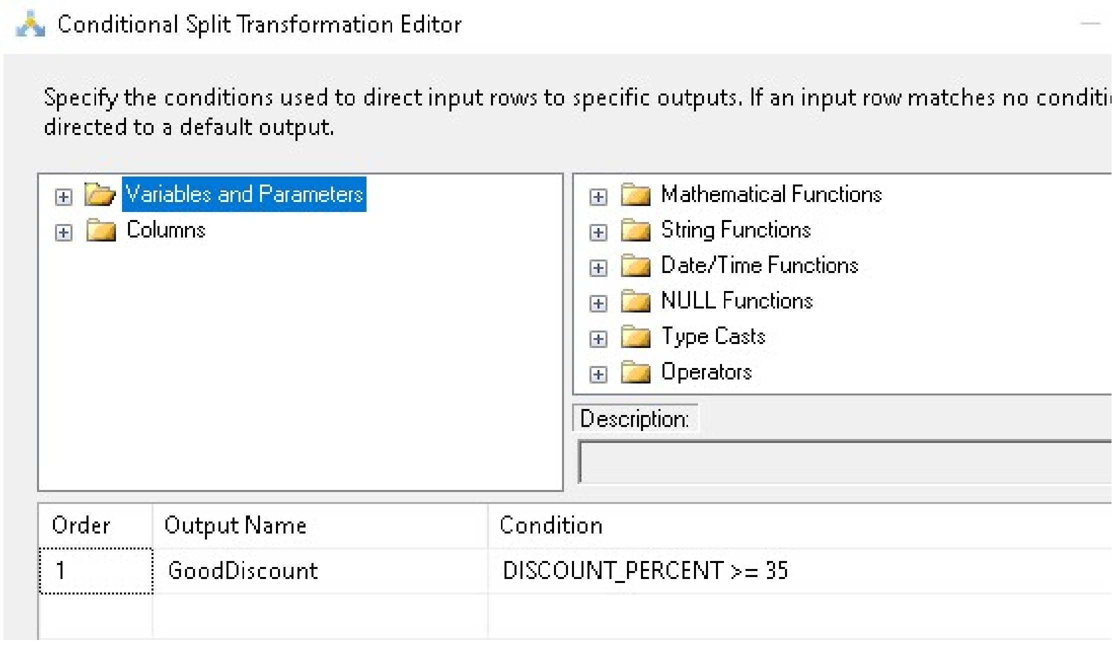
b. column named DISCOUNT_PERCENT, of datatype float, is derived from MSRP and PRICEEACH.


Derived Column Transformation Editor

Specify the expressions used to create new column values, and indicate whether the values update existing columns or populate new columns.

Derived Column Name	Derived Column	Expression	Data Type	Le
DISCOUNT_PERCENT	<add as new column>	(MSRP - PRICEEACH) * 100 / MSRP	float [DT_R4]	

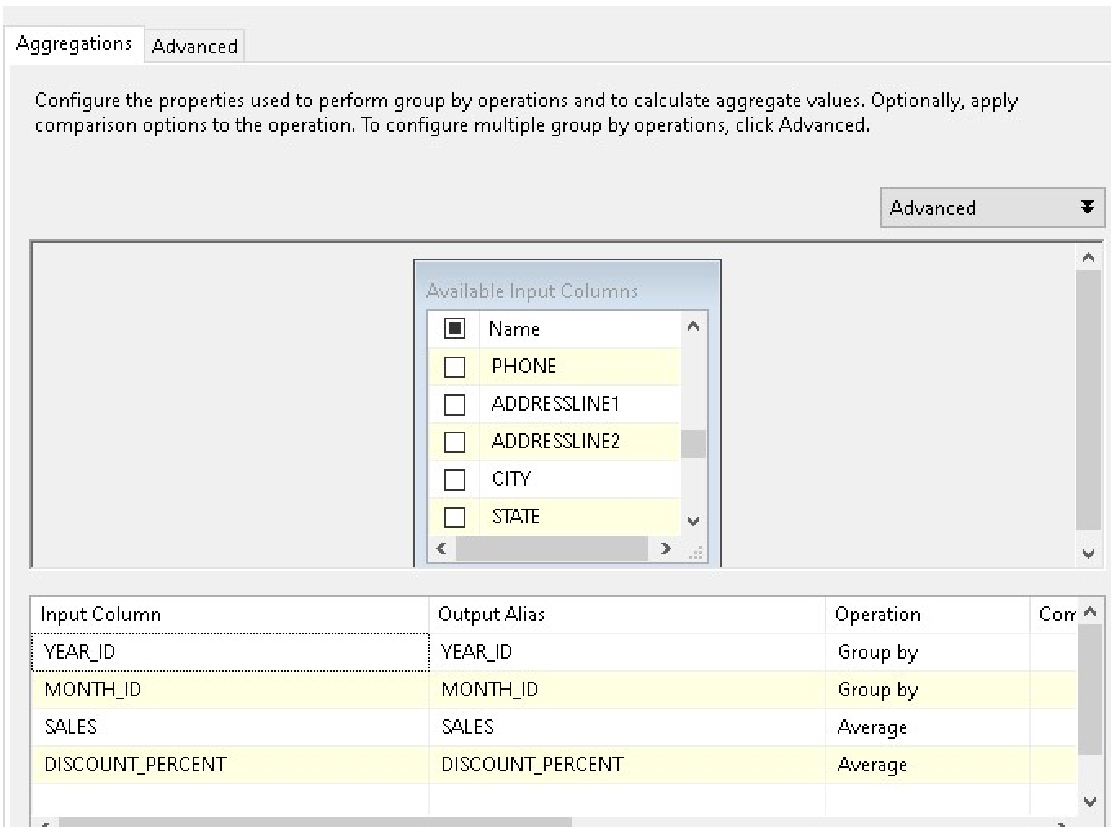
c. conditional split operation is used to select only those records where discount is greater than or equal to 35%.


Conditional Split Transformation Editor

Specify the conditions used to direct input rows to specific outputs. If an input row matches no conditi
directed to a default output.

Order	Output Name	Condition
1	GoodDiscount	DISCOUNT_PERCENT >= 35

d. aggregate operation groups the rows by YEAR_ID and MONTH_ID, and finds average of DISCOUNT_PERCENT and average of SALES.


Aggregations | Advanced

Configure the properties used to perform group by operations and to calculate aggregate values. Optionally, apply comparison options to the operation. To configure multiple group by operations, click Advanced.

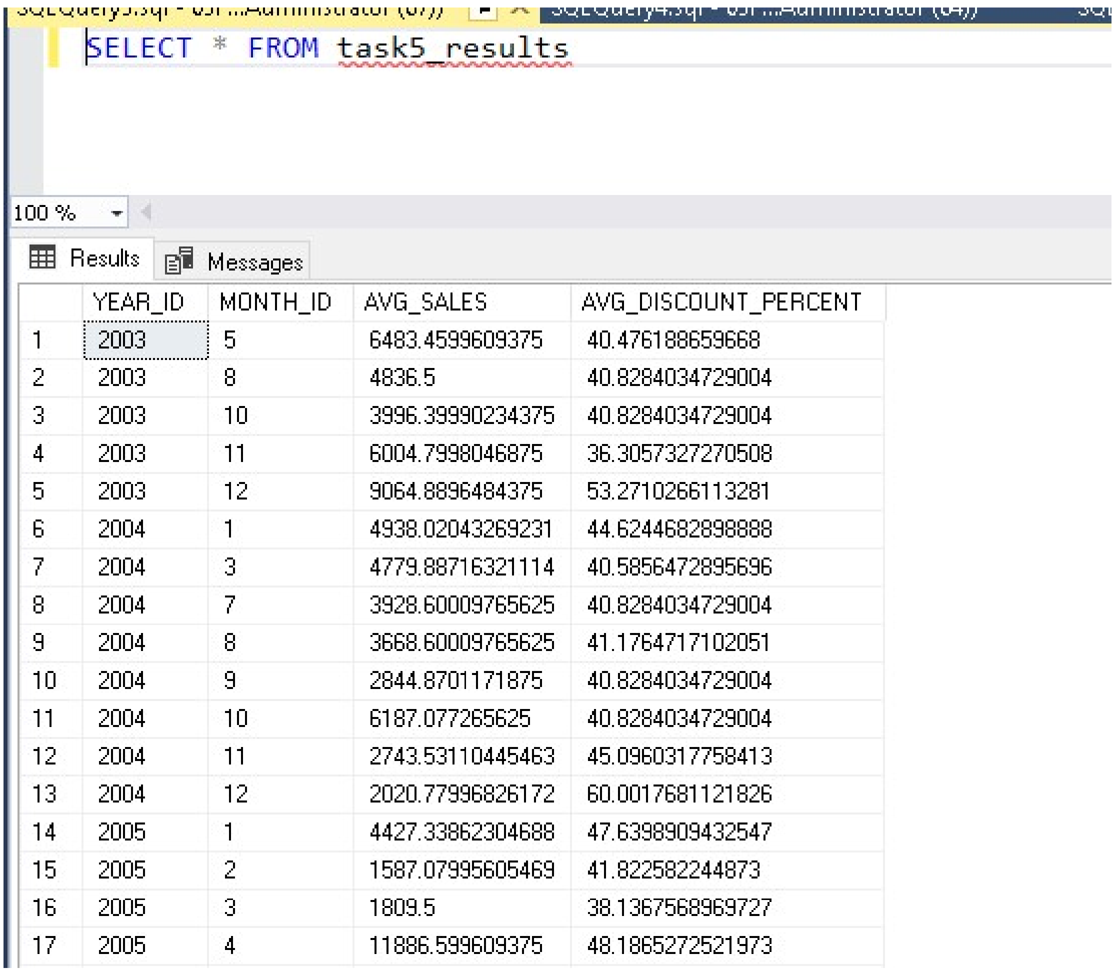
Advanced

Available Input Columns

	Name
▣	Name
☐	PHONE
☐	ADDRESSLINE1
☐	ADDRESSLINE2
☐	CITY
☐	STATE

Input Column	Output Alias	Operation	Com
YEAR_ID	YEAR_ID	Group by	
MONTH_ID	MONTH_ID	Group by	
SALES	SALES	Average	
DISCOUNT_PERCENT	DISCOUNT_PERCENT	Average	

e. final result is stored into SQL database

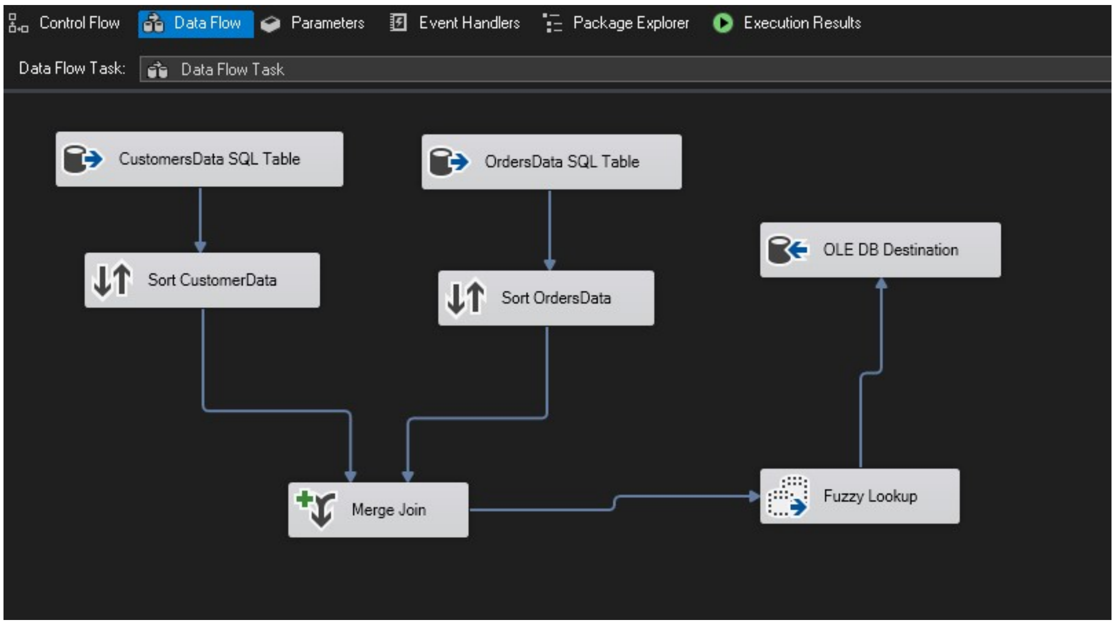


```
SELECT * FROM task5_results
```

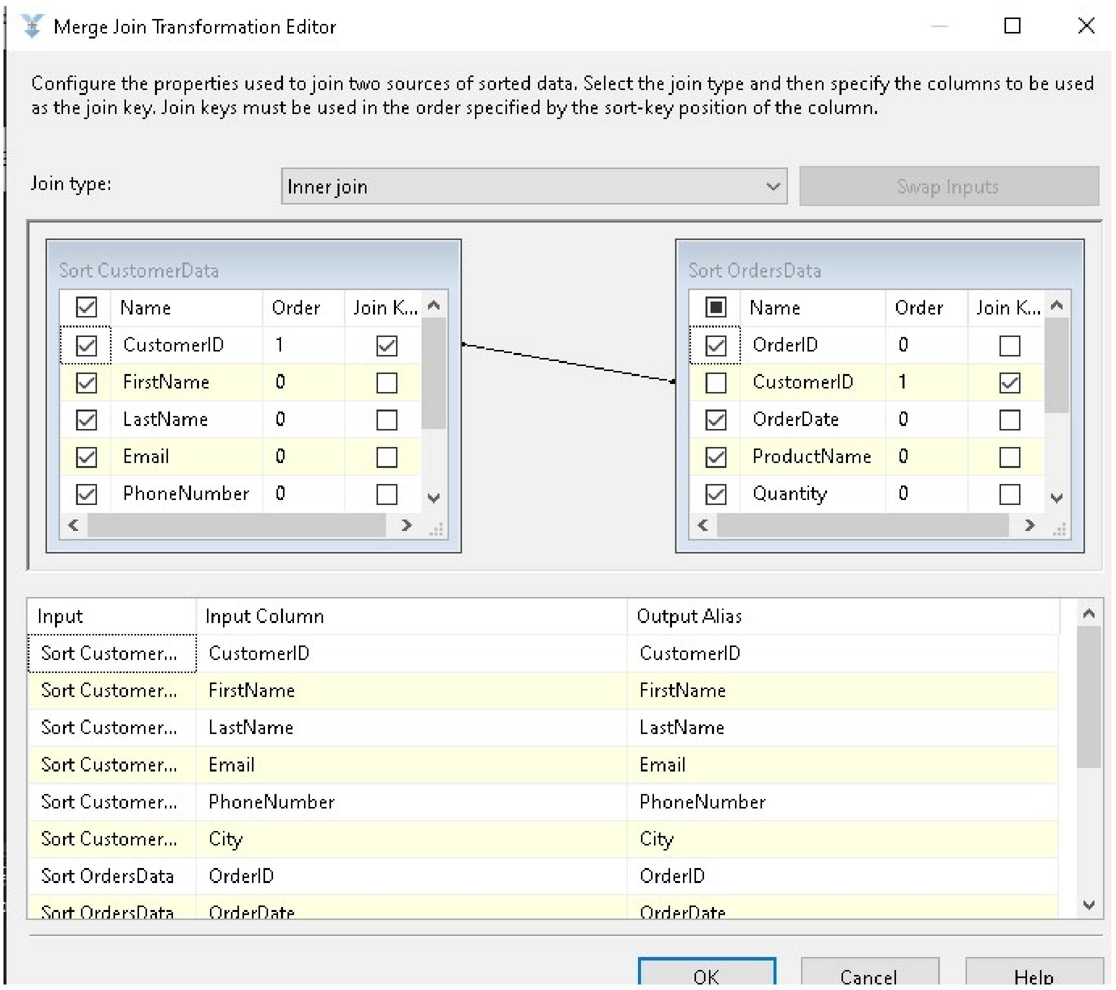
	YEAR_ID	MONTH_ID	AVG_SALES	AVG_DISCOUNT_PERCENT
1	2003	5	6483.4599609375	40.476188659668
2	2003	8	4836.5	40.8284034729004
3	2003	10	3996.39990234375	40.8284034729004
4	2003	11	6004.7998046875	36.3057327270508
5	2003	12	9064.8896484375	53.2710266113281
6	2004	1	4938.02043269231	44.6244682898888
7	2004	3	4779.88716321114	40.5856472895696
8	2004	7	3928.60009765625	40.8284034729004
9	2004	8	3668.60009765625	41.1764717102051
10	2004	9	2844.8701171875	40.8284034729004
11	2004	10	6187.077265625	40.8284034729004
12	2004	11	2743.53110445463	45.0960317758413
13	2004	12	2020.77996826172	60.0017681121826
14	2005	1	4427.33862304688	47.6398909432547
15	2005	2	1587.07995605469	41.822582244873
16	2005	3	1809.5	38.1367568969727
17	2005	4	11886.599609375	48.1865272521973

Task6: Merge and Fuzzy Lookup

a. main data flow for the task. CustomersData and OrdersData are two SQL table containing the necessary data. Both of them contain a common files - CustomerID. Both tables are sorted based on this field



b. Merge Join is used to join both these datasets into one to perform more functions. inner join is performed, by using CustomerID as the common key.



Configure the properties used to join two sources of sorted data. Select the join type and then specify the columns to be used as the join key. Join keys must be used in the order specified by the sort-key position of the column.

Join type: Inner join

Sort CustomerData

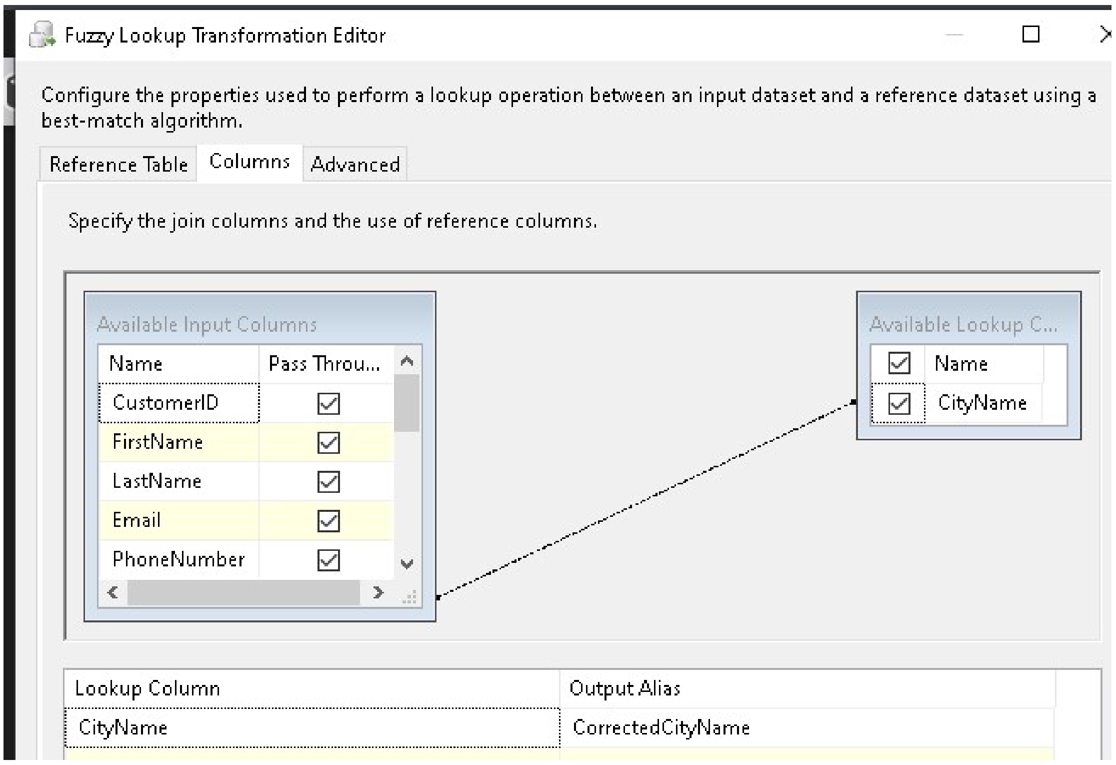
☑	Name	Order	Join K...
☑	CustomerID	1	☑
☑	FirstName	0	☐
☑	LastName	0	☐
☑	Email	0	☐
☑	PhoneNumber	0	☐

Sort OrdersData

■	Name	Order	Join K...
☑	OrderID	0	☐
☐	CustomerID	1	☑
☑	OrderDate	0	☐
☑	ProductName	0	☐
☑	Quantity	0	☐

Input	Input Column	Output Alias
Sort Customer...	CustomerID	CustomerID
Sort Customer...	FirstName	FirstName
Sort Customer...	LastName	LastName
Sort Customer...	Email	Email
Sort Customer...	PhoneNumber	PhoneNumber
Sort Customer...	City	City
Sort OrdersData	OrderID	OrderID
Sort OrdersData	OrderDate	OrderDate

c. Fuzzy lookup compares column 'City' with a list of city names stored in a database. Fuzzy logic is used to clean the data for the 'City' column.


Fuzzy Lookup Transformation Editor

Configure the properties used to perform a lookup operation between an input dataset and a reference dataset using a best-match algorithm.

Specify the join columns and the use of reference columns.

Available Input Columns

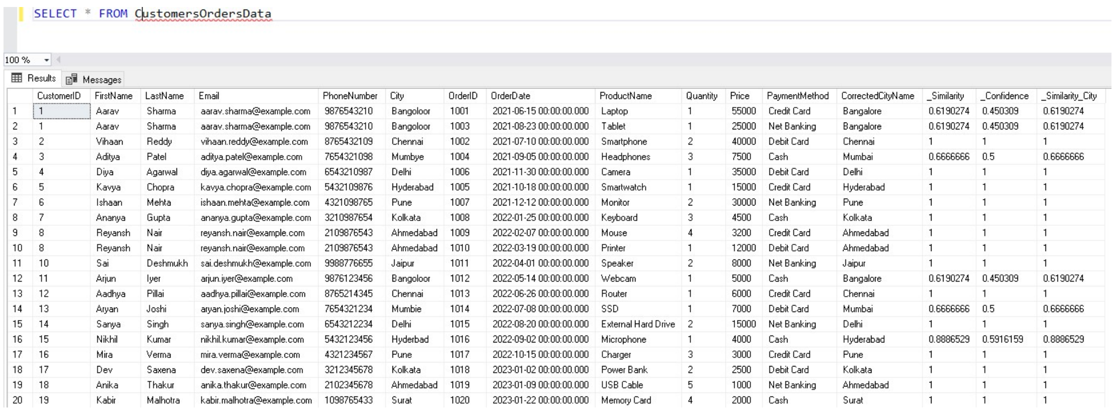
Name	Pass Throu...
CustomerID	☑
FirstName	☑
LastName	☑
Email	☑
PhoneNumber	☑

Available Lookup C...

	Name
☑	CityName

Lookup Column	Output Alias
CityName	CorrectedCityName

d. final output is stored into databse after cleaning city names

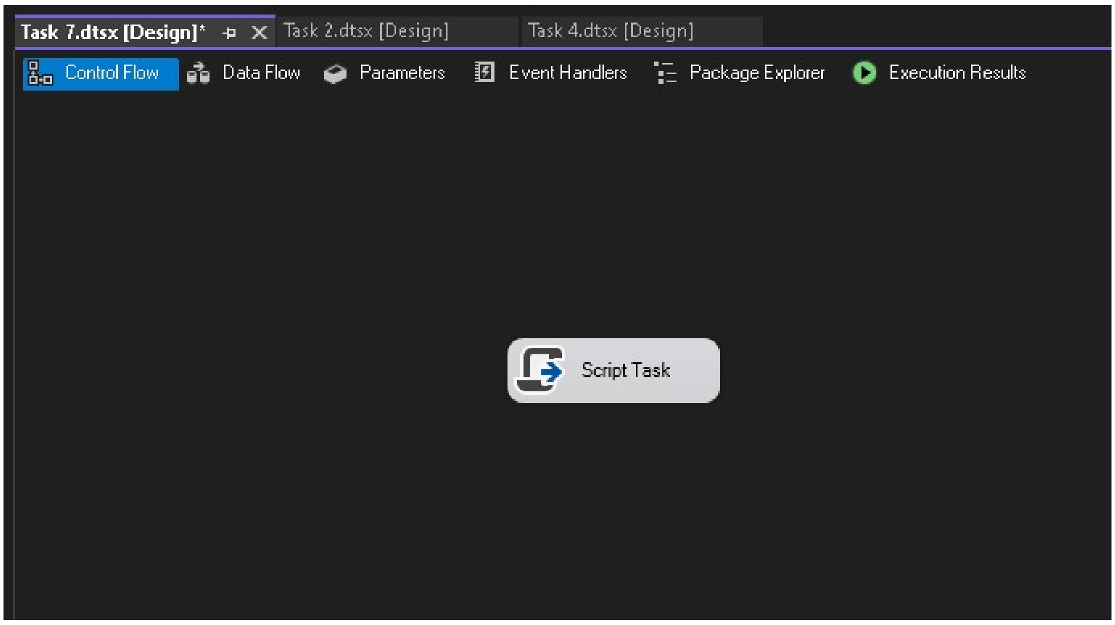


```
SELECT * FROM CustomersOrdersData
```

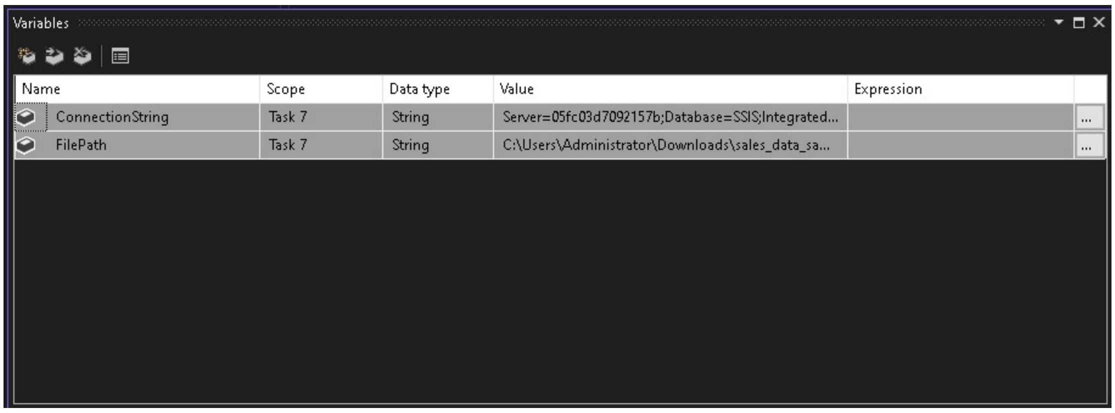
	CustomerID	FirstName	LastName	Email	PhoneNumber	City	OrderID	OrderDate	ProductName	Quantity	Price	PaymentMethod	CorrectedCityName	_Similarity	_Confidence	_Similarity_City
1	1	Aarav	Sharma	aarav.sharma@example.com	9876543210	Bangoloor	1001	2021-06-15 00:00:00.000	Laptop	1	55000	Credit Card	Bangalore	0.6190274	0.450309	0.6190274
2	1	Aarav	Sharma	aarav.sharma@example.com	9876543210	Bangoloor	1003	2021-08-23 00:00:00.000	Tablet	1	25000	Net Banking	Bangalore	0.6190274	0.450309	0.6190274
3	2	Vihaan	Reddy	vihaan.reddy@example.com	8765432109	Chennai	1002	2021-07-10 00:00:00.000	Smartphone	2	40000	Debit Card	Chennai	1	1	1
4	3	Aditya	Patel	aditya.patel@example.com	7654321098	Mumbye	1004	2021-09-05 00:00:00.000	Headphones	3	7500	Cash	Mumbai	0.6666666	0.5	0.6666666
5	4	Diya	Agarwal	diya.agarwal@example.com	6543210987	Delhi	1006	2021-11-30 00:00:00.000	Camera	1	35000	Debit Card	Delhi	1	1	1
6	5	Kavya	Chopra	kavya.chopra@example.com	5432109876	Hyderabad	1005	2021-10-18 00:00:00.000	Smartwatch	1	15000	Credit Card	Hyderabad	1	1	1
7	6	Ishaan	Mehta	ishaan.mehta@example.com	4321098765	Pune	1007	2021-12-12 00:00:00.000	Monitor	2	30000	Net Banking	Pune	1	1	1
8	7	Ananya	Gupta	ananya.gupta@example.com	3210987654	Kolkata	1008	2022-01-25 00:00:00.000	Keyboard	3	4500	Cash	Kolkata	1	1	1
9	8	Reyansh	Nair	reyansh.nair@example.com	2109876543	Ahmedabad	1009	2022-02-07 00:00:00.000	Mouse	4	3200	Credit Card	Ahmedabad	1	1	1
10	8	Reyansh	Nair	reyansh.nair@example.com	2109876543	Ahmedabad	1010	2022-03-19 00:00:00.000	Printer	1	12000	Debit Card	Ahmedabad	1	1	1
11	10	Sai	Deshmukh	sai.deshmukh@example.com	9988776655	Jaipur	1011	2022-04-01 00:00:00.000	Speaker	2	8000	Net Banking	Jaipur	1	1	1
12	11	Arjun	Iyer	arjun.iyer@example.com	9876123456	Bangoloor	1012	2022-05-14 00:00:00.000	Webcam	1	5000	Cash	Bangalore	0.6190274	0.450309	0.6190274
13	12	Aadhya	Pillai	aadhya.pillai@example.com	8765214345	Chennai	1013	2022-06-26 00:00:00.000	Router	1	6000	Credit Card	Chennai	1	1	1
14	13	Aryan	Joshi	aryan.joshi@example.com	7654321234	Mumbie	1014	2022-07-08 00:00:00.000	SSD	1	7000	Debit Card	Mumbai	0.6666666	0.5	0.6666666
15	14	Sanya	Singh	sanya.singh@example.com	6543212234	Delhi	1015	2022-08-20 00:00:00.000	External Hard Drive	2	15000	Net Banking	Delhi	1	1	1
16	15	Nikhil	Kumar	nikhil.kumar@example.com	5432123456	Hyderbad	1016	2022-09-02 00:00:00.000	Microphone	1	4000	Cash	Hyderabad	0.8886529	0.5916159	0.8886529
17	16	Mira	Verma	mira.verma@example.com	4321234567	Pune	1017	2022-10-15 00:00:00.000	Charger	3	3000	Credit Card	Pune	1	1	1
18	17	Dev	Saxena	dev.saxena@example.com	3212345678	Kolkata	1018	2023-01-02 00:00:00.000	Power Bank	2	2500	Debit Card	Kolkata	1	1	1
19	18	Anika	Thakur	anika.thakur@example.com	2102345678	Ahmedabad	1019	2023-01-09 00:00:00.000	USB Cable	5	1000	Net Banking	Ahmedabad	1	1	1
20	19	Kabir	Malhotra	kabir.malhotra@example.com	1098765433	Surat	1020	2023-01-22 00:00:00.000	Memory Card	4	2000	Cash	Surat	1	1	1

Task7: Using Script task

a. Add Script task to control flow



b. Create two variables ConnectionString and FilePath. ConnectionString is for connecting to the database, while FilePath specifies location of CSV source file


Variables

Name	Scope	Data type	Value	Expression
ConnectionString	Task 7	String	Server=05fc03d7092157b;Database=SSIS;Integrated...	
FilePath	Task 7	String	C:\Users\Administrator\Downloads\sales_data_sa...	

c. Open Script Task, click on Edit Script and type C# program to load from CSV and load into Table in Database

```
#region Help:  Introduction to the script task
/* The Script Task allows you to perform virtually any operation that can be accomplished in
 * a .Net application within the context of an Integration Services control flow.
 *
 * Expand the other regions which have "Help" prefixes for examples of specific ways to use
 * Integration Services features within this script task. */
#endregion
using System;
using System.Data;
using System.Data.SqlClient;
using System.IO;
using Microsoft.SqlServer.Dts.Runtime;

#region Namespaces
```

```
using System;
using System.Data;
using Microsoft.SqlServer.Dts.Runtime;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.IO;
#endregion

namespace ST_6a4faddaa9374e1ba866c9e413dd6719
{
   /// <summary>
   /// ScriptMain is the entry point class of the script.  Do not change the name, attributes,
   /// or parent of this class.
   /// </summary>
                [Microsoft.SqlServer.Dts.Tasks.ScriptTask.SSISScriptTaskEntryPointAttribute]
                public partial class ScriptMain : Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
                {
       #region Help:  Using Integration Services variables and parameters in a script
       /* To use a variable in this script, first ensure that the variable has been added to
        * either the list contained in the ReadOnlyVariables property or the list contained in
        * the ReadWriteVariables property of this script task, according to whether or not your
        * code needs to write to the variable.  To add the variable, save this script, close this instance of
        * Visual Studio, and update the ReadOnlyVariables and
        * ReadWriteVariables properties in the Script Transformation Editor window.
        * To use a parameter in this script, follow the same steps. Parameters are always read-only.
        *
        * Example of reading from a variable:
        *  DateTime startTime = (DateTime) Dts.Variables["System::StartTime"].Value;
        *
        * Example of writing to a variable:
        *  Dts.Variables["User::myStringVariable"].Value = "new value";
        *
        * Example of reading from a package parameter:
        *  int batchId = (int) Dts.Variables["$Package::batchId"].Value;
        *
        * Example of reading from a project parameter:
        *  int batchId = (int) Dts.Variables["$Project::batchId"].Value;
        *
        * Example of reading from a sensitive project parameter:
        *  int batchId = (int) Dts.Variables["$Project::batchId"].GetSensitiveValue();
        * */

       #endregion

       #region Help:  Firing Integration Services events from a script
       /* This script task can fire events for logging purposes.
        *
        * Example of firing an error event:
        *  Dts.Events.FireError(18, "Process Values", "Bad value", "", 0);
        *
        * Example of firing an information event:
        *  Dts.Events.FireInformation(3, "Process Values", "Processing has started", "", 0, ref fireAgain)
        *
        * Example of firing a warning event:
        *  Dts.Events.FireWarning(14, "Process Values", "No values received for input", "", 0);
        * */
       #endregion

       #region Help:  Using Integration Services connection managers in a script
       /* Some types of connection managers can be used in this script task.  See the topic
        * "Working with Connection Managers Programatically" for details.
        *
        * Example of using an ADO.Net connection manager:
        *  object rawConnection = Dts.Connections["Sales DB"].AcquireConnection(Dts.Transaction);
        *  SqlConnection myADONETConnection = (SqlConnection)rawConnection;
        *  //Use the connection in some code here, then release the connection
        *  Dts.Connections["Sales DB"].ReleaseConnection(rawConnection);
        *
        * Example of using a File connection manager
```

```
 *  object rawConnection = Dts.Connections["Prices.zip"].AcquireConnection(Dts.Transaction);
 *  string filePath = (string)rawConnection;
 *  //Use the connection in some code here, then release the connection
 *  Dts.Connections["Prices.zip"].ReleaseConnection(rawConnection);
 * */
#endregion


/// <summary>
/// This method is called when this script task executes in the control flow.
/// Before returning from this method, set the value of Dts.TaskResult to indicate success or failure.
/// To open Help, press F1.
/// </summary>
public void Main()
{
   string filePath = Dts.Variables["User::FilePath"].Value.ToString();
   string connectionString = Dts.Variables["User::ConnectionString"].Value.ToString();

   try
   {
      // Create a DataTable to hold CSV data
      DataTable dataTable = new DataTable();
      dataTable.Columns.Add("ORDERNUMBER", typeof(int));
      dataTable.Columns.Add("QUANTITYORDERED", typeof(int));
      dataTable.Columns.Add("PRICEEACH", typeof(decimal));
      dataTable.Columns.Add("ORDERLINENUMBER", typeof(int));
      dataTable.Columns.Add("SALES", typeof(decimal));
      dataTable.Columns.Add("ORDERDATE", typeof(string));
      dataTable.Columns.Add("STATUS", typeof(string));
      dataTable.Columns.Add("QTR_ID", typeof(int));
      dataTable.Columns.Add("MONTH_ID", typeof(int));
      dataTable.Columns.Add("YEAR_ID", typeof(int));
      dataTable.Columns.Add("PRODUCTLINE", typeof(string));
      dataTable.Columns.Add("MSRP", typeof(decimal));
      dataTable.Columns.Add("PRODUCTCODE", typeof(string));
      dataTable.Columns.Add("CUSTOMERNAME", typeof(string));
      dataTable.Columns.Add("PHONE", typeof(string));
      dataTable.Columns.Add("ADDRESSLINE1", typeof(string));
      dataTable.Columns.Add("ADDRESSLINE2", typeof(string));
      dataTable.Columns.Add("CITY", typeof(string));
      dataTable.Columns.Add("STATE", typeof(string));
      dataTable.Columns.Add("POSTALCODE", typeof(string));
      dataTable.Columns.Add("COUNTRY", typeof(string));
      dataTable.Columns.Add("TERRITORY", typeof(string));
      dataTable.Columns.Add("CONTACTLASTNAME", typeof(string));
      dataTable.Columns.Add("CONTACTFIRSTNAME", typeof(string));
      dataTable.Columns.Add("DEALSIZE", typeof(string));

      // Read data from the CSV file
      using (var reader = new StreamReader(filePath))
      {
         string headerLine = reader.ReadLine(); // Skip header
         while (!reader.EndOfStream)
         {
            var line = reader.ReadLine();
            var values = line.Split(',');

            // Ensure there are enough columns
            if (values.Length < 24) continue; // Adjust if necessary

            // Create a new row and populate it
            DataRow row = dataTable.NewRow();
            row["ORDERNUMBER"] = int.Parse(values[0]);
            row["QUANTITYORDERED"] = int.Parse(values[1]);
            row["PRICEEACH"] = decimal.Parse(values[2]);
            row["ORDERLINENUMBER"] = int.Parse(values[3]);
            row["SALES"] = decimal.Parse(values[4]);
            row["ORDERDATE"] = values[5];
            row["STATUS"] = values[6];
```

```
                row["QTR_ID"] = int.Parse(values[7]);
                row["MONTH_ID"] = int.Parse(values[8]);
                row["YEAR_ID"] = int.Parse(values[9]);
                row["PRODUCTLINE"] = values[10];
                row["MSRP"] = decimal.Parse(values[11]);
                row["PRODUCTCODE"] = values[12];
                row["CUSTOMERNAME"] = values[13];
                row["PHONE"] = values[14];
                row["ADDRESSLINE1"] = values[15];
                row["ADDRESSLINE2"] = values[16];
                row["CITY"] = values[17];
                row["STATE"] = values[18];
                row["POSTALCODE"] = values[19];
                row["COUNTRY"] = values[20];
                row["TERRITORY"] = values[21];
                row["CONTACTLASTNAME"] = values[22];
                row["CONTACTFIRSTNAME"] = values[23];
                row["DEALSIZE"] = values[24];
                dataTable.Rows.Add(row);
              }
            }

            // Insert data into SQL Server
            using (SqlConnection conn = new SqlConnection(connectionString))
            {
              conn.Open();
              foreach (DataRow row in dataTable.Rows)
              {
                using (SqlCommand cmd = new SqlCommand("INSERT INTO CustomTable (ORDERNUMBER, QUANTITYORDERED,
PRICEEACH, ORDERLINENUMBER, SALES, ORDERDATE, STATUS, QTR_ID, MONTH_ID, YEAR_ID, PRODUCTLINE, MSRP,
PRODUCTCODE, CUSTOMERNAME, PHONE, ADDRESSLINE1, ADDRESSLINE2, CITY, STATE, POSTALCODE, COUNTRY, TERRITORY,
CONTACTLASTNAME, CONTACTFIRSTNAME, DEALSIZE) VALUES (@ORDERNUMBER, @QUANTITYORDERED, @PRICEEACH,
@ORDERLINENUMBER, @SALES, @ORDERDATE, @STATUS, @QTR_ID, @MONTH_ID, @YEAR_ID, @PRODUCTLINE, @MSRP,
@PRODUCTCODE, @CUSTOMERNAME, @PHONE, @ADDRESSLINE1, @ADDRESSLINE2, @CITY, @STATE, @POSTALCODE,
@COUNTRY, @TERRITORY, @CONTACTLASTNAME, @CONTACTFIRSTNAME, @DEALSIZE)", conn))
                {
                  cmd.Parameters.AddWithValue("@ORDERNUMBER", row["ORDERNUMBER"]);
                  cmd.Parameters.AddWithValue("@QUANTITYORDERED", row["QUANTITYORDERED"]);
                  cmd.Parameters.AddWithValue("@PRICEEACH", row["PRICEEACH"]);
                  cmd.Parameters.AddWithValue("@ORDERLINENUMBER", row["ORDERLINENUMBER"]);
                  cmd.Parameters.AddWithValue("@SALES", row["SALES"]);
                  cmd.Parameters.AddWithValue("@ORDERDATE", row["ORDERDATE"]);
                  cmd.Parameters.AddWithValue("@STATUS", row["STATUS"]);
                  cmd.Parameters.AddWithValue("@QTR_ID", row["QTR_ID"]);
                  cmd.Parameters.AddWithValue("@MONTH_ID", row["MONTH_ID"]);
                  cmd.Parameters.AddWithValue("@YEAR_ID", row["YEAR_ID"]);
                  cmd.Parameters.AddWithValue("@PRODUCTLINE", row["PRODUCTLINE"]);
                  cmd.Parameters.AddWithValue("@MSRP", row["MSRP"]);
                  cmd.Parameters.AddWithValue("@PRODUCTCODE", row["PRODUCTCODE"]);
                  cmd.Parameters.AddWithValue("@CUSTOMERNAME", row["CUSTOMERNAME"]);
                  cmd.Parameters.AddWithValue("@PHONE", row["PHONE"]);
                  cmd.Parameters.AddWithValue("@ADDRESSLINE1", row["ADDRESSLINE1"]);
                  cmd.Parameters.AddWithValue("@ADDRESSLINE2", row["ADDRESSLINE2"]);
                  cmd.Parameters.AddWithValue("@CITY", row["CITY"]);
                  cmd.Parameters.AddWithValue("@STATE", row["STATE"]);
                  cmd.Parameters.AddWithValue("@POSTALCODE", row["POSTALCODE"]);
                  cmd.Parameters.AddWithValue("@COUNTRY", row["COUNTRY"]);
                  cmd.Parameters.AddWithValue("@TERRITORY", row["TERRITORY"]);
                  cmd.Parameters.AddWithValue("@CONTACTLASTNAME", row["CONTACTLASTNAME"]);
                  cmd.Parameters.AddWithValue("@CONTACTFIRSTNAME", row["CONTACTFIRSTNAME"]);
                  cmd.Parameters.AddWithValue("@DEALSIZE", row["DEALSIZE"]);
                  cmd.ExecuteNonQuery();
                }
              }
            }

            Dts.TaskResult = (int)ScriptResults.Success;
          }
          catch (Exception ex)
```

```
        {
            Dts.Events.FireError(0, "Script Task", ex.Message, String.Empty, 0);
            Dts.TaskResult = (int)ScriptResults.Failure;
        }
    }
}

    #region ScriptResults declaration
    /// <summary>
    /// This enum provides a convenient shorthand within the scope of this class for setting the
    /// result of the script.
    ///
    /// This code was generated automatically.
    /// </summary>
    enum ScriptResults
    {
        Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
        Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
    };
    #endregion

            }
```