CMM701 – DATA WAREHOUSING

Submission 02

Sahdiya 20231624/2330842

SECTION A

- **PART 1**

Kimball and Inmon represent two contrasting approaches to data warehouse design, each with its unique methodologies, strengths, and weaknesses. These approaches are chosen based on the specific needs of an organization, its business objectives, and the technical requirements.

The Inmon approach, pioneered by Bill Inmon, is a top-down architecture. It begins with the creation of a single, centralized enterprise-wide data warehouse and subsequently builds data marts to meet the diverse needs of the business and its analytical requirements. This approach defines a data warehouse as 'Subject Oriented, Integrated, Time-Variant, and Non-Volatile,' ensuring data consistency and eliminating the need for data overwriting. Inmon employs a logical model, often the Entity-Relationship (ER) model, for all entities and implements a normalized data structure. It is lauded for creating a single source of truth for the entire business.

On the other hand, the Kimball approach, introduced by Ralph Kimball, follows a bottom-up architecture. It begins by creating multiple data marts that cater to specific analytical needs and subsequently consolidates these data marts into a single data warehouse. In the Kimball approach, data is extracted and loaded into a staging area before populating a denormalized dimensional data warehouse. The data is organized into Fact tables containing transactional data and Dimension Tables providing supporting information. The star schema, which combines a fact table with several dimension tables, is the hallmark of this approach, along with the bus matrix, which outlines how these star schemas are constructed. Kimball primarily utilizes Star/Snowflake schema for data warehousing.

Inmon's approach excels in data consistency and maintains lower data redundancy due to its reliance on a normalized data model. However, it generally requires more time for implementation. The maintenance cost for Inmon is relatively lower as it can be segregated into data marts. On the other hand, the Kimball approach, while faster to implement, tends to be more maintenance-intensive as the project progresses.

Although both the designs follow ETL concepts and considers a centralized data warehouse, the way of modelling and loading data into the data warehouses differ from one another. Thus the designs are chosen based on factors like business requirements, time and cost:

- Inmon is well-suited for organizations with a strategic focus and a need for a large, enterprise-wide data warehouse. It is particularly useful in domains where a comprehensive overview of all entities is crucial, such as the insurance sector.
- Kimball is a better fit for organizations with fewer inter-process linkages, not necessitating an enterprise-wide data warehouse. It is ideal for businesses that require agility and quick access to specific analytical data, such as marketing firms.
- Inmon is suitable when a robust, normalized structure is preferred, and the organization can allocate more time to implementation.
- Kimball is the choice when speedy data warehousing implementation is a priority, especially when the business demands reports that focus on specific teams or processes.

Some organizations opt for a hybrid model that combines elements of both approaches, balancing the need for data consistency with agility and quick wins. Thus the choice between the Inmon and Kimball approaches is highly dependent on the specific needs and objectives of the organization. Both have their strengths and weaknesses, and the decision should align with the organization's strategic goals, business priorities, and available resources.

References

- https://www.astera.com/type/blog/data-warehouse-concepts/#:~:text=We've%20narrowed%20down%20a,opt%20for%20the%20Kimball%20method.
- https://www.geeksforgeeks.org/difference-between-kimball-and-inmon/
- https://medium.com/analytics-vidhya/theories-of-kimball-and-inmon-about-data-warehouse-design-c16260fab5e9
- https://www.zentut.com/data-warehouse/kimball-and-inmon-data-warehouse-architectures/#:~:text=Kimball%20uses%20the%20dimensional%20model,uses%20it%20for%20all%20data

- **PART 2**

Rise of Data Warehousing Automation

Data warehousing has undergone significant evolution over the years, and one of the latest state-of-the-art trends in this field is Data Warehousing Automation (DWA). This transformative technology has revolutionized the way organizations design, build, and manage their data warehouses, addressing the critical factors of time-to-market and business agility. DWA is a game-changer, minimizing manual overhead in the implementation of data warehouses and business intelligence analytics, thereby propelling organizations into a new era of efficiency and cost-effectiveness.

Data warehousing automation refers to the use of advanced tools and technologies to streamline the process of data warehouse design, development, and maintenance. Traditional data warehousing often involved manual and time-consuming tasks, but automation has changed the game. Modern data warehouses use latest technologies to automate the process of planning, data modelling and integration. It aims to accelerate the data warehousing lifecycle and reduce the need for manual intervention, resulting in increased efficiency and cost-effectiveness.

The history of data warehousing can be traced back to the 1960s when the emergence of disk storage created a need for efficient data storage and processing within databases. This period marked the beginnings of dimensional data marts and the development of entity relationships. As data warehousing evolved, it faced inherent challenges, prolonged development cycles, insufficient metadata management within existing data warehouses. Furthermore, the high resource costs associated with development of traditional data warehouse architectures were

unsuitable for the dynamic and competitive business landscape of the 21st century. Poorly integrated systems and fragmented data further exacerbated the challenges too. The limitations of traditional data warehousing paved the way for a more agile platform capable of automating Extract, Transform, Load (ETL) processes and seamlessly integrating with enterprise applications. DWA emerged as the solution to this growing need. DWA now excels in handling cloud technologies, real-time data extraction, and modern Application Programming Interfaces (APIs).

In today's world, DWA is progressing at a high pace due to several reasons. One reason lies in the growing maturity of DWA tools, which are relatively new compared to the well-established traditional data warehousing systems. Furthermore, the initial costs associated with implementing DWA tools can deter organizations, as they prioritize immediate data warehouse tasks, occasionally postponing these transformative projects. Additionally, it enables more data-driven decisions by providing easy access to reliable, high-quality data. DWA enhances productivity through automated code generation, connectors for data consolidation, and streamlined documentation while accelerating data platform deployment, ensuring faster implementation of data warehouse projects. Standardization is improved, as DWA elevates developers to higher levels of abstraction, reducing coding variations. DWA also enhances data quality by enforcing standards and consistency, automating data validation, transformation, and cleansing processes, ultimately reducing errors and duplicates. Nevertheless, there's a perceived risk associated with DWA, with concerns about the potential introduction of errors through code generation. Additionally, leveraging the full potential of DWA necessitates training and adaptation for technical personnel to become proficient in its use. The complexity of implementing DWA is also a challenge, as it aims to automate intricate processes such as ETL, data modelling, and testing within complex enterprise data warehouse systems. Thus, the growth of DWA is slow yet gaining momentum in the market due to its capabilities.

Automated data warehouse architecture, as facilitated by data warehouse automation software, offers a streamlined and code-free approach to the process of consolidating enterprise data from source systems into a data warehouse. In contrast to traditional data warehouse architecture, this software automates essential tasks such as ETL code deployment. The advantages of data warehouse software lie in its automation of ETL processes, achieved through auto-mapping and job scheduling, a user-friendly interface supporting drag-and-drop design, and pre-configured connectors for seamless integration with various enterprise applications, spanning Salesforce, SAP, CRM, and REST APIs. In essence, data warehouse automation software simplifies the creation and management of data warehouses when compared to traditional tools. While the capabilities of these solutions may differ, they typically offer common design patterns and functionalities to fulfill business objectives, making it advisable for enterprises to conduct a data warehouse cost comparison of various tools before selecting the most suitable data warehouse automation tool.

To underscore the practicality of DWA, let's look at how several organizations have leveraged this technology to improve their operations:

- LendingClub: A peer-to-peer lending platform, LendingClub turned to automation for its data warehousing needs. This transition allowed them to rapidly integrate and process vast

amounts of financial data, resulting in improved credit risk assessment and decision-making.

- Logitech: The global technology company Logitech introduced automation to its data warehousing processes. This led to faster product insights, enhanced supply chain management, and improved customer experiences.
- Astera: Astera DW Builder, an automated data warehousing solution, empowers users to create and deploy data warehouses without manual code writing. The software streamlines data operations, such as data mapping, ensuring a more efficient and cost-effective process.
- Zalando: The e-commerce giant Zalando utilized automation to accelerate data warehousing tasks, particularly in generating product recommendations and delivering personalized shopping experiences. This strategic move positioned them as a leading force in the fashion industry.

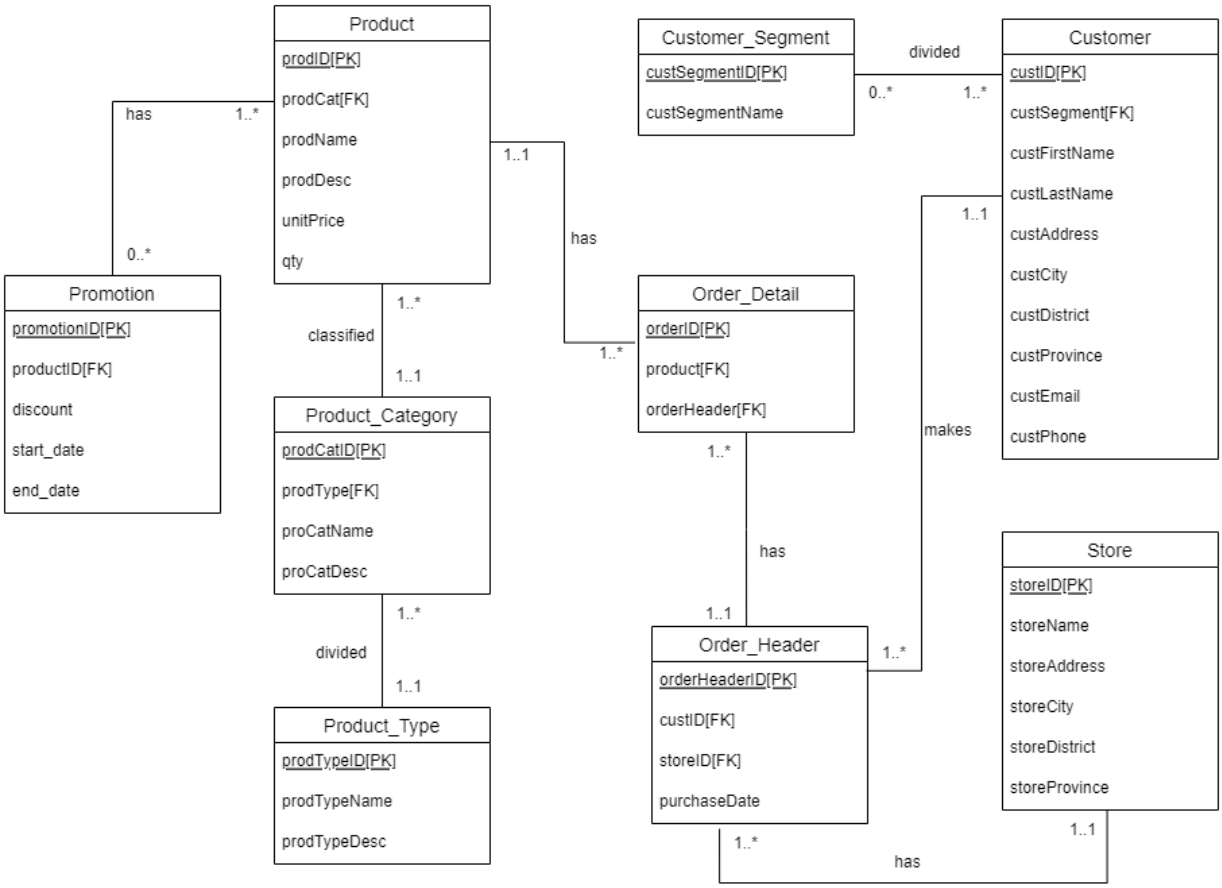
In conclusion, Data Warehousing Automation (DWA) represents a pivotal transformation in the data warehousing landscape, offering a solution that minimizes manual efforts, enhances efficiency, improves data quality, and fosters data-driven decision-making. As demonstrated by real-world examples from organizations like LendingClub, Logitech, Astera, and Zalando, DWA has become a strategic imperative, driving enhanced processes, faster insights, and superior customer experiences. Its evolution over the years has enabled it to adapt to the dynamic business environment, incorporating the latest technologies such as cloud data integration and real-time data extraction. In a world where data complexity and volume continue to grow, DWA stands as a crucial tool for organizations looking to maintain their competitive edge and excel in the realm of data management and business intelligence. Embracing this state-of-the-art technology is not merely a trend but a fundamental step toward staying at the forefront of data-savvy businesses in the modern landscape.

References:

1. https://www.astera.com/knowledge-center/data-warehouse-automation-a-complete-guide/
2. https://pixelplex.io/blog/data-warehouse-automation/
3. https://research.aimultiple.com/data-warehouse-automation/

SECTION B

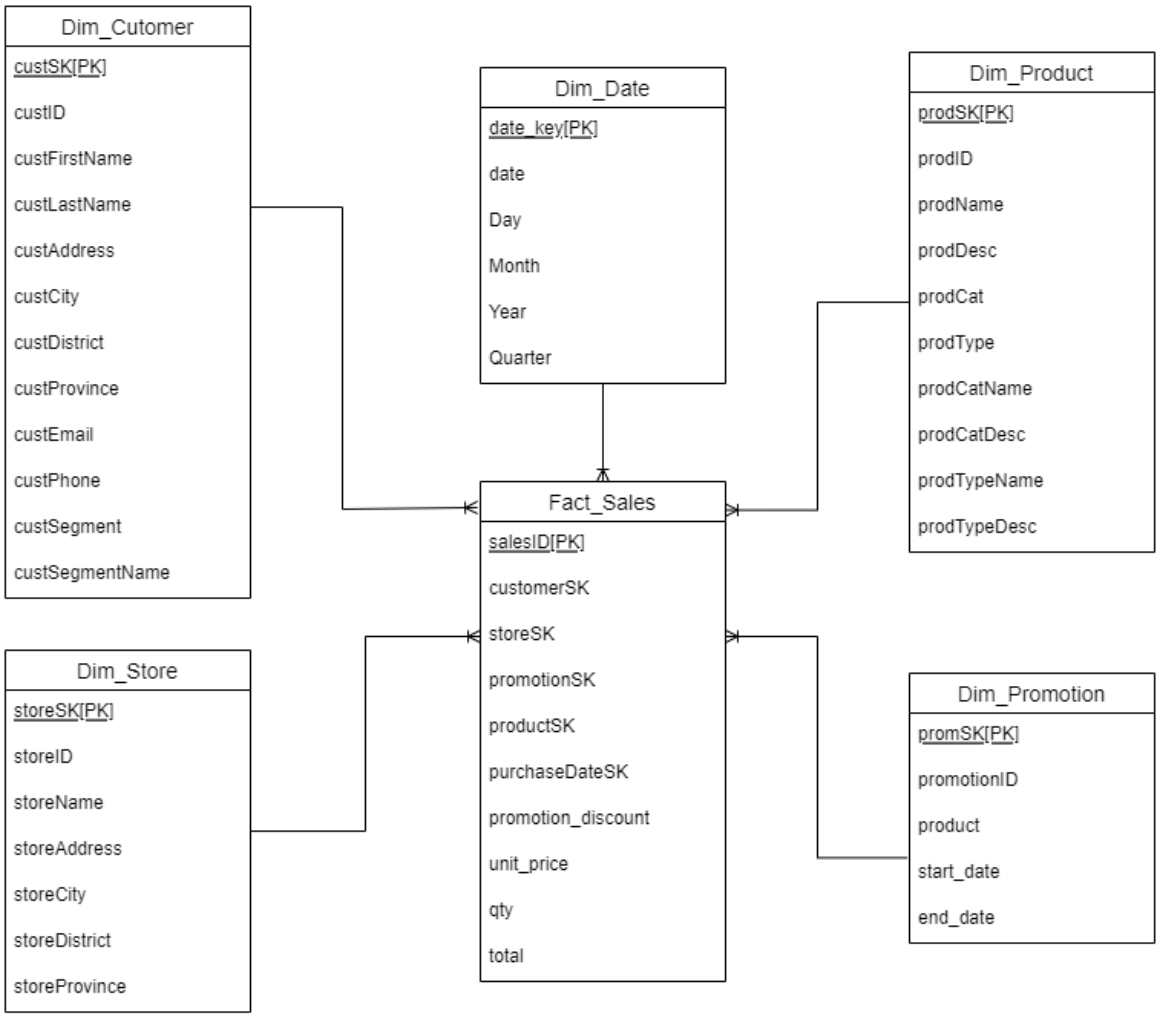
- **PART 1**



- **PART 2**

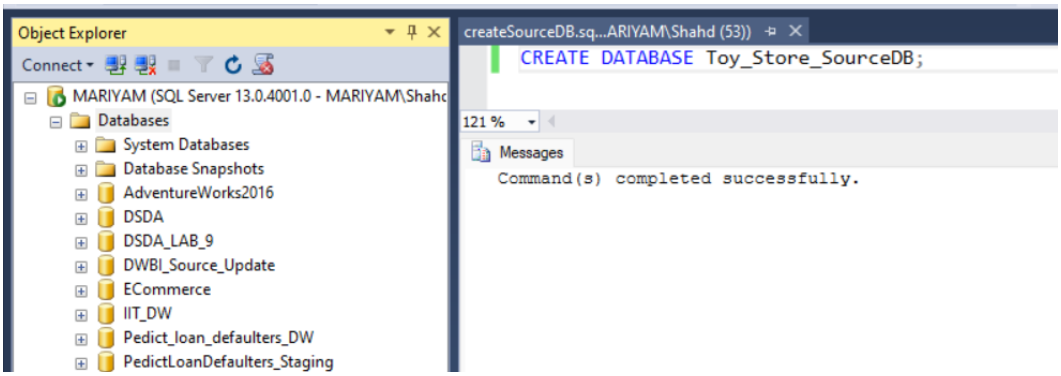
Source Table	Source Column Name	Target Table	Target Column	Dimension/Fact
Customer	custID	Dim_Customer	custID	Dimension
	custFirstName	Dim_Customer	custFirstName	Dimension
	custLastName	Dim_Customer	custLastName	Dimension
	custAddress	Dim_Customer	custAddress	Dimension
	custCity	Dim_Customer	custCity	Dimension
	custDistrict	Dim_Customer	custDistrict	Dimension
	custProvince	Dim_Customer	custProvince	Dimension
	custEmail	Dim_Customer	custEmail	Dimension
	custPhone	Dim_Customer	custPhone	Dimension
Customer_Segment	custSegmentID	Dim_Customer	custSegment	Dimension
	custSegmentName	Dim_Customer	custSegmentName	Dimension
Store	StoreID	Dim_Store	StoreID	Dimension
	storeName	Dim_Store	storeName	Dimension
	storeAddress	Dim_Store	storeAddress	Dimension
	storeCity	Dim_Store	storeCity	Dimension
	storeDistrict	Dim_Store	storeDistrict	Dimension
	storeProvince	Dim_Store	storeProvince	Dimension
Product_Type	prodTypeID	Dim_Product	prodType	Dimension
	prodTypeName	Dim_Product	prodTypeName	Dimension
	prodTypeDesc	Dim_Product	prodTypeDesc	Dimension
Product_Category	prodCatID	Dim_Product	prodCat	Dimension
	prodCatName	Dim_Product	prodCatName	Dimension
	prodCatDesc	Dim_Product	prodCatDesc	Dimension

- **PART 3**

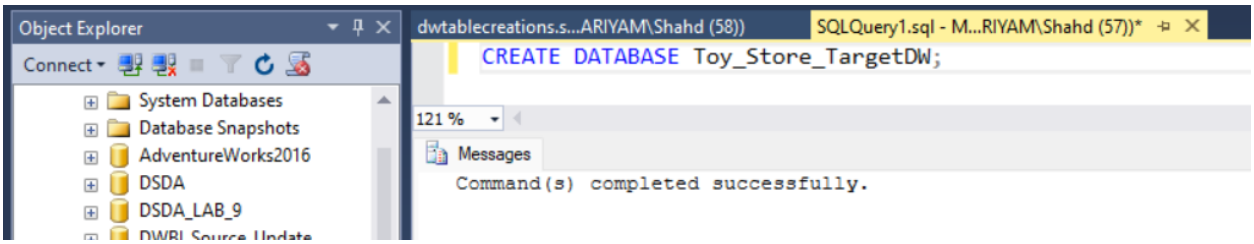


- **PART 4**

❖ Creating database for source tables

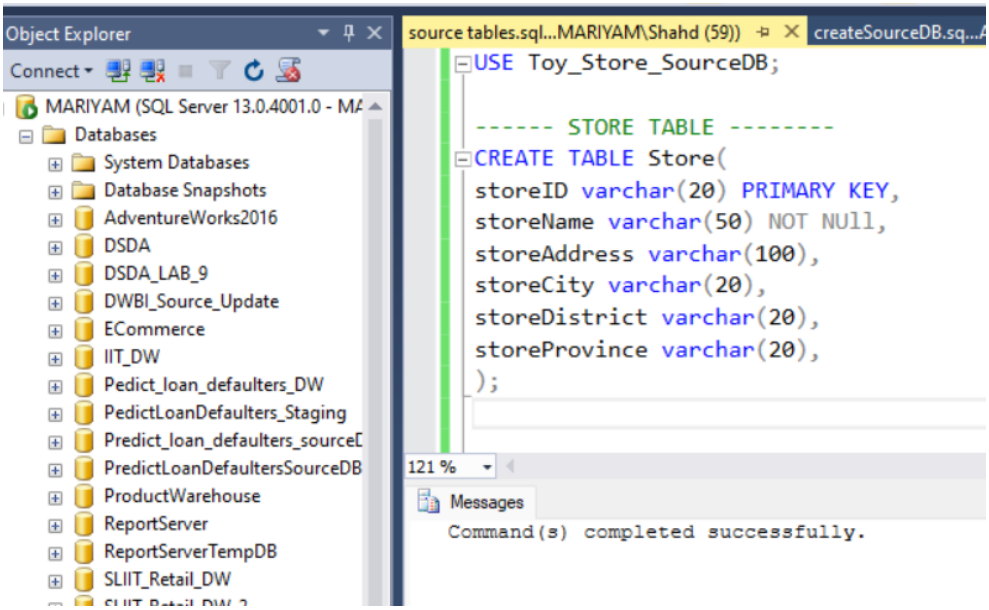


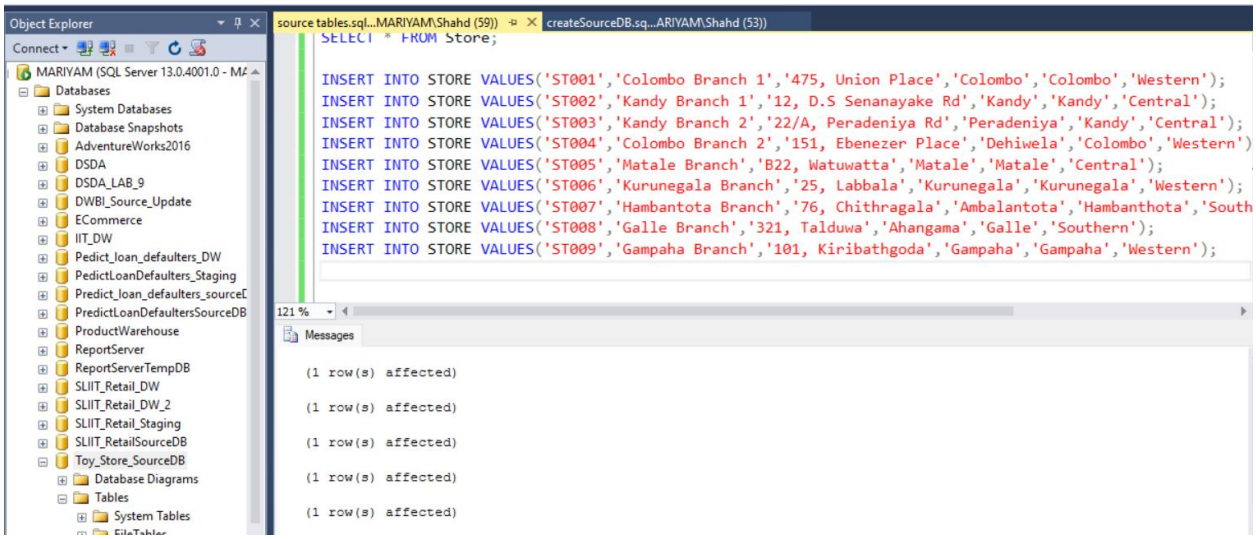
❖ Creating the target database



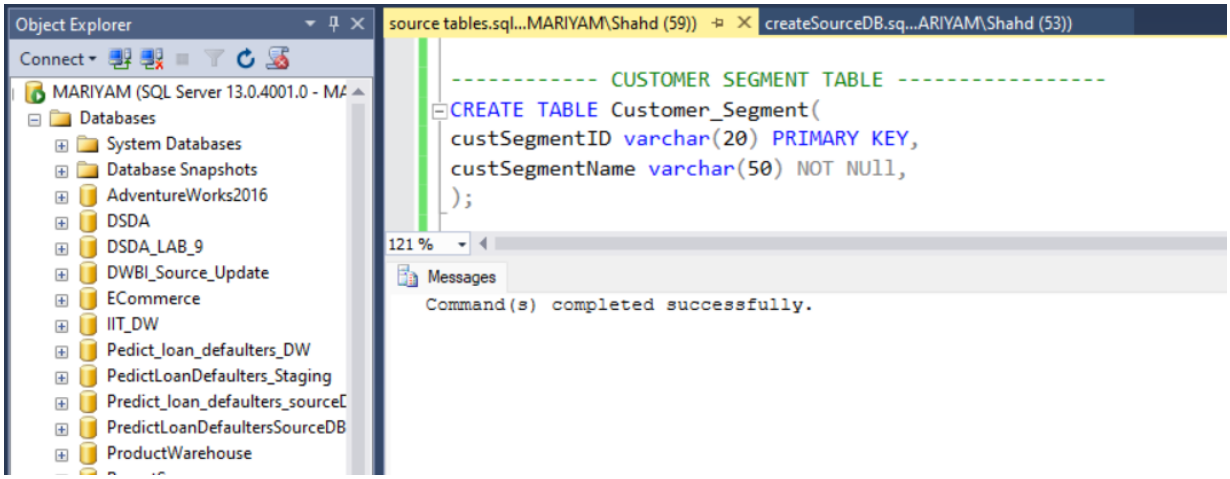
❖ Creating Source Tables

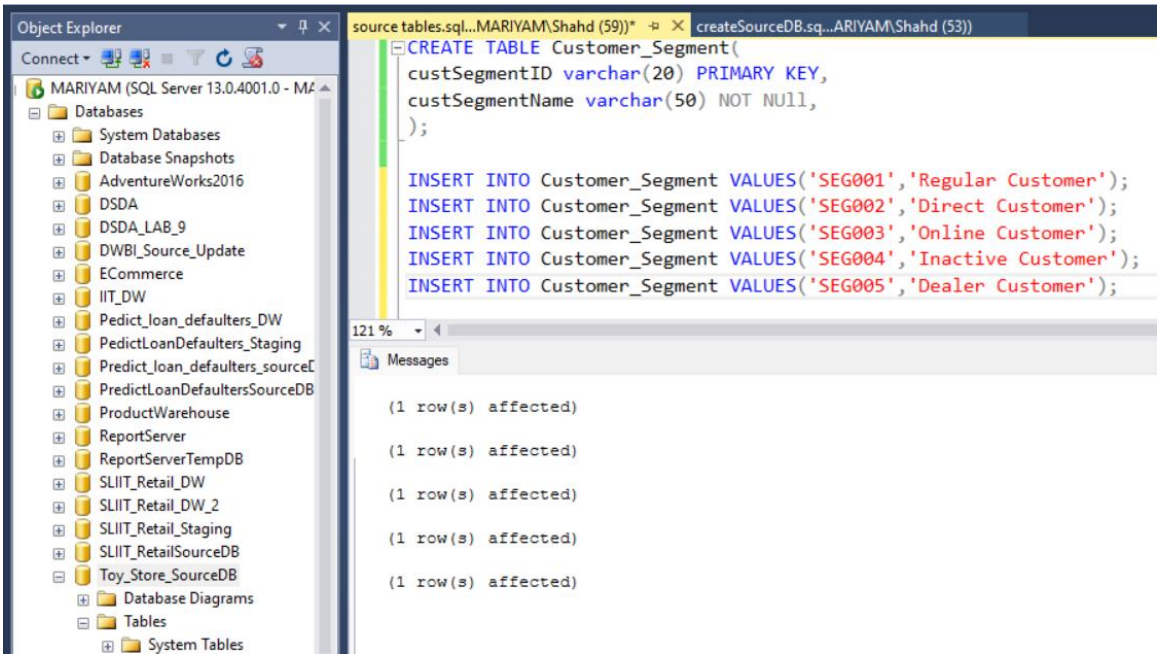
- Store table



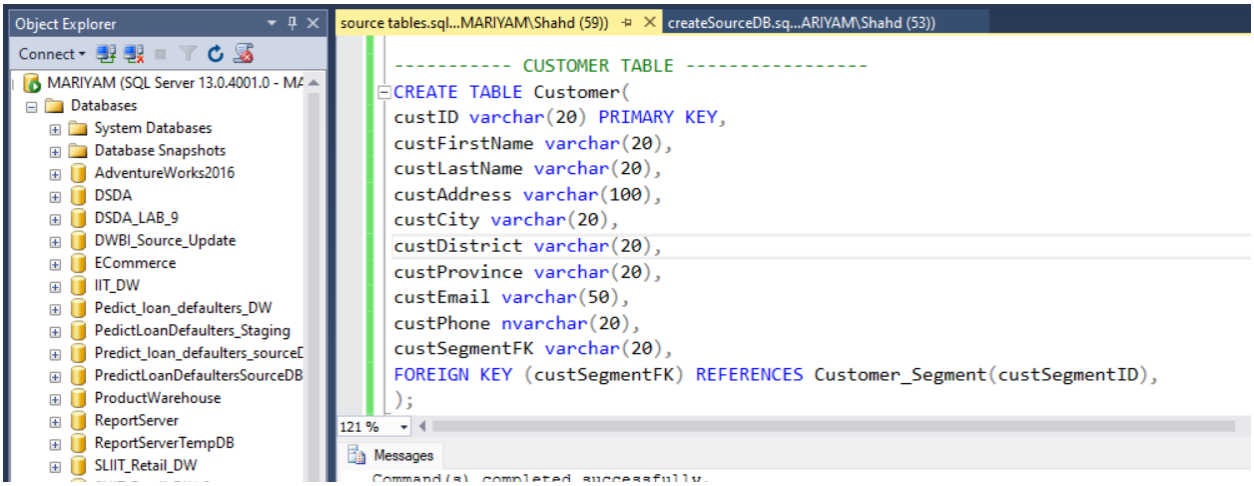


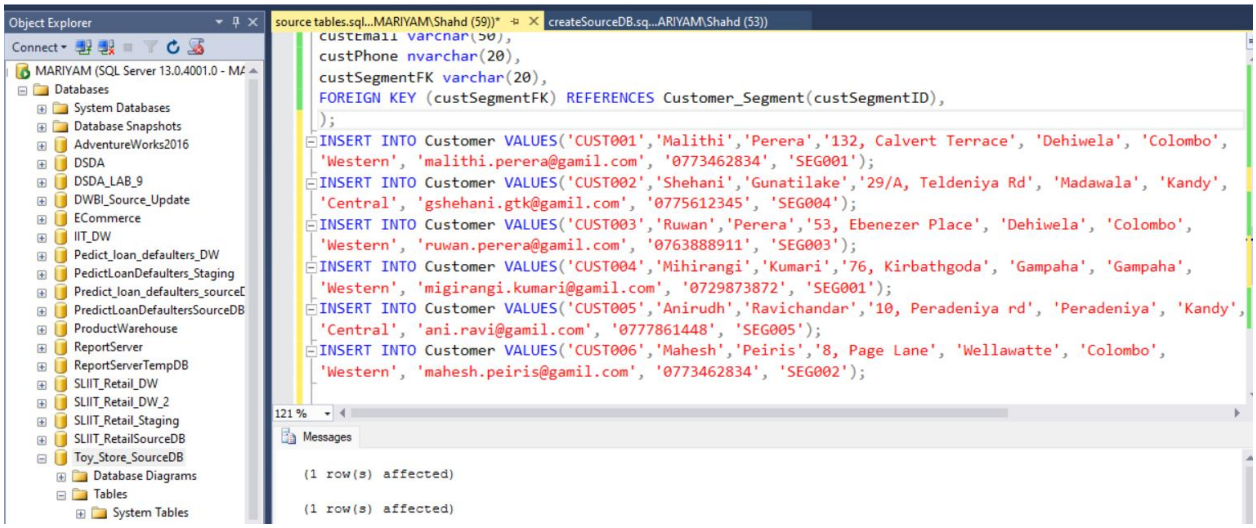
- Customer Segment Table



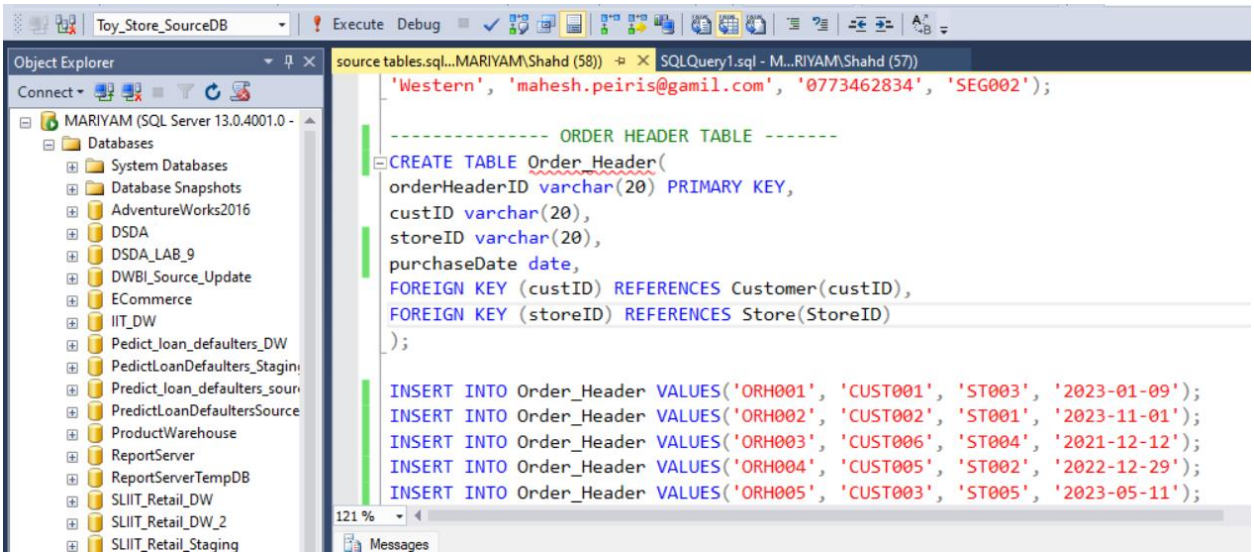


- Customer Table

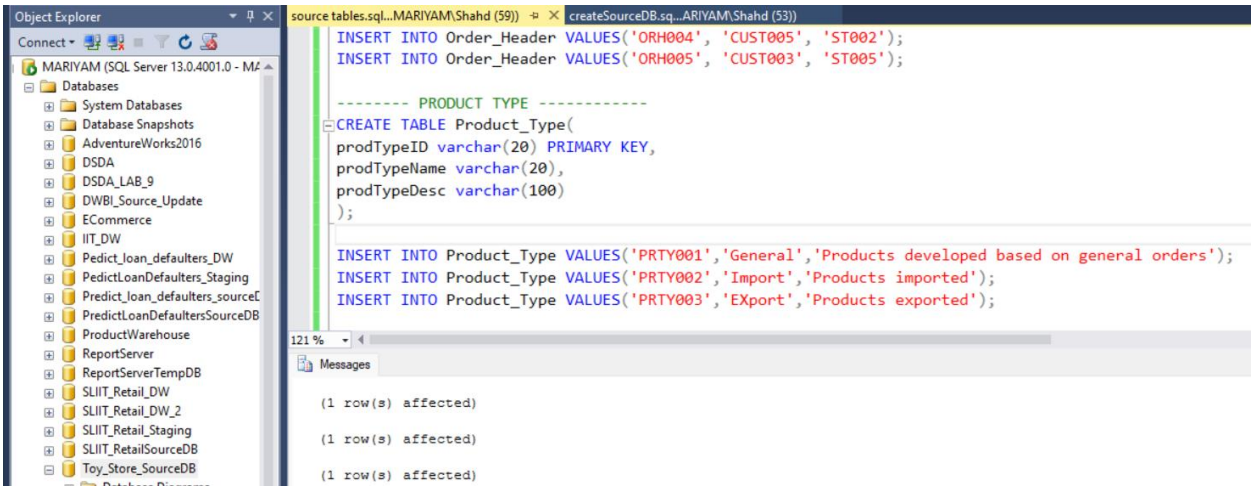




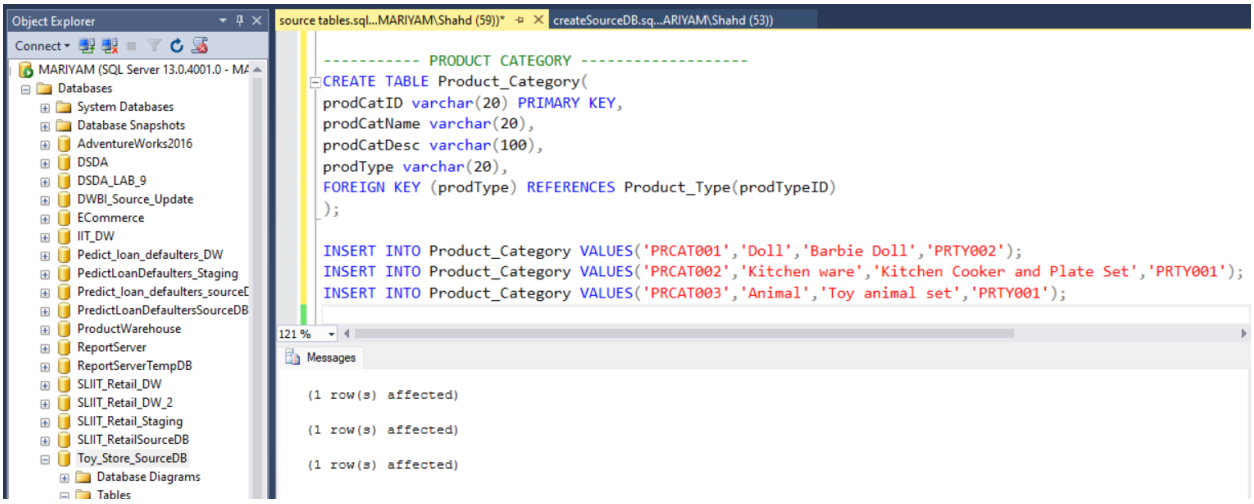
- Order Header Table



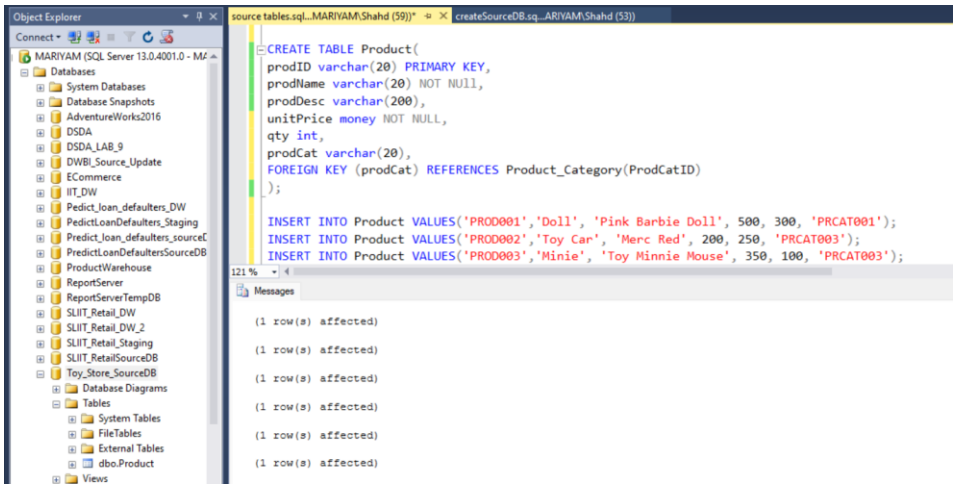
- Product Type Table



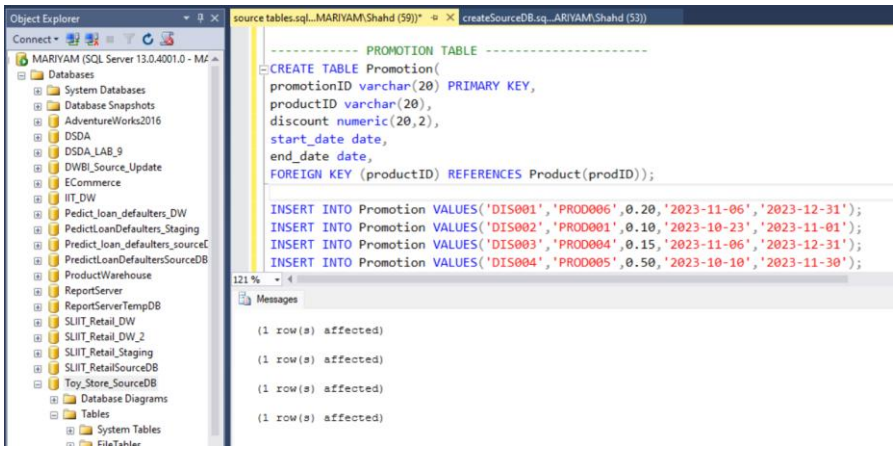
- Product Category Table



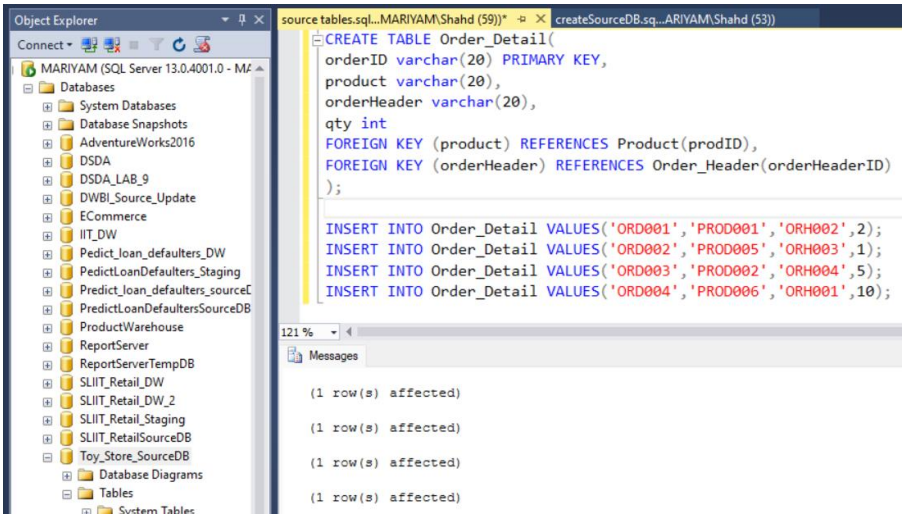
- Product Table



- Promotion Table

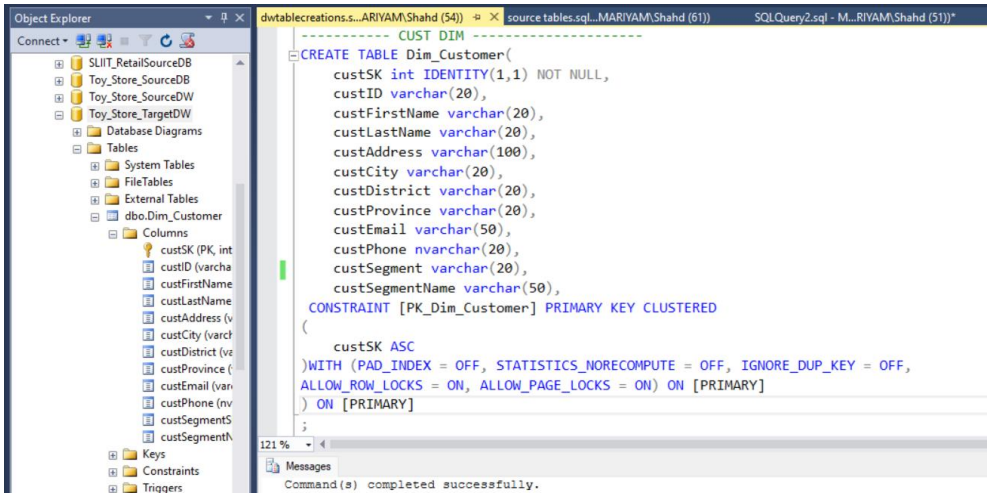


- Order Details Table

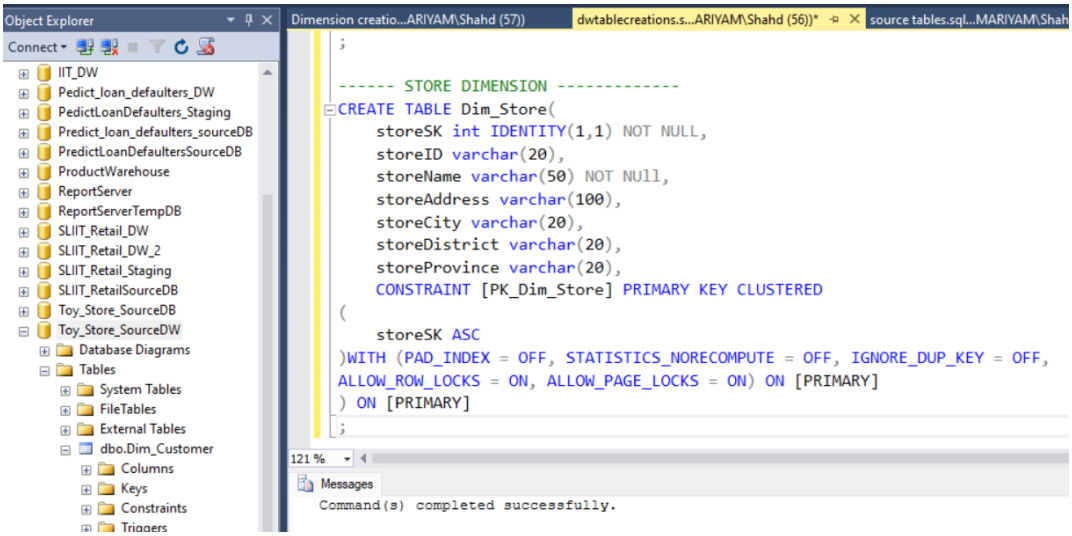


❖ Datawarehouse Dimension and Fact Creation

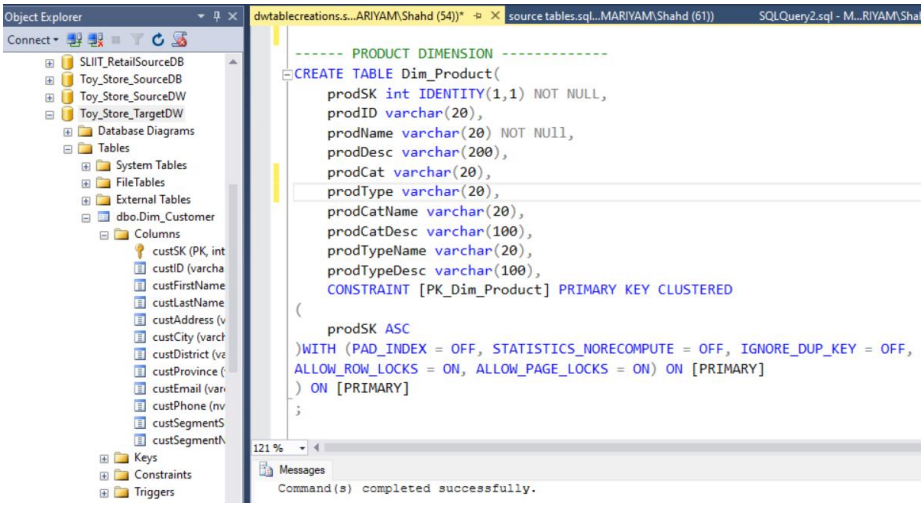
- Dim Customer



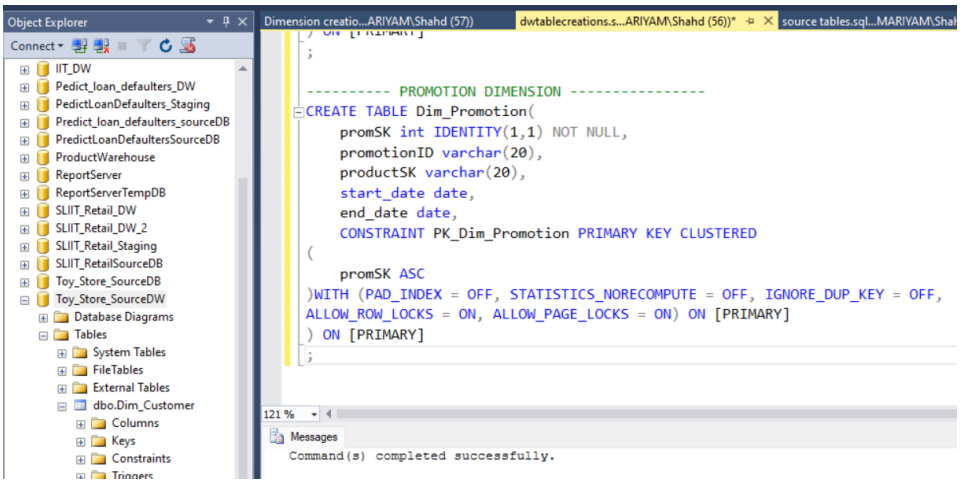
- Store Dimension



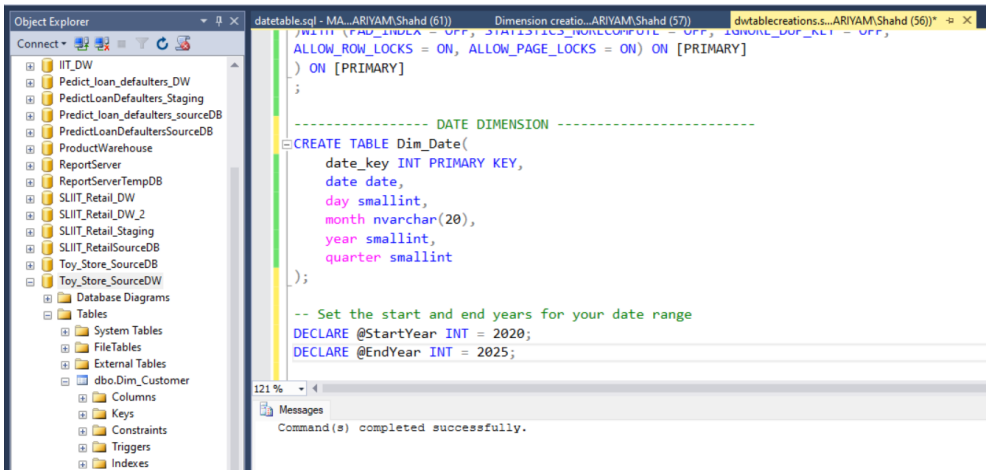
- Product Dimension



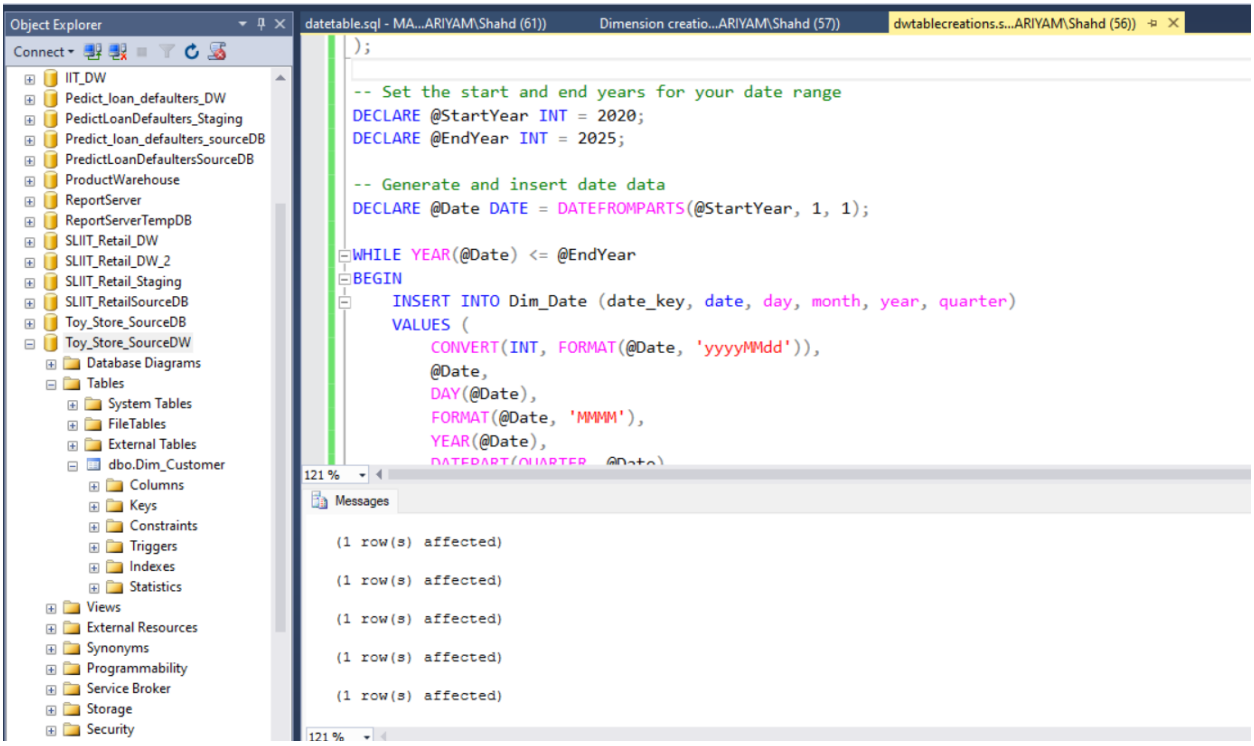
- Promotion Dimension



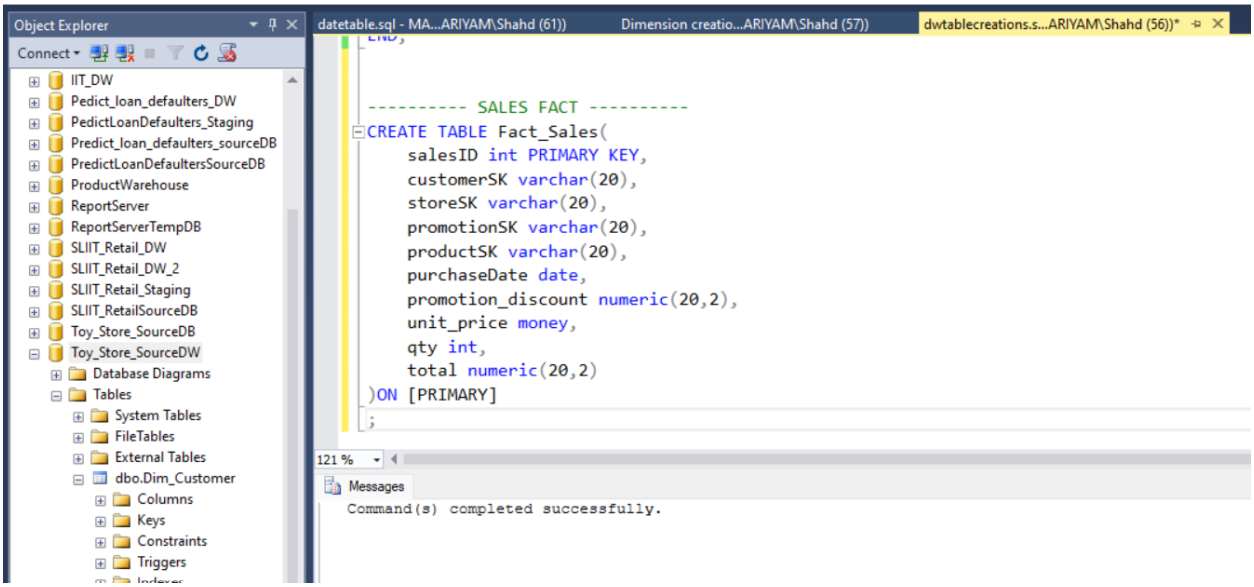
- Date Dimension



Loading Data to Date Dimension



- Fact Sales


Object Explorer
Connect
IIT_DW
Pedict_loan_defaulters_DW
PedictLoanDefaulters_Staging
Predict_loan_defaulters_sourceDB
PredictLoanDefaultersSourceDB
ProductWarehouse
ReportServer
ReportServerTempDB
SLIIT_Retail_DW
SLIIT_Retail_DW_2
SLIIT_Retail_Staging
SLIIT_RetailSourceDB
Toy_Store_SourceDB
Toy_Store_SourceDW
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Dim_Customer
Columns
Keys
Constraints
Triggers
datetable.sql - MA...ARIYAM\Shahd (61))
Dimension creatio...ARIYAM\Shahd (57))
dwtablecreations.s...ARIYAM\Shahd (56))*
END;
---------- SALES FACT ----------
CREATE TABLE Fact_Sales(
salesID int PRIMARY KEY,
customerSK varchar(20),
storeSK varchar(20),
promotionSK varchar(20),
productSK varchar(20),
purchaseDate date,
promotion_discount numeric(20,2),
unit_price money,
qty int,
total numeric(20,2)
)ON [PRIMARY]
;
121 %
Messages
Command(s) completed successfully.

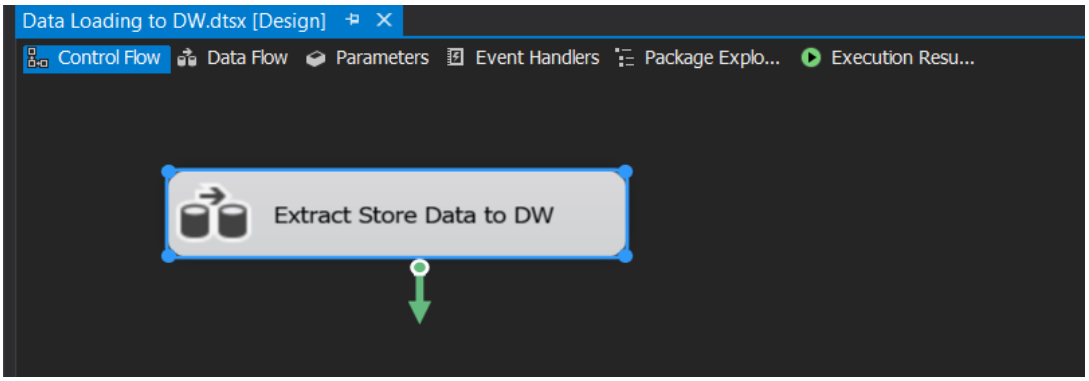
SECTION C

- **Part 1**

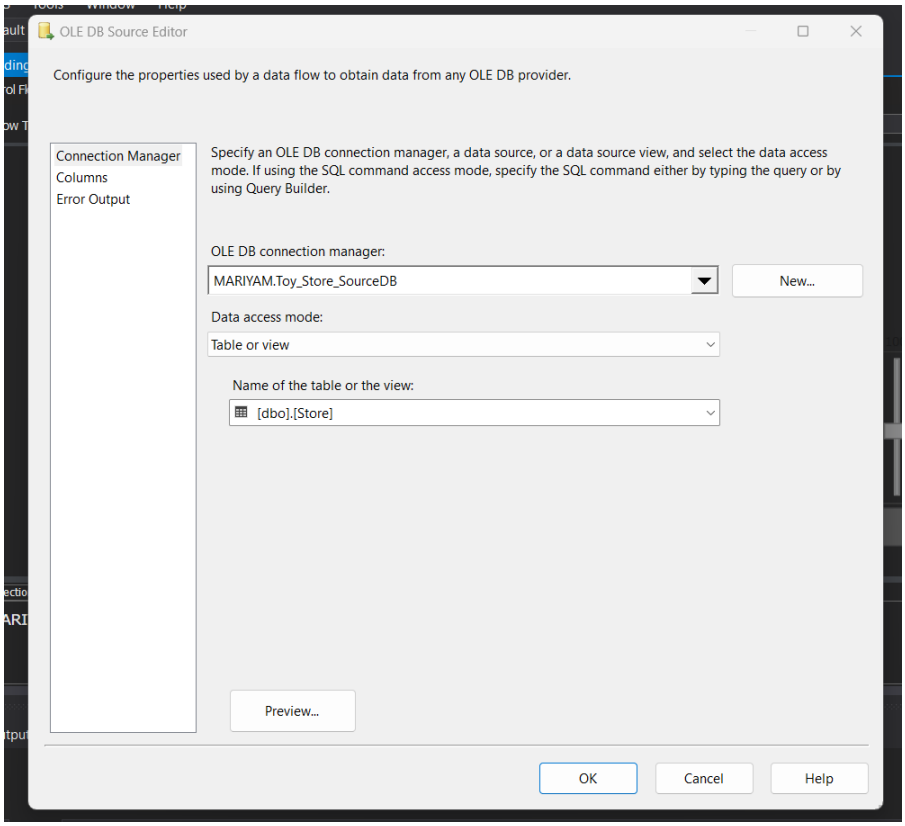
A new integration project is created for extracting and loading data to data warehouse.

❖ Store dimension

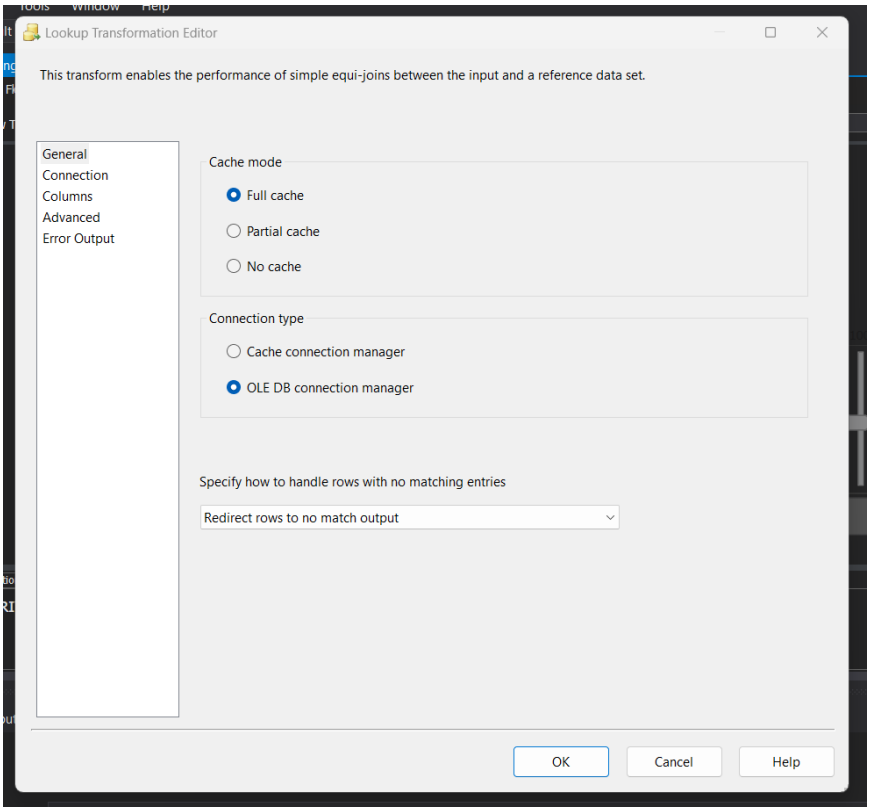
- Creating a ‘Data Flow Task’ to extract data from Store table to Store Dimension in the Control Flow area



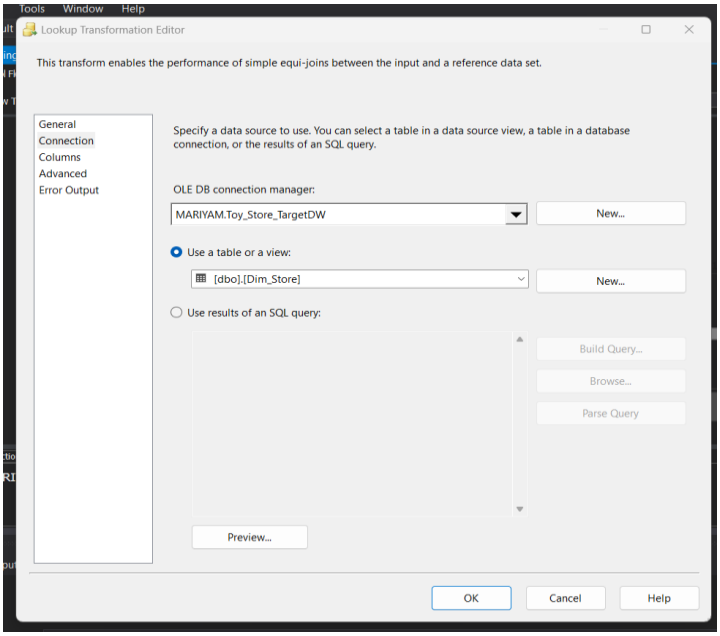
- Adding Source Assistant/ OLE DB Source and connecting it to the Store Table in the source database



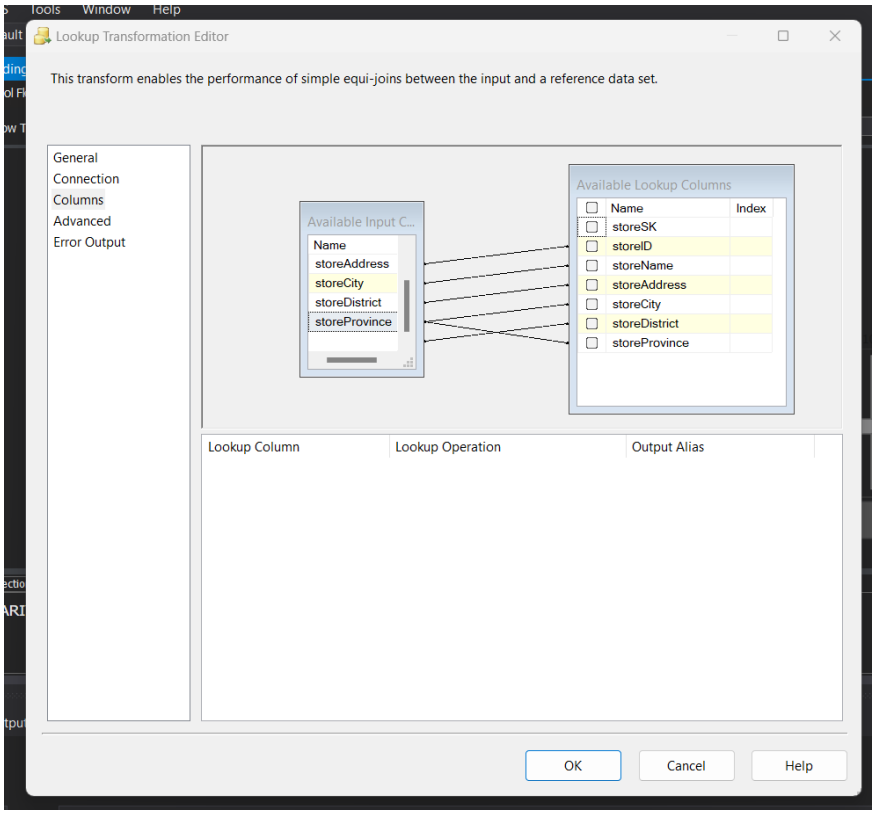
- Adding a Lookup transformation:
  - To handle rows we select ‘Redirect rows to no match output’ so that it fetches only the new records



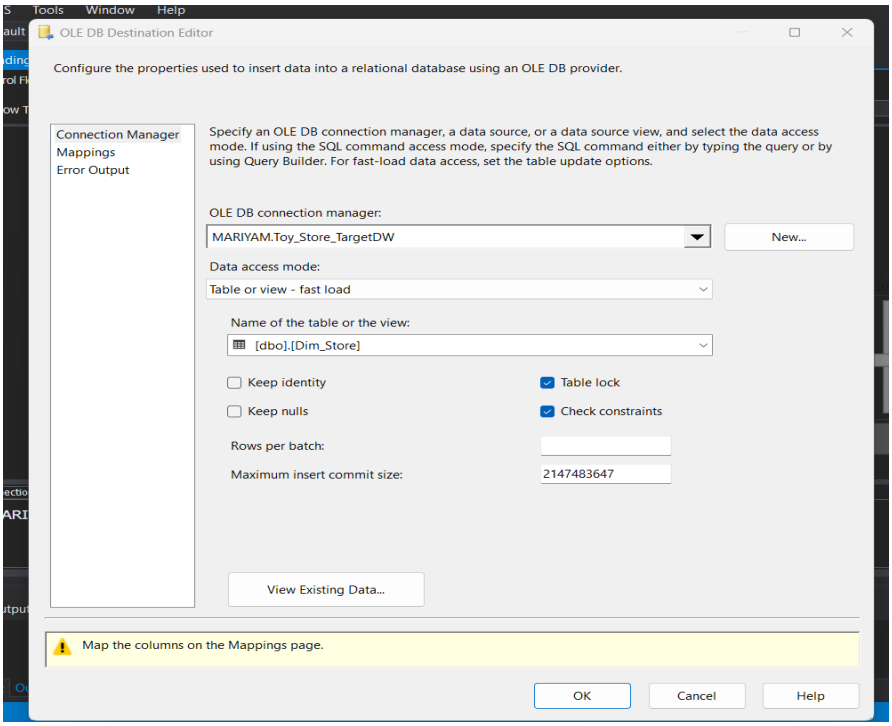
  - Selecting the target data warehouse and the respective dimension table



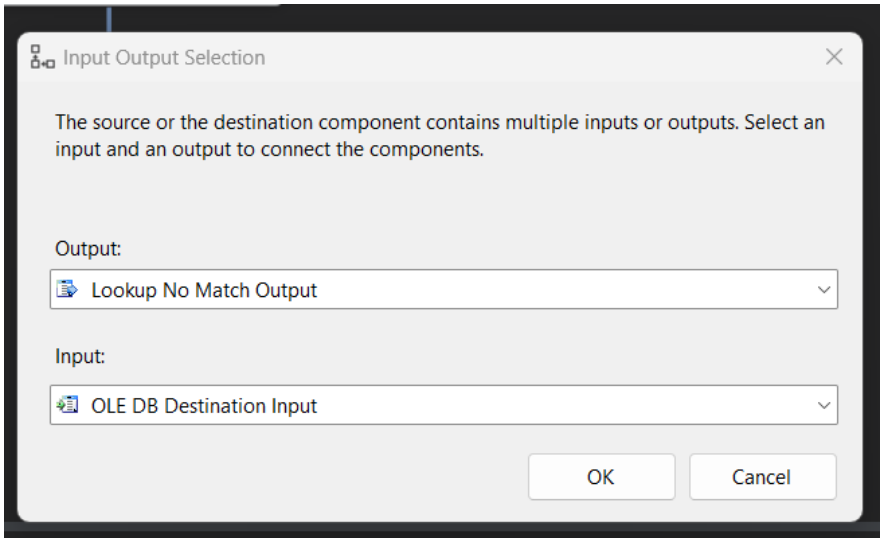
  - Mapping the columns between source and target dimension table



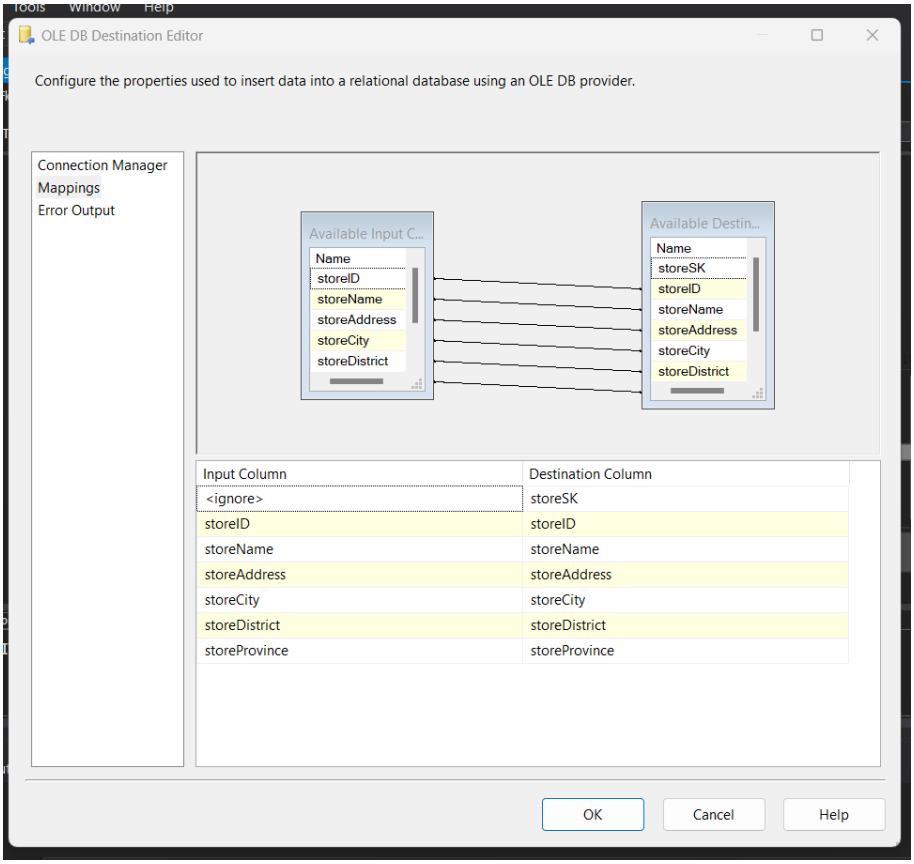
- Adding Destination Assistant/ OLE DB Destination and connecting it to the Store Dimension in the Target Data warehouse



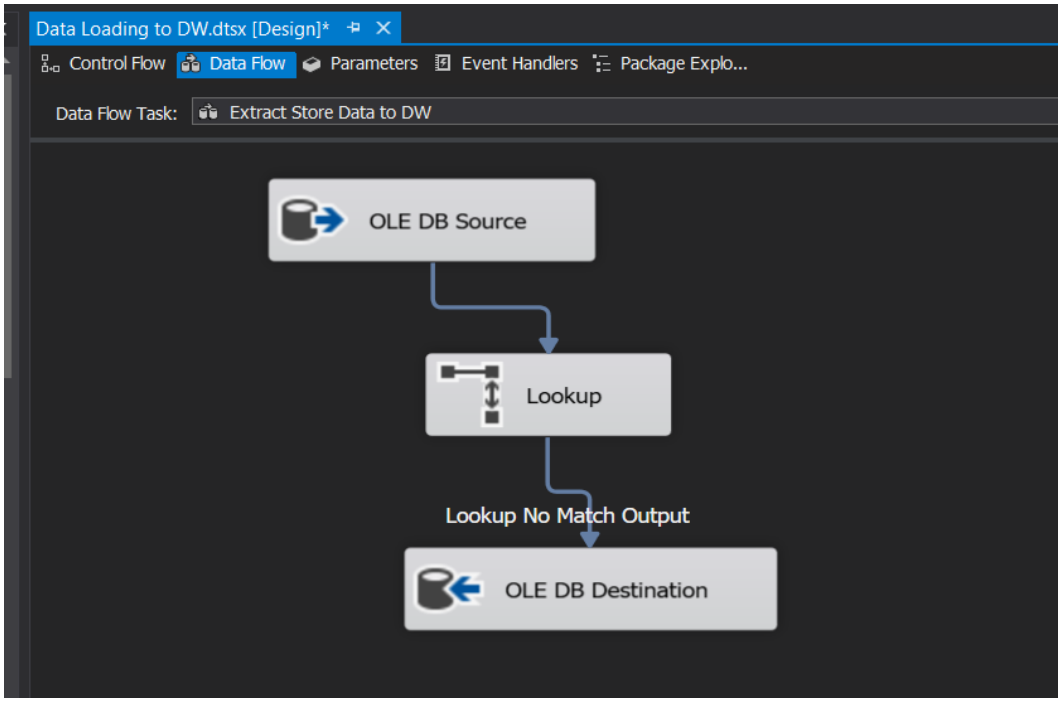
- Now connect the blue arrow from lookup with the destination assistant. Select the configurations as represented below



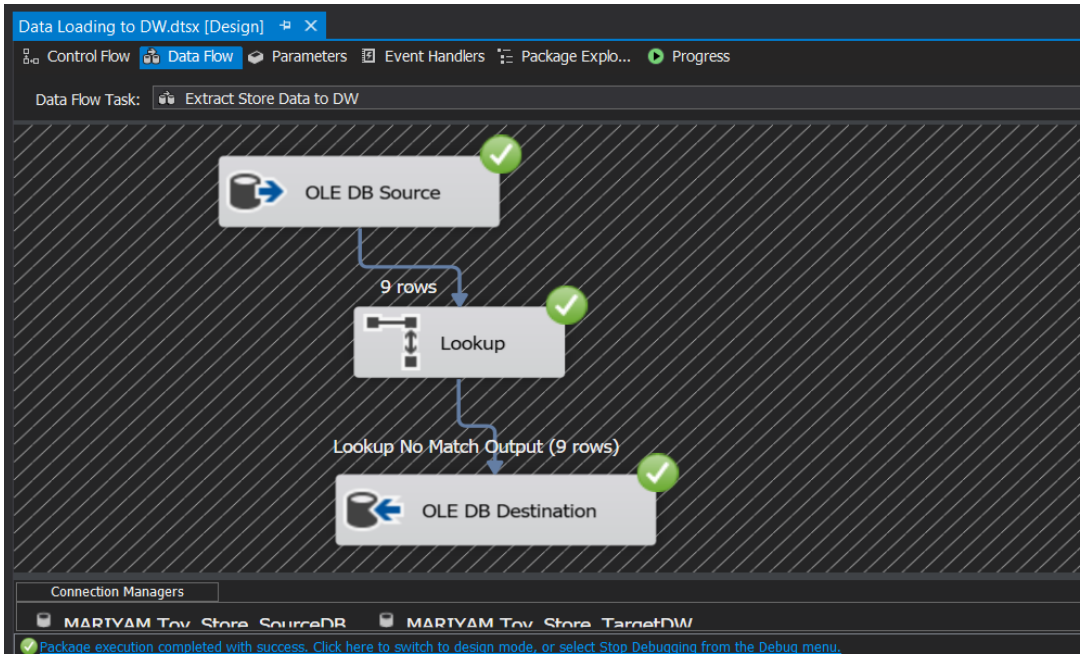
- Double click on the destination assistant and verify if the mappings are correct.



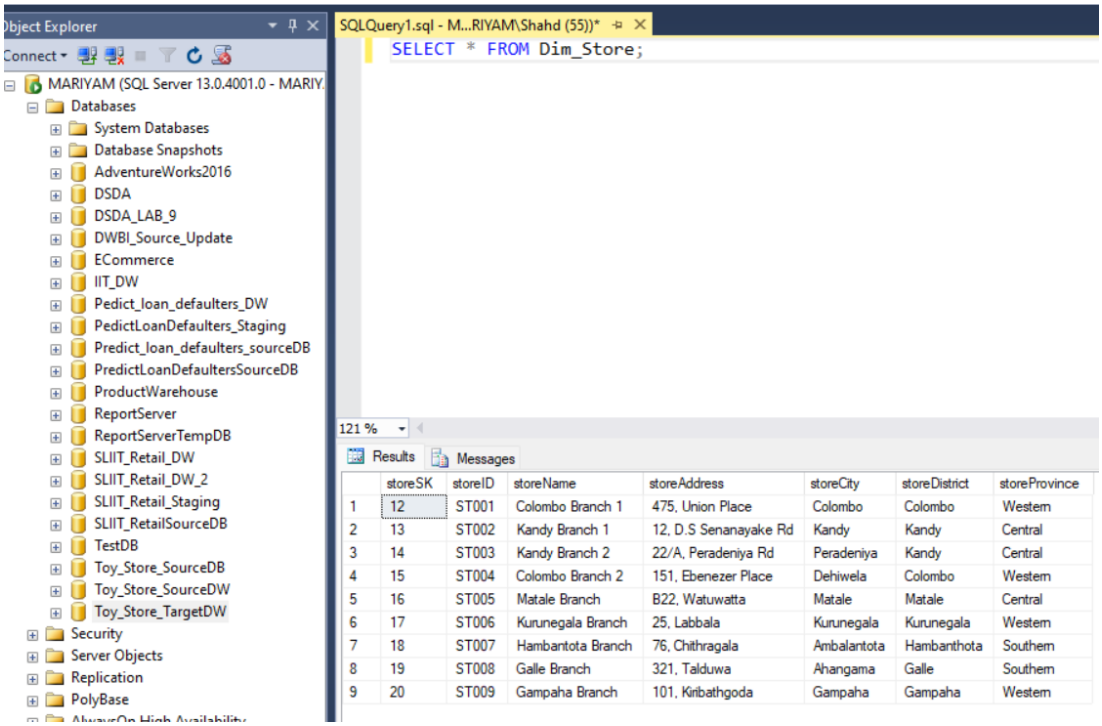
- Completed Data Flow Task is represented below



- Executing the package



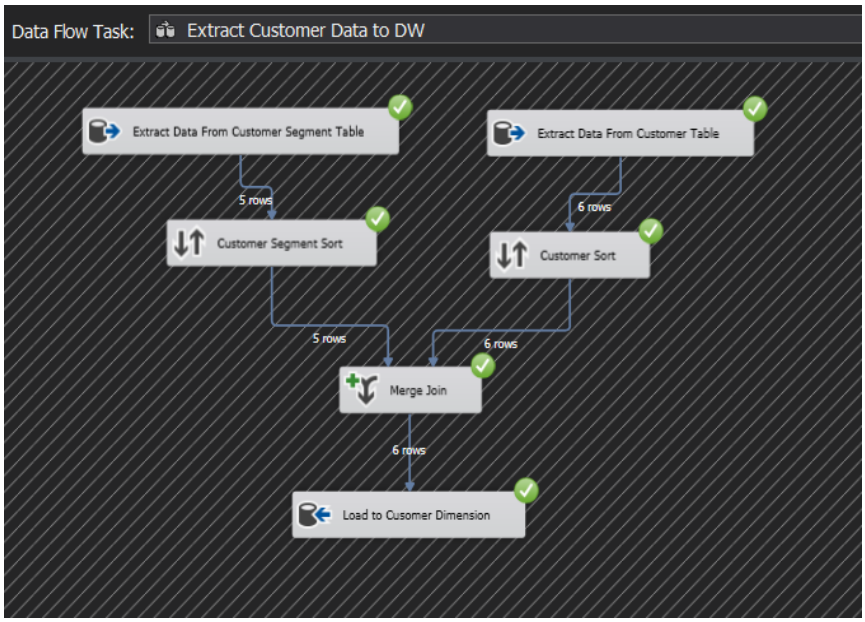
- Verifying it in MS SQL Server Management Studio

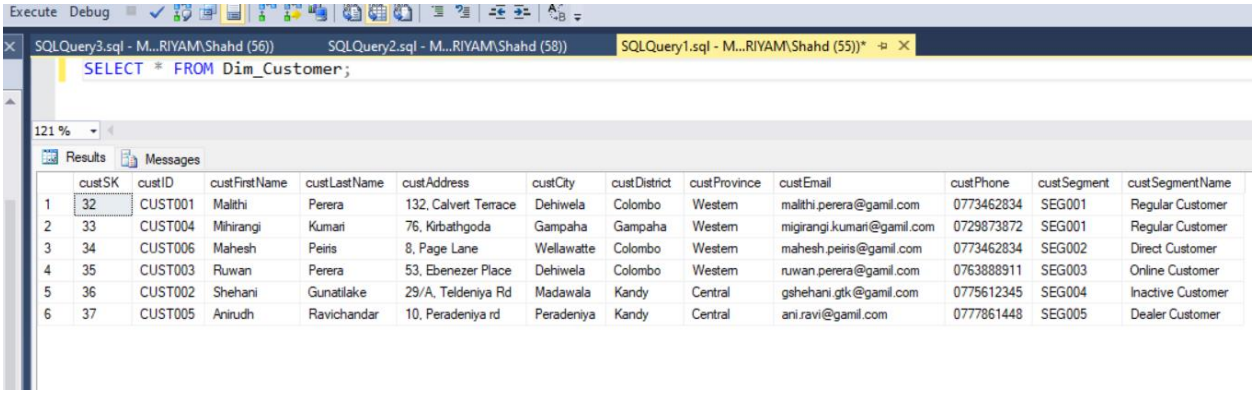


	storeSK	storeID	storeName	storeAddress	storeCity	storeDistrict	storeProvince
1	12	ST001	Colombo Branch 1	475, Union Place	Colombo	Colombo	Western
2	13	ST002	Kandy Branch 1	12, D.S Senanayake Rd	Kandy	Kandy	Central
3	14	ST003	Kandy Branch 2	22/A, Peradeniya Rd	Peradeniya	Kandy	Central
4	15	ST004	Colombo Branch 2	151, Ebenezer Place	Dehiwela	Colombo	Western
5	16	ST005	Matale Branch	B22, Watuwatta	Matale	Matale	Central
6	17	ST006	Kurunegala Branch	25, Labbala	Kurunegala	Kurunegala	Western
7	18	ST007	Hambantota Branch	76, Chithragala	Ambalantota	Hambanthota	Southern
8	19	ST008	Galle Branch	321, Talduwa	Ahangama	Galle	Southern
9	20	ST009	Gampaha Branch	101, Kiribathgoda	Gampaha	Gampaha	Western

❖ Follow the same procedure to extract the data from other source tables to their respective dimensions using correct methods

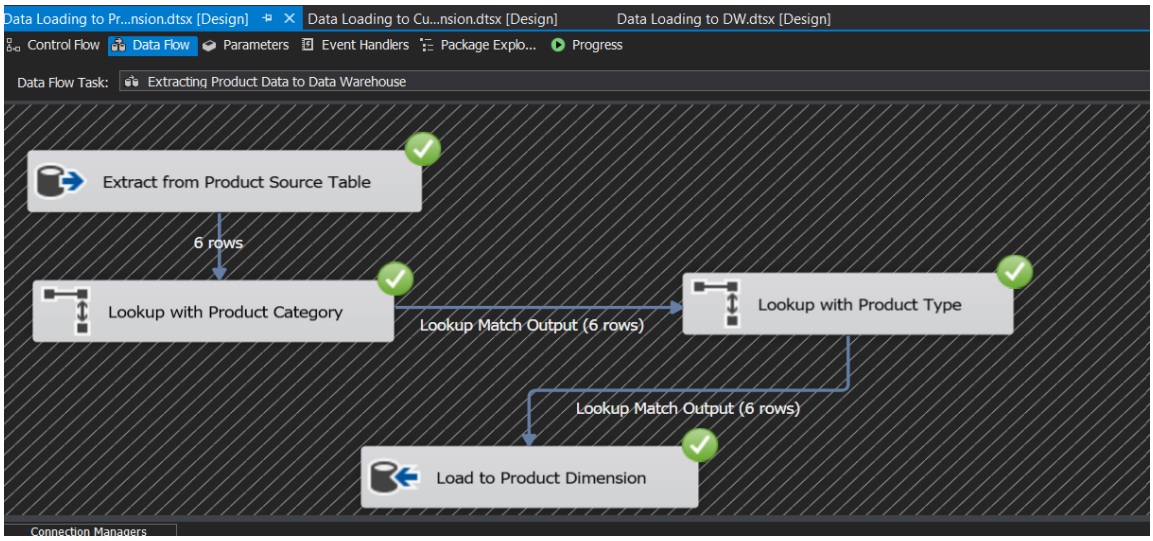
- Loading data to Customer dimension

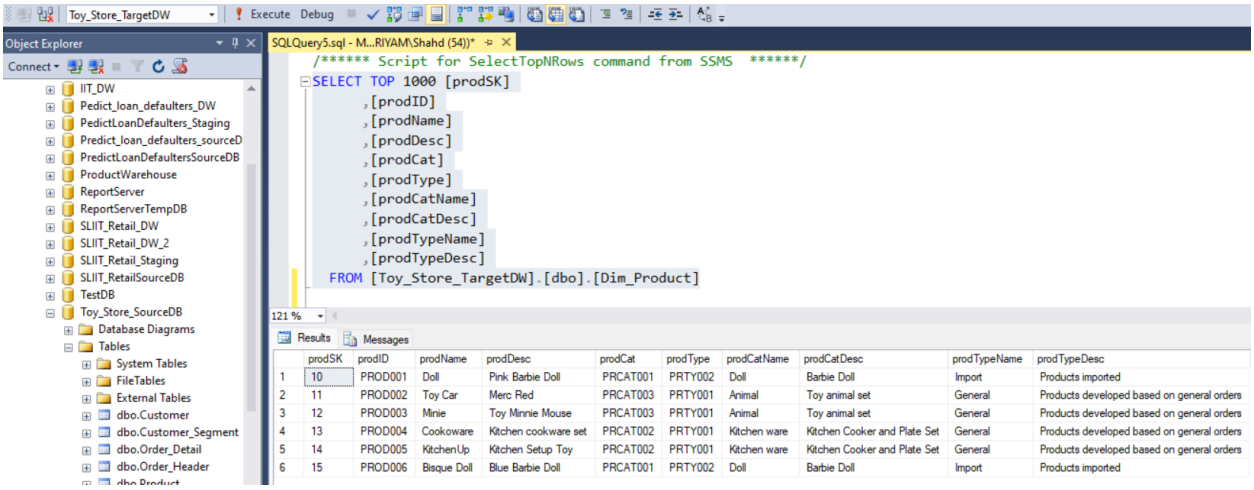




	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName
1	32	CUST001	Malithi	Perera	132, Calvert Terrace	Dehiwela	Colombo	Western	malithi.perera@gamil.com	0773462834	SEG001	Regular Customer
2	33	CUST004	Mihirangi	Kumari	76, Kirbathgoda	Gampaha	Gampaha	Western	migirangi.kumari@gamil.com	0729873872	SEG001	Regular Customer
3	34	CUST006	Mahesh	Peiris	8, Page Lane	Wellawatte	Colombo	Western	mahesh.peiris@gamil.com	0773462834	SEG002	Direct Customer
4	35	CUST003	Ruwan	Perera	53, Ebenezer Place	Dehiwela	Colombo	Western	ruwan.perera@gamil.com	0763888911	SEG003	Online Customer
5	36	CUST002	Shehani	Gunatilake	29/A, Teldeniya Rd	Madawala	Kandy	Central	gshehani.gtk@gamil.com	0775612345	SEG004	Inactive Customer
6	37	CUST005	Anirudh	Ravichandar	10, Peradeniya rd	Peradeniya	Kandy	Central	ani.ravi@gamil.com	0777861448	SEG005	Dealer Customer

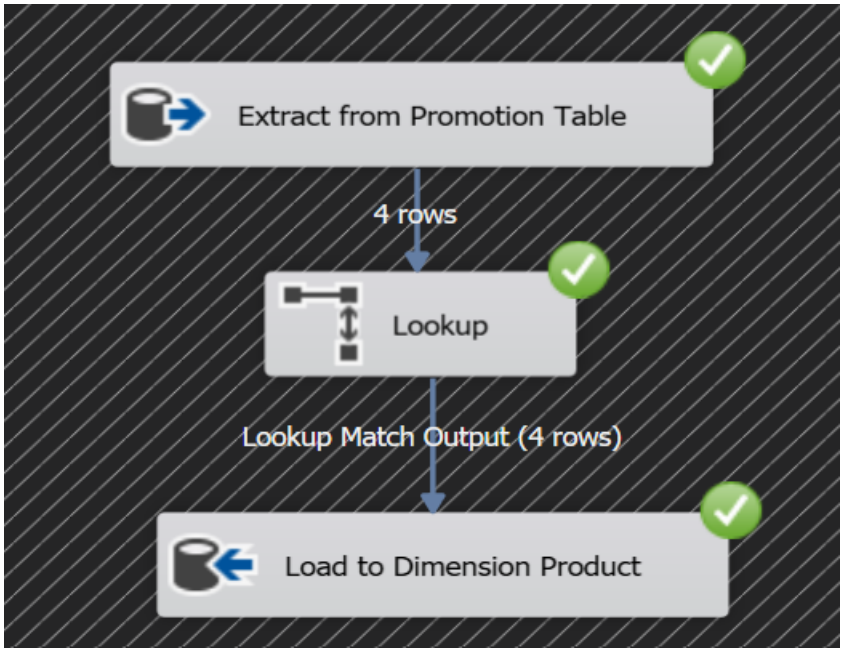
- Loading Data to Product Dimension





	prodSK	prodID	prodName	prodDesc	prodCat	prodType	prodCatName	prodCatDesc	prodTypeName	prodTypeDesc
1	10	PROD001	Doll	Pink Barbie Doll	PRCAT001	PRTY002	Doll	Barbie Doll	Import	Products imported
2	11	PROD002	Toy Car	Merc Red	PRCAT003	PRTY001	Animal	Toy animal set	General	Products developed based on general orders
3	12	PROD003	Minie	Toy Minnie Mouse	PRCAT003	PRTY001	Animal	Toy animal set	General	Products developed based on general orders
4	13	PROD004	Cookoware	Kitchen cookware set	PRCAT002	PRTY001	Kitchen ware	Kitchen Cooker and Plate Set	General	Products developed based on general orders
5	14	PROD005	KitchenUp	Kitchen Setup Toy	PRCAT002	PRTY001	Kitchen ware	Kitchen Cooker and Plate Set	General	Products developed based on general orders
6	15	PROD006	Bisque Doll	Blue Barbie Doll	PRCAT001	PRTY002	Doll	Barbie Doll	Import	Products imported

- Promotion Dimension Loading




```
SELECT TOP 1000 [promSK]
      ,[promotionID]
      ,[productSK]
      ,[start_date]
      ,[end_date]
  FROM [Toy_Store_TargetDW].[dbo].[Dim_Promotion]
```

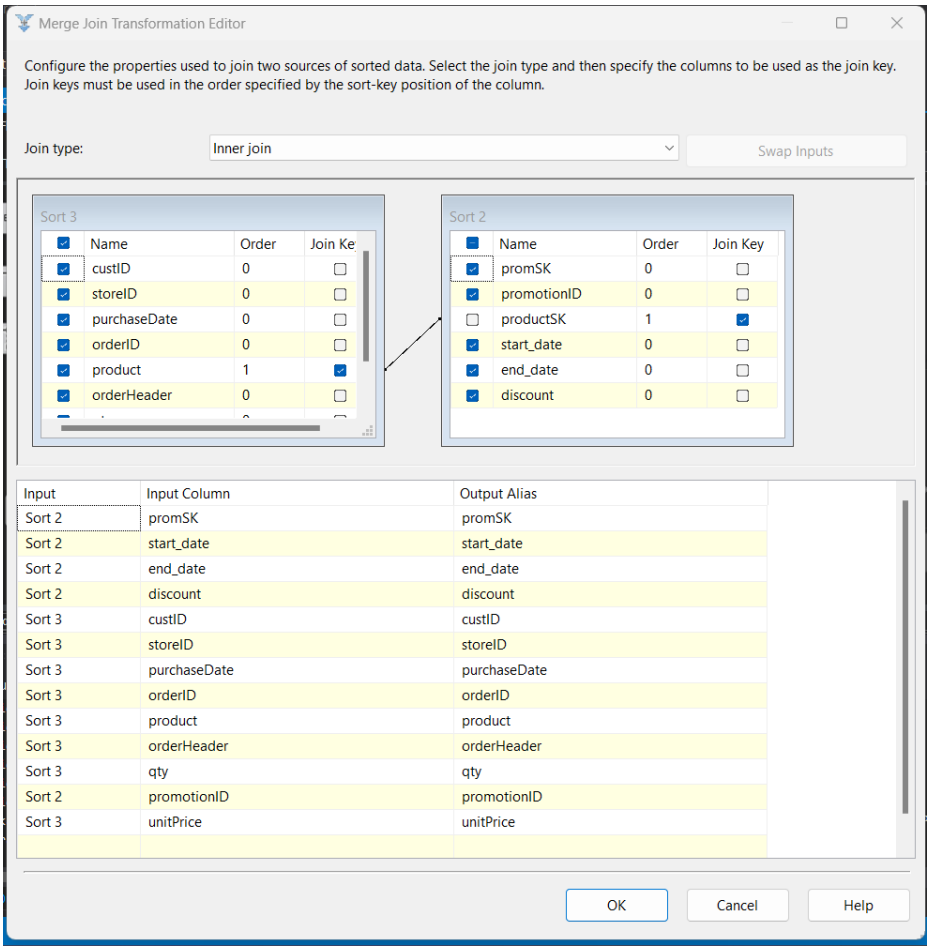
1 %

Results | Messages

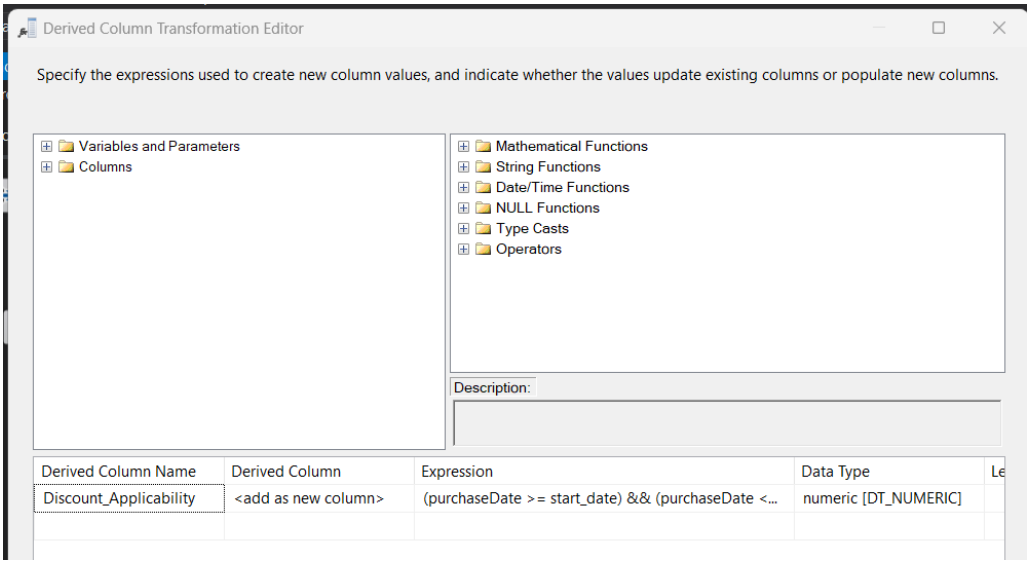
	promSK	promotionID	productSK	start_date	end_date
1	9	DIS001	PROD006	2023-11-06	2023-12-31
2	10	DIS002	PROD001	2023-10-23	2023-11-01
3	11	DIS003	PROD004	2023-11-06	2023-12-31
4	12	DIS004	PROD005	2023-10-10	2023-11-30

❖ All dimension tables are now loaded with data. Next step is to load data to the Fact table:

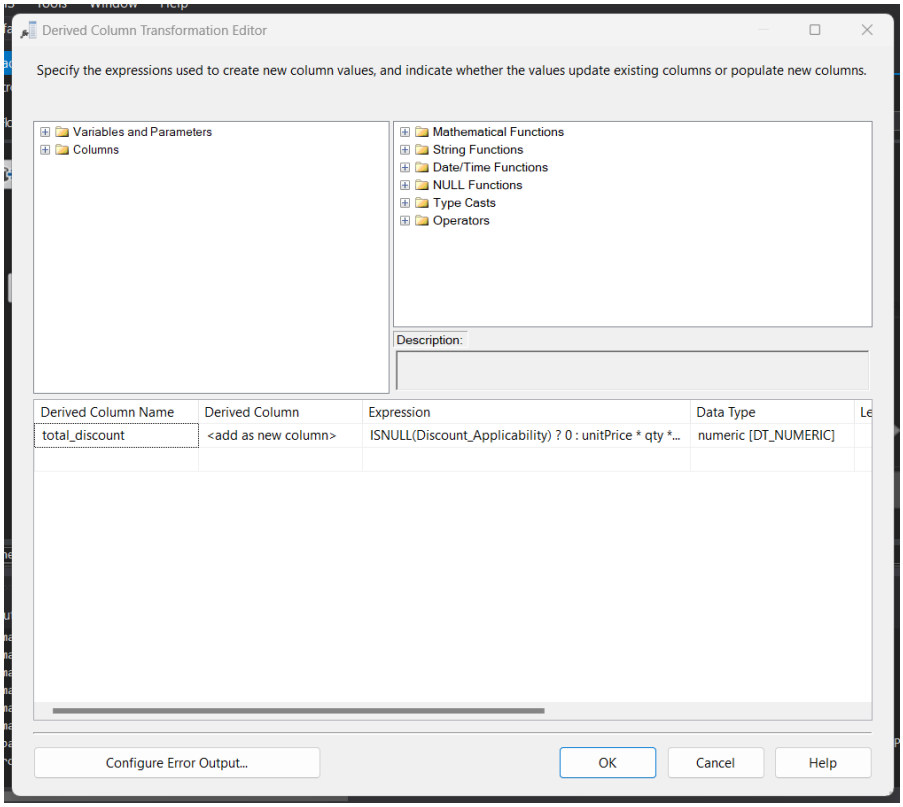
1. Initially a lookup is performed between '*Order_Detail*' table and '*Dim_Product*' to filter based on '*productID'*
2. Another lookup is performed to derive the '*unitPrice'* of the product from product table in source.
3. Lookup is performed between '*Order_Header*', '*Dim_Customer*,' and '*Dim_Store'* to extract the necessary details
4. The two lookup outputs (Step 2 and 3) are sorted and merged based on the '*orderHeader*' ID.
5. Lookup is performed between '*Dim_Promotion*' and Promotion table in source to extract the discount amounts and the output is sorted based on '*productID'*
6. The output from step 4 is again sorted based on '*productID'* and merged with output of step 5
7. Now, we have all required columns to load necessary data to the Fact Table



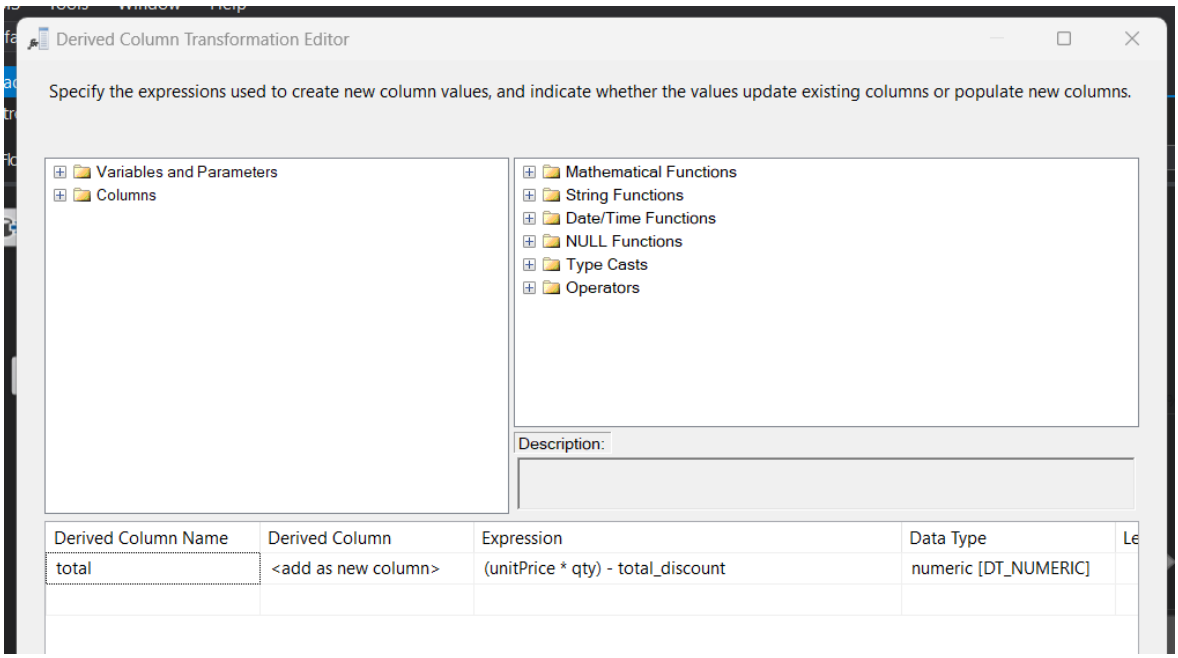
8. Check the applicability of discount for the purchase as the discounts are seasonal. The expression checks if the '*purchaseDate'* is greater than or equal to the '*StartDate'* and less than or equal to the '*EndDate'*. If the condition is true, it assigns 1; otherwise, it assigns 0. The expression is as follows:
(purchaseDate >= start_date) && (purchaseDate <= end_date) ? discount : 0



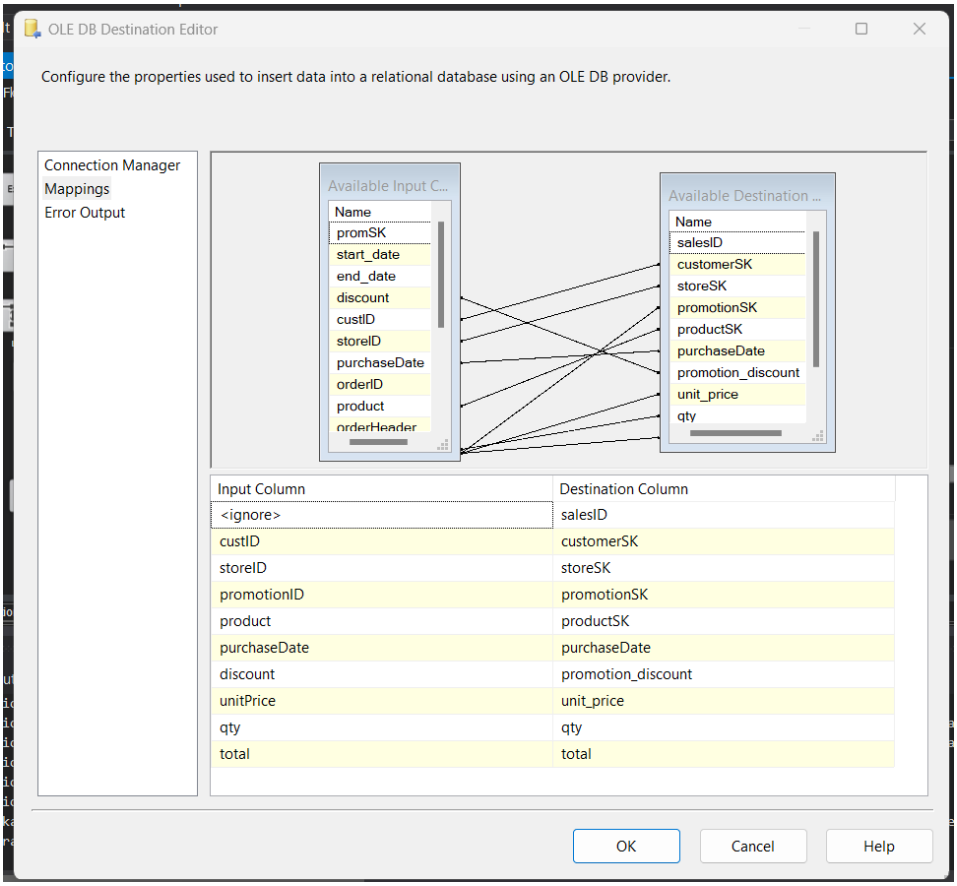
9. Total discount amount is calculated based on the discount applicability:
ISNULL(Discount_Applicability) ? 0 : unitPrice * qty * Discount_Applicability



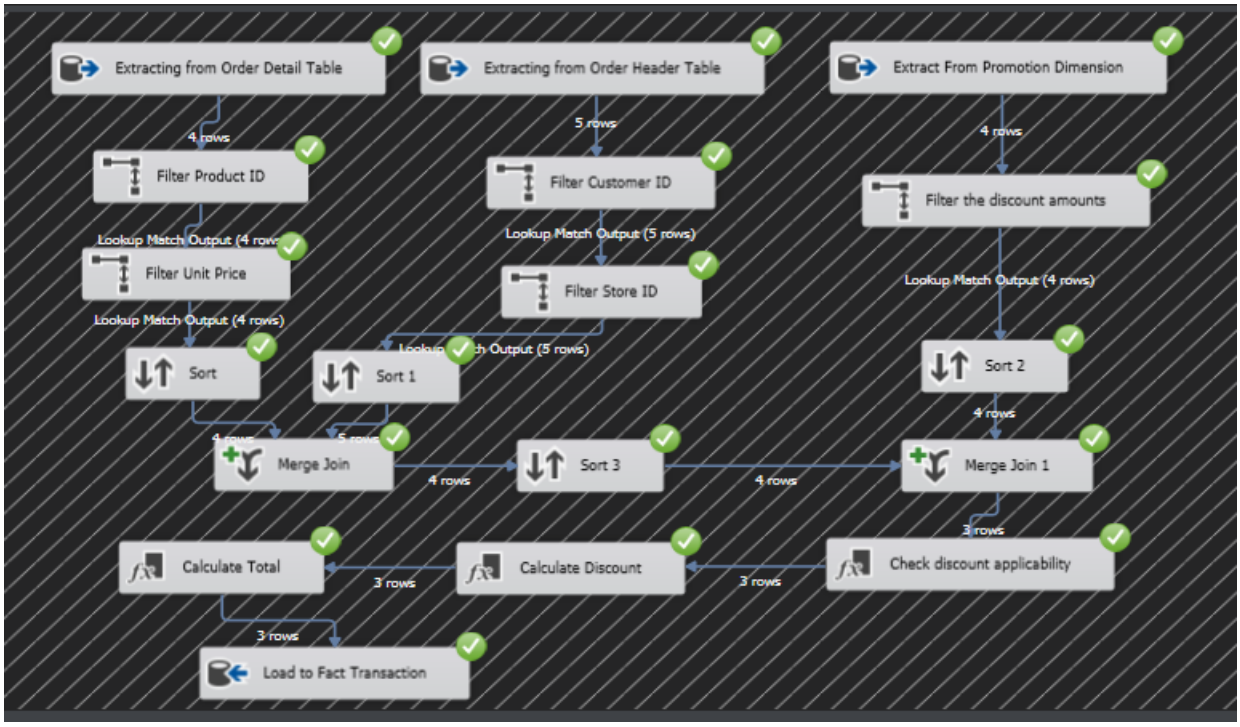
10. Finally, the total bill amount is calculated with discount:
(unitPrice * qty) - total_discount



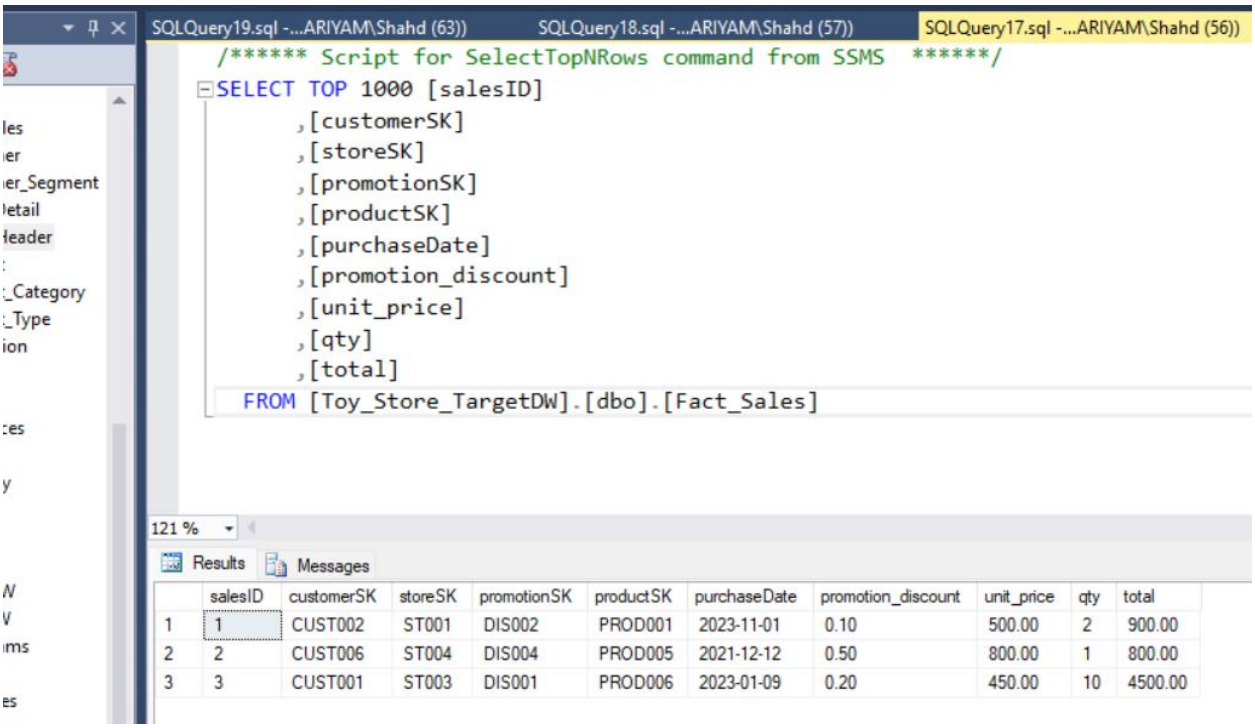
11. Since all column values are derived, data is mapped to load data to the fact table



12. Package is executed



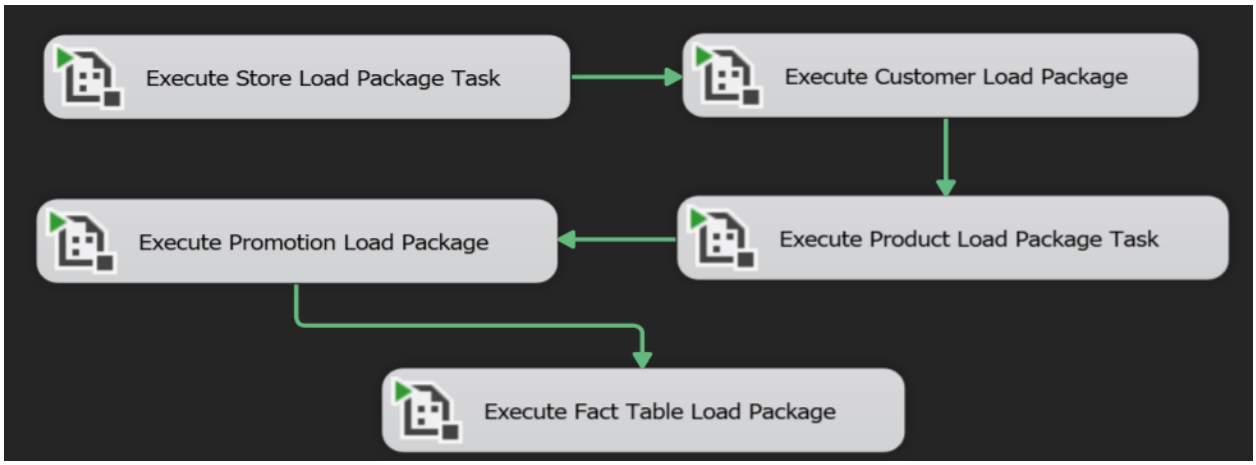
13. Populated fact table is represented below



```
/****** Script for SelectTopNRows command from SSMS  ******/
SELECT TOP 1000 [salesID]
      ,[customerSK]
      ,[storeSK]
      ,[promotionSK]
      ,[productSK]
      ,[purchaseDate]
      ,[promotion_discount]
      ,[unit_price]
      ,[qty]
      ,[total]
  FROM [Toy_Store_TargetDW].[dbo].[Fact_Sales]
```

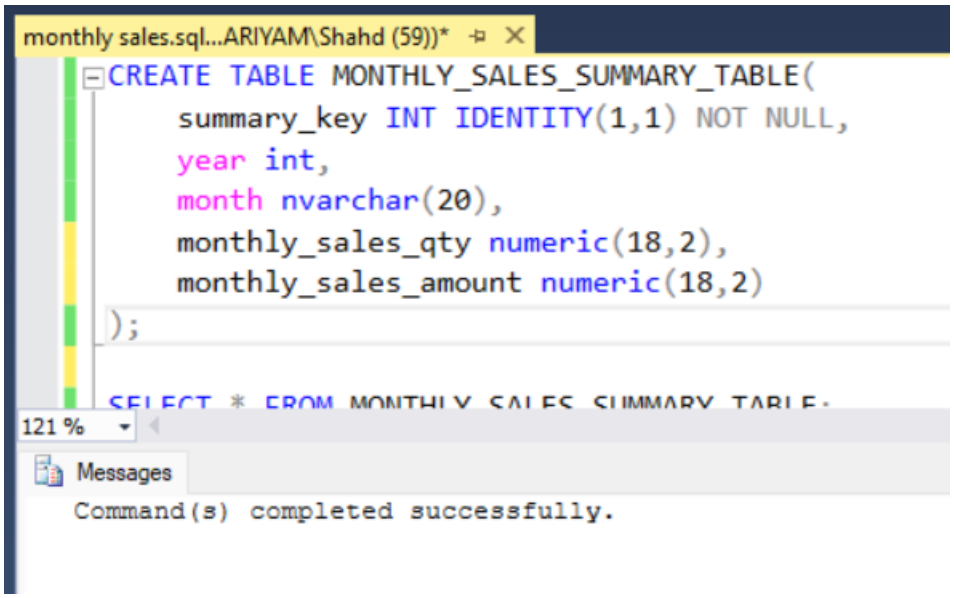
	salesID	customerSK	storeSK	promotionSK	productSK	purchaseDate	promotion_discount	unit_price	qty	total
1	1	CUST002	ST001	DIS002	PROD001	2023-11-01	0.10	500.00	2	900.00
2	2	CUST006	ST004	DIS004	PROD005	2021-12-12	0.50	800.00	1	800.00
3	3	CUST001	ST003	DIS001	PROD006	2023-01-09	0.20	450.00	10	4500.00

- Each dimension and fact table was populated using individual packages. To streamline the execution of all these packages in single click, a new package is designed, incorporating Execute Package tasks to run all the packages from a centralized point.



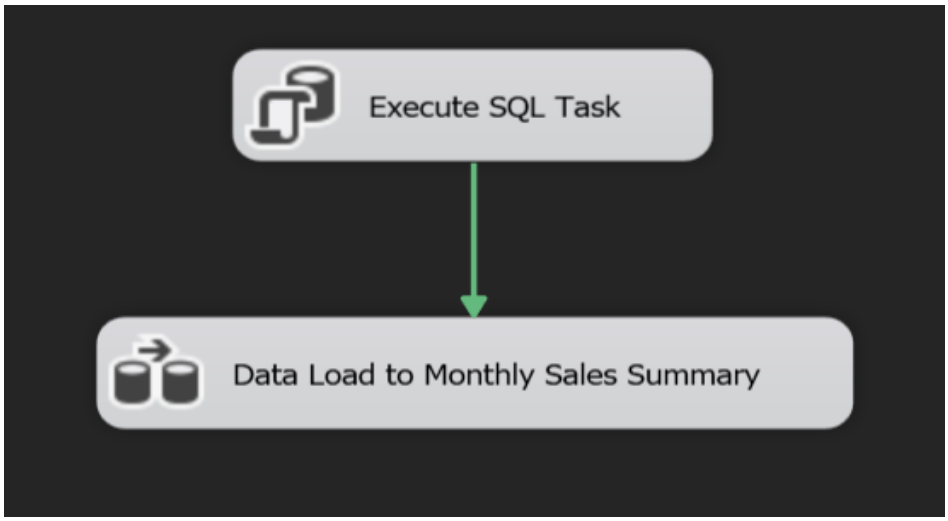
- **Part 2**

Initially a table is created in the target data warehouse to store the monthly sales summary



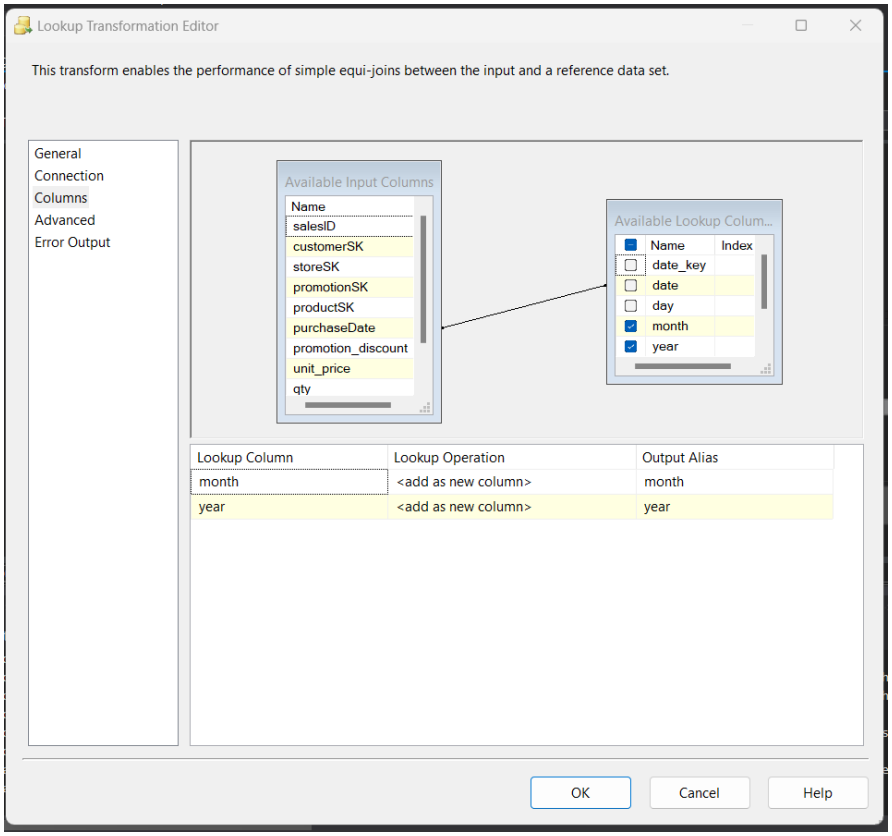
Next, data is loaded to the table using ETL

- ❖ An EXEXUTE SQL TASK is added to truncate the table in the control area followed by Data Flow task.

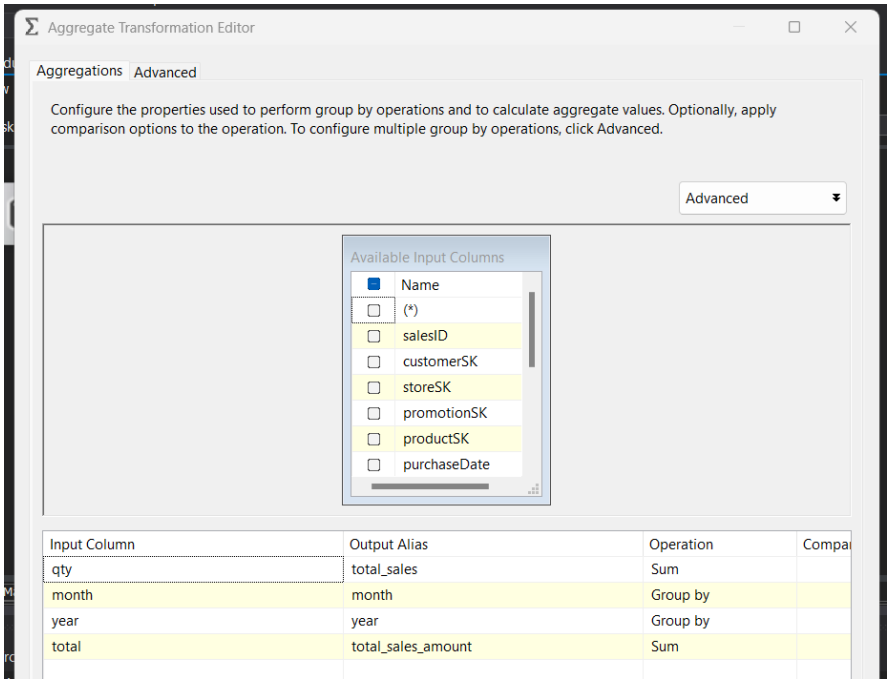


- ❖ Source assistant is added in data flow area and connected to the FACT_SALES table in data warehouse

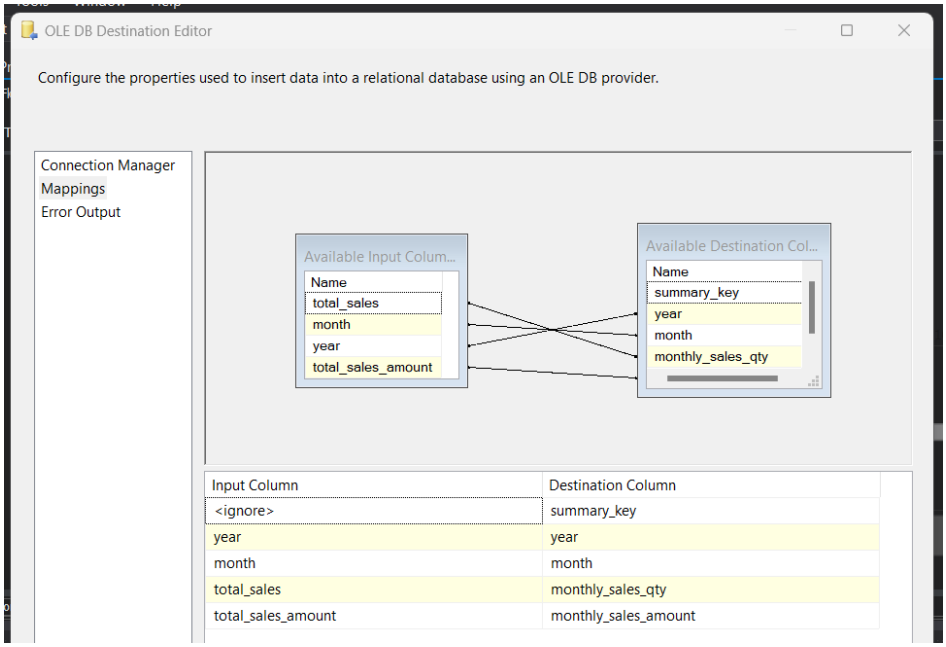
❖ A Lookup is done with the Dim_Date to get the year and month using the date key



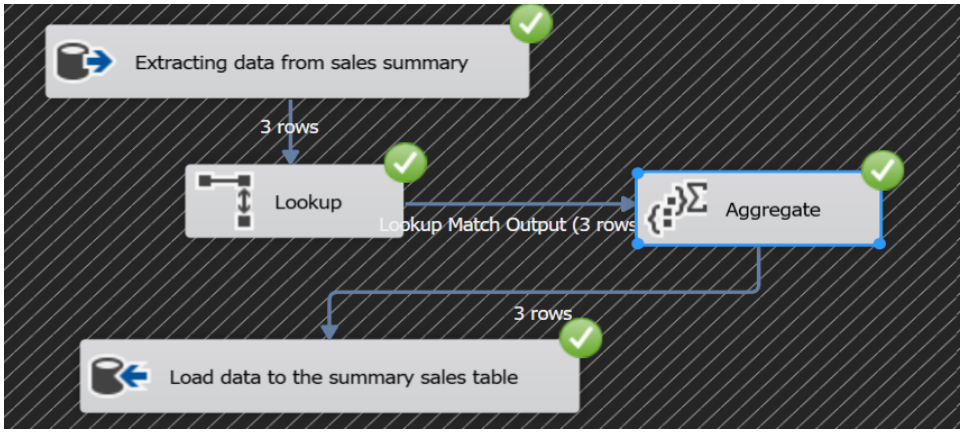
❖ To get the monthly sales summary, an Aggregate task is added and sum operation is applied to the quantity and total, grouped by the year and month



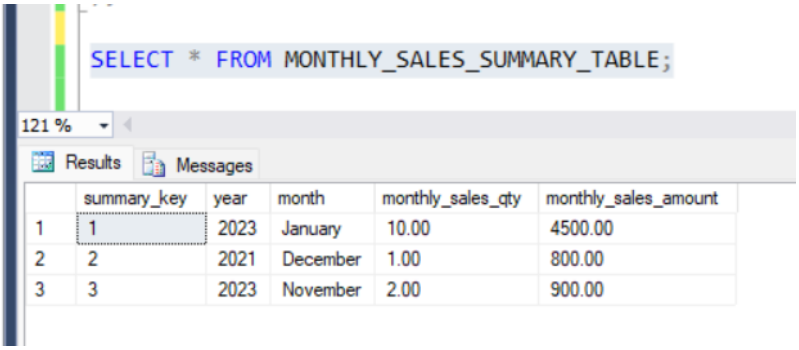
- Destination assistant is added in data flow area and connected to the newly created sales summary table in data warehouse. The columns are mapped accordingly.



- Execute the package



- Verifying the populated data



```
SELECT * FROM MONTHLY_SALES_SUMMARY_TABLE;
```

121 %

Results | Messages

	summary_key	year	month	monthly_sales_qty	monthly_sales_amount
1	1	2023	January	10.00	4500.00
2	2	2021	December	1.00	800.00
3	3	2023	November	2.00	900.00

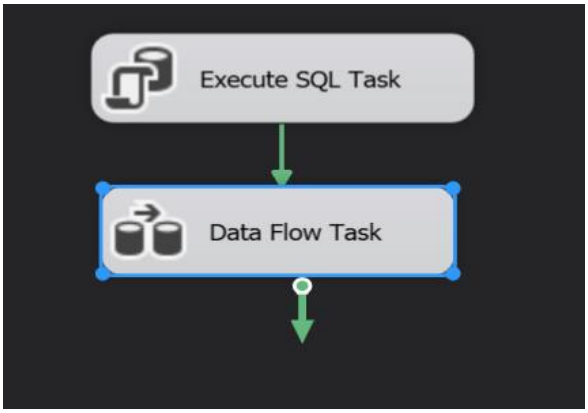
PART 3

- Initially the fact table to store the market basket is created

```
---------------- MARKET BASKET ----------------------
CREATE TABLE FACT_MARKET_BASKET(
    Date_key INT,
    Product_A INT,
    Product_B INT,
    Store INT,
    Product_A_qty INT,
    Product_B_qty INT,
    Product_A_SalesAmnt NUMERIC(18,2),
    Product_B_SalesAmnt NUMERIC(18,2),
);
```

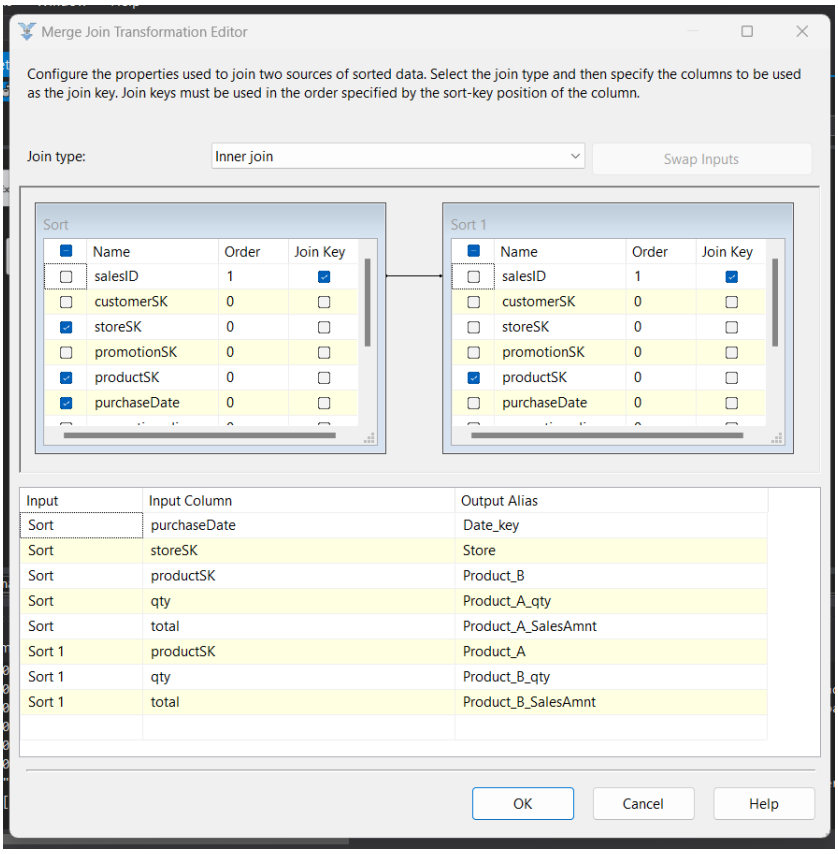
```
121 %
Messages
Command(s) completed successfully.
```

- Now, we populate data using ETL
  - EXECUTE SQL JOB task is added to truncate table followed by a data flow task.

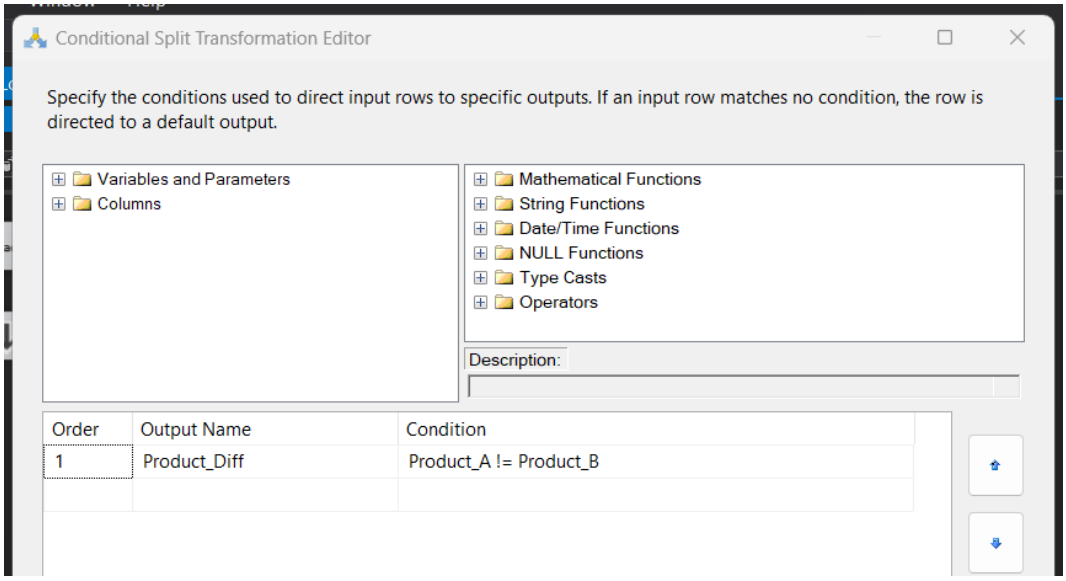


  - Add two source assistants and connect both to Fact Sales, since we have to extract details for two products

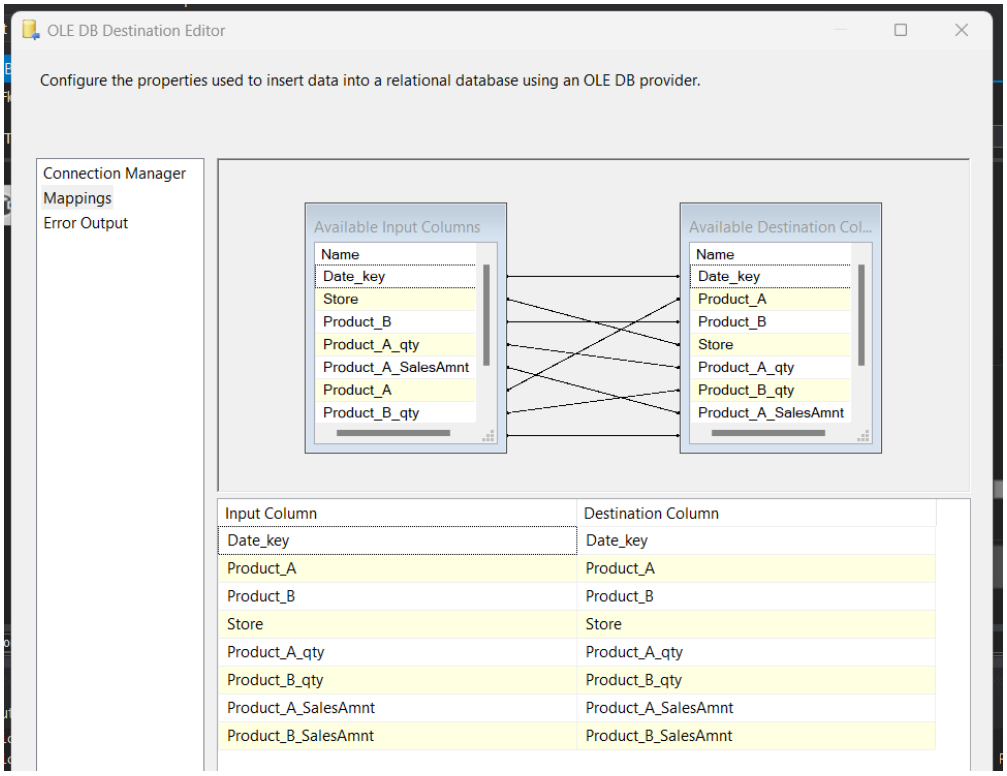
- The two are sorted and merged using the key (sales id)



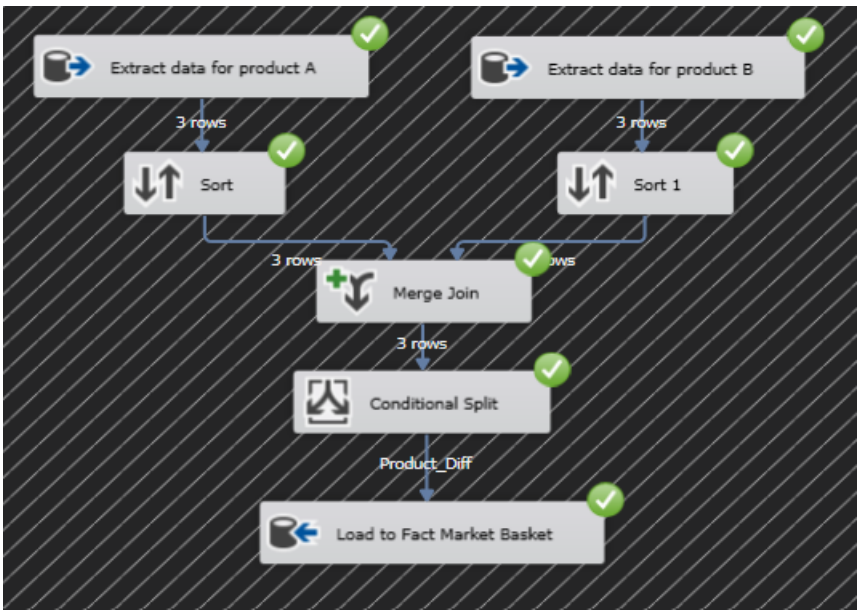
- Add a conditional split to identify the two products separately



- Add destination assistant and load data to the created table



- Execute the package



❖ Verify the populated data

```
SELECT * FROM FACT_MARKET_BASKET;
```

121 %

Results | Messages

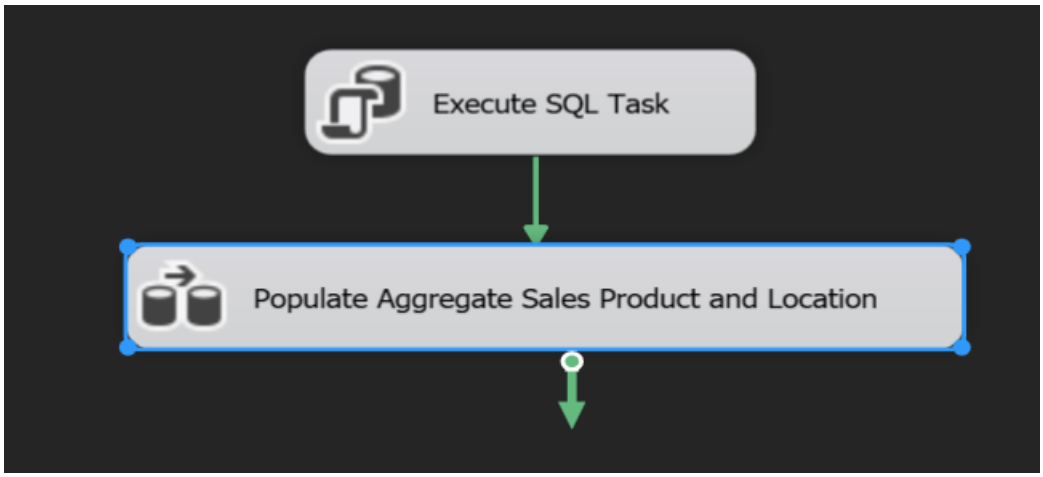
	Date_key	Product_A	Product_B	Store	Product_A_qty	Product_B_qty	Product_A_SalesAmnt	Product_B_SalesAmnt
1	2023-10-01	3	2	ST001	1	1	10.00	2.00

PART 4

- ❖ Create a table in the data warehouse to store the aggregated sales data

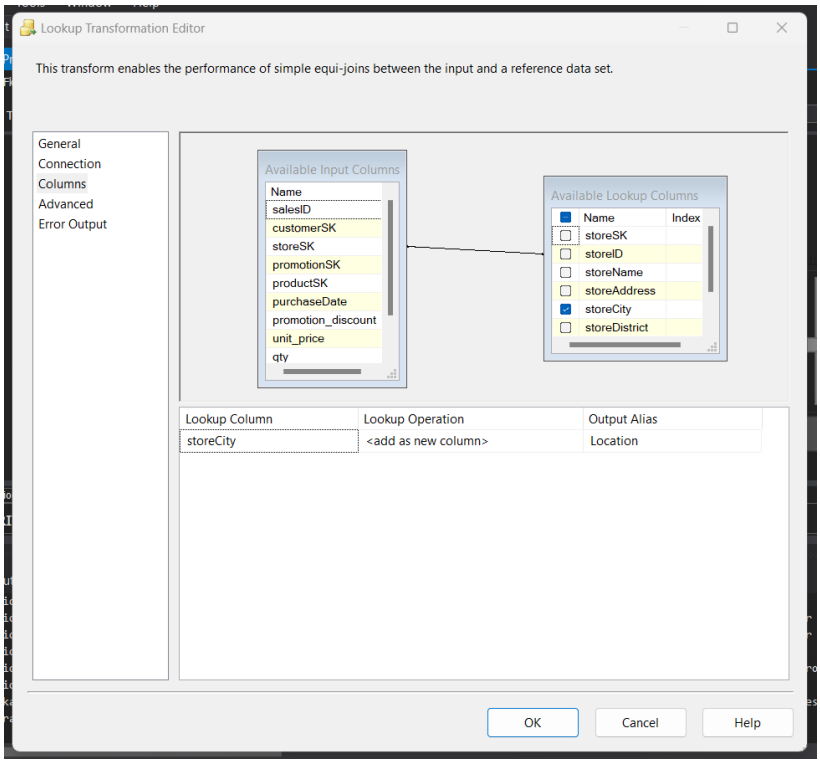
```
---------------- SALES PRODUCT AND LOCATION ----
CREATE TABLE SALES_PROD_LOC(
    Product varchar(20),
    Location varchar(20),
    total_sales_qty numeric(18,2),
    total_sales_amount numeric(18,2),
);
```

- ❖ Data is populated to the table using ETL
  - Add an EXECUTE SQL TASK in the control flow area to truncate the table followed by a data flow task

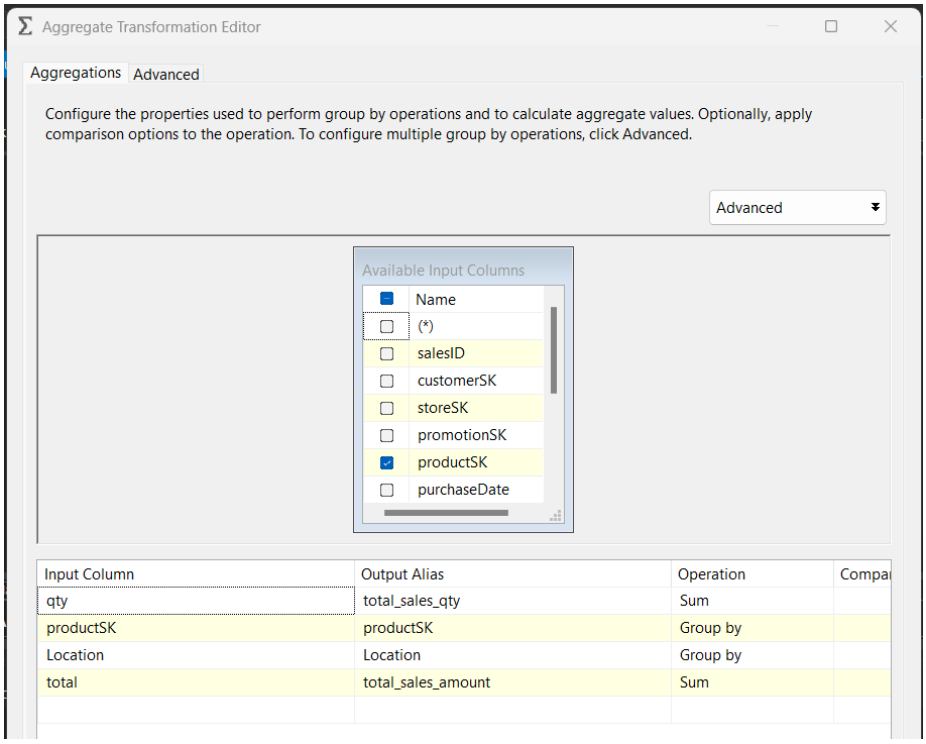


  - Add source assistant and connect it with the Fact_Sales

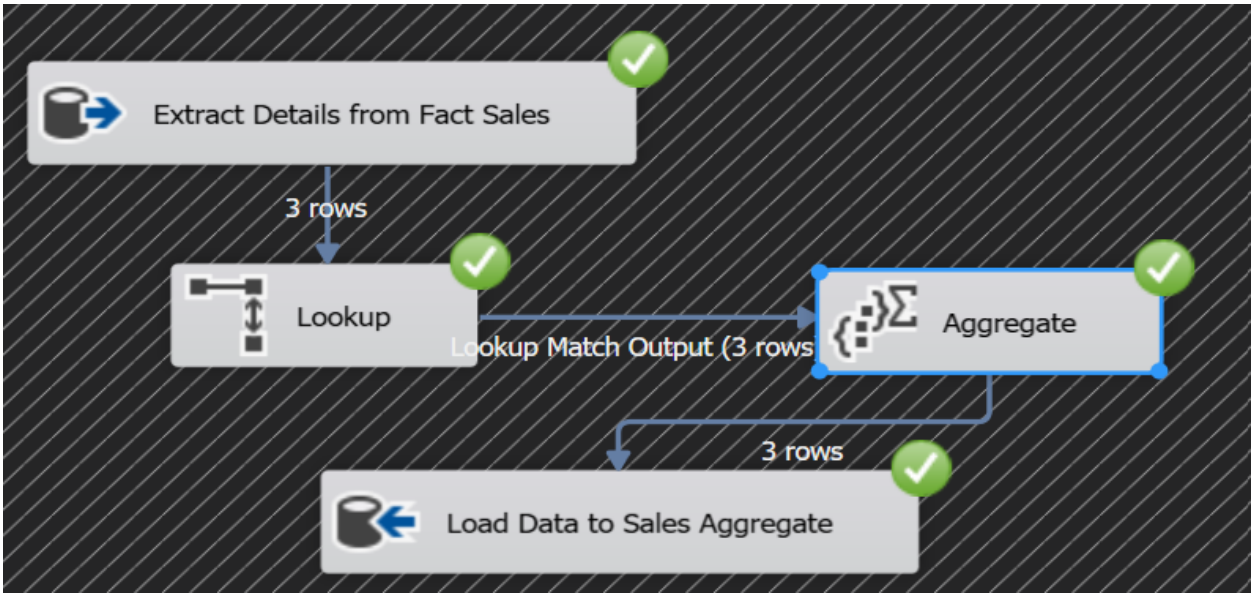
- A lookup is done with Dim_Store to fetch the store location using storeSK and store ID



- Aggregate task is added to derive the sales quantity and sales amount



- Execute the package



❖ Verify the populated data

```
SELECT * FROM SALES_PROD_LOC;
```

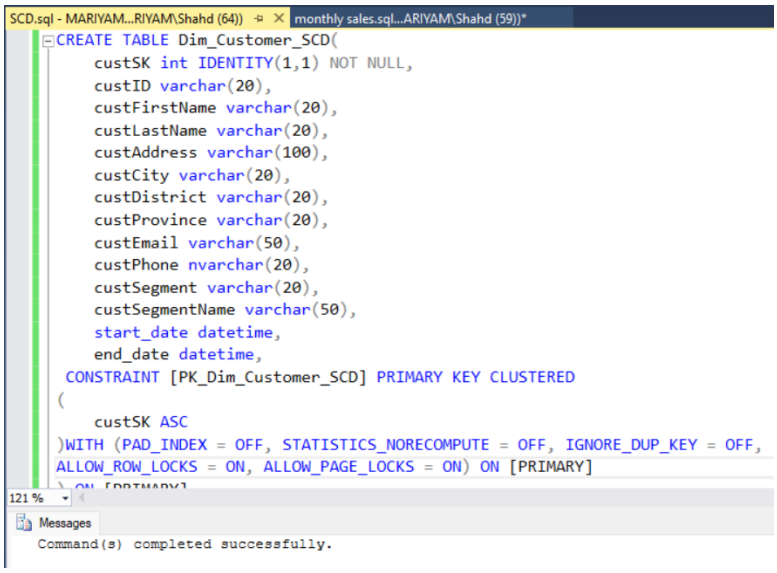
121 %

Results | Messages

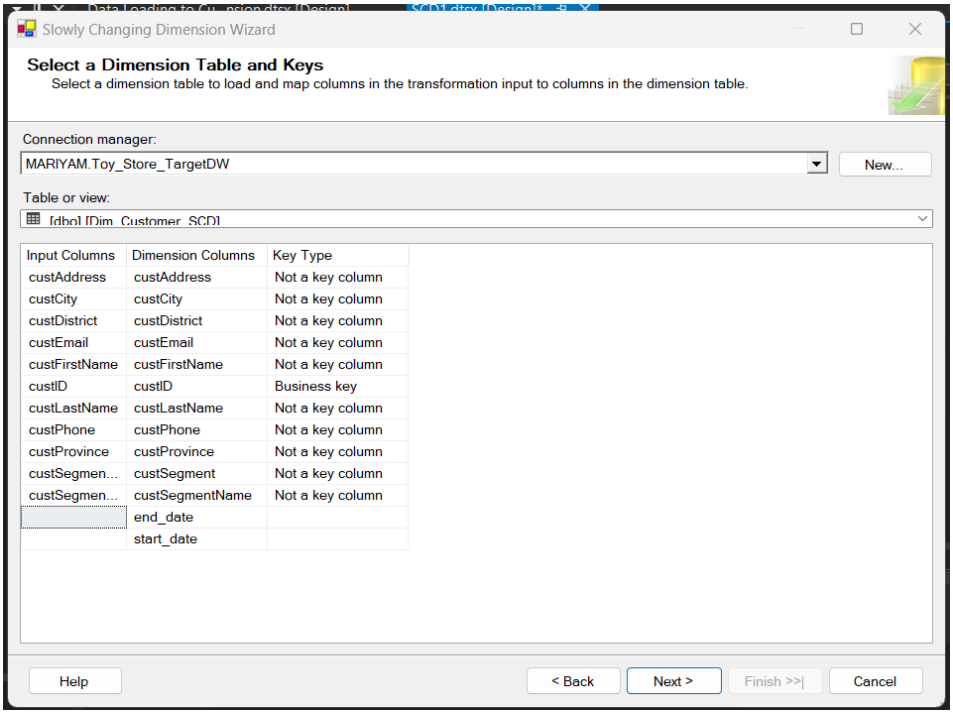
	Product	Location	total_sales_qty	total_sales_amount
1	PROD001	Colombo	2.00	900.00
2	PROD005	Dehiwela	1.00	800.00
3	PROD006	Peradeniya	10.00	4500.00

PART 5

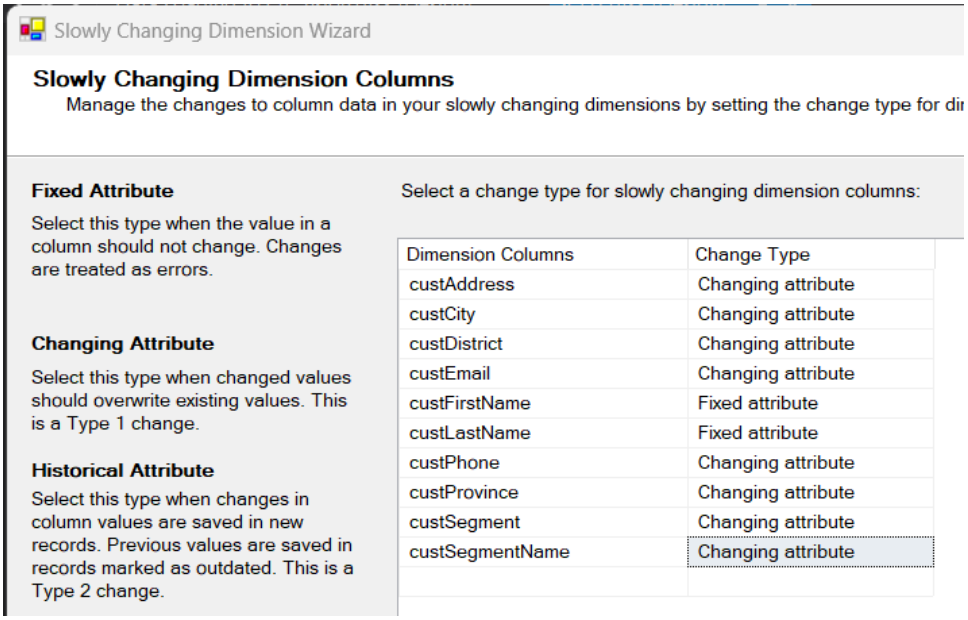
- A new dimension is created in the target warehouse to implement slowly changing dimension:



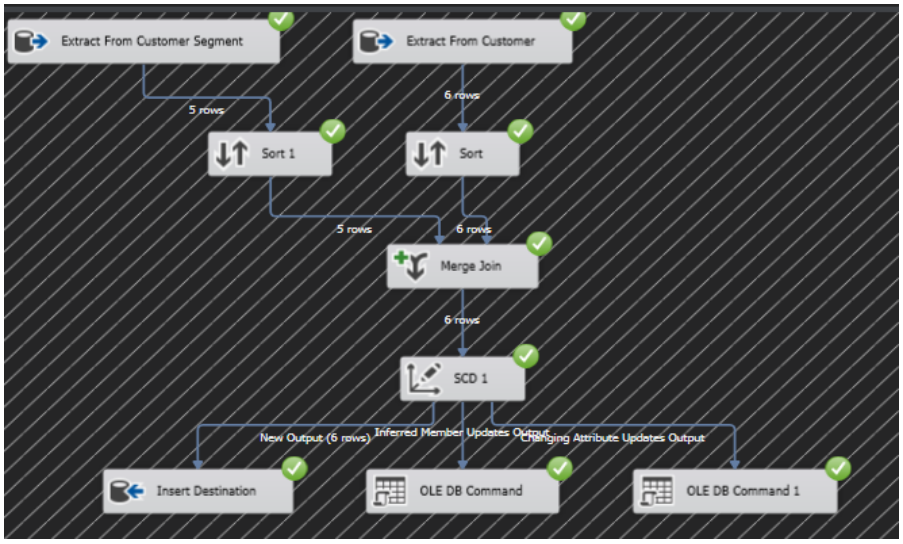
- SCD – Type 1 : This type does not maintain history. It maintains only the current state of the subject by updating directly.
  - Once the merging process between the customer and customer segment tables are complete, incorporate the Slowly Changing Dimension task and connect it with the merged output.
  - Assign the 'custID' as the business key



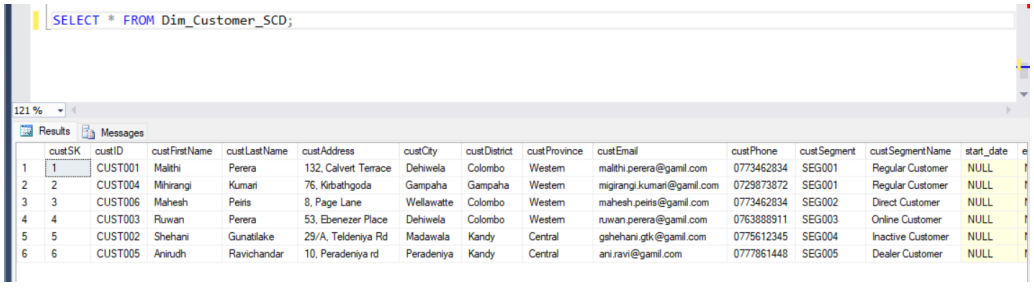
- The ‘custFirstName’ and ‘custLastName’ are kept as fixed attributes and the others as changing attribute



- Execute the package



- View all inserted data



```
SELECT * FROM Dim_Customer_SCD;
```

121 %

Results | Messages

	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName	start_date	e
1	1	CUST001	Malithi	Perera	132, Calvert Terrace	Dehiwela	Colombo	Western	malithi.perera@gamil.com	0773462834	SEG001	Regular Customer	NULL	N
2	2	CUST004	Mihirangi	Kumari	76, Kirbathgoda	Gampaha	Gampaha	Western	migirangi.kumari@gamil.com	0729873872	SEG001	Regular Customer	NULL	N
3	3	CUST006	Mahesh	Peiris	8, Page Lane	Wellawatte	Colombo	Western	mahesh.peiris@gamil.com	0773462834	SEG002	Direct Customer	NULL	N
4	4	CUST003	Ruwan	Perera	53, Ebenezer Place	Dehiwela	Colombo	Western	ruwan.perera@gamil.com	0763888911	SEG003	Online Customer	NULL	N
5	5	CUST002	Shehani	Gunatilake	29/A, Teldeniya Rd	Madawala	Kandy	Central	gshehani.gtk@gamil.com	0775612345	SEG004	Inactive Customer	NULL	N
6	6	CUST005	Anirudh	Ravichandar	10, Peradeniya rd	Peradeniya	Kandy	Central	ani.ravi@gamil.com	0777861448	SEG005	Dealer Customer	NULL	N

- Update a record in the source table

```
UPDATE Customer
SET custPhone = '0777123123'
WHERE custID = 'CUST006';
```

```
21 %
Messages

(1 row(s) affected)
```

- Check that particular customer in the dimension before execution of package

```
SELECT * FROM Dim_Customer_SCD
WHERE custID = 'CUST006';
```

121 %

Results | Messages

	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName
1	3	CUST006	Mahesh	Peiris	8, Page Lane	Wellawatte	Colombo	Western	mahesh.peiris@gamil.com	0773462834	SEG002	Direct Customer

- Execute the package and check the record again. Since the update has taken place directly, the SCD is now implemented in type 1

```
SELECT * FROM Dim_Customer_SCD
WHERE custID = 'CUST006';
```

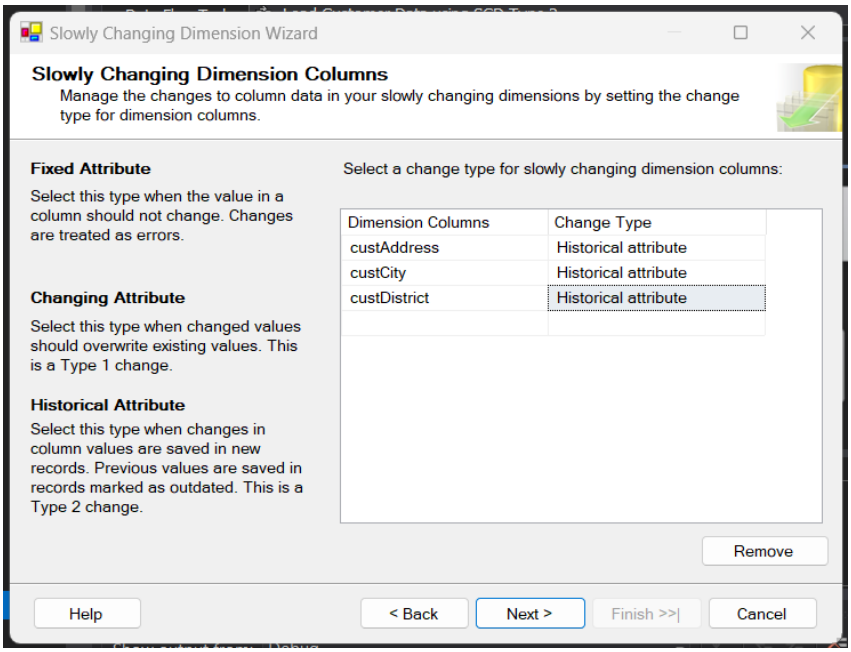
21 %

Results | Messages

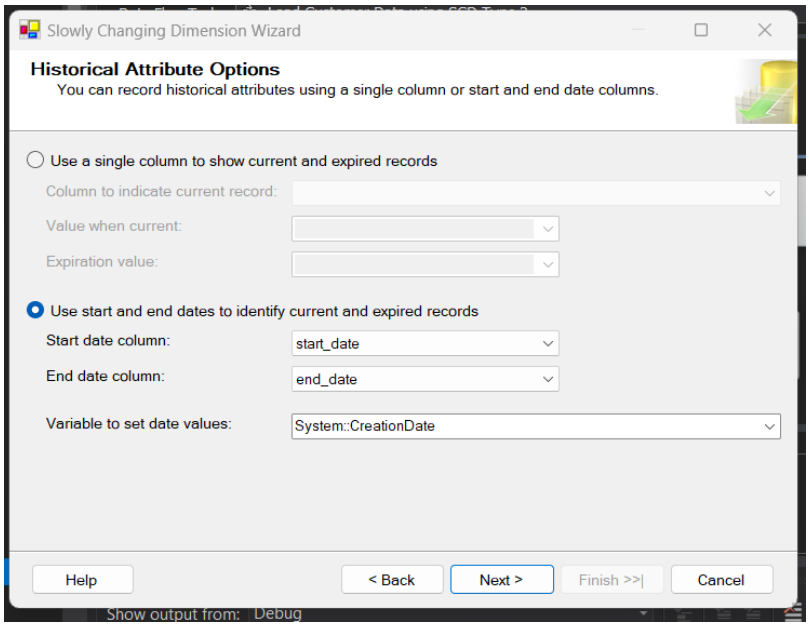
	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName	start_date
1	3	CUST006	Mahesh	Peiris	8, Page Lane	Wellawatte	Colombo	Western	mahesh.peiris@gamil.com	0777123123	SEG002	Direct Customer	NULL

❖ SCD – Type 2: History attributes are maintained in this type. It identifies updates to a row. All changes are captured as version records with a flag and the date.

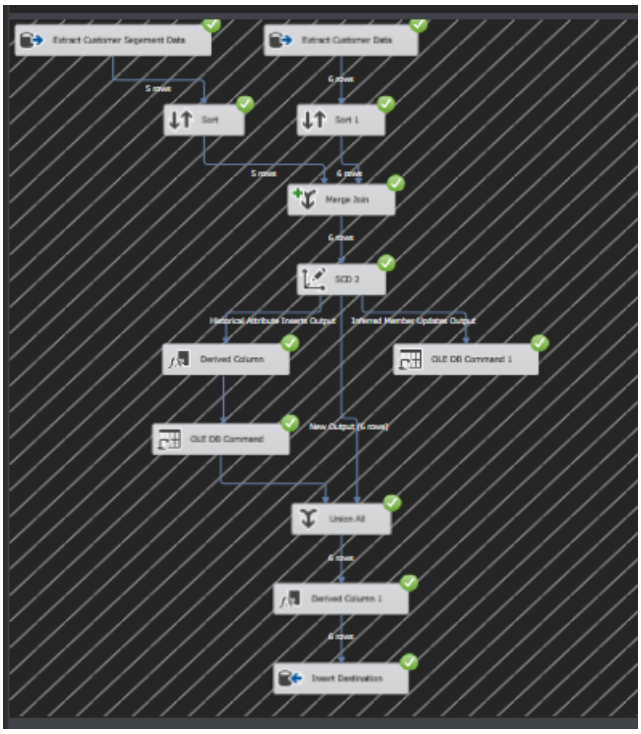
- Follow the same procedure followed in SCD until assigning the change type for the attributes.
- Assign custAddress, custCity, custDistrict as historic attributes



- In historic attribute options, select the option to use start and end dates to identify current or expired records



- Execute the package



- View the inserted data

```
SELECT * FROM Dim_Customer_SCD;
```

21 %

Results | Messages

	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName	start_date	end_dat
1	1	CUST001	Malithi	Perera	132, Calvert Terrace	Dehiwela	Colombo	Western	malithi.perera@gamil.com	0773462834	SEG001	Regular Customer	2023-12-19 02:30:16.000	NULL
2	2	CUST004	Mihirangi	Kumari	76, Kirbathgoda	Gampaha	Gampaha	Western	migirangi.kumari@gamil.com	0729873872	SEG001	Regular Customer	2023-12-19 02:30:16.000	NULL
3	3	CUST006	Mahesh	Peiris	8, Page Lane	Wellawatte	Colombo	Western	mahesh.peiris@gamil.com	0777123123	SEG002	Direct Customer	2023-12-19 02:30:16.000	NULL
4	4	CUST003	Ruwan	Perera	53, Ebenezer Place	Dehiwela	Colombo	Western	ruwan.perera@gamil.com	0763888911	SEG003	Online Customer	2023-12-19 02:30:16.000	NULL
5	5	CUST002	Shehani	Gunatilake	29/A, Teldeniya Rd	Madawala	Kandy	Central	gshehani.gtk@gamil.com	0775612345	SEG004	Inactive Customer	2023-12-19 02:30:16.000	NULL
6	6	CUST005	Anirudh	Ravichandar	10, Peradeniya rd	Peradeniya	Kandy	Central	ani.ravi@gamil.com	0777861448	SEG005	Dealer Customer	2023-12-19 02:30:16.000	NULL

- Update the customer address in the source table

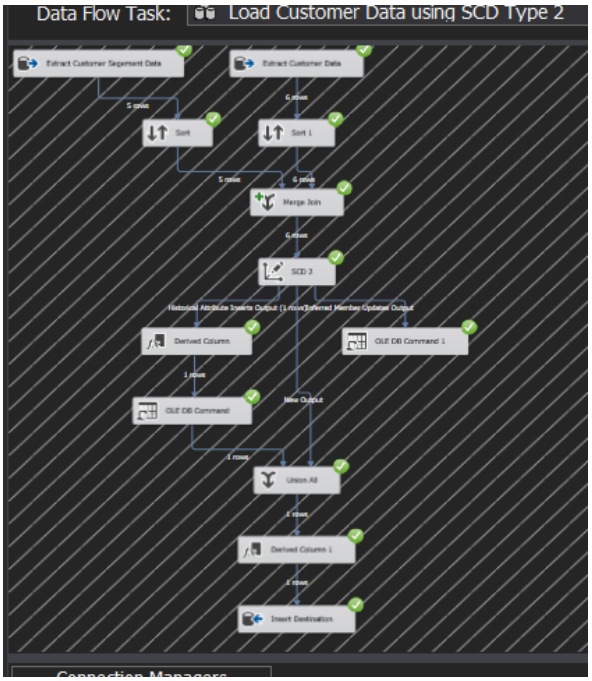
```
UPDATE Customer
SET custAddress = '18, Page Street', custCity = 'Mahaiyawa', custDistrict = 'Colombo'
WHERE custID = 'CUST006';
```

121 %

Messages

```
(1 row(s) affected)
```

- Re-execute the package



- Check for the particular customer data. Two records will be available which means the Slowly changing dimension is implemented using type 2

```
SELECT * FROM Dim_Customer_SCD
WHERE custID = 'CUST006'
```

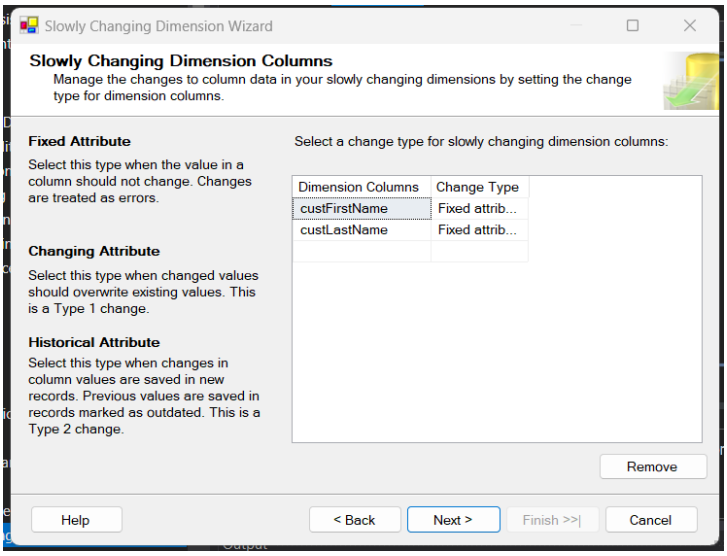
121 %

Results | Messages

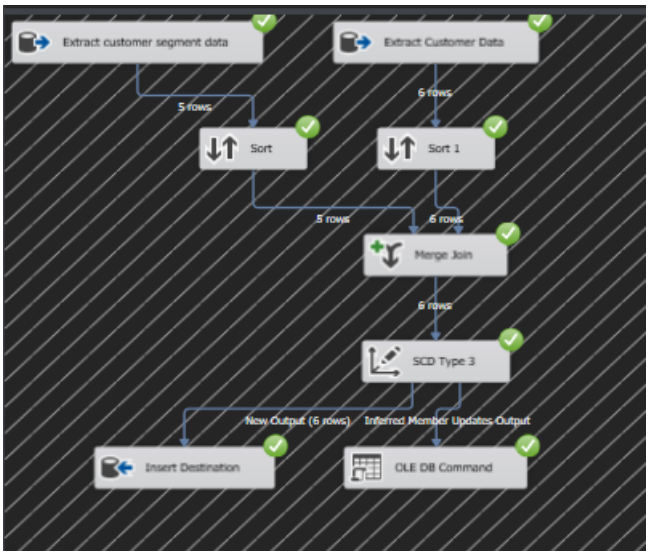
	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName	start_date	end_date
1	3	CUST006	Mahesh	Peiris	8, Page Lane	Wellawatte	Colombo	Western	mahesh.peiris@gamil.com	0777123123	SEG002	Direct Customer	2023-12-19 02:30:16.000	2023-12-19 02:
2	7	CUST006	Mahesh	Peiris	18, Page Street	Mahaiyawa	Colombo	Western	mahesh.peiris@gamil.com	0777123123	SEG002	Direct Customer	2023-12-19 02:30:16.000	NULL

❖ SCD – Type 3: Monitor changes to a particular attribute by introducing a column to track the previous value and modified changes.

- Follow same steps followed in SCD 1 and 2 until assigning the change types to attributes.
- We will use the first name and last name as the attribute to track changes. Assign the two variables and fixed attributes.



- Execute the package



- Check the populated data

```
SELECT * FROM Dim_Customer_SCD;
```

121 %

Results | Messages

	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName	start_date	end_date
1	1	CUST001	Malithi	Perera	132, Calvert Terrace	Dehiwela	Colombo	Western	malithi.perera@gamil.com	0773462834	SEG001	Regular Customer	NULL	NULL
2	2	CUST004	Mihirangi	Kumari	76, Kirbathgoda	Gampaha	Gampaha	Western	migirangi.kumari@gamil.com	0729873872	SEG001	Regular Customer	NULL	NULL
3	3	CUST006	Maheshika	Peiris	18, Page Street	Mahaiyawa	Colombo	Western	mahesh.peiris@gamil.com	0777123123	SEG002	Direct Customer	NULL	NULL
4	4	CUST003	Ruwan	Perera	53, Ebenezer Place	Dehiwela	Colombo	Western	ruwan.perera@gamil.com	0763888911	SEG003	Online Customer	NULL	NULL
5	5	CUST002	Shehani	Gunatilake	29/A, Teldeniya Rd	Madawala	Kandy	Central	gshehani.gtk@gamil.com	0775612345	SEG004	Inactive Customer	NULL	NULL
6	6	CUST005	Anirudh	Ravichandar	10, Peradeniya rd	Peradeniya	Kandy	Central	ani.ravi@gamil.com	0777861448	SEG005	Dealer Customer	NULL	NULL

- Update the customer's name in the source

```
UPDATE Customer
SET custFirstName = 'Mahesh'
WHERE custID = 'CUST006';
```

```
121 %
Messages

(1 row(s) affected)
```

- Execute the package
- Check the updated customer data in the dimension. The value will remain the same although the change is made. This means the previous value is maintained in the first and last name, thereby enabling type 3 slowly changing dimension

```
SELECT * FROM Dim_Customer_SCD
WHERE custID = 'CUST006'
```

121 %

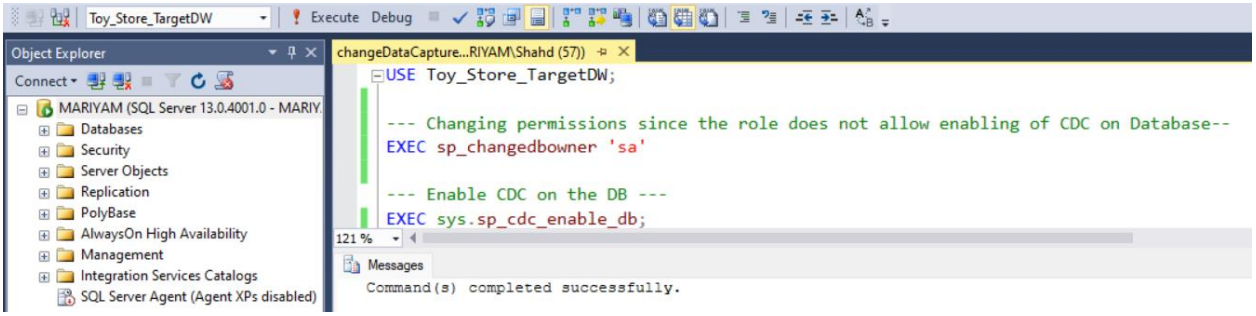
Results | Messages

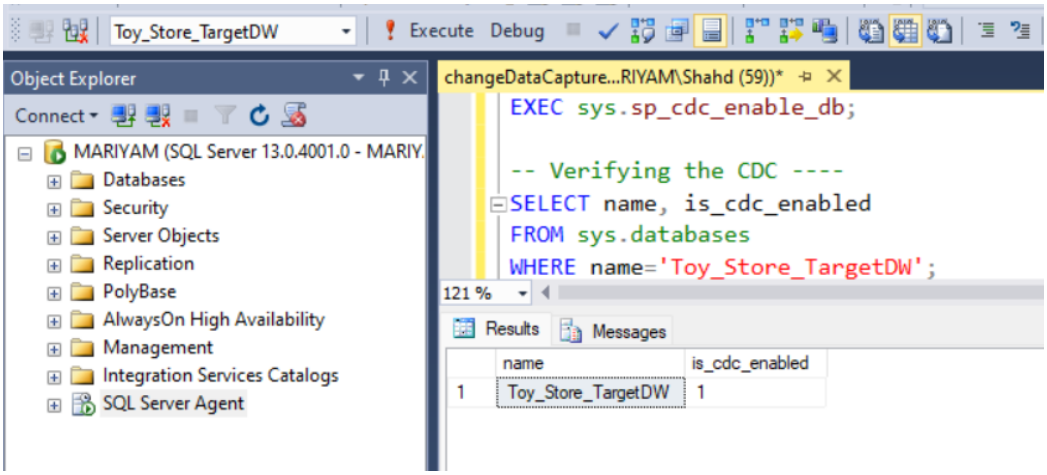
	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName	start_date	end_date
1	3	CUST006	Maheshika	Peiris	18, Page Street	Mahaiyawa	Colombo	Western	mahesh.peiris@gamil.com	0777123123	SEG002	Direct Customer	NULL	NULL

SECTION D

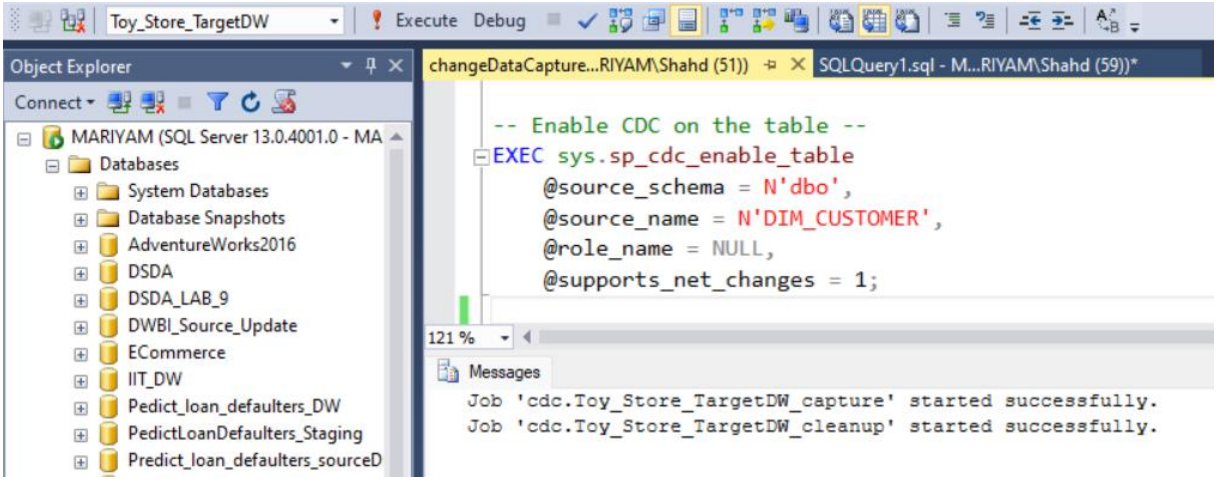
- **PART 1**

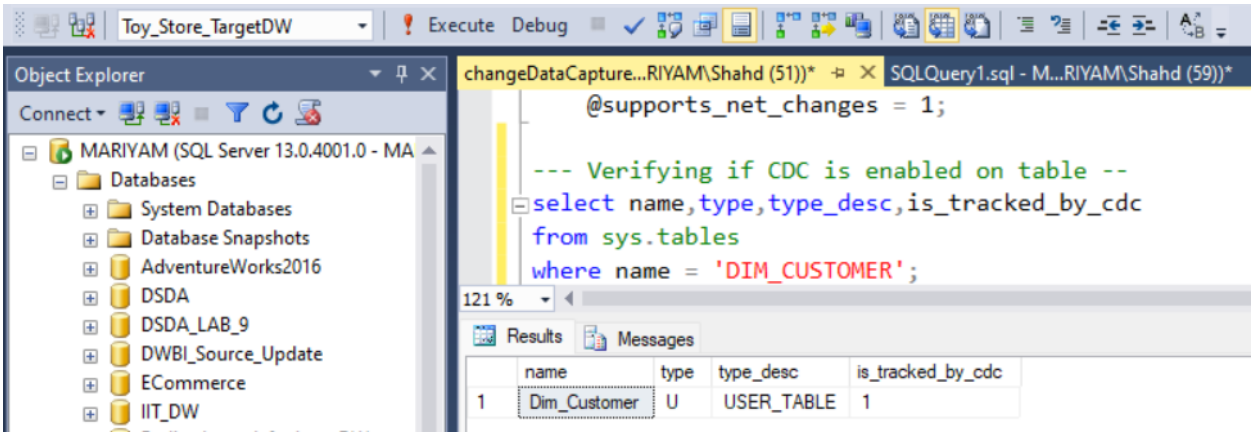
❖ Enabling Change Data Capture on Database



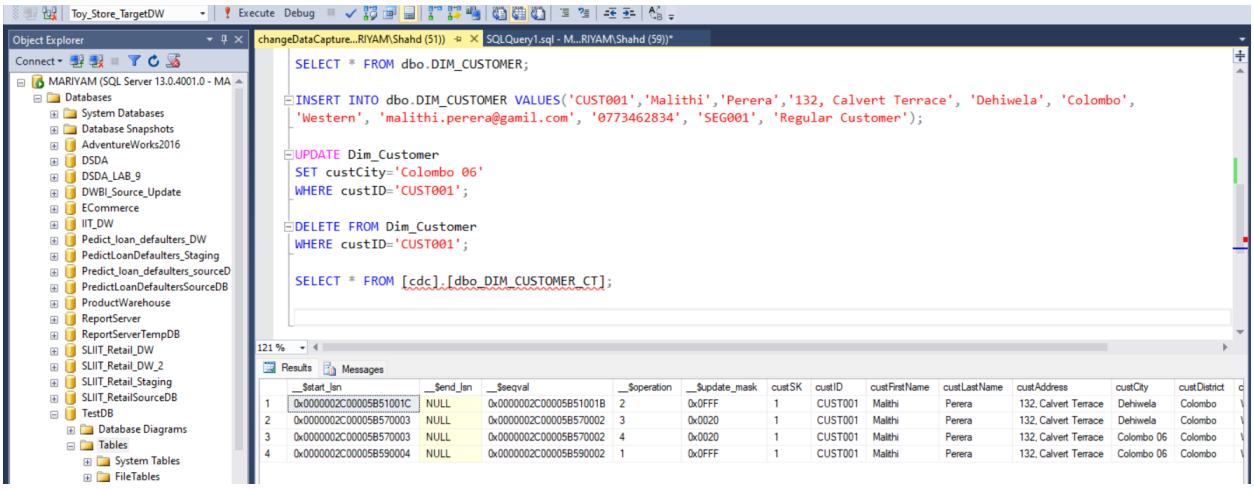


❖ Enabling Change Data Capture on Customer Dimension





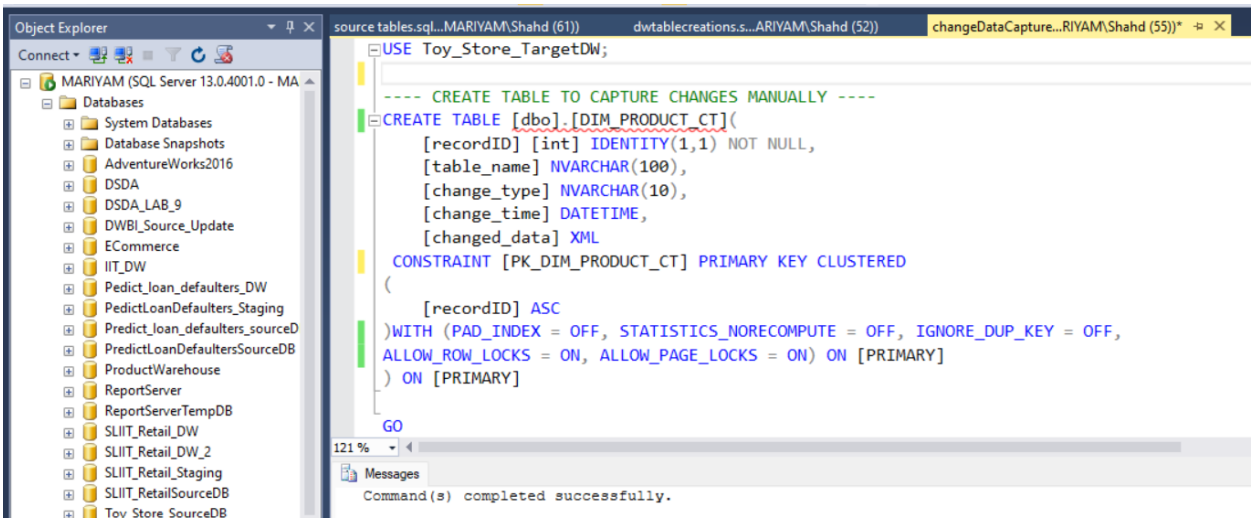
- Inserting, updating and deleting rows in Customer Dimension Table and checking if the change is captured.



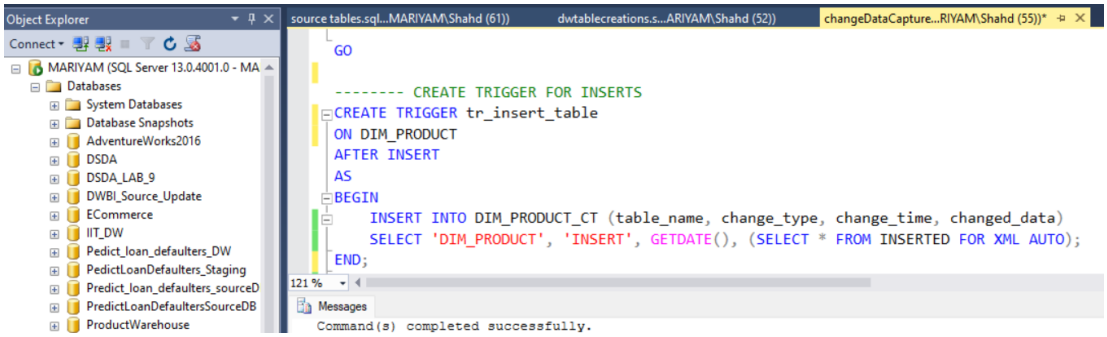
According to the output it is evident that CDC is now enabled and working.

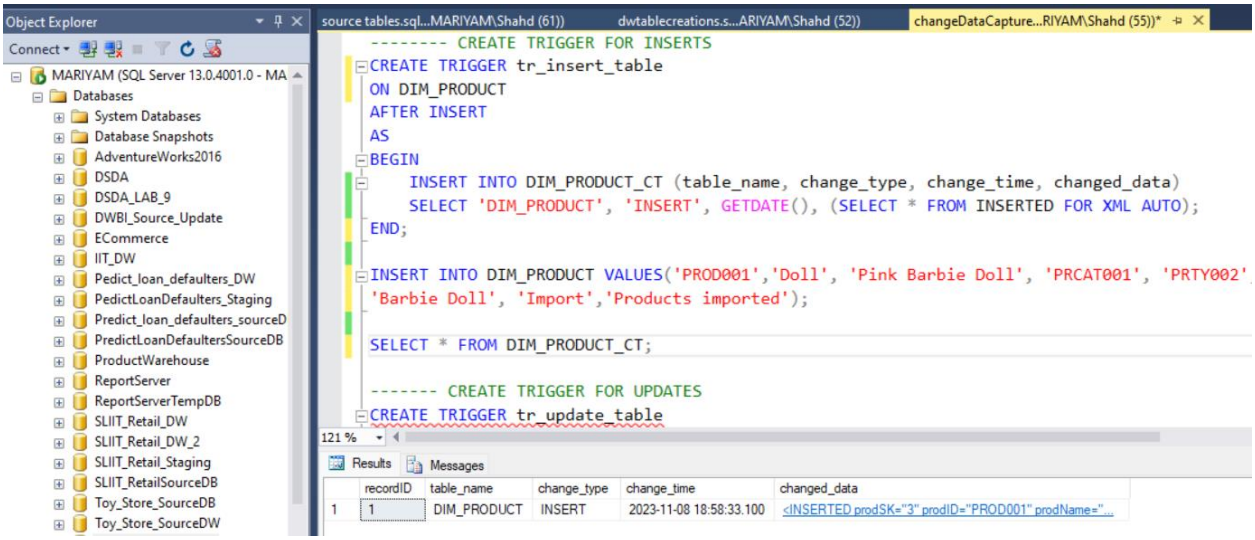
- **PART 2**

❖ Creating a new database to record all the changes

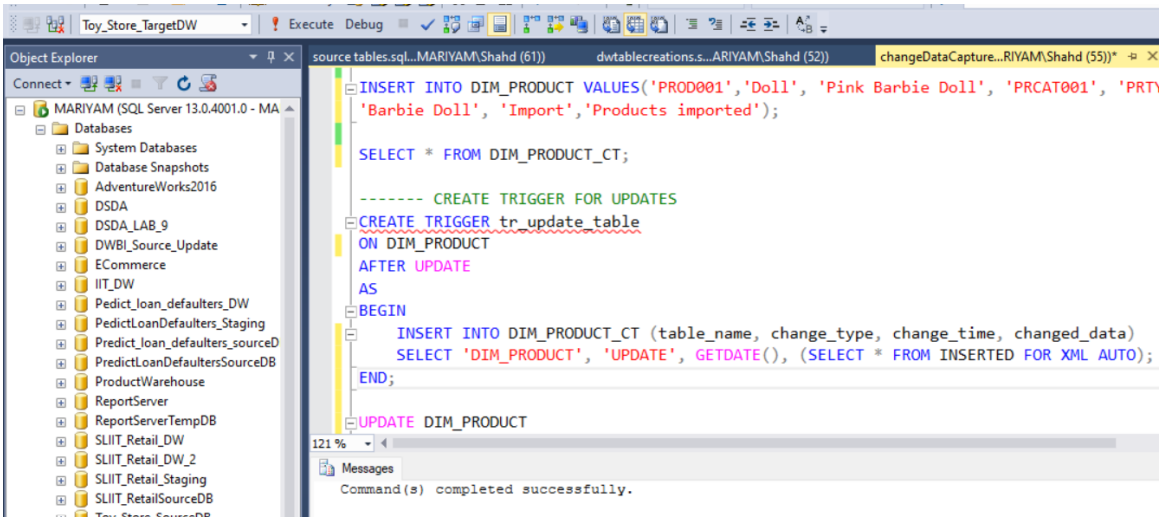


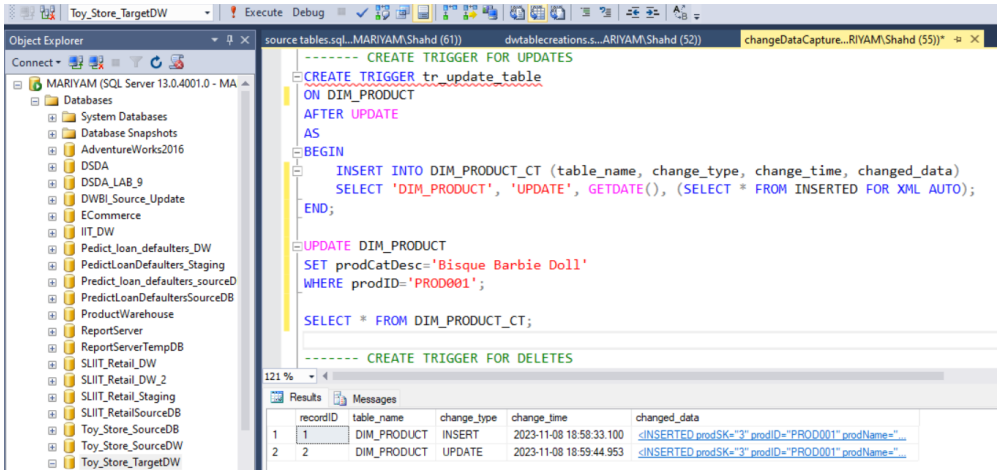
❖ Creating a trigger to capture the insert operations



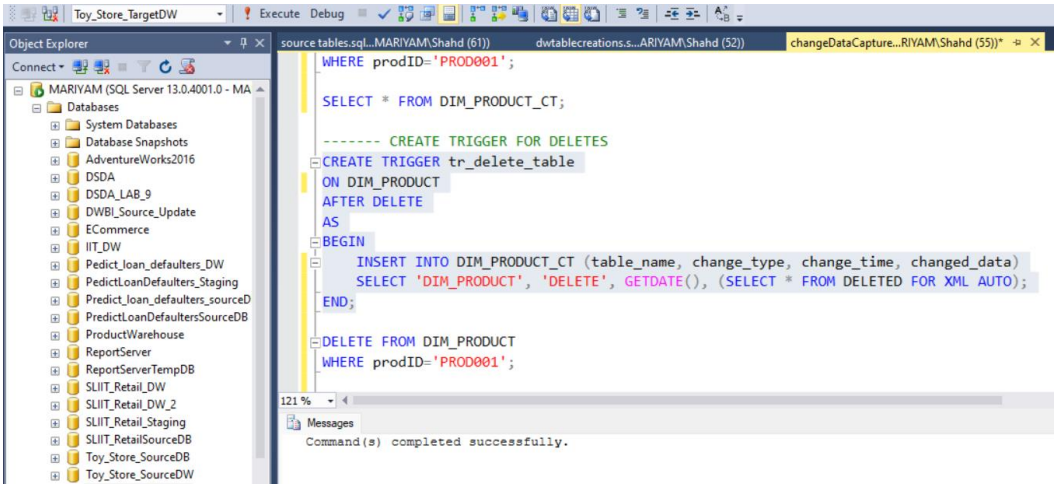


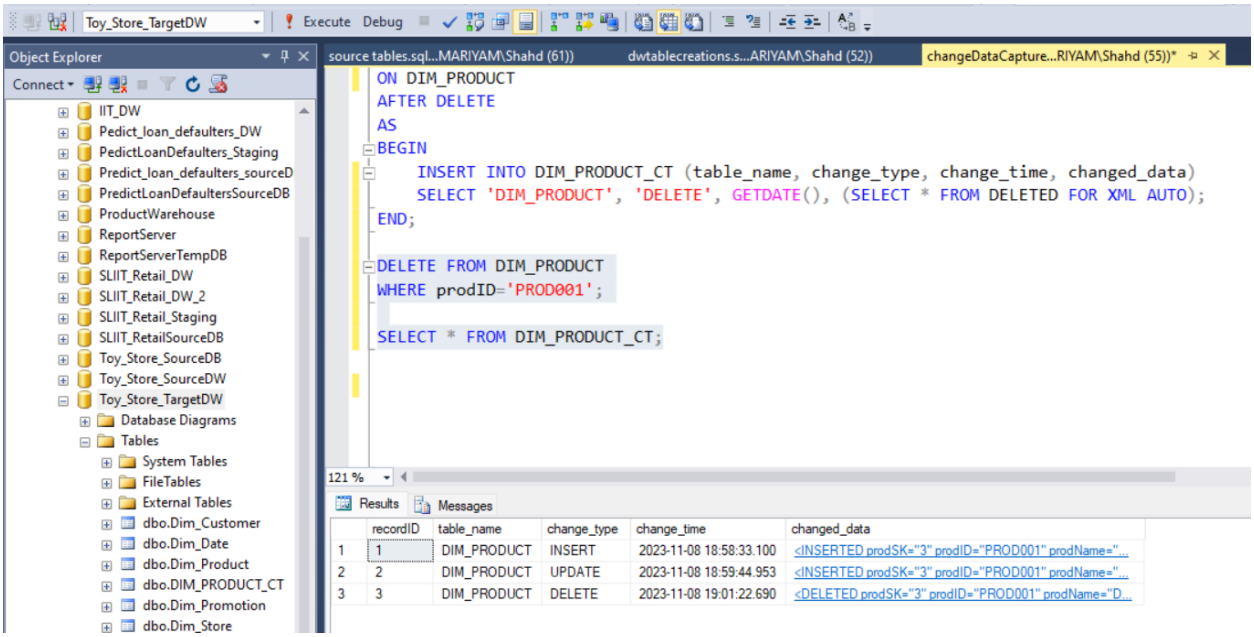
❖ Creating trigger for update operations





❖ Creating trigger for delete operations




Toy_Store_TargetDW
Execute
Debug
Object Explorer
Connect
IIT_DW
Pedict_loan_defaulters_DW
PedictLoanDefaulters_Staging
Predict_loan_defaulters_sourceD
PredictLoanDefaultersSourceDB
ProductWarehouse
ReportServer
ReportServerTempDB
SLIIT_Retail_DW
SLIIT_Retail_DW_2
SLIIT_Retail_Staging
SLIIT_RetailSourceDB
Toy_Store_SourceDB
Toy_Store_SourceDW
Toy_Store_TargetDW
Database Diagrams
Tables
System Tables
FileTables
External Tables
dbo.Dim_Customer
dbo.Dim_Date
dbo.Dim_Product
dbo.DIM_PRODUCT_CT
dbo.Dim_Promotion
dbo.Dim_Store
source tables.sql...MARIYAM\Shahd (61))
dwtablecreations.s...ARIYAM\Shahd (52))
changeDataCapture...RIYAM\Shahd (55))*
ON DIM_PRODUCT
AFTER DELETE
AS
BEGIN
INSERT INTO DIM_PRODUCT_CT (table_name, change_type, change_time, changed_data)
SELECT 'DIM_PRODUCT', 'DELETE', GETDATE(), (SELECT * FROM DELETED FOR XML AUTO);
END;
DELETE FROM DIM_PRODUCT
WHERE prodID='PROD001';
SELECT * FROM DIM_PRODUCT_CT;
121 %
Results
Messages
recordID table_name change_type change_time changed_data
1 1 DIM_PRODUCT INSERT 2023-11-08 18:58:33.100 <INSERTED prodSK="3" prodID="PROD001" prodName="...
2 2 DIM_PRODUCT UPDATE 2023-11-08 18:59:44.953 <INSERTED prodSK="3" prodID="PROD001" prodName="...
3 3 DIM_PRODUCT DELETE 2023-11-08 19:01:22.690 <DELETED prodSK="3" prodID="PROD001" prodName="D...

SECTION E

- **PART 1**

The scenario requires the implementation to be dynamic. SQL Server Management Studio does not have this feature, thus live SQL is used.

- Create the sales table and insert sample records

```
--CREATE THE SALES TABLE
CREATE TABLE PRODUCT_SALES(
    PRODUCT VARCHAR(20),
    YEAR INT,
    SALE NUMBER(20,2)
);

-- Insert Data to PRODUCT_SALES table
INSERT INTO PRODUCT_SALES VALUES('Prod1',2020,45000);
INSERT INTO PRODUCT_SALES VALUES('Prod1',2021,999999);
INSERT INTO PRODUCT_SALES VALUES('Prod1',2022,155000);
INSERT INTO PRODUCT_SALES VALUES('Prod2',2020,9999);
INSERT INTO PRODUCT_SALES VALUES('Prod2',2021,50000);
INSERT INTO PRODUCT_SALES VALUES('Prod2',2022,100000);
INSERT INTO PRODUCT_SALES VALUES('Prod3',2020,99999);
```

```
1 row(s) inserted.
```

- Create another table to store the years as it is required to generate the dynamic view

```

-- CREATE A TABLE TO STORE THE YEARS
CREATE TABLE YEAR_TAB(
    YEARS clob
);
```

```
Table created.
```

- Create the materialized view by fetching the data from sales table, followed by extracting the year dynamically to trigger and refresh on demand.

```sql
-- CREATE VIEW
DECLARE
    dynamic_years CLOB;
    sql_query CLOB;
BEGIN
    -- Clear existing data in YEAR_TAB
    DELETE FROM YEAR_TAB;

    -- Insert unique years from PRODUCT_SALES into YEAR_TAB
    INSERT INTO YEAR_TAB (YEARS)
    SELECT DISTINCT TO_CHAR(YEAR) FROM PRODUCT_SALES;

    -- Get the dynamic years from YEAR_TAB
    SELECT LISTAGG(YEARS, ',') WITHIN GROUP (ORDER BY YEARS DESC)
    INTO dynamic_years
    FROM (
        SELECT TO_CHAR(YEARS) AS YEARS
        FROM YEAR_TAB
        ORDER BY YEARS DESC
    );

    EXECUTE IMMEDIATE 'DROP MATERIALIZED VIEW MV_VIEW1';

    -- Build the dynamic SQL query
    sql_query := 'CREATE MATERIALIZED VIEW mv_view1
                  BUILD IMMEDIATE
                  REFRESH FORCE
                  ON DEMAND
                  ENABLE QUERY REWRITE
                  AS
                  SELECT * FROM (
                      SELECT ps.PRODUCT,
                             TO_CHAR(ps.YEAR) AS YEAR,
                             COALESCE(ps.SALE, 0) AS SALE
                      FROM YEAR_TAB yt
                      LEFT JOIN PRODUCT_SALES ps ON TO_CHAR(yt.YEARS) = TO_CHAR(ps.YEAR)
                  )
                  PIVOT (
                      MAX(SALE) FOR YEAR IN (' || dynamic_years || ')
                  )
                  ORDER BY PRODUCT';

    -- Execute the dynamic SQL query
    EXECUTE IMMEDIATE sql_query;

    -- Refresh the materialized view
    DBMS_MVIEW.REFRESH('MV_VIEW1');
END;
```

Statement processed.

- The view output

```sql
-- View the output
SELECT * FROM MV_VIEW1;
```

PRODUCT	2022	2021	2020
Prod1	155000	999999	45000
Prod2	100000	50000	9999
Prod3	480000	175000	99999

- Insert records to sales table and refresh the view to check if records are added

```
75
76  -- Insert more records to the sales table
77  INSERT INTO PRODUCT_SALES VALUES('Prod1',2023,500000);
78  INSERT INTO PRODUCT_SALES VALUES('Prod2',2023,175000);
79  INSERT INTO PRODUCT_SALES VALUES('Prod3',2023,480000);
80
81  -- REFRESH VIEW
82  EXECUTE DBMS_MVIEW.REFRESH('MV_VIEW1');
83  SELECT * FROM MV_VIEW1;
```

PRODUCT	2023	2022	2021	2020
Prod1	500000	155000	999999	45000
Prod2	175000	100000	50000	9999
Prod3	480000	480000	175000	99999

Accordingly, the view is updated when a new record is inserted

- **PART 2**

The product table is created and sample data are inserted for the illustration

```
Product Hieararch...ARIYAM\Shahd (57))
CREATE TABLE PRODUCT (
    PRODUCT_ID INT IDENTITY(1,1),
    PRODUCT VARCHAR(100),
    PRODUCT_CATEGORY VARCHAR(50),
    PRODUCT_TYPE VARCHAR(50),
    SALES NUMERIC(20,2)
);

-- Insert sample data
INSERT INTO PRODUCT VALUES ('Prod1', 'Cat1', 'Type1', 1000);
INSERT INTO PRODUCT VALUES ('Prod2', 'Cat1', 'Type1', 1500);
INSERT INTO PRODUCT VALUES ('Prod3', 'Cat2', 'Type2', 2000);
INSERT INTO PRODUCT VALUES ('Prod4', 'Cat2', 'Type2', 1200);
INSERT INTO PRODUCT VALUES ('Prod5', 'Cat3', 'Type1', 800);
```

121 %

Messages

```
(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)

(1 row(s) affected)
```

1. Hierarchy Summary

```
-- Summary hierarchy
SELECT
    PRODUCT_TYPE,
    PRODUCT_CATEGORY,
    PRODUCT,
    SUM(SALES) AS TOTAL_SALES
FROM
    PRODUCT
GROUP BY
ROLLUP
    (PRODUCT_TYPE, PRODUCT_CATEGORY, PRODUCT);
```

21 %

Results | Messages

	PRODUCT_TYPE	PRODUCT_CATEGORY	PRODUCT	TOTAL_SALES
1	Type1	Cat1	Prod1	1000.00
2	Type1	Cat1	Prod2	1500.00
3	Type1	Cat1	NULL	2500.00
4	Type1	Cat3	Prod5	800.00
5	Type1	Cat3	NULL	800.00
6	Type1	NULL	NULL	3300.00
7	Type2	Cat2	Prod3	2000.00
8	Type2	Cat2	Prod4	1200.00
9	Type2	Cat2	NULL	3200.00
10	Type2	NULL	NULL	3200.00
11	NULL	NULL	NULL	6500.00

2. Summary for combination

```
-- Hierarchy for each combination
SELECT
    PRODUCT_TYPE,
    PRODUCT_CATEGORY,
    PRODUCT,
    SUM(SALES) AS TOTAL_SALES
FROM
    PRODUCT
GROUP BY
CUBE
    (PRODUCT_TYPE, PRODUCT_CATEGORY, PRODUCT);
```

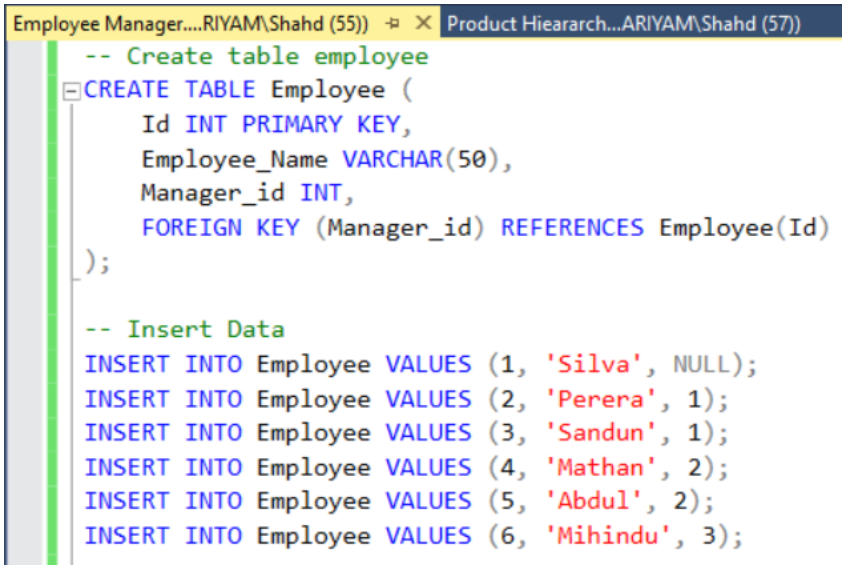
121 %

Results | Messages

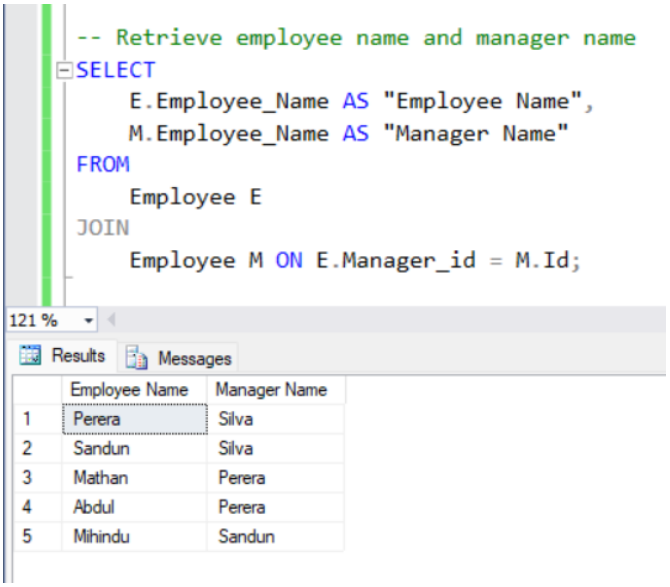
	PRODUCT_TYPE	PRODUCT_CATEGORY	PRODUCT	TOTAL_SALES
1	Type1	Cat1	Prod1	1000.00
2	NULL	Cat1	Prod1	1000.00
3	NULL	NULL	Prod1	1000.00
4	Type1	Cat1	Prod2	1500.00
5	NULL	Cat1	Prod2	1500.00
6	NULL	NULL	Prod2	1500.00
7	Type2	Cat2	Prod3	2000.00
8	NULL	Cat2	Prod3	2000.00
9	NULL	NULL	Prod3	2000.00
10	Type2	Cat2	Prod4	1200.00
11	NULL	Cat2	Prod4	1200.00
12	NULL	NULL	Prod4	1200.00
13	Type1	Cat3	Prod5	800.00
14	NULL	Cat3	Prod5	800.00
15	NULL	NULL	Prod5	800.00

- **PART 3**

Create the employee table and insert sample data



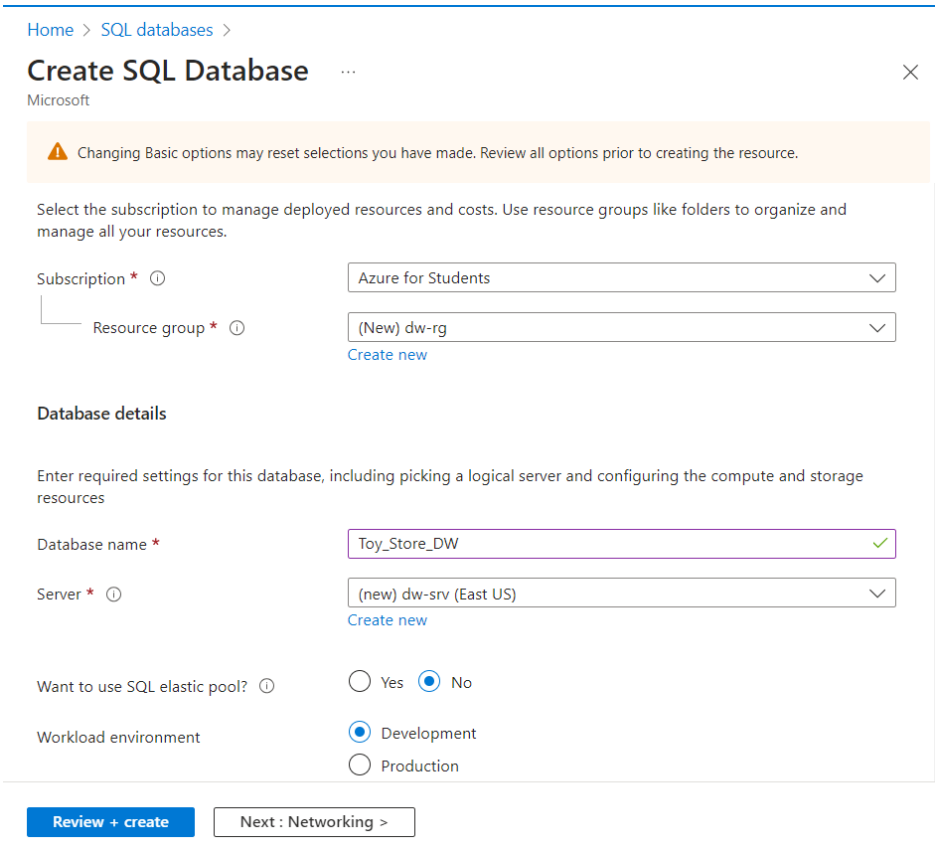
Query and Output are as follows:

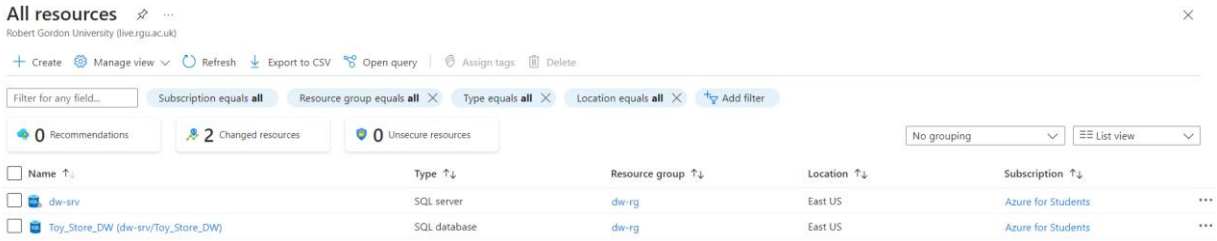


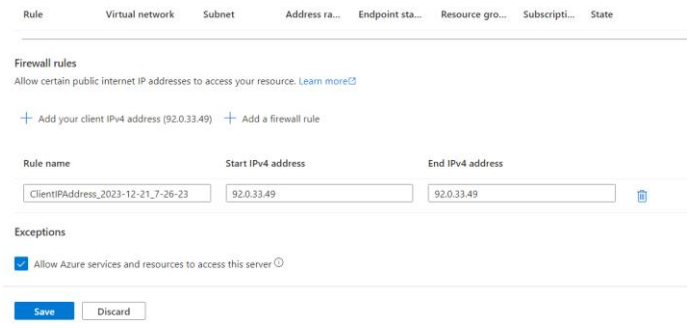
	Employee Name	Manager Name
1	Perera	Silva
2	Sandun	Silva
3	Mathan	Perera
4	Abdul	Perera
5	Mihindu	Sandun

SECTION F
PART 1

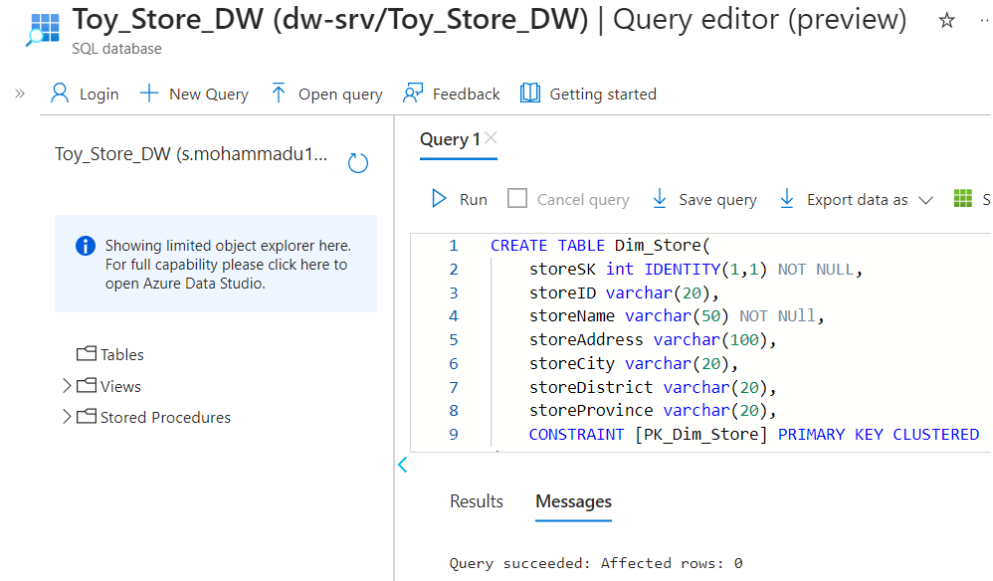
- Initially, a SQL database and a server is created using SQL Database service in Azure Portal. After creation, the server has to be modified by adding a firewall rule to provide IPV4 access to the database.



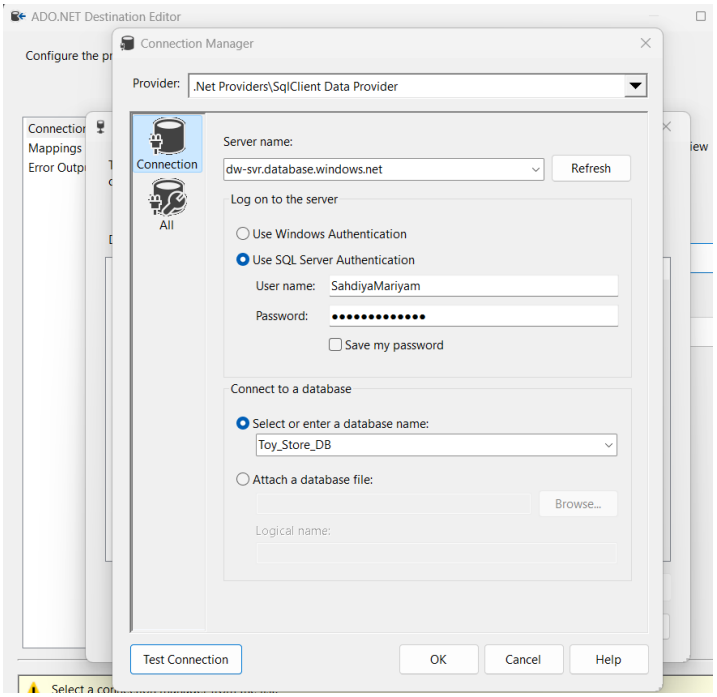




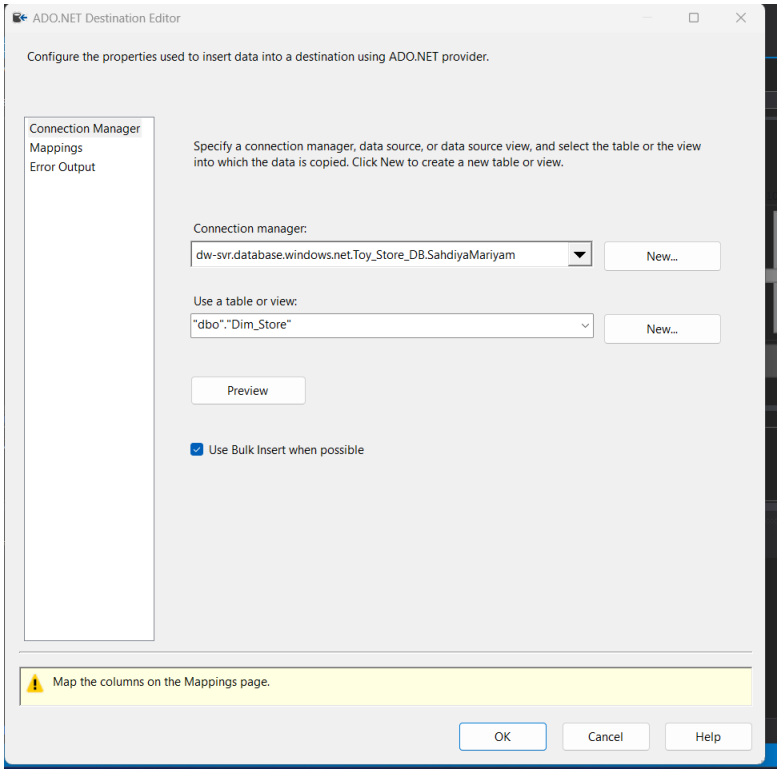
- Create the Store Dimension table using query editor option in the created database



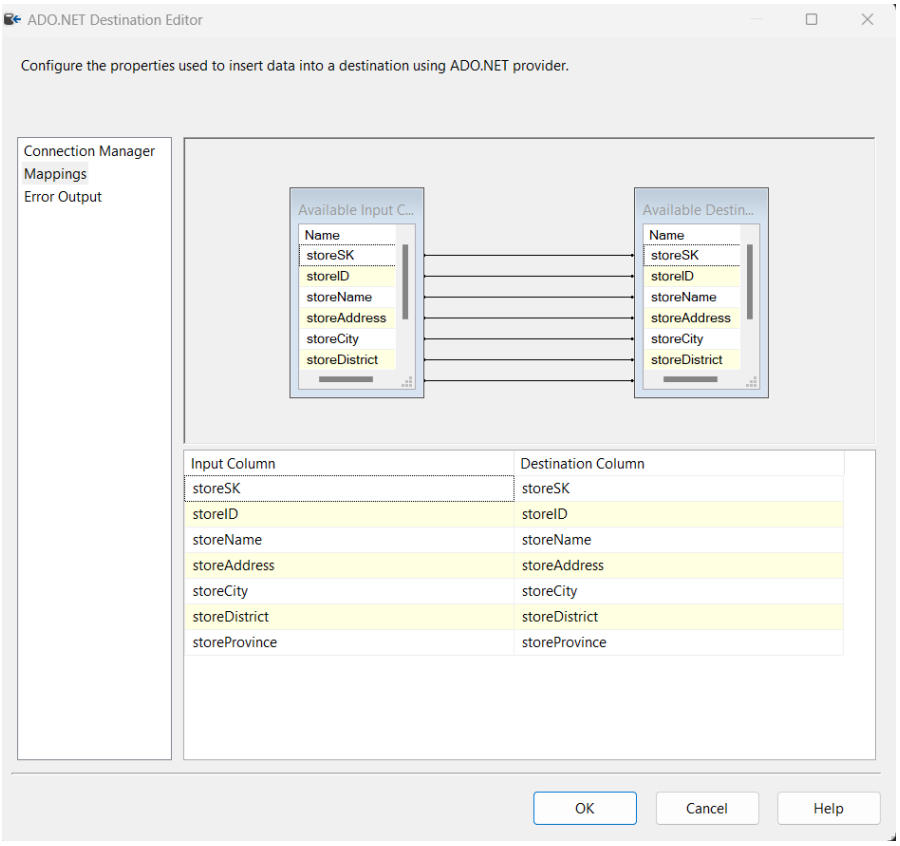
- Open SSIS and create a new integration project
- Add a Data Flow task in Control Area
- Add a source assistant/OLE DB Source and connect it to the Store dimension
- Add 'ADO.NET' destination task and connect to the server created in Azure Cloud.



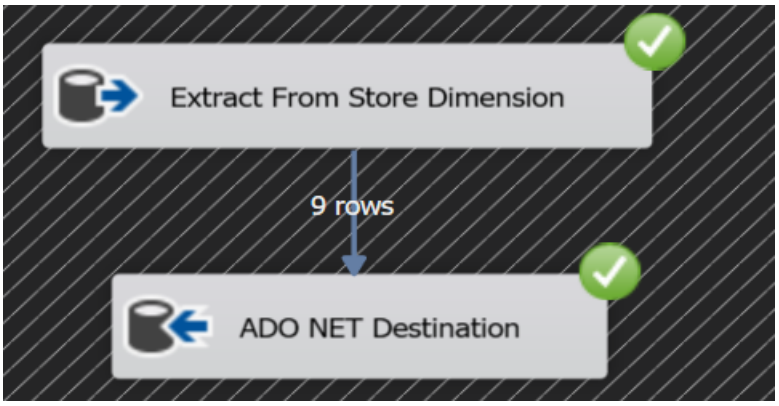
- In the connection manager connect to the table created in Azure Cloud.



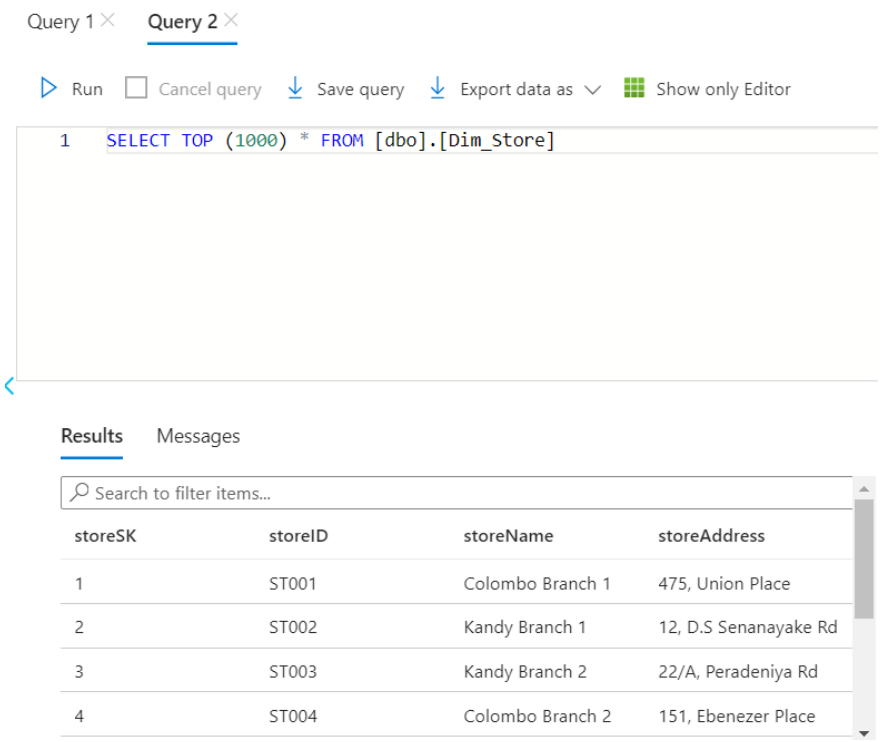
- Map the columns as required



- Execute the package



- Verify the data load in Azure Cloud



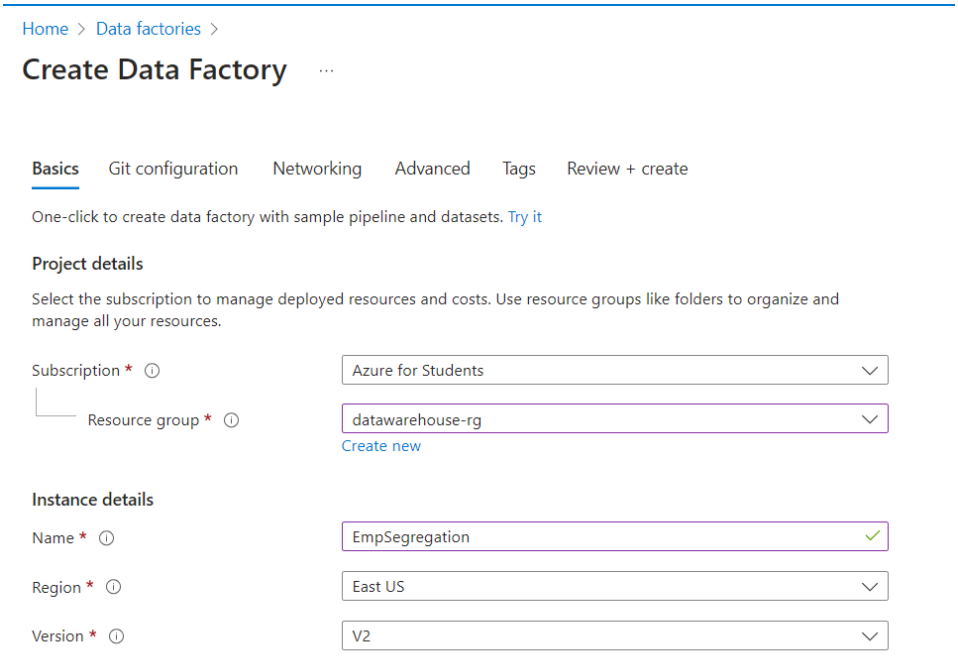
PART 2

- Create three tables ‘Employee,’ ‘ExecutiveEmployee,’ and ‘NonExecutiveEmployee’ and insert data into the Employee Table

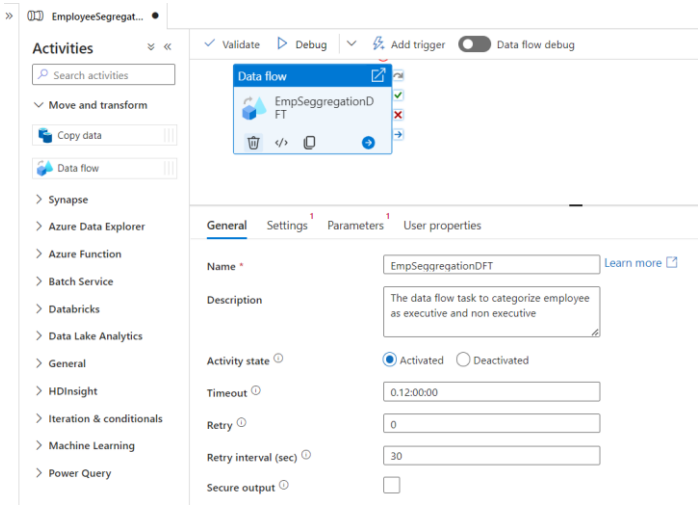
```sql
-- CREATE TABLE EMPLOYEE
CREATE TABLE Employee(
emp_id NVARCHAR(20) PRIMARY KEY NOT NULL,
emp_name NVARCHAR(20),
designation NVARCHAR(20),
);

-- CREATE TABLE EXECUTIVE EMPLOYEE
CREATE TABLE ExecutiveEmployee(
emp_id NVARCHAR(20) PRIMARY KEY NOT NULL,
emp_name NVARCHAR(20),
designation NVARCHAR(20),
);

-- CREATE TABLE NON-EXECUTIVE EMPLOYEE
CREATE TABLE NonExecutiveEmployee(
emp_id NVARCHAR(20) PRIMARY KEY NOT NULL,
emp_name NVARCHAR(20),
designation NVARCHAR(20),
);
```

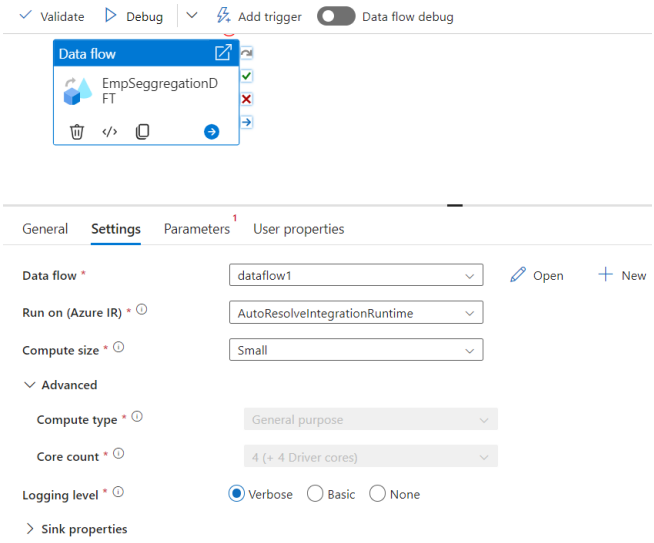
```sql
-- INSERT DATA
INSERT INTO Employee VALUES ('EMP001','Murugan','Executive');
INSERT INTO Employee VALUES ('EMP002','Arunan','Executive');
INSERT INTO Employee VALUES ('EMP003','Perera','Non-Executive');
INSERT INTO Employee VALUES ('EMP004','Steve','Executive');
INSERT INTO Employee VALUES ('EMP005','Ayeisha','Non-Executive');
INSERT INTO Employee VALUES ('EMP006','Nimalka','Non-Executive');
INSERT INTO Employee VALUES ('EMP007','Jacob','Non-Executive');
INSERT INTO Employee VALUES ('EMP008','Sachini','Executive');
INSERT INTO Employee VALUES ('EMP009','Shalini','Non-Executive');
```

- Navigate to Azure Data Factory service and create a new data factory

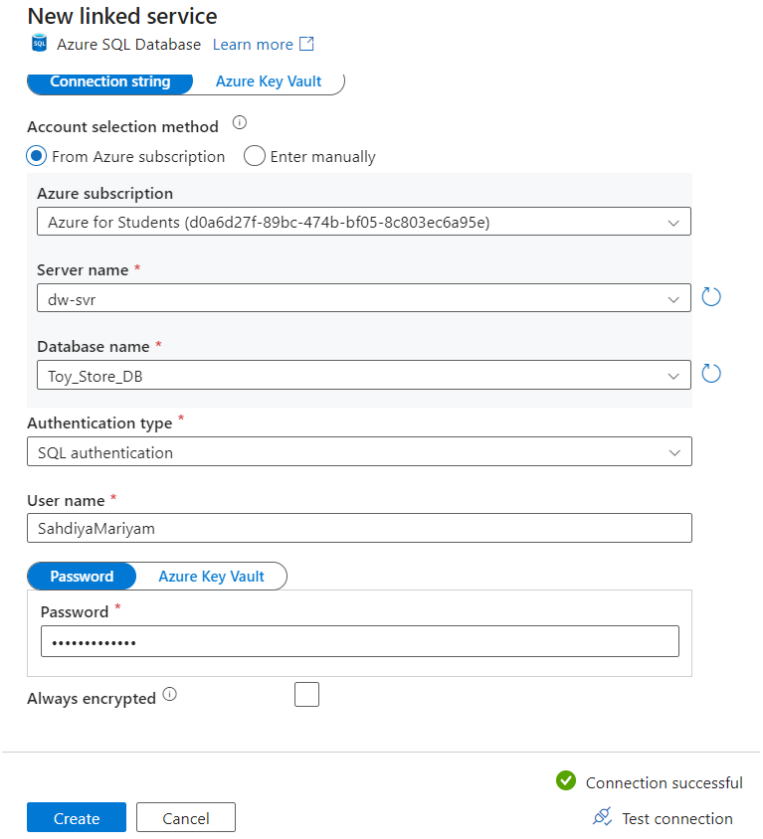


- Launch the Data Factory Studio for the created Data Factory
- Create a new Data Pipeline
- Add a data flow task and configure:

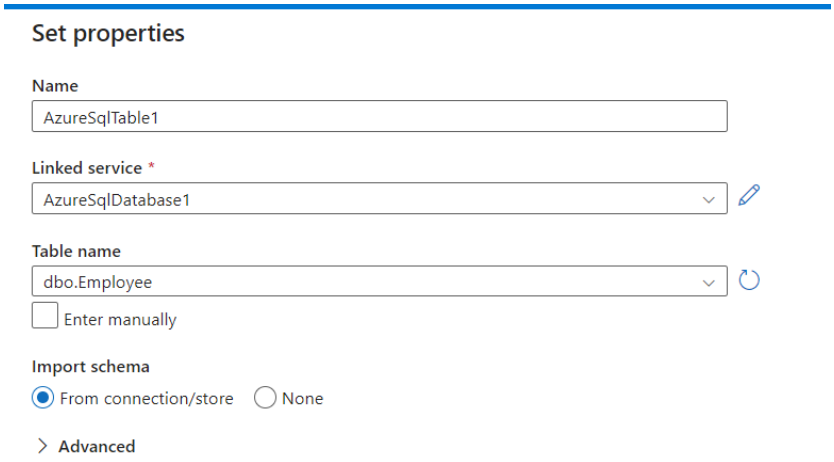




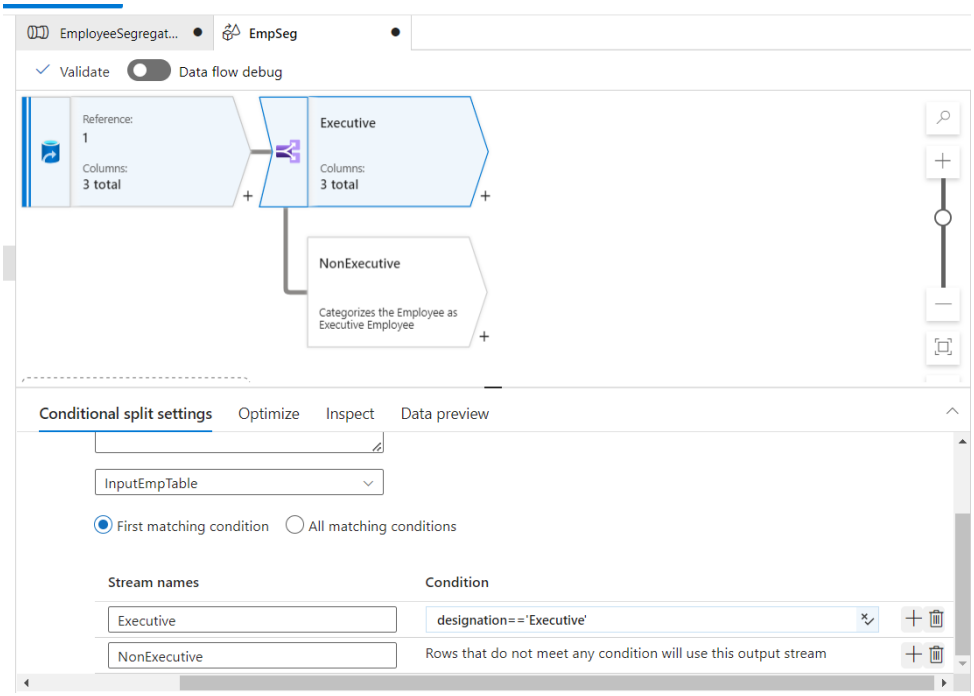
- Move to the data flow task and provide the input source as the Employee table. A linked service must be created to connect to the Database server.



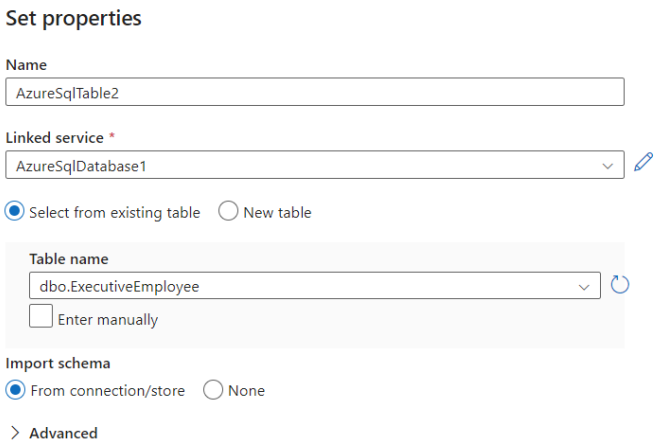
- Once the linked service is created, specify the table name



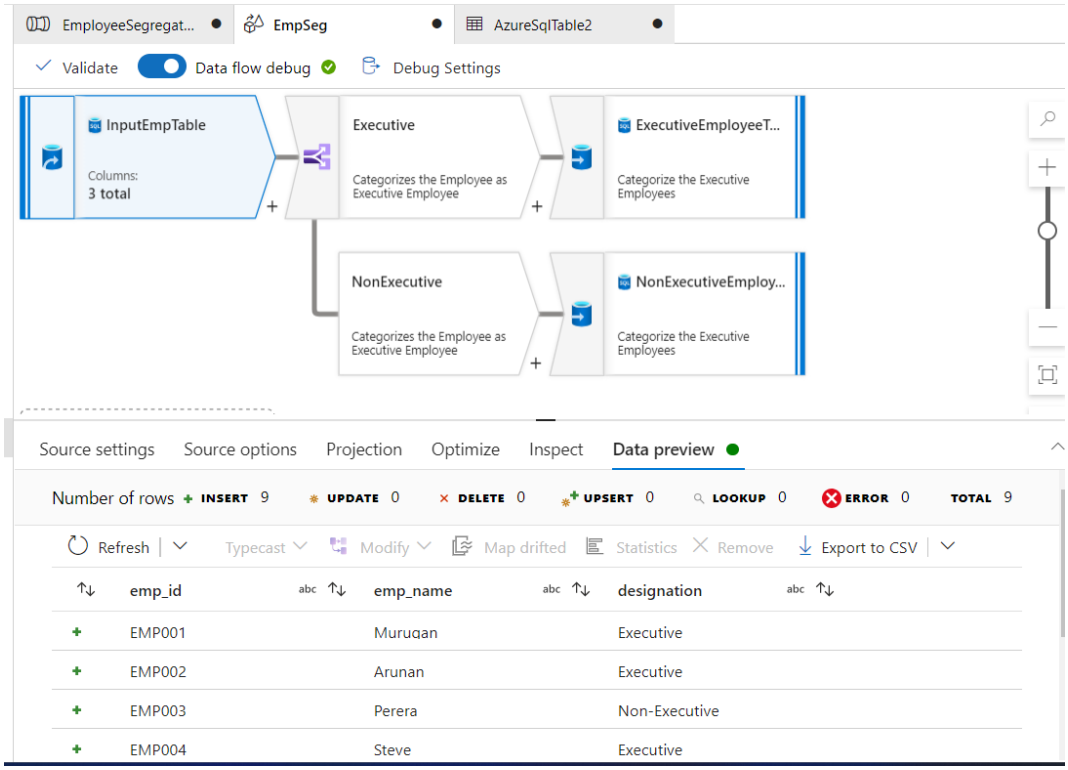
- After adding the source, add a conditional split and configure as follows to segregate the employees



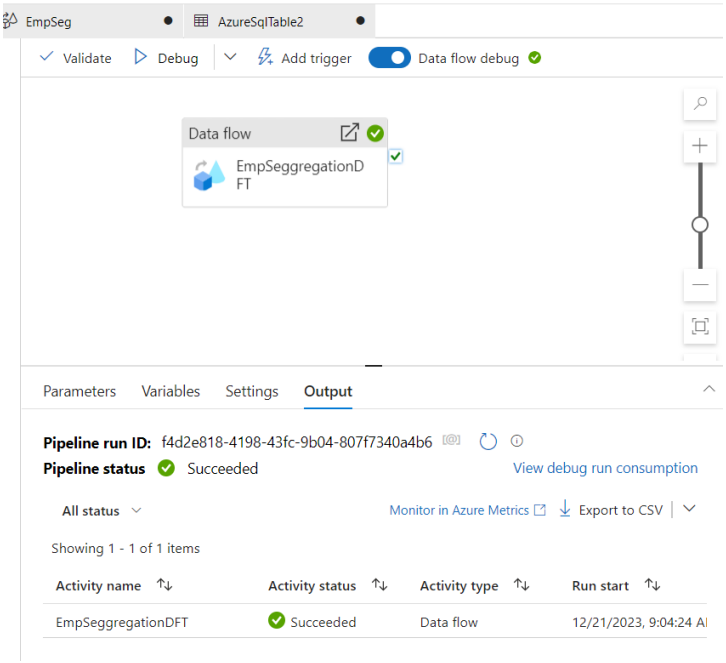
- Now add destination to both the splits and configure it to connect to the Executive and Non-Executive Employee Tables



- Debug to preview the split



- Debug the data flow task to segregate the employees to their respective groups



- Query the tables to verify the implementation

Results Messages

Search to filter items...

emp_id	emp_name	designation
EMP001	Murugan	Executive
EMP002	Arunan	Executive
EMP004	Steve	Executive
EMP008	Sachini	Executive

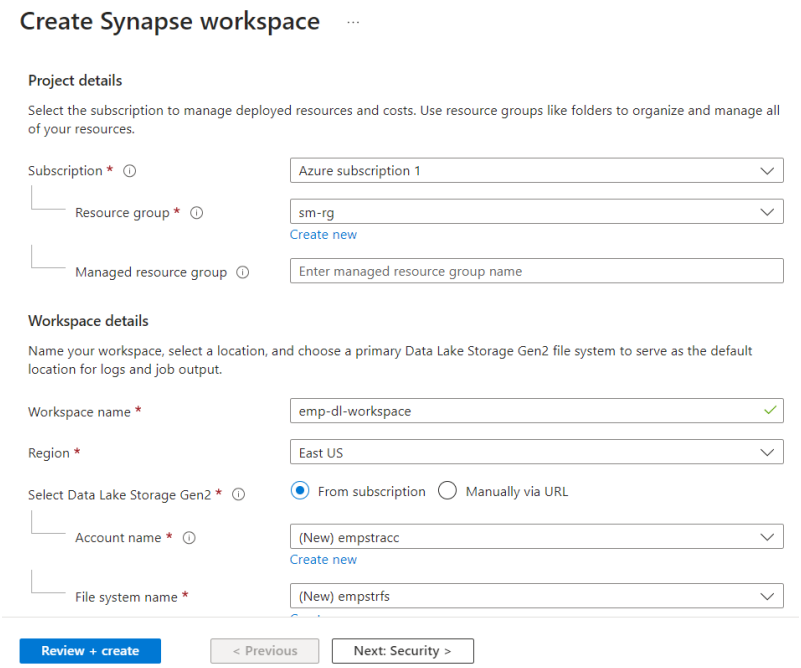
Results Messages

Search to filter items...

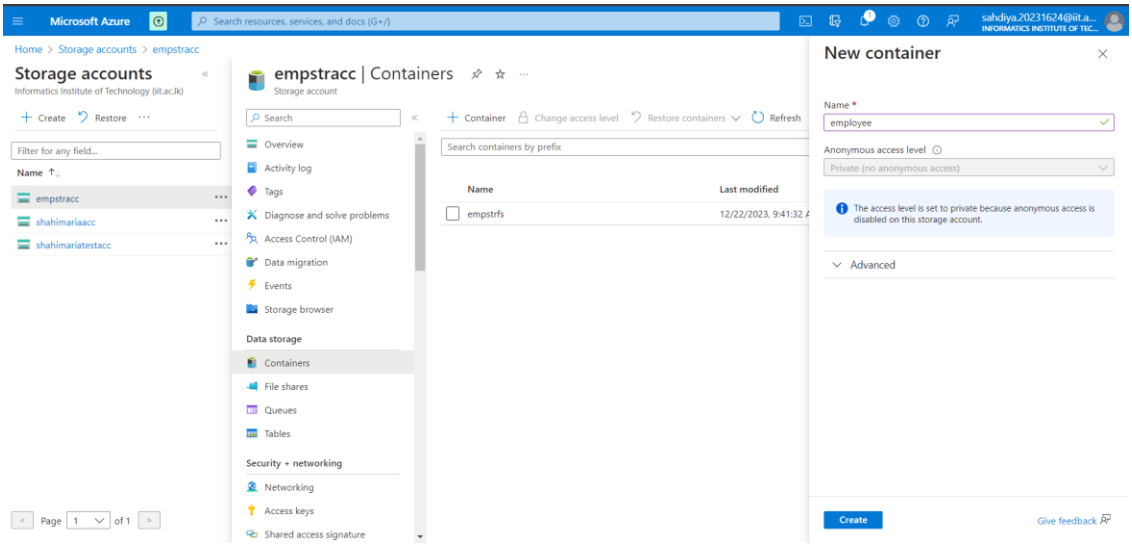
emp_id	emp_name	designation
EMP003	Perera	Non-Executive
EMP005	Ayeisha	Non-Executive
EMP006	Nimalka	Non-Executive
EMP007	Jacob	Non-Executive
EMP009	Shalini	Non-Executive

PART 3

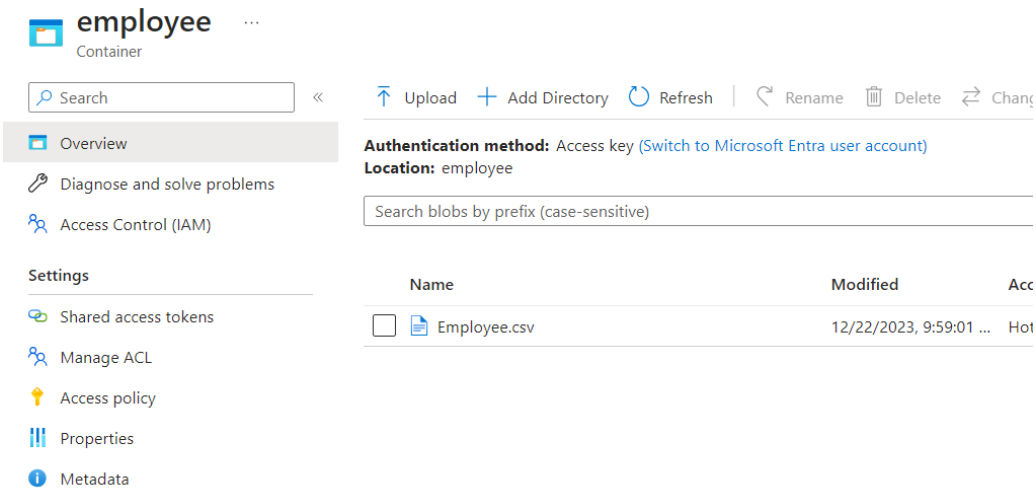
- Open Azure Synapse Analytics and create a workspace by providing Storage Account Name



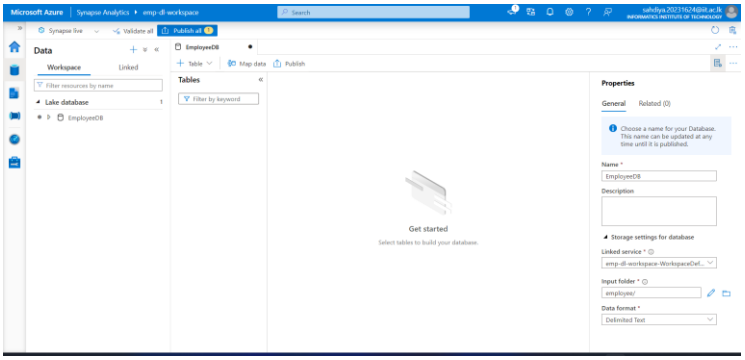
- Move to Azure Storage Accounts. Select the storage account created above and create a container

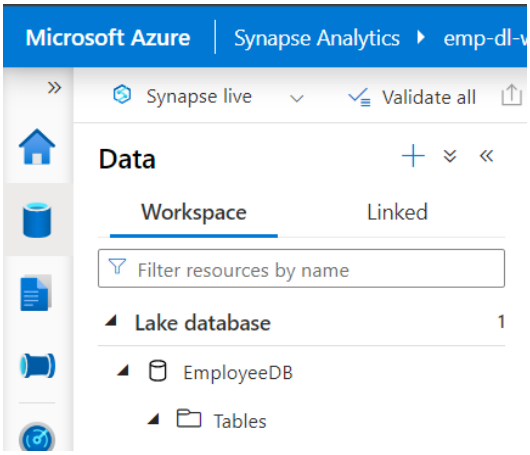


- Azure Synapse does not support direct creation of tables. Therefore, we will upload a csv file with dataset to the container

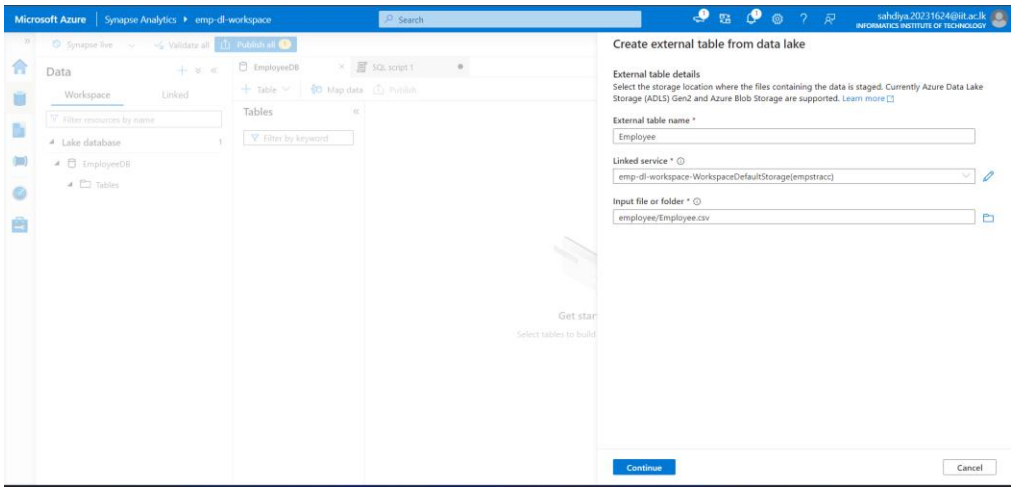


- Open Synapse Studio
- Create a Lake Database using the data option

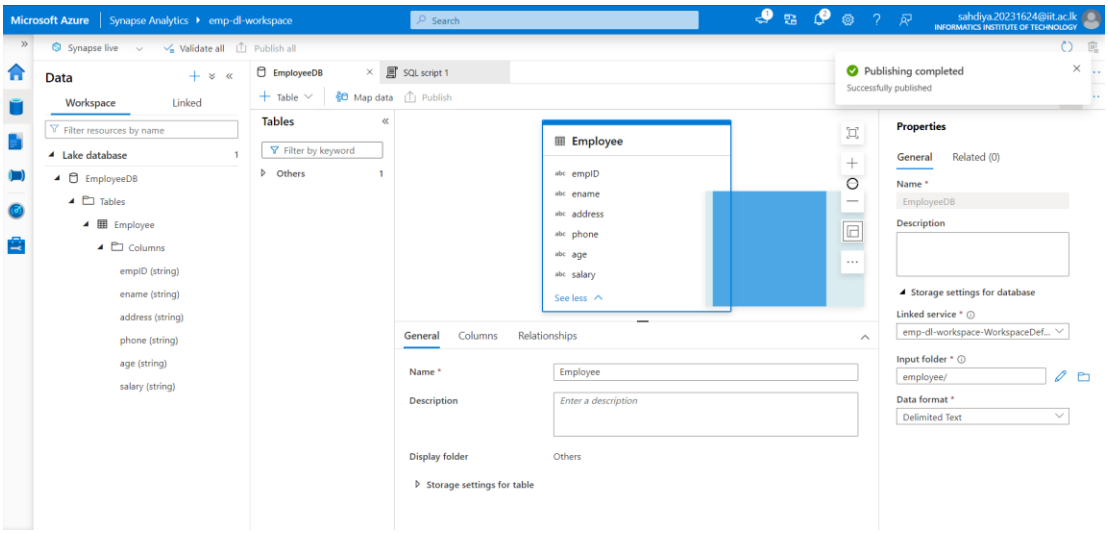




- Open the ‘EmployeeDB’ and create a table from Data Lake. Specify the location to the csv file in the container. Ensure to check the option to infer column names from first row.



- Now publish all changes to create the table



SECTION G

PART 1

In large-scale business solutions, data partitioning enhances scalability, mitigates contention, and optimizes performance. Choosing the right strategy is crucial. Three common approaches are:

1. **Range Partitioning**: Groups records with identical key field values into equal-sized partitions, improving sorting operation efficiency. For instance, in a sales database, partitioning by transaction dates enhances storage and retrieval for specific time intervals.
2. **List Partitioning**: Explicitly maps data rows to partitions based on unique values, facilitating natural organization. For a customer database, list partitioning based on customer types streamlines data organization for targeted marketing.
3. **Hash Partitioning**: Ensures equitable data distribution across partitions based on hash key fields, improving system load balancing. Applied to user IDs in a user database, it optimizes overall system efficiency.

These strategies are vital for efficient data management and tailored database architecture.

PART 2

- Range Partitioning

Creating the table with range portioning

```
-- Create a table with range partitioning for sales data, partitioned by year
CREATE TABLE sales_data_range_partition (
    sale_id INT PRIMARY KEY,
    product_id INT,
    year INT,
    sale_amount DECIMAL(10, 2),
    location VARCHAR(50)
)
PARTITION BY RANGE (year) (
    PARTITION p2016 VALUES LESS THAN (2017),
    PARTITION p2017 VALUES LESS THAN (2018),
    PARTITION p2018 VALUES LESS THAN (2019),
    PARTITION p2019 VALUES LESS THAN (2020),
    PARTITION p2020 VALUES LESS THAN (2021),
    PARTITION p2021 VALUES LESS THAN (2022),
    PARTITION p2022 VALUES LESS THAN (2023),
    PARTITION p2023 VALUES LESS THAN (MAXVALUE)
);
```

```
Table created.
```

Inserting dummy data to the table

```
-- Insert sample data into the sales_data table
INSERT INTO sales_data_range_partition VALUES (1, 101, 2017, 500.00, 'Colombo');
INSERT INTO sales_data_range_partition VALUES (2, 102, 2018, 700.50, 'Gampaha');
INSERT INTO sales_data_range_partition VALUES (3, 101, 2019, 450.25, 'Kandy');
INSERT INTO sales_data_range_partition VALUES (4, 103, 2020, 900.75, 'Colombo');
INSERT INTO sales_data_range_partition VALUES (5, 104, 2021, 400.00, 'Anuradhapura');
INSERT INTO sales_data_range_partition VALUES (6, 105, 2022, 350.85, 'Galle');
INSERT INTO sales_data_range_partition VALUES (7, 106, 2023, 1000.00, 'Kandy');
```

```
1 row(s) inserted.

1 row(s) inserted.
```

View the range partitions

```
-- View Partion
SELECT
    TABLE_NAME,
    PARTITION_NAME,
    HIGH_VALUE
FROM
    USER_TAB_PARTITIONS
WHERE
    TABLE_NAME = 'SALES_DATA_RANGE_PARTITION';
```

TABLE_NAME	PARTITION_NAME	HIGH_VALUE
SALES_DATA_RANGE_PARTITION	P2016	2017
SALES_DATA_RANGE_PARTITION	P2017	2018
SALES_DATA_RANGE_PARTITION	P2018	2019
SALES_DATA_RANGE_PARTITION	P2019	2020

❖ List Partitioning

Creating the table with list partitioning

```
-- Create a table with list partitioning for s
CREATE TABLE sales_data_list_partition (
    sale_id INT PRIMARY KEY,
    product_id INT,
    year INT,
    sale_amount DECIMAL(10, 2),
    location VARCHAR(50)
)
PARTITION BY LIST (location) (
    PARTITION p_colombo VALUES ('Colombo'),
    PARTITION p_kandy VALUES ('Kandy'),
    PARTITION p_matara VALUES ('Matara'),
    PARTITION p_galle VALUES ('Galle'),
    PARTITION p_other VALUES (DEFAULT)
);
```

```
Table created.
```

Inserting dummy data

```
-- Insert sample data into the sales_data_list_partitioned table
INSERT INTO sales_data_list_partition VALUES (1, 101, 2020, 500.00, 'Colombo');
INSERT INTO sales_data_list_partition VALUES (2, 102, 2020, 700.50, 'Kandy');
INSERT INTO sales_data_list_partition VALUES (3, 201, 2021, 450.25, 'Matara');
INSERT INTO sales_data_list_partition VALUES (4, 301, 2021, 900.75, 'Galle');

```

```
1 row(s) inserted.

1 row(s) inserted.
```

Viewing the partitions

```
-- View Partion
SELECT
    TABLE_NAME,
    PARTITION_NAME,
    HIGH_VALUE
FROM
    USER_TAB_PARTITIONS
WHERE
    TABLE_NAME = 'SALES_DATA_LIST_PARTITION';
```

TABLE_NAME	PARTITION_NAME	HIGH_VALUE
SALES_DATA_LIST_PARTITION	P_COLOMBO	'Colombo'
SALES_DATA_LIST_PARTITION	P_GALLE	'Galle'
SALES_DATA_LIST_PARTITION	P_KANDY	'Kandy'
SALES_DATA_LIST_PARTITION	P_MATARA	'Matara'
SALES_DATA_LIST_PARTITION	P_OTHER	DEFAULT

❖ Hash Partitioning

Creating the hash partitioning table

```
76  -- Create a table with hash partitioning for sales da
77  CREATE TABLE sales_data_hash_partition (
78      sale_id INT PRIMARY KEY,
79      product_id INT,
80      year INT,
81      sale_amount DECIMAL(10, 2),
82      location VARCHAR(50)
83  )
84  PARTITION BY HASH (product_id)
85  PARTITIONS 4;
86
```

```
Table created.
```

Inserting dummy data

```
87  -- Insert sample data into the sales_data_hash_partitioned table
88  INSERT INTO sales_data_hash_partition VALUES (1, 101, 2020, 500.00, 'Colombo');
89  INSERT INTO sales_data_hash_partition VALUES (2, 102, 2020, 700.50, 'Kandy');
90  INSERT INTO sales_data_hash_partition VALUES (3, 201, 2021, 450.25, 'Matara');
91  INSERT INTO sales_data_hash_partition VALUES (4, 301, 2021, 900.75, 'Galle');
92
```

```
1 row(s) inserted.

1 row(s) inserted.
```

Viewing the partition

```
93  -- View Partion
94  SELECT
95      TABLE_NAME,
96      PARTITION_NAME,
97      HIGH_VALUE
98  FROM
99      USER_TAB_PARTITIONS
100 WHERE
101     TABLE_NAME = 'SALES_DATA_HASH_PARTITION';
```

TABLE_NAME	PARTITION_NAME	HIGH_VALUE
SALES_DATA_HASH_PARTITION	SYS_P587377	-
SALES_DATA_HASH_PARTITION	SYS_P587378	-
SALES_DATA_HASH_PARTITION	SYS_P587379	-
SALES_DATA_HASH_PARTITION	SYS_P587380	-

Download CSV

PART 3

A similar table to the above tables is created with the name ‘SALES_DATA’ and dummy data is inserted.

The table is altered to create partitions

```
118
119  -- ALTER TABLE TO PARTITION
120  ALTER TABLE SALES_DATA MODIFY
121      PARTITION BY RANGE (year) (
122      PARTITION p2016 VALUES LESS THAN (2017),
123      PARTITION p2017 VALUES LESS THAN (2018),
124      PARTITION p2018 VALUES LESS THAN (2019),
125      PARTITION p2019 VALUES LESS THAN (2020),
126      PARTITION p2020 VALUES LESS THAN (2021),
127      PARTITION p2021 VALUES LESS THAN (2022),
128      PARTITION p2022 VALUES LESS THAN (2023),
129      PARTITION p2023 VALUES LESS THAN (MAXVALUE)
```

```
Table altered.
```

View the partition

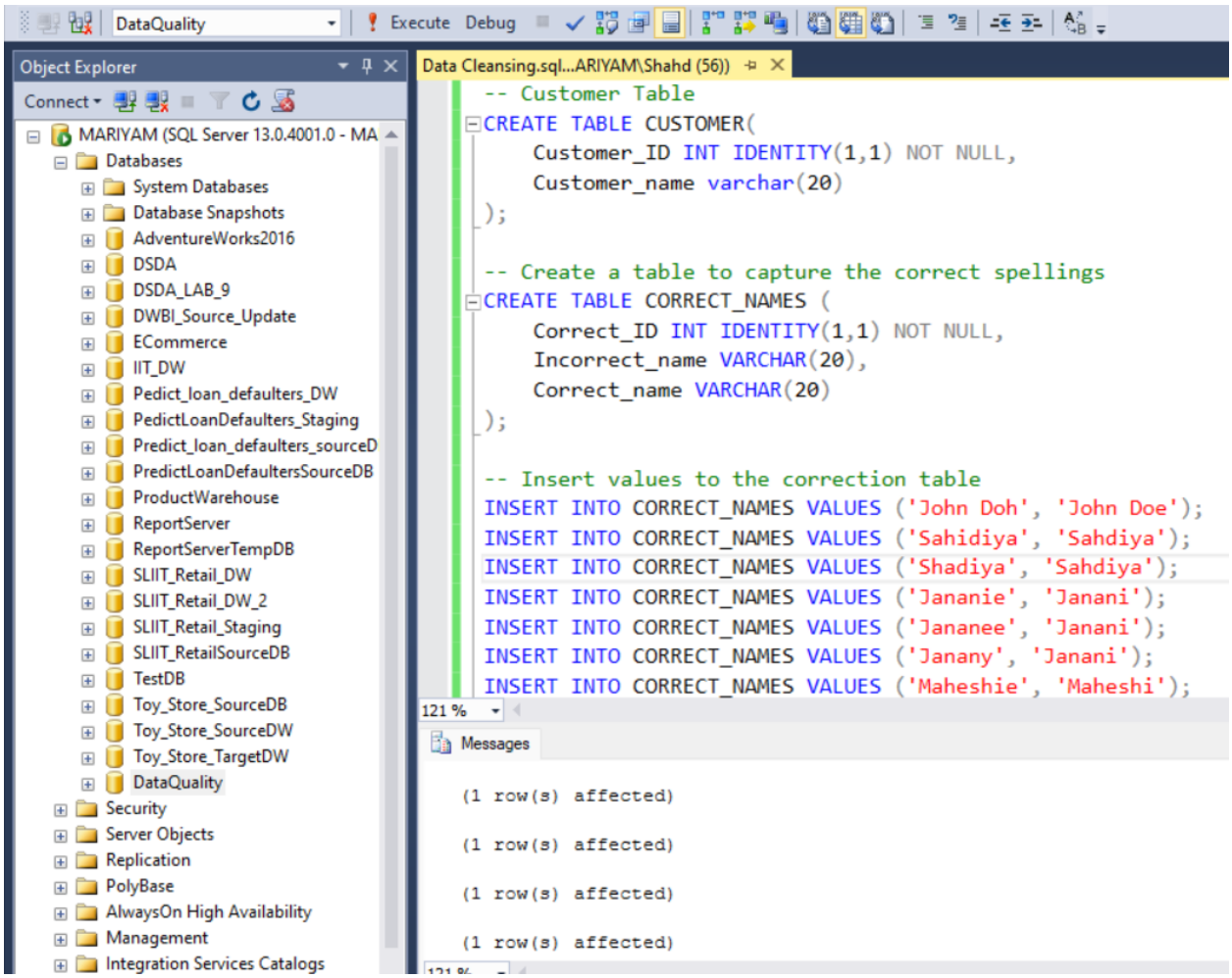
```
131
132
133  -- VIEW PARTITION
134  SELECT
135      TABLE_NAME,
136      PARTITION_NAME,
137      HIGH_VALUE
138  FROM
139      USER_TAB_PARTITIONS
140  WHERE
141      TABLE_NAME = 'SALES_DATA';
142
```

TABLE_NAME	PARTITION_NAME	HIGH_VALUE
SALES_DATA	P2016	2017
SALES_DATA	P2017	2018
SALES_DATA	P2018	2019
SALES_DATA	P2019	2020
SALES_DATA	P2020	2021

SECTION H

When a customer's name is consistently misspelled, it's vital to resolve the issue through effective data cleansing. Using a SQL script, we will construct a simple data model in the SQL Server database to tackle this discrepancy.

- Create a table that contains incorrect and correct names and insert data to the table.



- Create a trigger to check the name of customer when a record is inserted or updated and automatically update it to the correct name if there is a mistake

```
Data Cleansing.sql...ARIYAM\Shahd (56))
-- Create a trigger to ensure correct names are inserted or updated in CUSTOMER
CREATE TRIGGER trigger_EnforceCorrectNames
ON CUSTOMER
INSTEAD OF INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Update incorrect names to their correct forms
    UPDATE c
    SET Customer_name = cn.Correct_name
    FROM CUSTOMER c
    JOIN inserted i ON c.Customer_ID = i.Customer_ID
    JOIN CORRECT_NAMES cn ON c.Customer_name = cn.Incorrect_name;

    -- Insert only correct names
    INSERT INTO CUSTOMER (Customer_name)
    SELECT cn.Correct_name
    FROM inserted i
    JOIN CORRECT_NAMES cn ON i.Customer_name = cn.Incorrect_name
    WHERE NOT EXISTS (
        SELECT 1
        FROM CUSTOMER c
        WHERE c.Customer_ID = i.Customer_ID
    );
END;
121 %
Messages
Command(s) completed successfully.
121 %
```

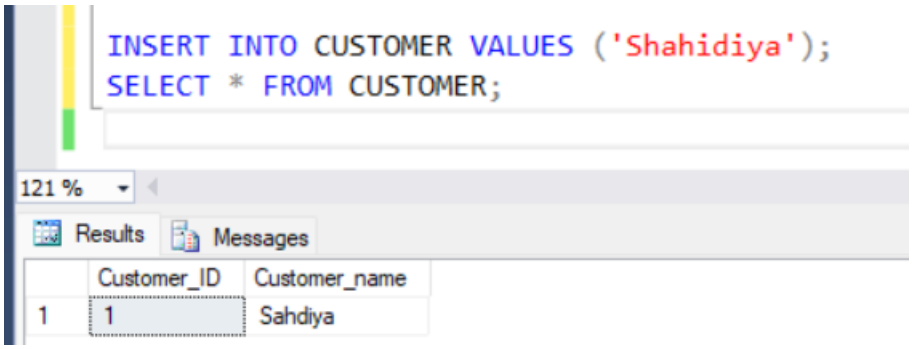
- Verify the trigger by inserting a record with incorrect name

```
-- Test by inserting a record with wrong name
INSERT INTO CUSTOMER VALUES ('Shadiya');
SELECT * FROM CUSTOMER;
21 %
Results
Messages
```

	Customer_ID	Customer_name
1	1	Sahdiya

As shown above, even when the name is specified incorrectly, the trigger enforces to insert the correct name.

In a case where the inserted incorrect name is not specified in the database, the trigger will automatically rollback the action and avoid inserting the name.



As shown in the image above, the inserted name is not specified in the correction database, therefore it avoids inserting it to the database.