

# 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 consider 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.

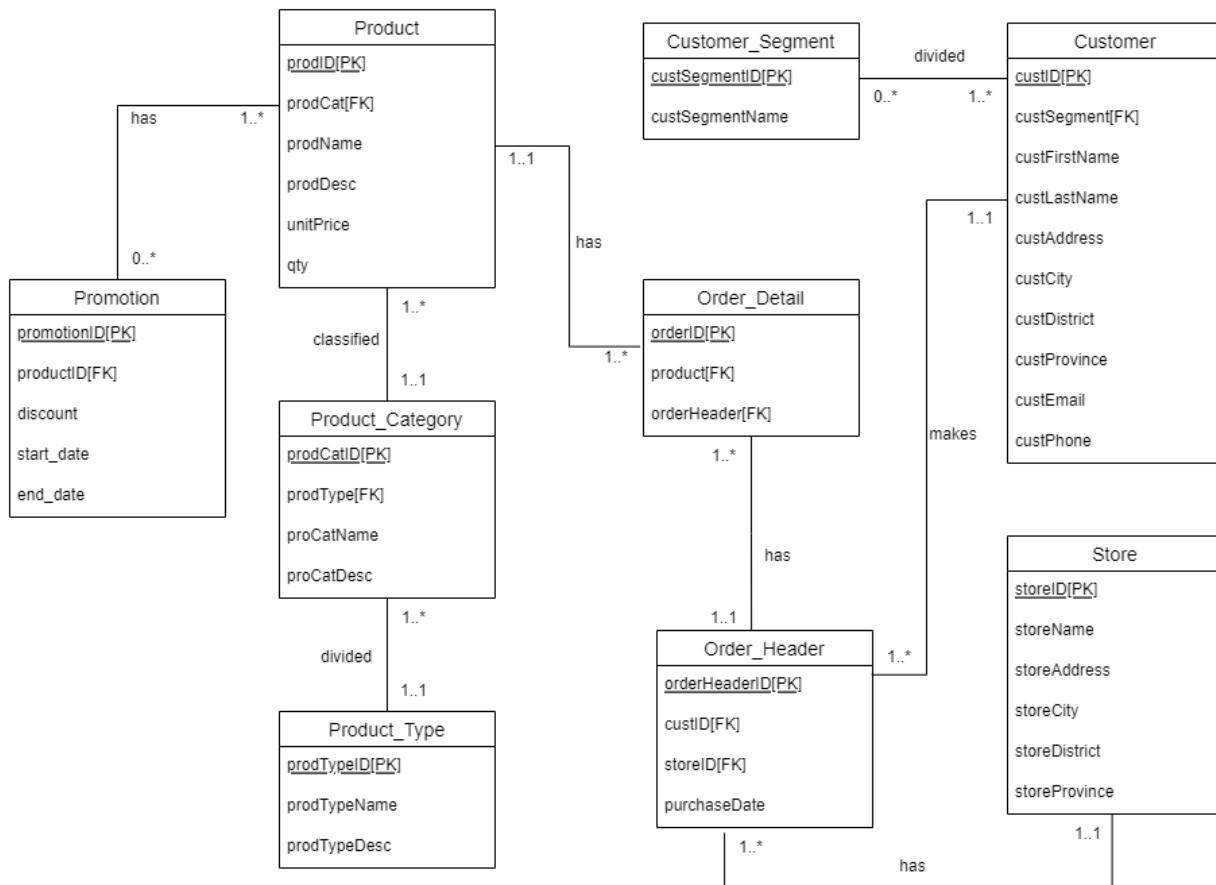
In conclusion, Data Warehousing Automation (DWA) represents a pivotal transformation in the data warehousing landscape, offering a solution that minimizes manual efforts, enhances efficiency, improves data quality, and fosters data-driven decision-making. As demonstrated by real-world examples from organizations 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

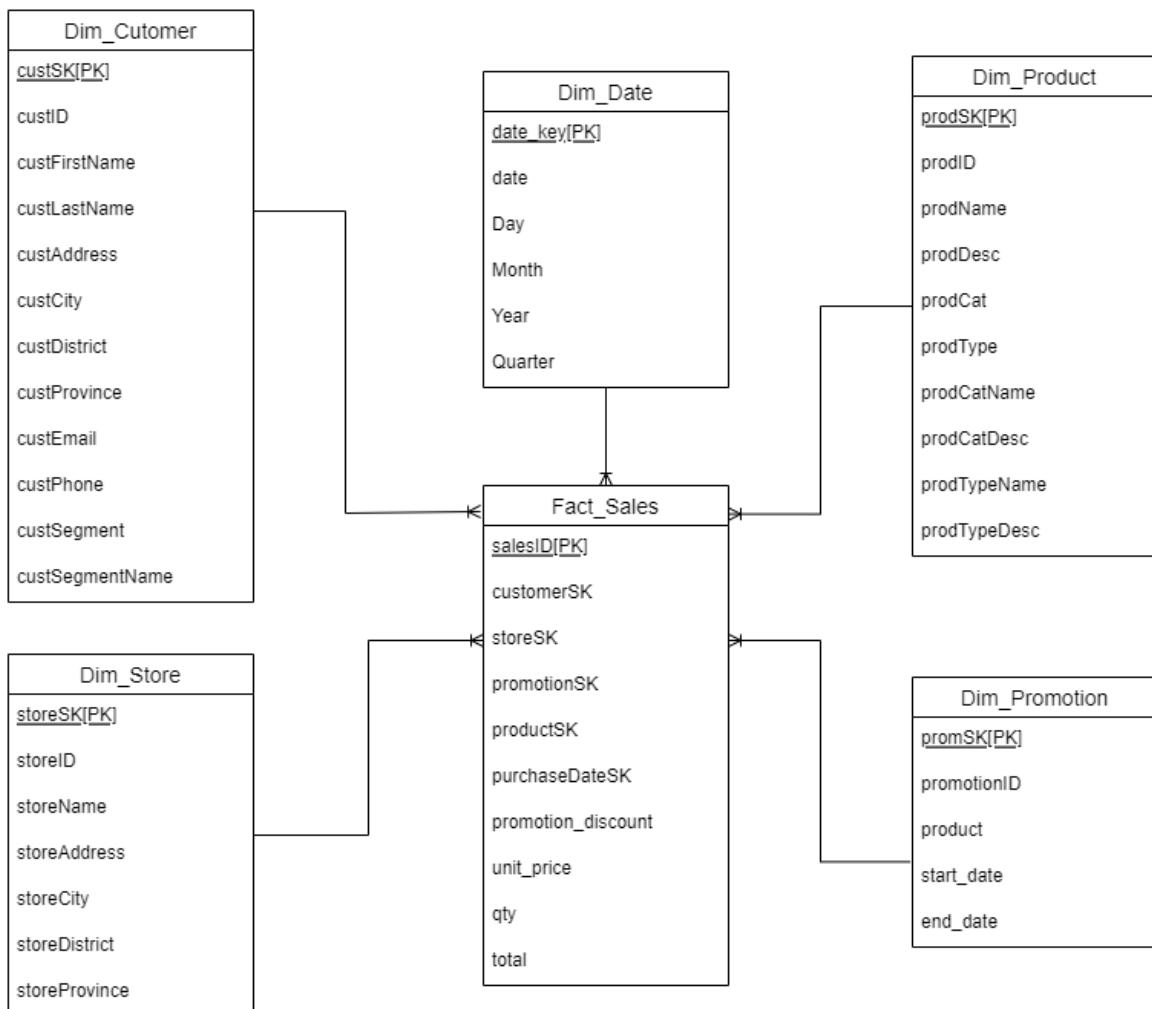
- PART 1



- PART 2

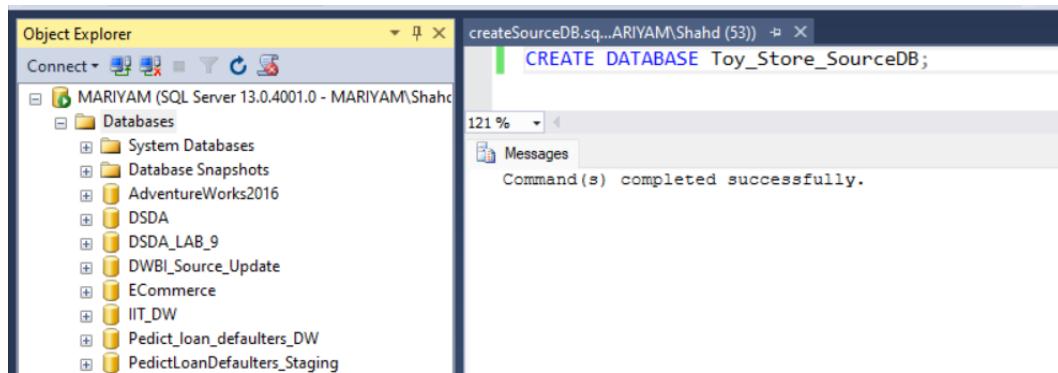
Source Table	Source Column Name	Target Table	Target Column	Dimension/Fact
Customer	custID	Dim_Customer	custID	Dimension
	custFirstName	Dim_Customer	custFirstName	Dimension
	custLastName	Dim_Customer	custLastName	Dimension
	custAddress	Dim_Customer	custAddress	Dimension
	custCity	Dim_Customer	custCity	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_Cat	prodCatID	Dim_Product	prodCat	Dimension
	prodCatName	Dim_Product	prodCatName	Dimension
	prodCatDesc	Dim_Product	prodCatDesc	Dimension
Product	prodID	Dim_Product	prodID	Dimension

- PART 3



- PART 4

- ❖ Creating database for source tables



❖ Creating the target database

The screenshot shows the Object Explorer on the left with databases like System Databases, Database Snapshots, AdventureWorks2016, DSDA, DSDA\_LAB\_9, and DWBI\_Source\_Update. The SQL Query Editor on the right has the following code:

```
CREATE DATABASE Toy_Store_TargetDW;
```

The Messages pane at the bottom shows the command completed successfully.

❖ Creating Source Tables

- Store table

The screenshot shows the Object Explorer on the left with databases like MARIYAM, System Databases, Database Snapshots, AdventureWorks2016, DSDA, DSDA\_LAB\_9, DWBI\_Source\_Update, ECommerce, IIT\_DW, Predict\_loan\_defaulters\_DW, PredictLoanDefaulters\_Staging, Predict\_loan\_defaulters\_sourceDB, PredictLoanDefaultersSourceDB, ProductWarehouse, ReportServer, ReportServerTempDB, SLIIT\_Retail\_DB, and SLIIT\_Retail\_DW. The SQL Query Editor on the right has the following code:

```
USE Toy_Store_SourceDB;

----- STORE TABLE -----
CREATE TABLE Store(
    storeID varchar(20) PRIMARY KEY,
    storeName varchar(50) NOT NULL,
    storeAddress varchar(100),
    storeCity varchar(20),
    storeDistrict varchar(20),
    storeProvince varchar(20),
);
```

The Messages pane at the bottom shows the command completed successfully.

The screenshot shows the Object Explorer on the left with the same list of databases as the previous screenshot. The SQL Query Editor on the right has the following code:

```
SELECT * FROM Store;
```

Below this, there is a series of INSERT INTO statements for the 'Store' table:

```

INSERT INTO STORE VALUES('ST001','Colombo Branch 1','475, Union Place','Colombo','Colombo','Western');
INSERT INTO STORE VALUES('ST002','Kandy Branch 1','12, D.S Senanayake Rd','Kandy','Kandy','Central');
INSERT INTO STORE VALUES('ST003','Kandy Branch 2','22/A, Peradeniya Rd','Peradeniya','Kandy','Central');
INSERT INTO STORE VALUES('ST004','Colombo Branch 2','151, Ebenezer Place','Dehiwela','Colombo','Western');
INSERT INTO STORE VALUES('ST005','Matale Branch','B22, Watuwatta','Matale','Matale','Central');
INSERT INTO STORE VALUES('ST006','Kurunegala Branch','25, Labbala','Kurunegala','Kurunegala','Western');
INSERT INTO STORE VALUES('ST007','Hambantota Branch','76, Chithragala','Ambalantota','Hambantota','South');
INSERT INTO STORE VALUES('ST008','Galle Branch','321, Talduwa','Ahangama','Galle','Southern');
INSERT INTO STORE VALUES('ST009','Gampaha Branch','101, Kiribathgoda','Gampaha','Gampaha','Western');
```

The Messages pane at the bottom shows multiple (1 row(s) affected) messages, indicating successful insertions.

- Customer Segment Table

The screenshot shows two separate sessions in SQL Server Management Studio's Object Explorer. Both sessions are connected to the same database, MARIYAM (SQL Server 13.0.4001.0 - MARIYAM).

**Session 1 (Top):**

```
----- CUSTOMER SEGMENT TABLE -----
CREATE TABLE Customer_Segment(
    custSegmentID varchar(20) PRIMARY KEY,
    custSegmentName varchar(50) NOT NULL,
);
```

Messages: Command(s) completed successfully.

**Session 2 (Bottom):**

```
CREATE TABLE Customer_Segment(
    custSegmentID varchar(20) PRIMARY KEY,
    custSegmentName varchar(50) NOT NULL,
);

INSERT INTO Customer_Segment VALUES('SEG001','Regular Customer');
INSERT INTO Customer_Segment VALUES('SEG002','Direct Customer');
INSERT INTO Customer_Segment VALUES('SEG003','Online Customer');
INSERT INTO Customer_Segment VALUES('SEG004','Inactive Customer');
INSERT INTO Customer_Segment VALUES('SEG005','Dealer Customer');
```

Messages:

- (1 row(s) affected)

### ○ Customer Table

The screenshot shows the SSMS interface. The Object Explorer on the left lists databases including MARIYAM, AdventureWorks2016, DSDA, DSDA\_LAB\_9, DWBI\_Source\_Update, ECommerce, IIT\_DW, Predict\_loan\_defaulters\_DW, PredictLoanDefaulters\_Staging, Predict\_loan\_defaulters\_sourceE, PredictLoanDefaultersSourceDB, ProductWarehouse, ReportServer, ReportServerTempDB, and SLIIT\_Retail\_DW. The query editor window on the right contains T-SQL code for creating a 'Customer' table with various columns and a foreign key constraint.

```
----- CUSTOMER TABLE -----
CREATE TABLE Customer(
    custID varchar(20) PRIMARY KEY,
    custFirstName varchar(20),
    custLastName varchar(20),
    custAddress varchar(100),
    custCity varchar(20),
    custDistrict varchar(20),
    custProvince varchar(20),
    custEmail varchar(50),
    custPhone nvarchar(20),
    custSegmentFK varchar(20),
    FOREIGN KEY (custSegmentFK) REFERENCES Customer_Segment(custSegmentID),
);
```

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. The Object Explorer on the left lists various databases and tables under the 'MARIYAM' database. The central pane displays a T-SQL script for creating a table named 'Customer'. The script includes columns for custID (int), custName (varchar(50)), custPhone (nvarchar(20)), custSegmentFK (varchar(20)), and a FOREIGN KEY constraint referencing the 'Customer\_Segment' table. Below the script, several INSERT statements are shown, each adding a new customer record with a unique ID and specific details like name, phone, and segment. The status bar at the bottom indicates 121 rows affected.

```
source tables.sql...MARIYAM\Shahd (59)* ↗ x createSourceDB.sql...ARIYAM\Shahd (53)
CREATE TABLE [Customer]
(
    custID int,
    custName varchar(50),
    custPhone nvarchar(20),
    custSegmentFK varchar(20),
    FOREIGN KEY (custSegmentFK) REFERENCES Customer_Segment(custSegmentID),
);
INSERT INTO Customer VALUES('CUST001','Malithi','Perera','132, Calvert Terrace', 'Dehiwela', 'Colombo', 'Western', 'malithi.perera@gamil.com', '0773462834', 'SEG001');
INSERT INTO Customer VALUES('CUST002','Shehani','Gunatilake','29/A, Teldeniya Rd', 'Madawala', 'Kandy', 'Central', 'gshehani	gtk@gamil.com', '0775612345', 'SEG004');
INSERT INTO Customer VALUES('CUST003','Ruwan','Perera','53, Ebenezer Place', 'Dehiwela', 'Colombo', 'Western', 'ruwan.perera@gamil.com', '0763888911', 'SEG003');
INSERT INTO Customer VALUES('CUST004','Mihirangi','Kumari','76, Kirbathgoda', 'Gampaha', 'Gampaha', 'Western', 'migirangi.kumari@gamil.com', '0729873872', 'SEG001');
INSERT INTO Customer VALUES('CUST005','Anirudh','Ravichandar','10, Peradeniya rd', 'Peradeniya', 'Kandy', 'Central', 'ani.ravi@gamil.com', '0777861448', 'SEG005');
INSERT INTO Customer VALUES('CUST006','Mahesh','Peiris','8, Page Lane', 'Wellawatte', 'Colombo', 'Western', 'mahesh.peiris@gamil.com', '0773462834', 'SEG002');
```

### ○ Order Header Table

The screenshot shows the SQL Server Management Studio interface. The Object Explorer on the left lists databases including 'MARIYAM' (selected), 'AdventureWorks2016', 'DSDA', 'DSDA\_LAB\_9', 'DWBI\_Source\_Update', 'ECommerce', 'IIT\_DW', 'Predict\_loan\_defaulters\_DW', 'PredictLoanDefaulters\_Staging', 'Predict\_loan\_defaulters\_source', 'PredictLoanDefaultersSource', 'ProductWarehouse', 'ReportServer', 'ReportServerTempDB', 'SIIIT\_Retail\_DW', 'SIIIT\_Retail\_DW\_2', and 'SIIIT\_Retail\_Staging'. The central pane displays a SQL script for creating a table named 'Order\_Header' and inserting data into it. The script includes columns for orderHeaderID, custID, storeID, and purchaseDate, along with foreign key constraints referencing 'Customer' and 'Store' tables. It also includes five INSERT statements for the Order\_Header table.

```
'Western', 'mahesh.peiris@gmail.com', '0773462834', 'SEG002');

----- ORDER HEADER TABLE -----

CREATE TABLE Order_Header(
    orderHeaderID varchar(20) PRIMARY KEY,
    custID varchar(20),
    storeID varchar(20),
    purchaseDate date,
    FOREIGN KEY (custID) REFERENCES Customer(custID),
    FOREIGN KEY (storeID) REFERENCES Store(StoreID)
);

INSERT INTO Order_Header VALUES('ORH001', 'CUST001', 'ST003', '2023-01-09');
INSERT INTO Order_Header VALUES('ORH002', 'CUST002', 'ST001', '2023-11-01');
INSERT INTO Order_Header VALUES('ORH003', 'CUST006', 'ST004', '2021-12-12');
INSERT INTO Order_Header VALUES('ORH004', 'CUST005', 'ST002', '2022-12-29');
INSERT INTO Order_Header VALUES('ORH005', 'CUST003', 'ST005', '2023-05-11');
```

## ○ Product Type Table

The screenshot shows the Object Explorer on the left and a query window on the right. The query window contains the following SQL code:

```

source tables.sql...MARIYAM\Shahd (59) ✎ × createSourceDB.sql...ARIYAM\Shahd (53)

INSERT INTO Order_Header VALUES('ORH004', 'CUST005', 'ST002');
INSERT INTO Order_Header VALUES('ORH005', 'CUST003', 'ST005');

----- PRODUCT TYPE -----
CREATE TABLE Product_Type(
    prodTypeID varchar(20) PRIMARY KEY,
    prodTypeName varchar(20),
    prodTypeDesc varchar(100)
);

INSERT INTO Product_Type VALUES('PRTY001', 'General', 'Products developed based on general orders');
INSERT INTO Product_Type VALUES('PRTY002', 'Import', 'Products imported');
INSERT INTO Product_Type VALUES('PRTY003', 'Export', 'Products exported');

```

The 'Messages' pane at the bottom shows three rows affected by the insert statements.

## ○ Product Category Table

The screenshot shows the Object Explorer on the left and a query window on the right. The query window contains the following SQL code:

```

source tables.sql...MARIYAM\Shahd (59)* ✎ × createSourceDB.sql...ARIYAM\Shahd (53)

----- PRODUCT CATEGORY -----
CREATE TABLE Product_Category(
    prodCatID varchar(20) PRIMARY KEY,
    prodCatName varchar(20),
    prodCatDesc varchar(100),
    prodType varchar(20),
    FOREIGN KEY (prodType) REFERENCES Product_Type(prodTypeID)
);

INSERT INTO Product_Category VALUES('PRCAT001', 'Doll', 'Barbie Doll', 'PRTY002');
INSERT INTO Product_Category VALUES('PRCAT002', 'Kitchen ware', 'Kitchen Cooker and Plate Set', 'PRTY001');
INSERT INTO Product_Category VALUES('PRCAT003', 'Animal', 'Toy animal set', 'PRTY001');

```

The 'Messages' pane at the bottom shows three rows affected by the insert statements.

## ○ Product Table

The screenshot shows the Object Explorer on the left and a query window on the right. The query window contains the following SQL code:

```

source tables.sql...MARIYAM\Shahd (59)* ✎ × createSourceDB.sql...ARIYAM\Shahd (53)

CREATE TABLE Product(
    prodID varchar(20) PRIMARY KEY,
    prodName varchar(20) NOT NULL,
    prodDesc varchar(200),
    unitPrice money NOT NULL,
    qty int,
    prodCat varchar(20),
    FOREIGN KEY (prodCat) REFERENCES Product_Category(ProdCatID)
);

INSERT INTO Product VALUES('PROD001', 'Doll', 'Pink Barbie Doll', 500, 300, 'PRCAT001');
INSERT INTO Product VALUES('PROD002', 'Toy Car', 'Merc Red', 200, 250, 'PRCAT003');
INSERT INTO Product VALUES('PROD003', 'Minie', 'Toy Minnie Mouse', 350, 100, 'PRCAT003');

```

The 'Messages' pane at the bottom shows four rows affected by the insert statements.

- Promotion Table

The screenshot shows the Object Explorer on the left with the database 'MARIYAM' selected. The right pane displays the T-SQL code for creating the 'Promotion' table and inserting four rows of data. The execution results show four rows affected.

```

----- PROMOTION TABLE -----
CREATE TABLE Promotion(
    promotionID varchar(20) PRIMARY KEY,
    productID varchar(20),
    discount numeric(20,2),
    start_date date,
    end_date date,
    FOREIGN KEY (productID) REFERENCES Product(prodID));

INSERT INTO Promotion VALUES('DIS001','PROD006',0.20,'2023-11-06','2023-12-31');
INSERT INTO Promotion VALUES('DIS002','PROD001',0.10,'2023-10-23','2023-11-01');
INSERT INTO Promotion VALUES('DIS003','PROD004',0.15,'2023-11-06','2023-12-31');
INSERT INTO Promotion VALUES('DIS004','PROD005',0.50,'2023-10-10','2023-11-30');

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

```

- Order Details Table

The screenshot shows the Object Explorer on the left with the database 'MARIYAM' selected. The right pane displays the T-SQL code for creating the 'Order\_Detail' table and inserting five rows of data. The execution results show five rows affected.

```

----- ORDER_DETAIL -----
CREATE TABLE Order_Detail(
    orderID varchar(20) PRIMARY KEY,
    product varchar(20),
    orderHeader varchar(20),
    qty int,
    FOREIGN KEY (product) REFERENCES Product(prodID),
    FOREIGN KEY (orderHeader) REFERENCES Order_Header(orderHeaderID)
);

INSERT INTO Order_Detail VALUES('ORD001','PROD001','ORH002',2);
INSERT INTO Order_Detail VALUES('ORD002','PROD005','ORH003',1);
INSERT INTO Order_Detail VALUES('ORD003','PROD002','ORH004',5);
INSERT INTO Order_Detail VALUES('ORD004','PROD006','ORH001',10);

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

```

- ❖ Datawarehouse Dimension and Fact Creation

- Dim Customer

The screenshot shows the Object Explorer on the left with the database 'SLIIT\_RetailSourceDB' selected. The right pane displays the T-SQL code for creating the 'Dim\_Customer' dimension table. The execution results show the command completed successfully.

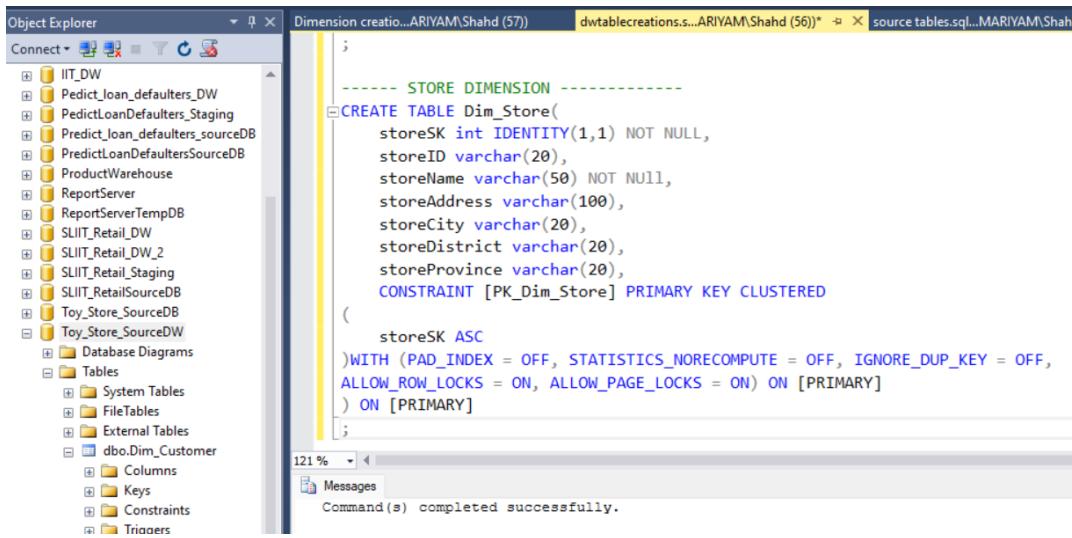
```

----- CUST DIM -----
CREATE TABLE Dim_Customer(
    custSK int IDENTITY(1,1) NOT NULL,
    custID varchar(20),
    custFirstName varchar(20),
    custLastName varchar(20),
    custAddress varchar(100),
    custCity varchar(20),
    custDistrict varchar(20),
    custProvince varchar(20),
    custEmail varchar(50),
    custPhone nvarchar(20),
    custSegment varchar(20),
    custSegmentName varchar(50),
    CONSTRAINT [PK_Dim_Customer] PRIMARY KEY CLUSTERED
    (
        custSK ASC
    )
    WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
);

Command(s) completed successfully.

```

- Store Dimension



The screenshot shows the Object Explorer on the left with various databases listed. The 'dwtablecreations.s...ARIYAM\Shahd (57)\*' tab is active in the center, displaying the following SQL code:

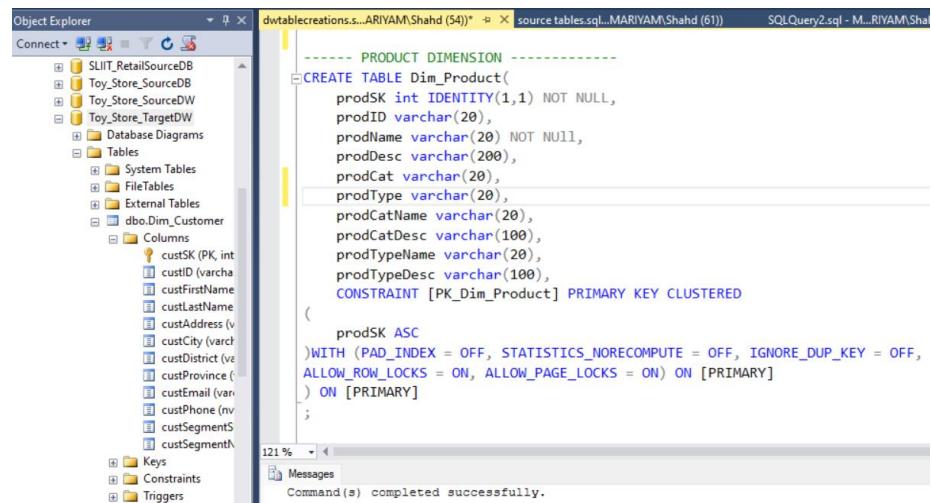
```

;----- STORE DIMENSION -----
CREATE TABLE Dim_Store(
    storeSK int IDENTITY(1,1) NOT NULL,
    storeID varchar(20),
    storeName varchar(50) NOT NULL,
    storeAddress varchar(100),
    storeCity varchar(20),
    storeDistrict varchar(20),
    storeProvince varchar(20),
    CONSTRAINT [PK_Dim_Store] PRIMARY KEY CLUSTERED
)
(
    storeSK ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
;

```

The 'Messages' pane at the bottom right shows the message: "Command(s) completed successfully."

- Product Dimension



The screenshot shows the Object Explorer on the left with various databases listed. The 'dwtablecreations.s...ARIYAM\Shahd (54)\*' tab is active in the center, displaying the following SQL code:

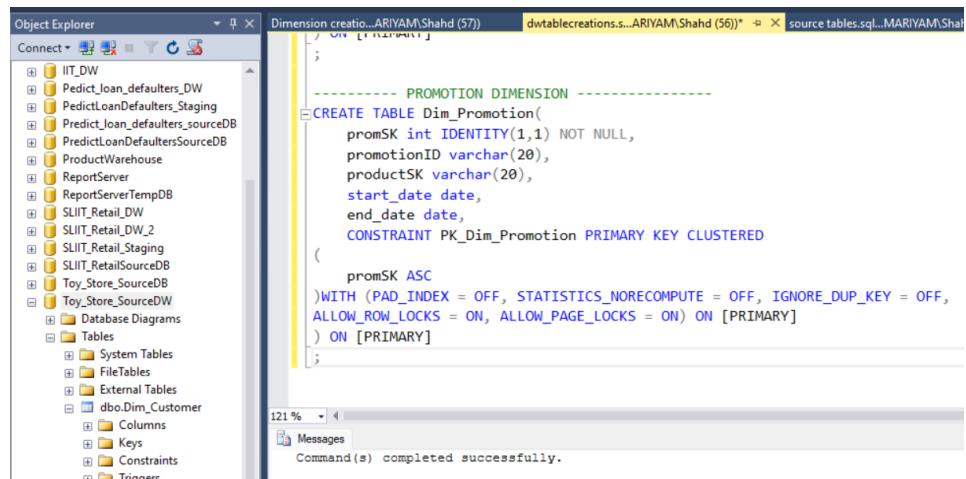
```

;----- PRODUCT DIMENSION -----
CREATE TABLE Dim_Product(
    prodSK int IDENTITY(1,1) NOT NULL,
    prodID varchar(20),
    prodName varchar(20) NOT NULL,
    prodDesc varchar(200),
    prodCat varchar(20),
    prodType varchar(20),
    prodCatName varchar(20),
    prodCatDesc varchar(100),
    prodTypeName varchar(20),
    prodTypeDesc varchar(100),
    CONSTRAINT [PK_Dim_Product] PRIMARY KEY CLUSTERED
)
(
    prodSK ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
;

```

The 'Messages' pane at the bottom right shows the message: "Command(s) completed successfully."

- Promotion Dimension



The screenshot shows the Object Explorer on the left with various databases listed. The 'dwtablecreations.s...ARIYAM\Shahd (56)\*' tab is active in the center, displaying the following SQL code:

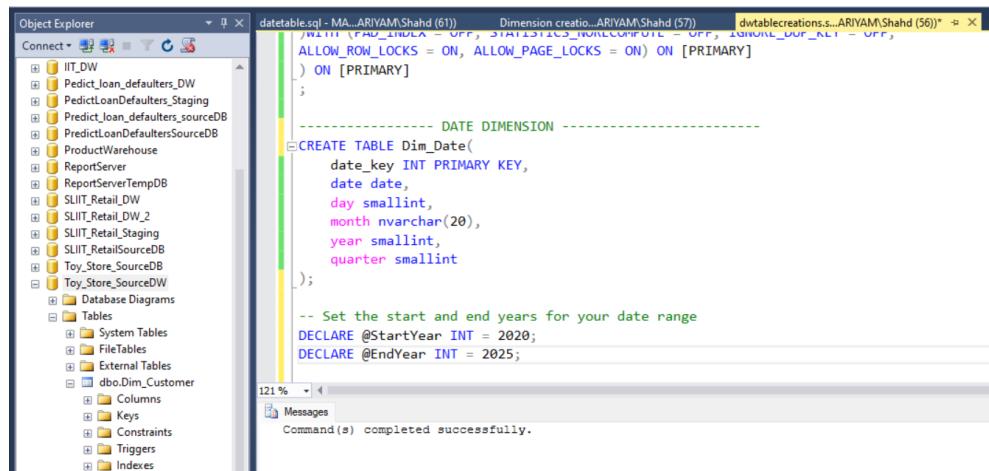
```

;----- PROMOTION DIMENSION -----
CREATE TABLE Dim_Promotion(
    promSK int IDENTITY(1,1) NOT NULL,
    promotionID varchar(20),
    productSK varchar(20),
    start_date date,
    end_date date,
    CONSTRAINT PK_Dim_Promotion PRIMARY KEY CLUSTERED
)
(
    promSK ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
;

```

The 'Messages' pane at the bottom right shows the message: "Command(s) completed successfully."

## o Date Dimension



The screenshot shows the Object Explorer on the left with various databases listed. The central pane displays a SQL script for creating a Date Dimension table. The code includes a primary key constraint, column definitions for date\_key, date, day, month, year, and quarter, and a comment indicating the start and end years for the date range. The command is successfully completed.

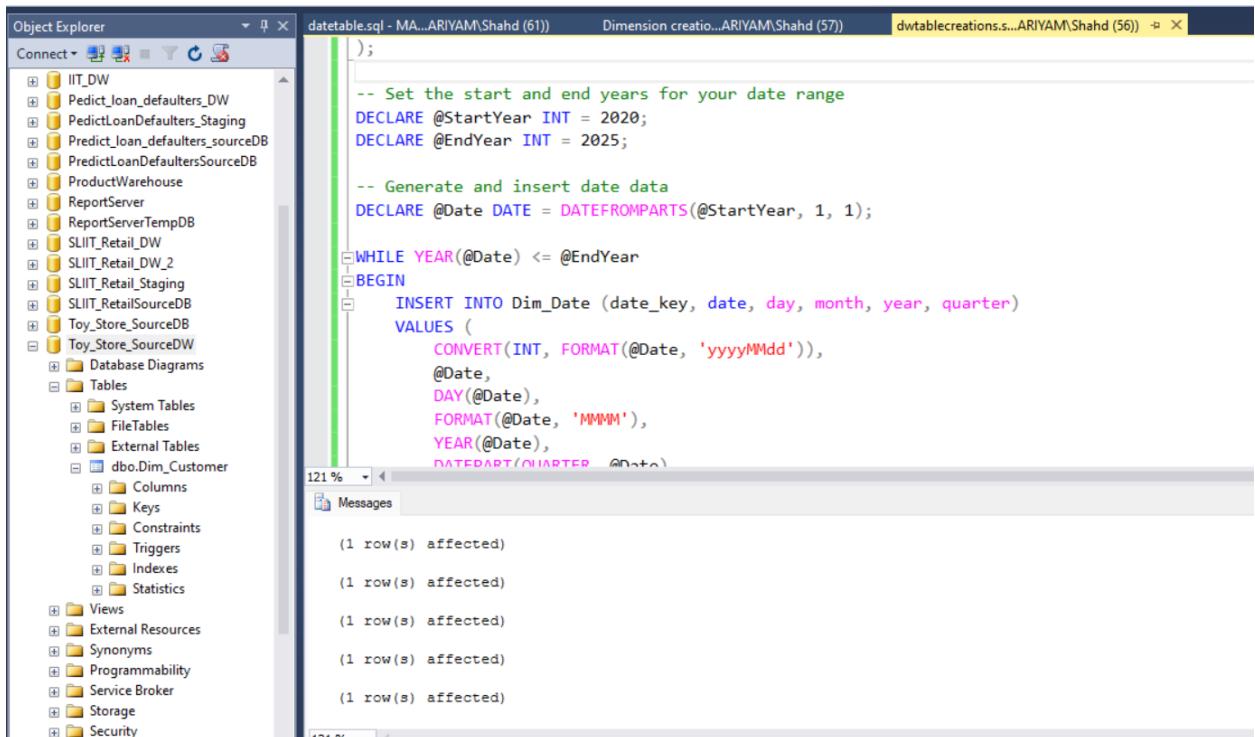
```

CREATE TABLE Dim_Date(
    date_key INT PRIMARY KEY,
    date date,
    day smallint,
    month nvarchar(20),
    year smallint,
    quarter smallint
);

-- Set the start and end years for your date range
DECLARE @StartYear INT = 2020;
DECLARE @EndYear INT = 2025;

```

## Loading Data to Date Dimension



The screenshot shows the Object Explorer on the left. The central pane displays a SQL script for loading data into the Dim\_Date table. It uses a WHILE loop to generate dates from 2020 to 2025 and inserts them into the table. The script includes a conversion of the date to a specific format and the use of DATEPART to determine the quarter. The execution messages at the bottom show that 121 rows were affected.

```

-- Generate and insert date data
DECLARE @Date DATE = DATEFROMPARTS(@StartYear, 1, 1);

WHILE YEAR(@Date) <= @EndYear
BEGIN
    INSERT INTO Dim_Date (date_key, date, day, month, year, quarter)
    VALUES (
        CONVERT(INT, FORMAT(@Date, 'yyyyMMdd')),
        @Date,
        DAY(@Date),
        FORMAT(@Date, 'MMMM'),
        YEAR(@Date),
        DATEPART(QUARTER, @Date)
    );
END

```

- o Fact Sales

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. On the left, the Object Explorer pane displays a tree view of databases and objects, including IIT\_DW, Predict\_Loan\_defaulters\_DW, PredictLoanDefaulters\_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, and dbo.Dim\_Customer. Under dbo.Dim\_Customer, there are columns, keys, constraints, and triggers. In the center, the main query editor window contains the following SQL code:

```
-- 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]
;
```

The status bar at the bottom of the query editor shows "121 %". Below the query editor, the Messages pane displays the message "Command(s) completed successfully."

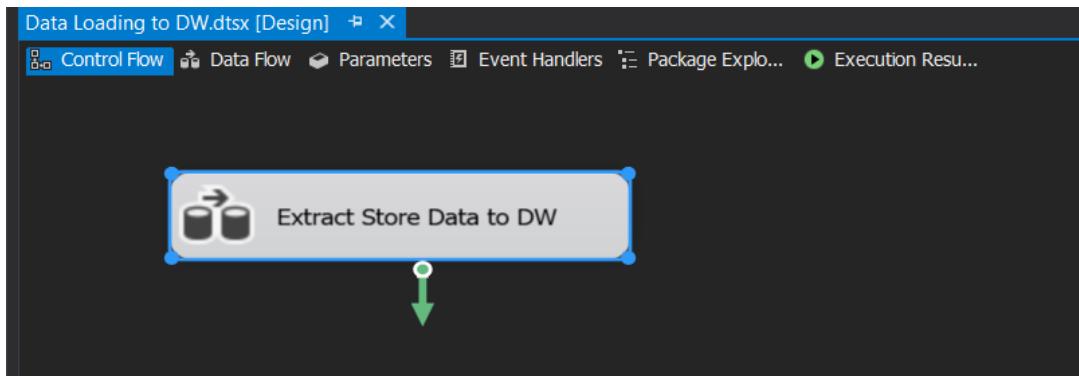
## SECTION C

- **Part 1**

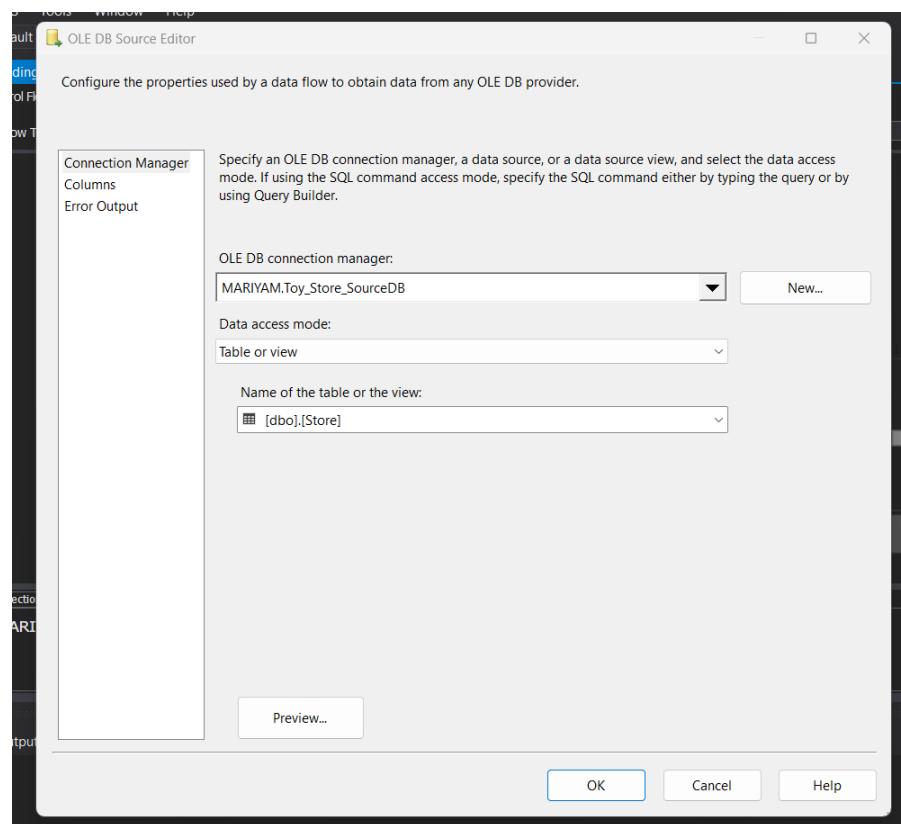
A new integration project is created for extracting and loading data to data warehouse.

- ❖ **Store dimension**

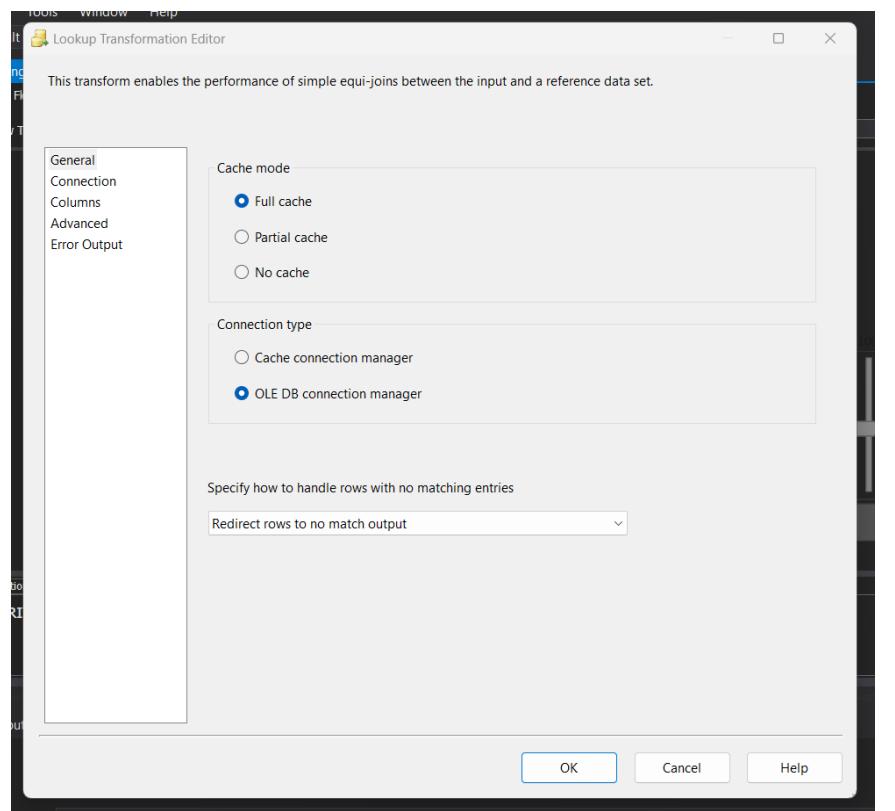
- Creating a ‘Data Flow Task’ to extract data from Store table to Store Dimension in the Control Flow area



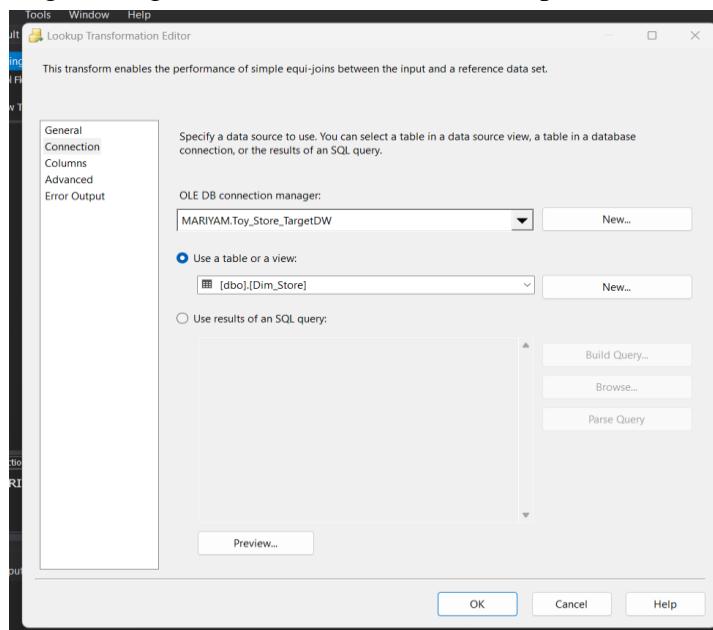
- Adding Source Assistant/ OLE DB Source and connecting it to the Store Table in the source database



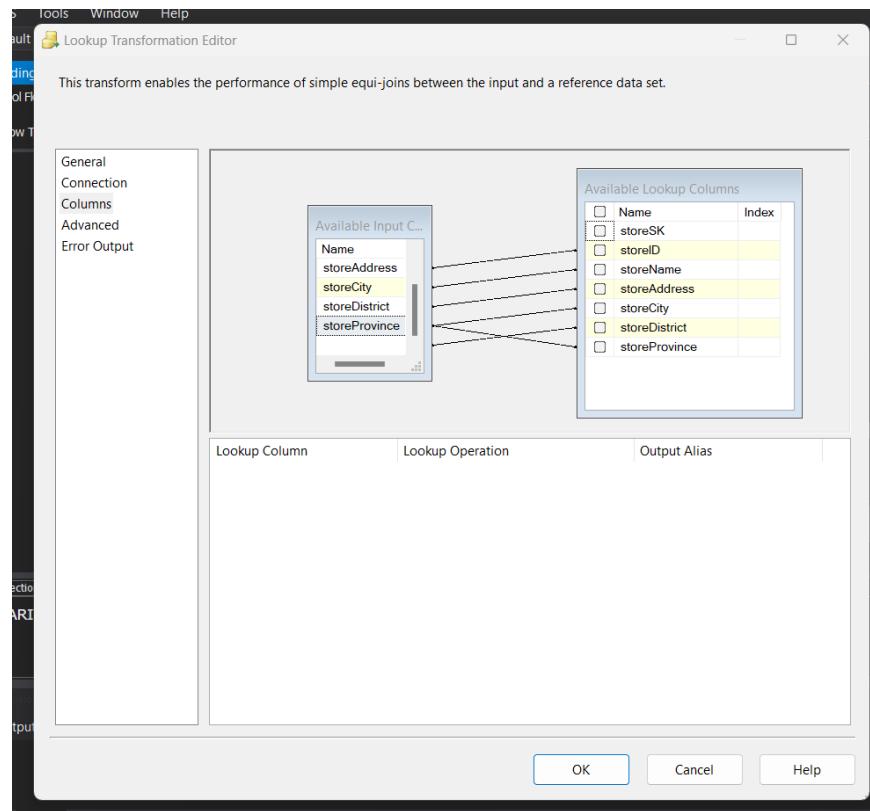
- Adding a Lookup transformation:
  - To handle rows we select ‘Redirect rows to no match output’ so that it fetches only the new records



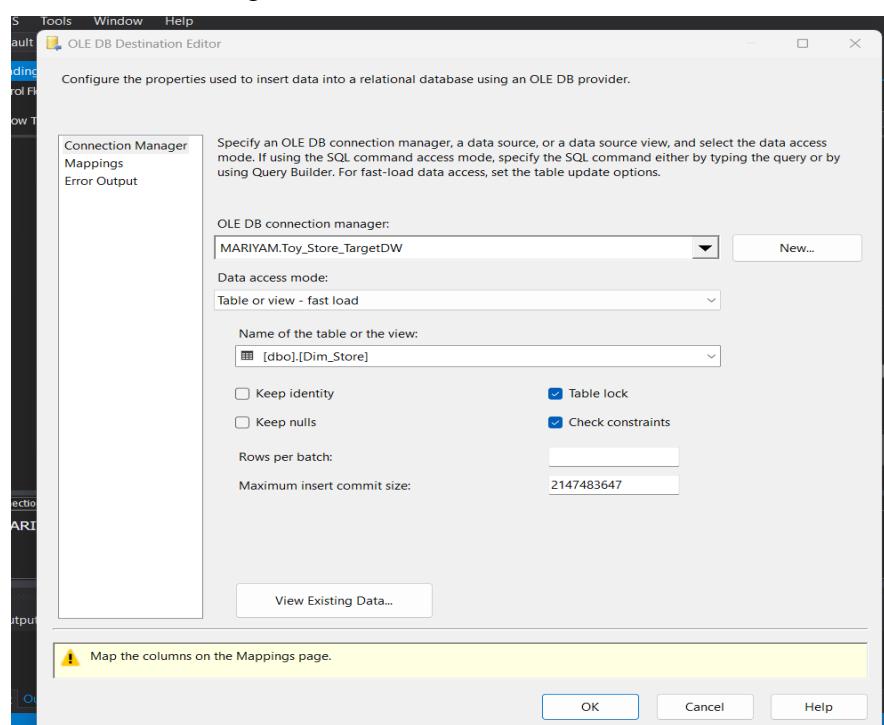
- Selecting the target data warehouse and the respective dimension table



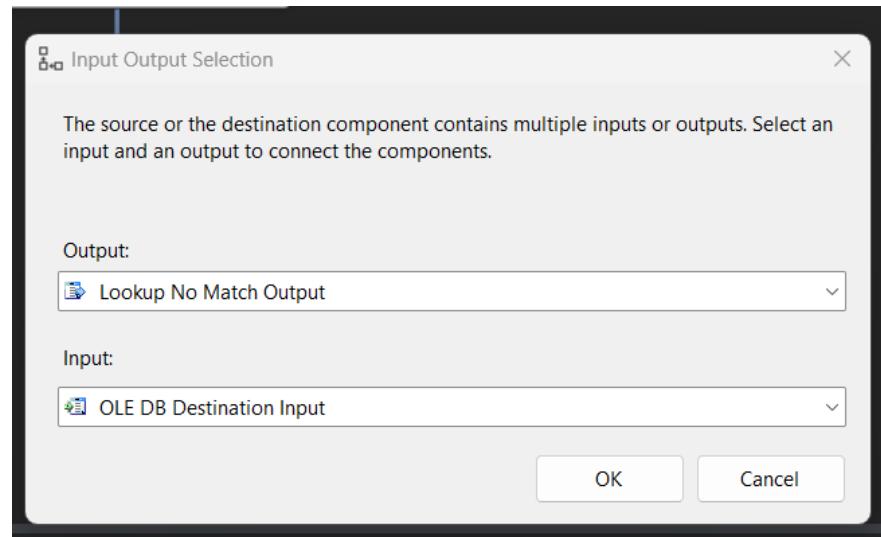
- Mapping the columns between source and target dimension table



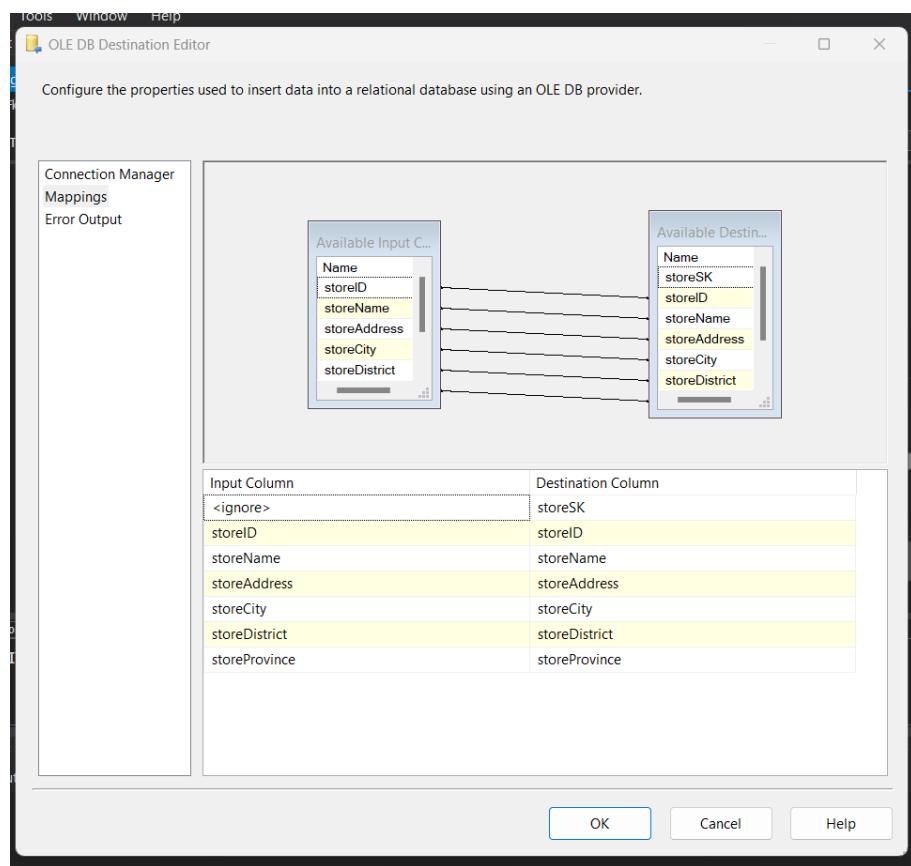
- Adding Destination Assistant/ OLE DB Destination and connecting it to the Store Dimension in the Target Data warehouse



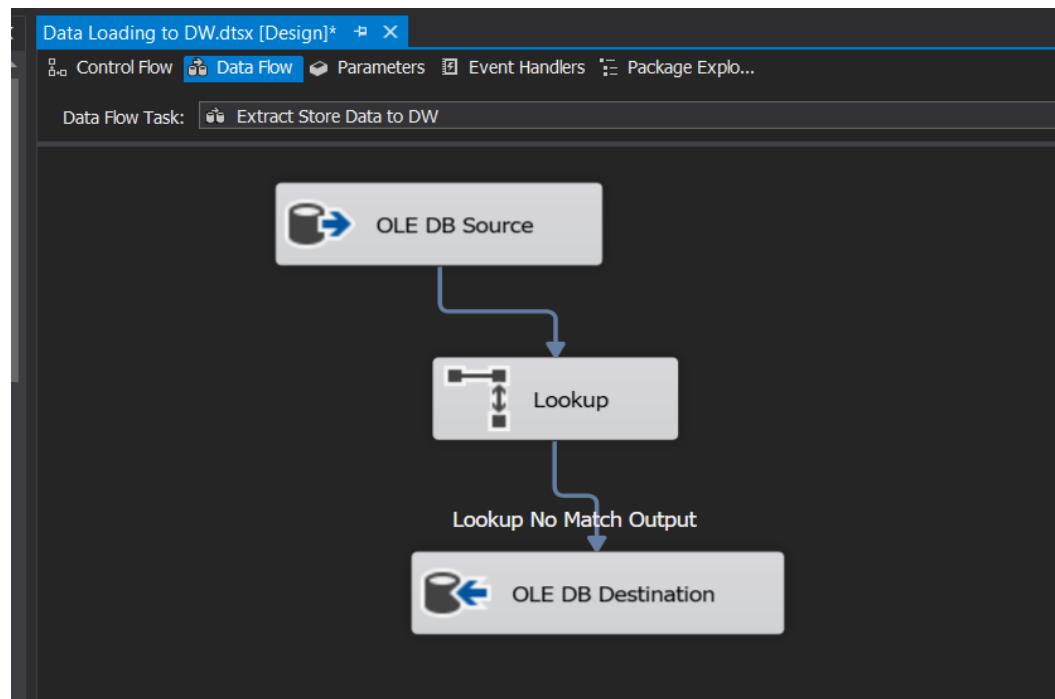
- Now connect the blue arrow from lookup with the destination assistant. Select the configurations as represented below



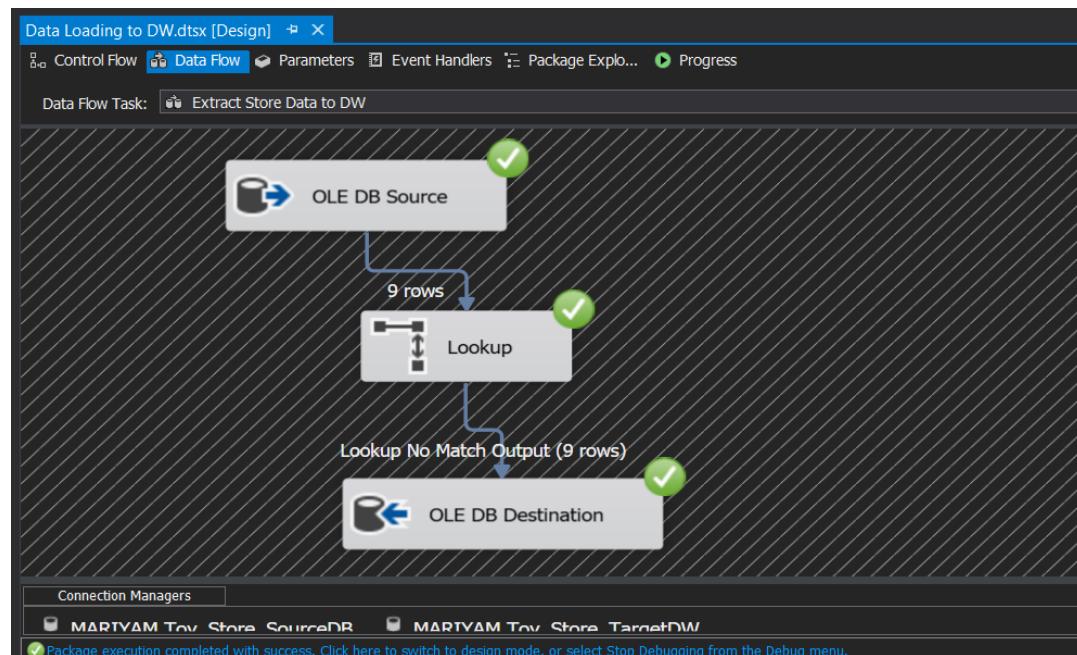
- Double click on the destination assistant and verify if the mappings are correct.



- Completed Data Flow Task is represented below



- Executing the package



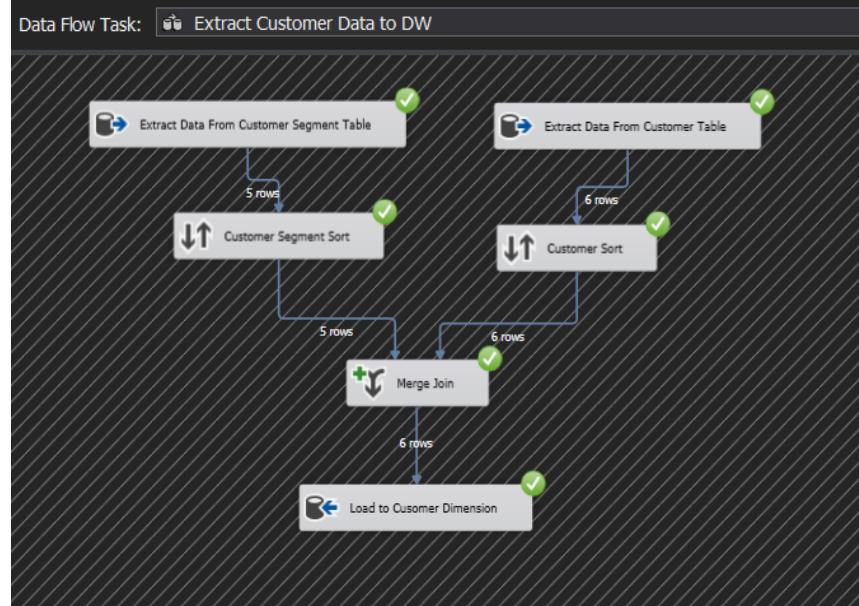
- Verifying it in MS SQL Server Management Studio

The screenshot shows the Object Explorer on the left with a connection to 'MARIYAM (SQL Server 13.0.4001.0 - MARIYAM)'. The 'Databases' node is expanded, showing various databases including 'AdventureWorks2016', 'DSDA', 'DSDA\_LAB\_9', 'DWBI\_Source\_Update', 'ECommerce', 'IIT\_DW', 'Predict\_loan\_defaulters\_DW', 'PredictLoanDefaulters\_Staging', 'Predict\_loan\_defaulters\_sourceDB', 'PredictLoanDefaultersSourceDB', 'ProductWarehouse', 'ReportServer', 'ReportServerTempDB', 'SLIIT\_Retail\_DW', 'SLIIT\_Retail\_DW\_2', 'SLIIT\_Retail\_Staging', 'SLIIT\_RetailSourceDB', 'TestDB', 'Toy\_Store\_SourceDB', 'Toy\_Store\_SourceDW', and 'Toy\_Store\_TargetDW'. The 'Security', 'Server Objects', 'Replication', and 'PolyBase' nodes are also visible.

The central pane displays the results of the query 'SELECT \* FROM Dim\_Store;'. The results are shown in a table with the following columns: storeSK, storeID, storeName, storeAddress, storeCity, storeDistrict, and storeProvince. The data consists of 9 rows:

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



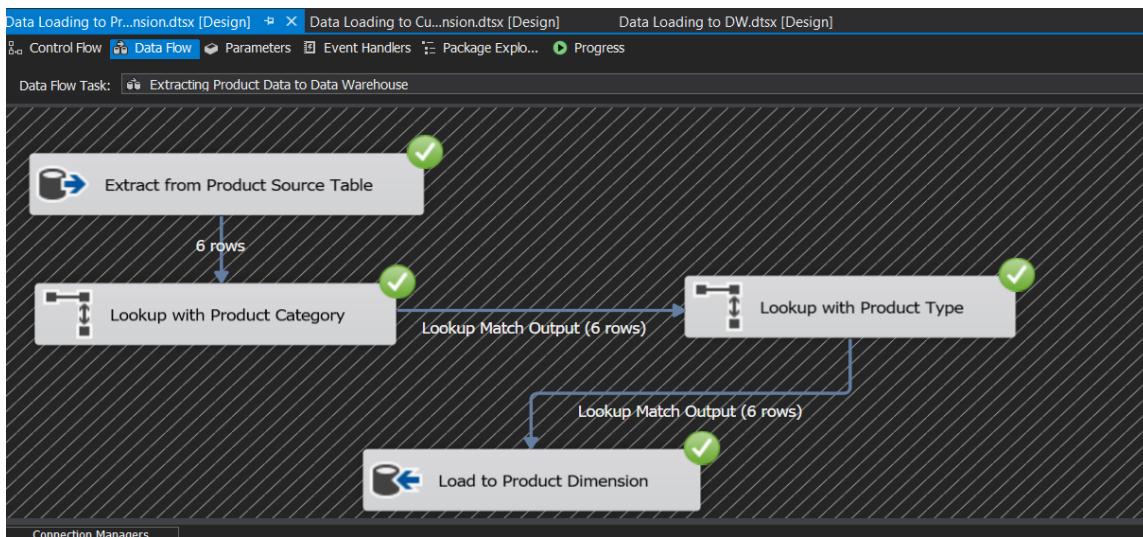
Execute Debug SQLQuery3.sql - M...RIYAM\Shahd (56) SQLQuery2.sql - M...RIYAM\Shahd (58) SQLQuery1.sql - M...RIYAM\Shahd (55)\*

SELECT \* FROM Dim\_Customer;

Results Messages

	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName
1	32	CUST001	Malithi	Perera	132, Calvett Terrace	Dehiwela	Colombo	Western	malithi.perera@gamil.com	0773462834	SEG001	Regular Customer
2	33	CUST004	Mirangi	Kumari	76, Kiribathgoda	Gampaha	Gampaha	Western	mirangi.kumar@gamil.com	0729873872	SEG001	Regular Customer
3	34	CUST006	Mahe	Peiris	8, Page Lane	Wellawatte	Colombo	Western	maheh.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



Object Explorer

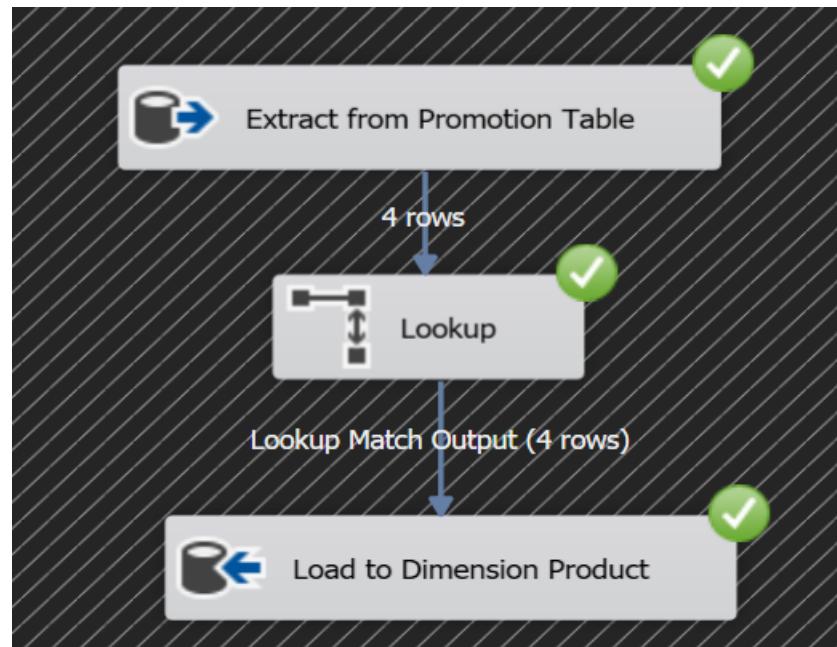
SQLQuery5.sql - M...RIYAM\Shahd (54)\*

```
***** Script for SelectTopNRows command from SSMS *****/
SELECT TOP 1000 [prodsk]
,[prodID]
,[prodName]
,[prodDesc]
,[prodCat]
,[prodType]
,[prodCatName]
,[prodCatDesc]
,[prodTypeName]
,[prodTypeDesc]
FROM [Toy_Store_TargetDW].[dbo].[Dim_Product]
```

Results Messages

	prodSK	prodID	prodName	prodDesc	prodCat	prodType	prodCatName	prodCatDesc	prodTypeName	prodTypeDesc
1	10	PR0D001	Doll	Pink Barbie Doll	PRCAT001	PRTY002	Doll	Barbie Doll	Import	Products imported
2	11	PR0D002	Toy Car	Merc Red	PRCAT003	PRTY001	Animal	Toy animal set	General	Products developed based on general orders
3	12	PR0D003	Minnie	Toy Minnie Mouse	PRCAT003	PRTY001	Animal	Toy animal set	General	Products developed based on general orders
4	13	PR0D004	Cookware	Kitchen cookware set	PRCAT002	PRTY001	Kitchen ware	Kitchen Cooker and Plate Set	General	Products developed based on general orders
5	14	PR0D005	KitchenUp	Kitchen Setup Toy	PRCAT002	PRTY001	Kitchen ware	Kitchen Cooker and Plate Set	General	Products developed based on general orders
6	15	PR0D006	Bisque Doll	Blue Barbie Doll	PRCAT001	PRTY002	Doll	Barbie Doll	Import	Products imported

- Promotion Dimension Loading

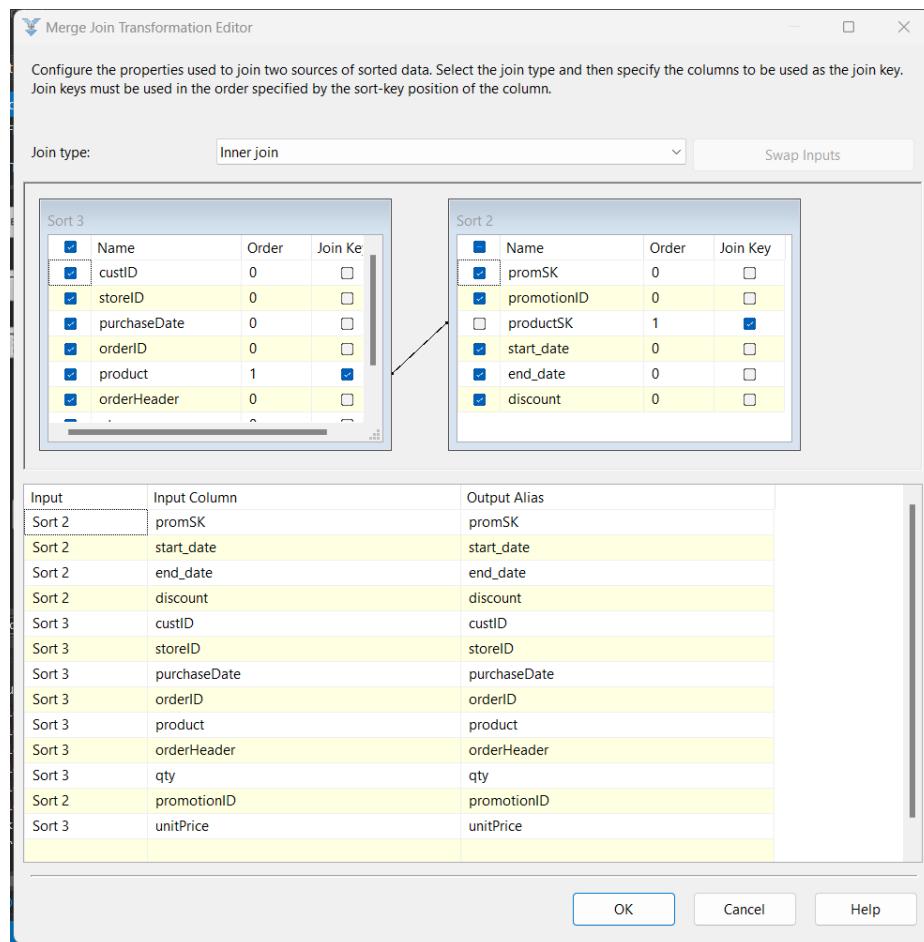


SELECT TOP 1000 [promSK]  
, [promotionID]  
, [productSK]  
, [start\_date]  
, [end\_date]  
FROM [Toy\_Store\_TargetDW].[dbo].[Dim\_Promotion]

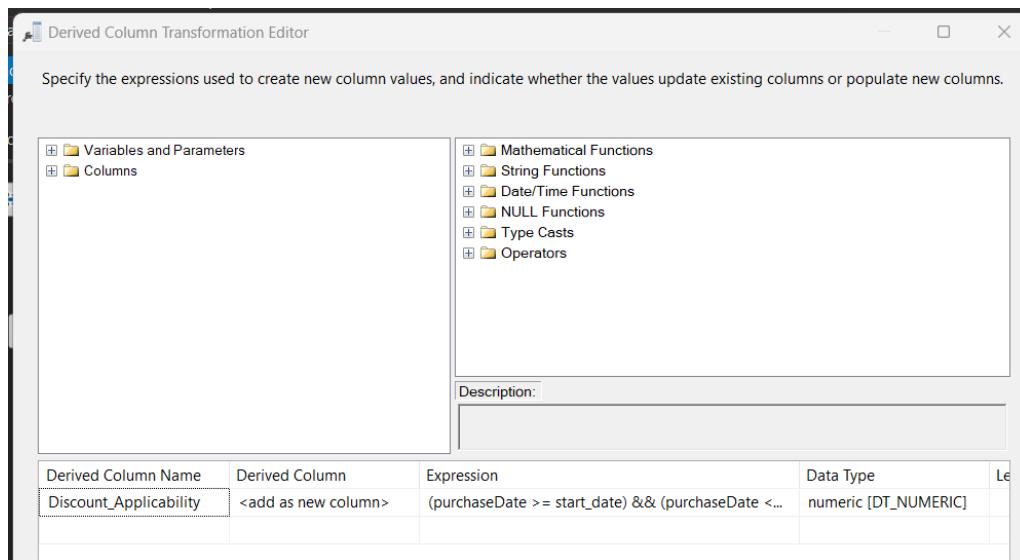
	promSK	promotionID	productSK	start_date	end_date
1	9	DIS001	PROD006	2023-11-06	2023-12-31
2	10	DIS002	PROD001	2023-10-23	2023-11-01
3	11	DIS003	PROD004	2023-11-06	2023-12-31
4	12	DIS004	PROD005	2023-10-10	2023-11-30

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

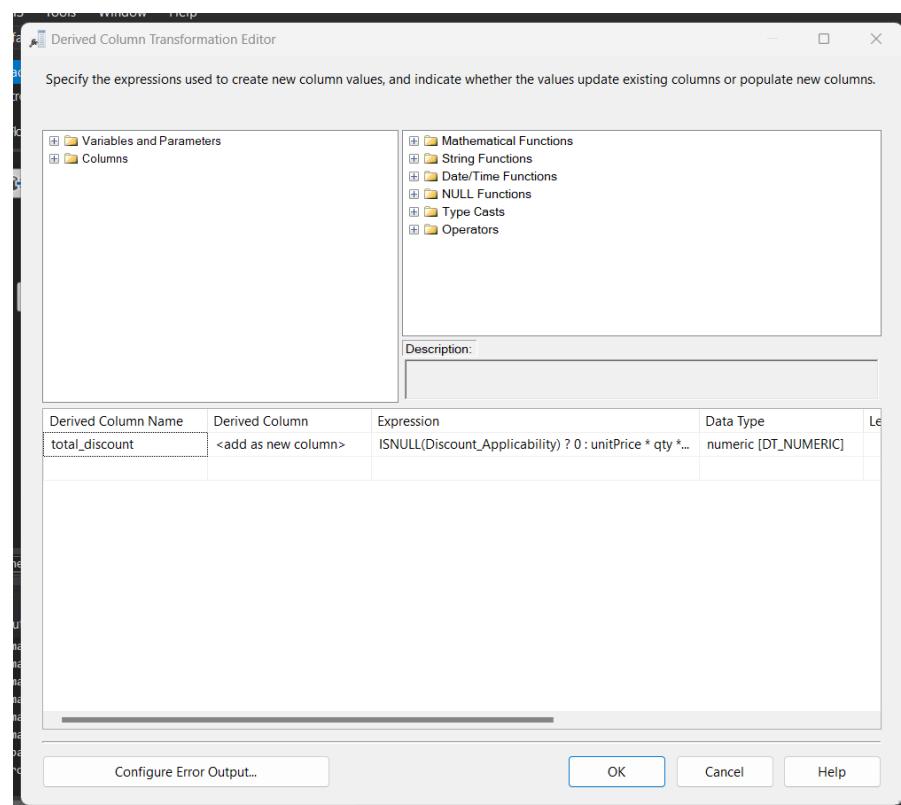
1. Initially a lookup is performed between ‘Order\_Detail’ table and ‘Dim\_Product’ to filter based on ‘productID’
2. Another lookup is performed to derive the ‘unitPrice’ of the product from product table in source.
3. Lookup is performed between ‘Order\_Header’, ‘Dim\_Customer,’ and ‘Dim\_Store’ to 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



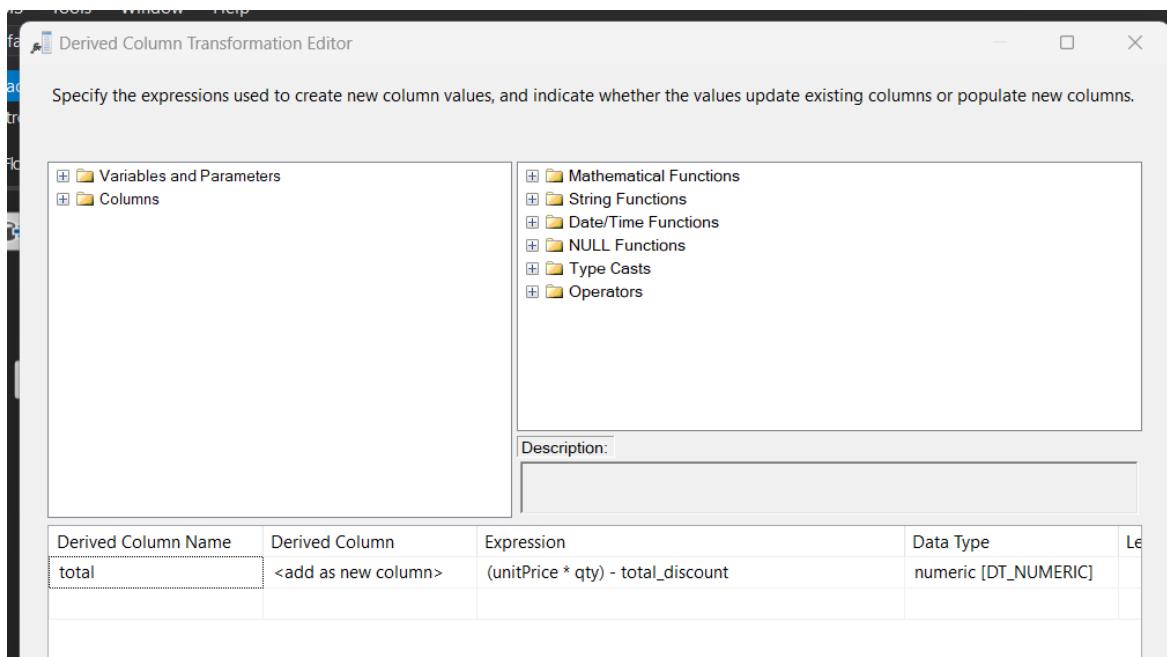
8. Check the applicability of discount for the purchase as the discounts are seasonal. The expression checks if the '*purchaseDate*' is greater than or equal to the '*StartDate*' and less than or equal to the '*EndDate*'. If the condition is true, it assigns 1; otherwise, it assigns 0. The expression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

$$(\text{purchaseDate} \geq \text{start\_date}) \&\& (\text{purchaseDate} \leq \text{end\_date}) ? \text{discount} : 0$$


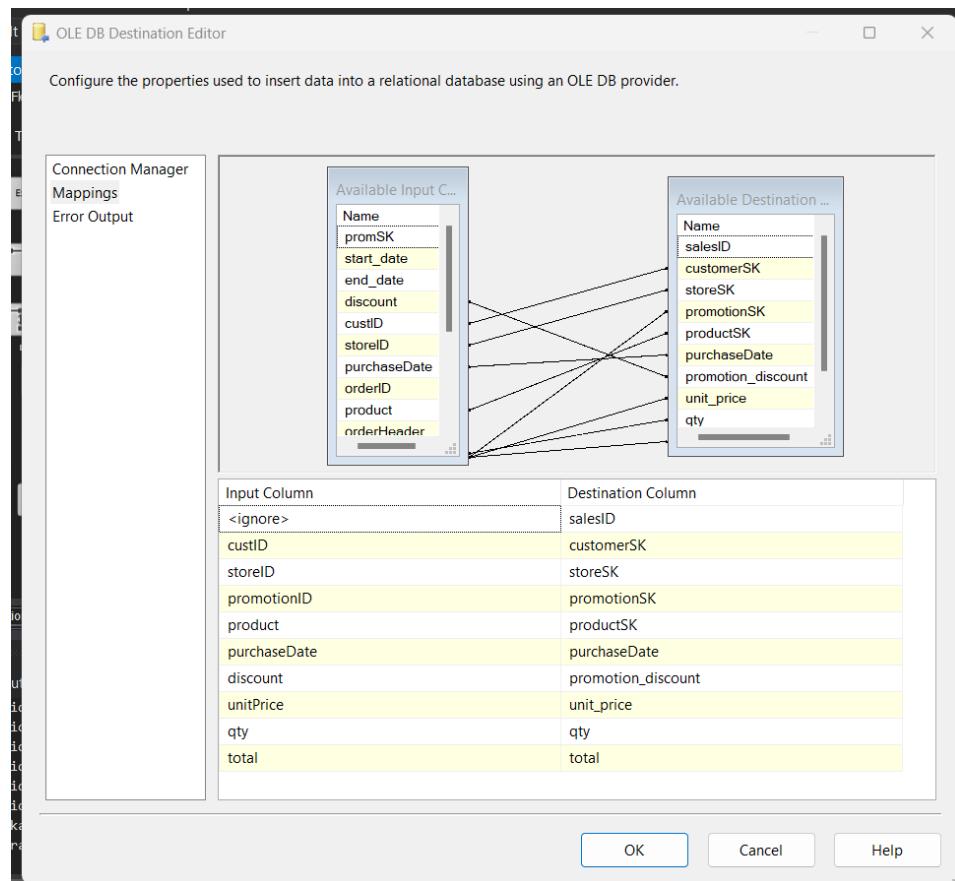
9. Total discount amount is calculated based on the discount applicability:

$$\text{ISNULL}(\text{Discount_Applicability}) ? 0 : \text{unitPrice} * \text{qty} * \text{Discount_Applicability}$$


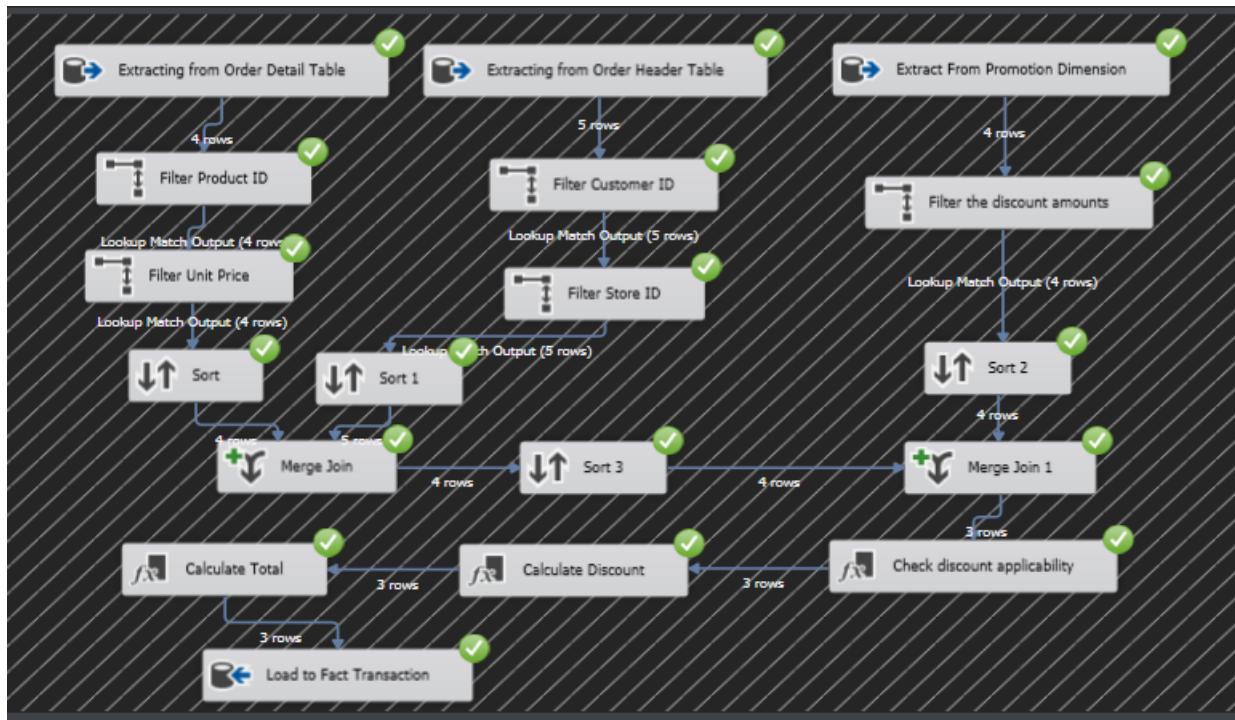
10. Finally, the total bill amount is calculated with discount:  
 $(unitPrice * qty) - total\_discount$



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

SQLQuery19.sql -...ARIYAM\Shahd (63))      SQLQuery18.sql -...ARIYAM\Shahd (57))      SQLQuery17.sql -...ARIYAM\Shahd (56))

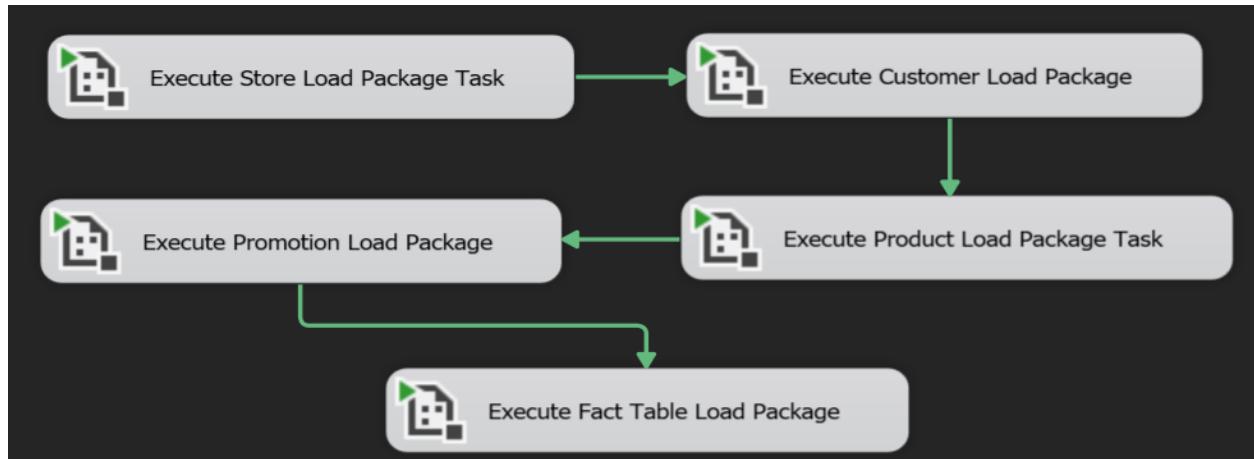
```

***** 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]
  
```

Results

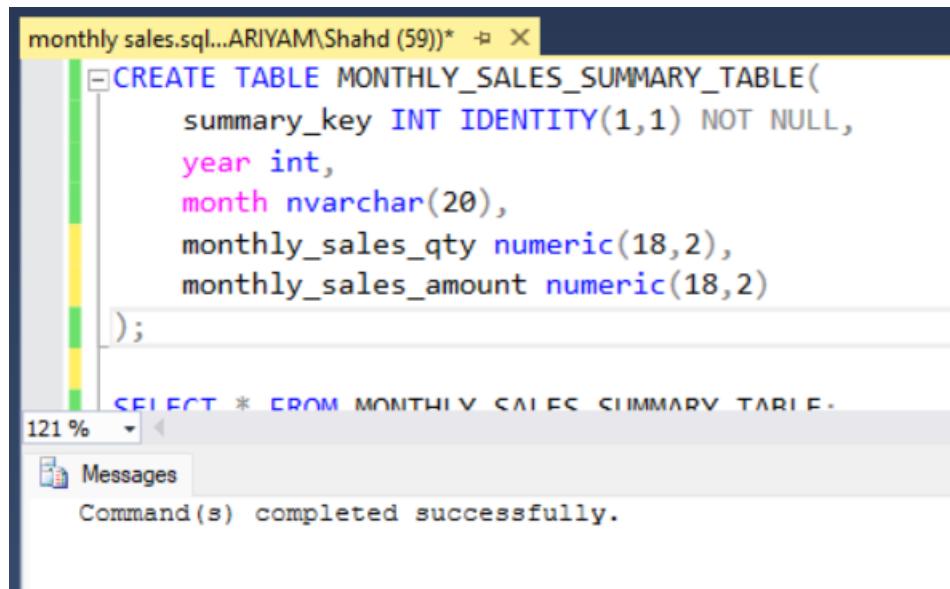
	salesID	customerSK	storeSK	promotionSK	productSK	purchaseDate	promotion_discount	unit_price	qty	total
1	1	CUST002	ST001	DIS002	PROD001	2023-11-01	0.10	500.00	2	900.00
2	2	CUST006	ST004	DIS004	PROD005	2021-12-12	0.50	800.00	1	800.00
3	3	CUST001	ST003	DIS001	PROD006	2023-01-09	0.20	450.00	10	4500.00

- ❖ 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



The screenshot shows a SQL query window in SSMS. The title bar says "monthly sales.sql...ARIYAM\Shahd (59)\*". The main pane contains the following SQL code:

```
CREATE TABLE MONTHLY_SALES_SUMMARY_TABLE(
    summary_key INT IDENTITY(1,1) NOT NULL,
    year int,
    month nvarchar(20),
    monthly_sales_qty numeric(18,2),
    monthly_sales_amount numeric(18,2)
);
```

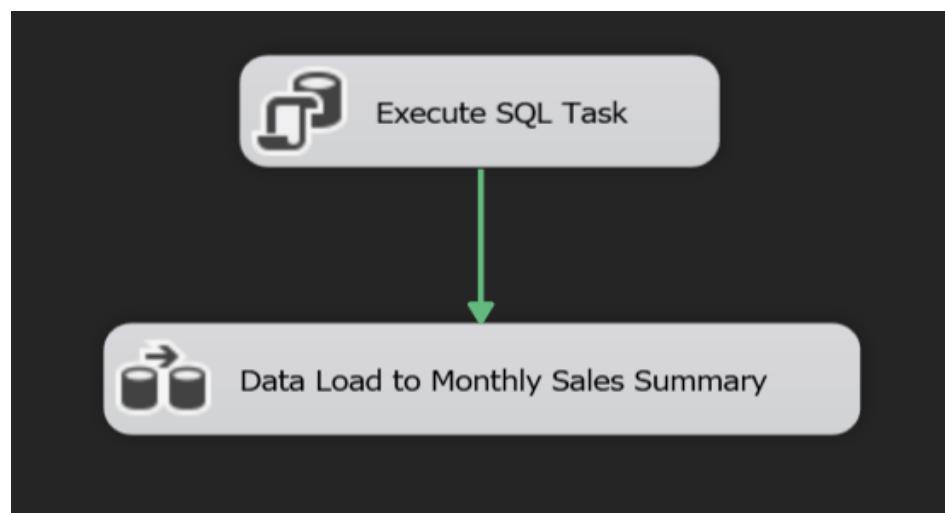
Below the code, there is a partially visible SELECT statement:

```
SELECT * FROM MONTHLY_SALES_SUMMARY_TABLE
```

In the status bar at the bottom left, it says "121 %". At the bottom right, there is a "Messages" tab with the message "Command(s) completed successfully."

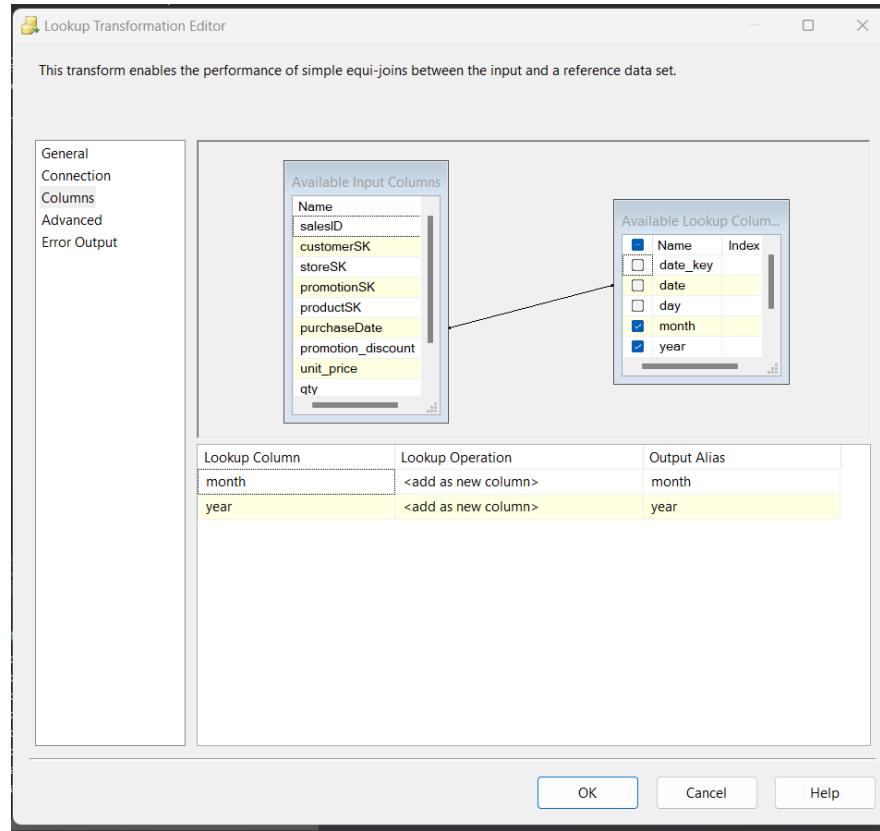
Next, data is loaded to the table using ETL

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

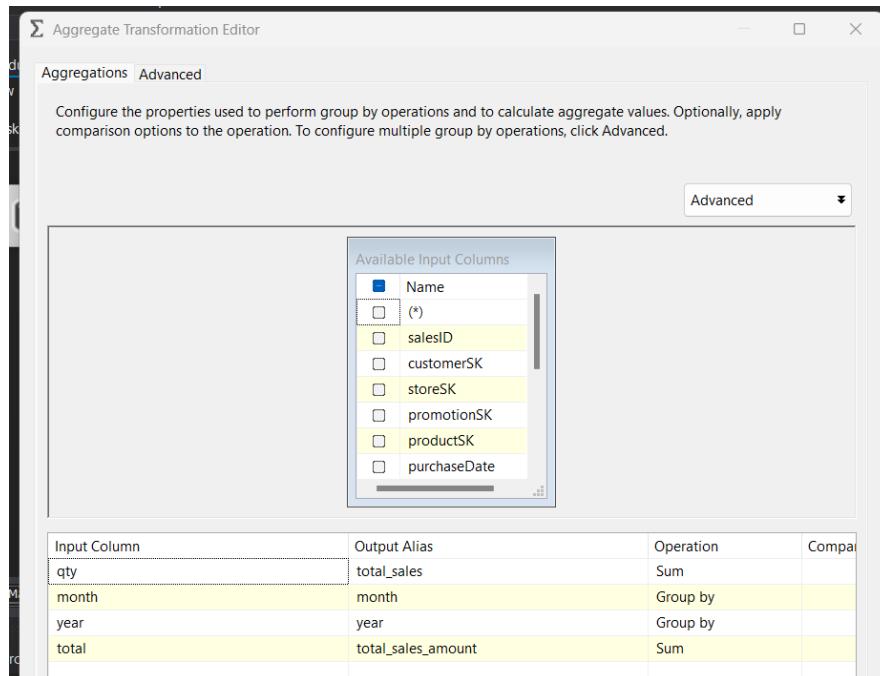


- ❖ Source assistant is added in data flow area and connected to the FACT\_SALES table in data warehouse

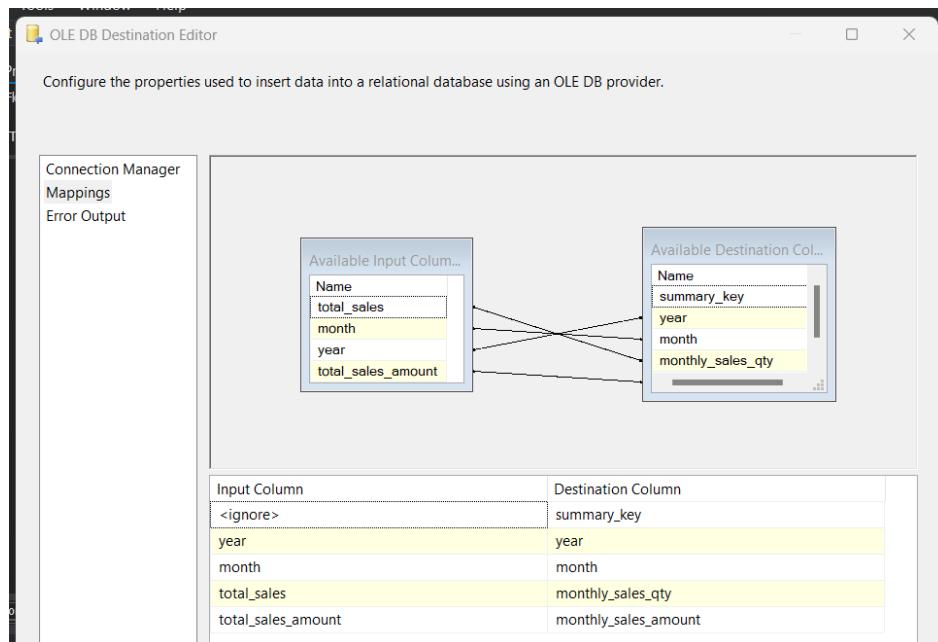
- ❖ A Lookup is done with the Dim\_Date to get the year and month using the date key



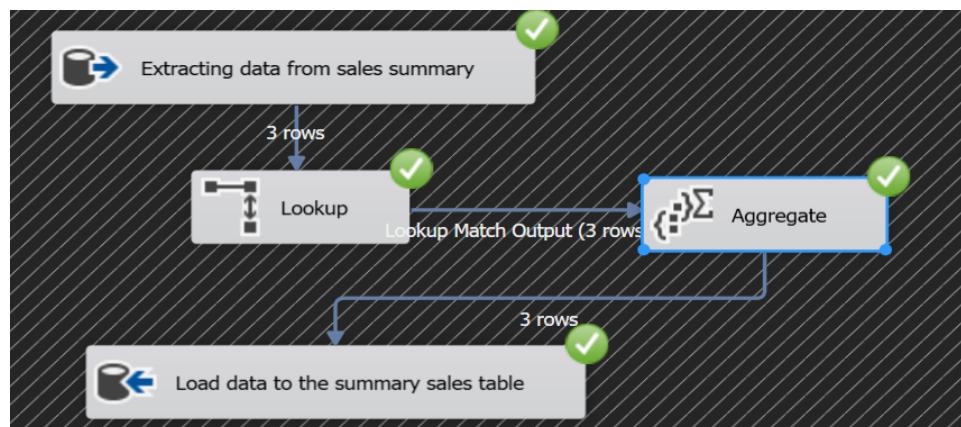
- ❖ To get the monthly sales summary, an Aggregate task is added and sum operation is applied to the quantity and total, grouped by the year and month



- ❖ 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

	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

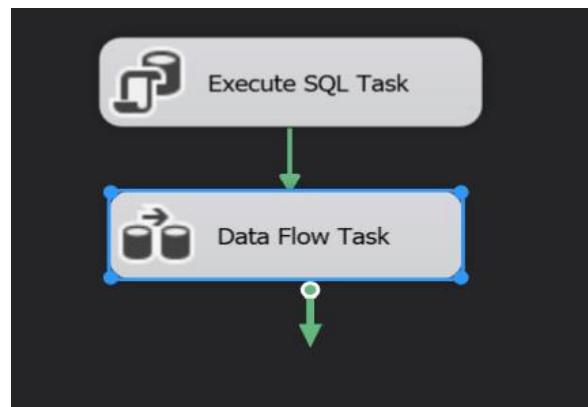
- ❖ Initially the fact table to store the market basket is created

The screenshot shows a SQL query window in SSMS. The query is:

```
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),
);
```

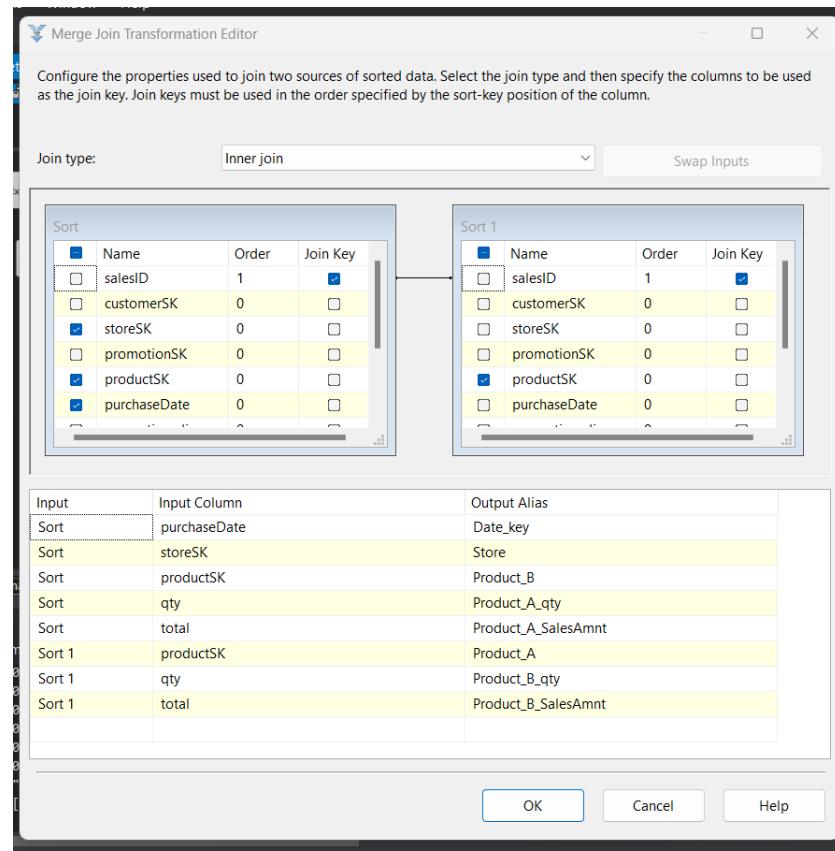
The status bar at the bottom indicates "Command(s) completed successfully."

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

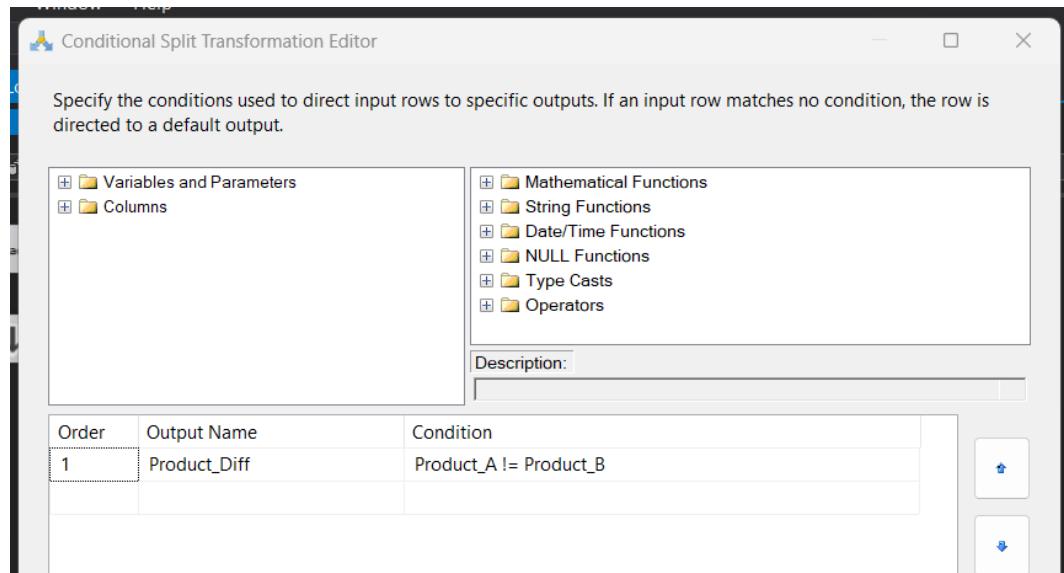


- Add two source assistants and connect both to Fact Sales, since we have to extract details for two products

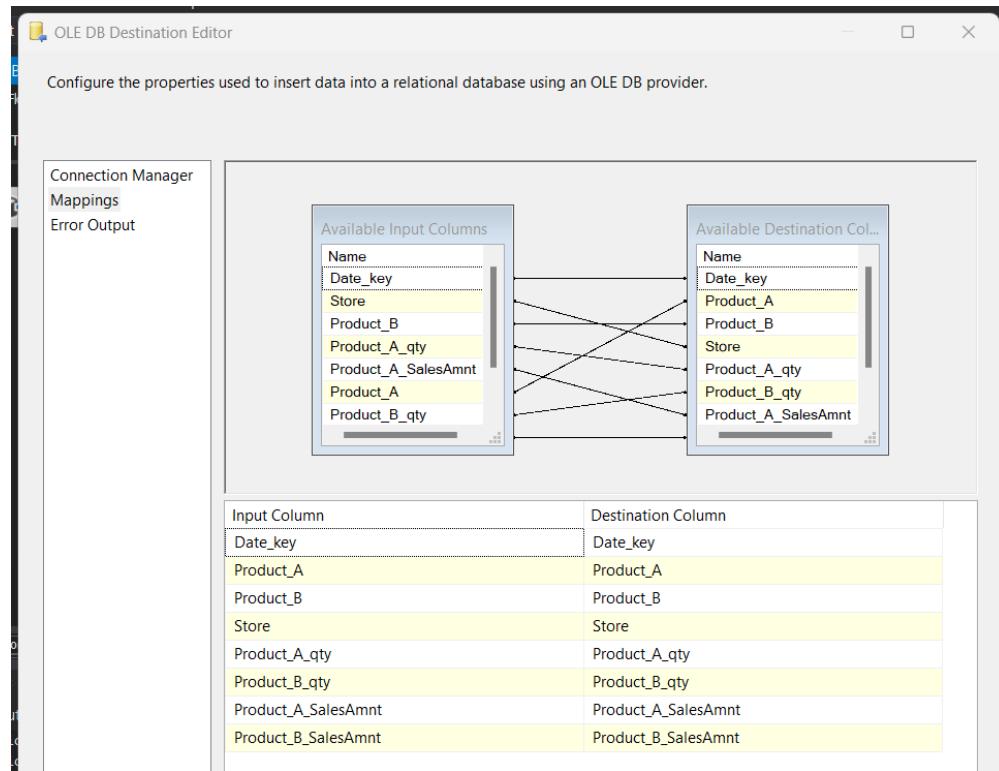
- The two are sorted and merged using the key (sales id)



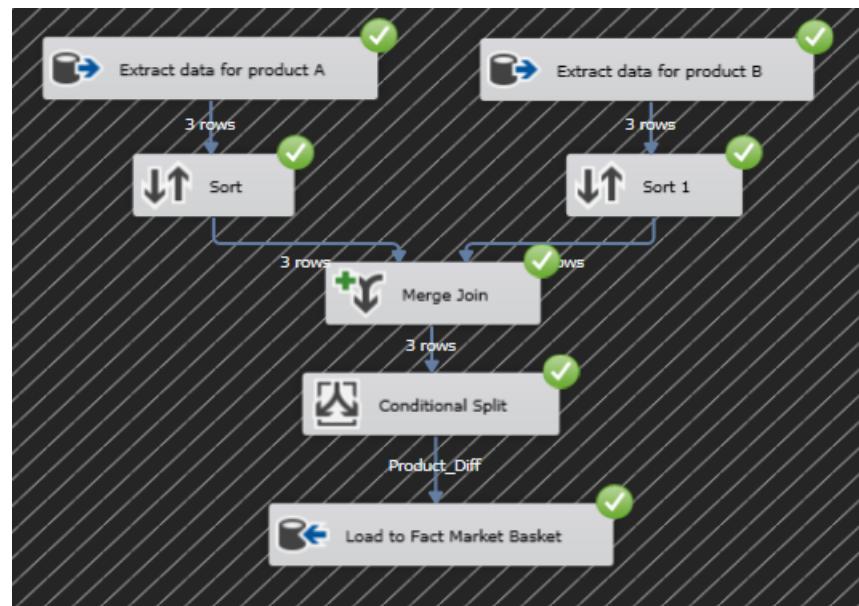
- Add a conditional split to identify the two products separately



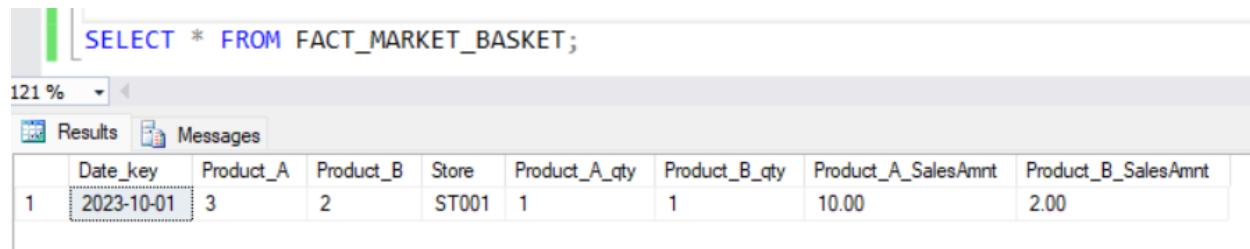
- Add destination assistant and load data to the created table



- Execute the package



- ❖ Verify the populated data



A screenshot of a SQL query results window. The query is:

```
SELECT * FROM FACT_MARKET_BASKET;
```

The results show one row of data:

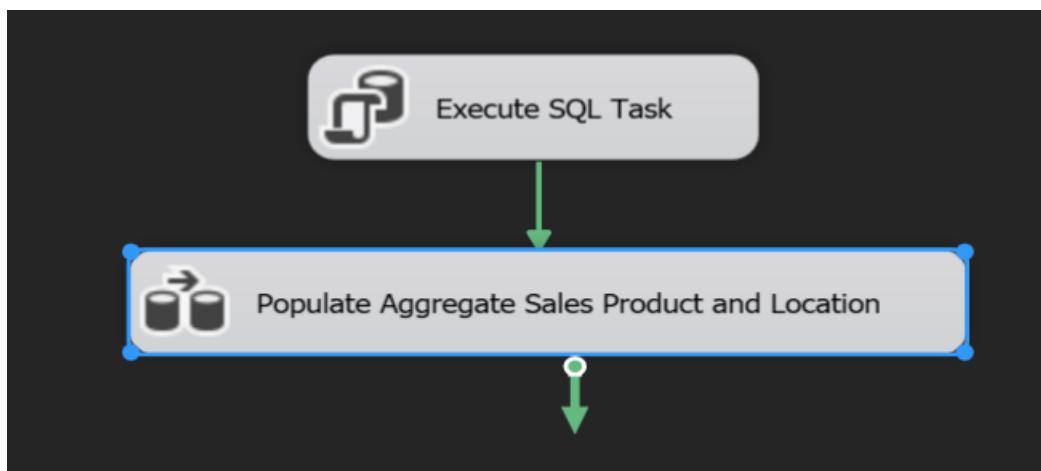
	Date_key	Product_A	Product_B	Store	Product_A_qty	Product_B_qty	Product_A_SalesAmnt	Product_B_SalesAmnt
1	2023-10-01	3	2	ST001	1	1	10.00	2.00

## PART 4

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

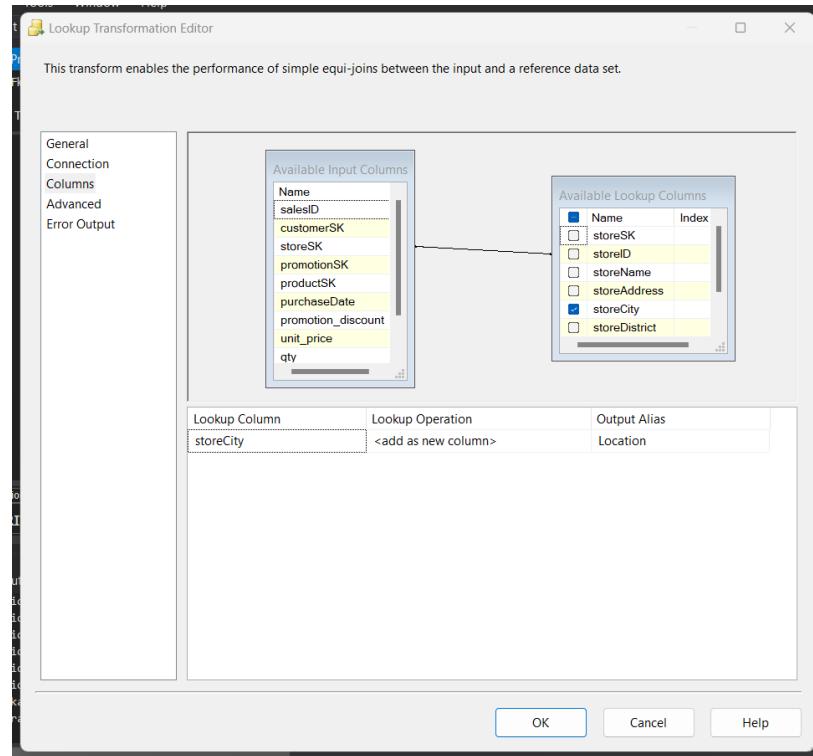
```
----- SALES PRODUCT AND LOCATION -----
CREATE TABLE SALES_PROD_LOC(
    Product varchar(20),
    Location varchar(20),
    total_sales_qty numeric(18,2),
    total_sales_amount numeric(18,2),
);
```

- ❖ 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

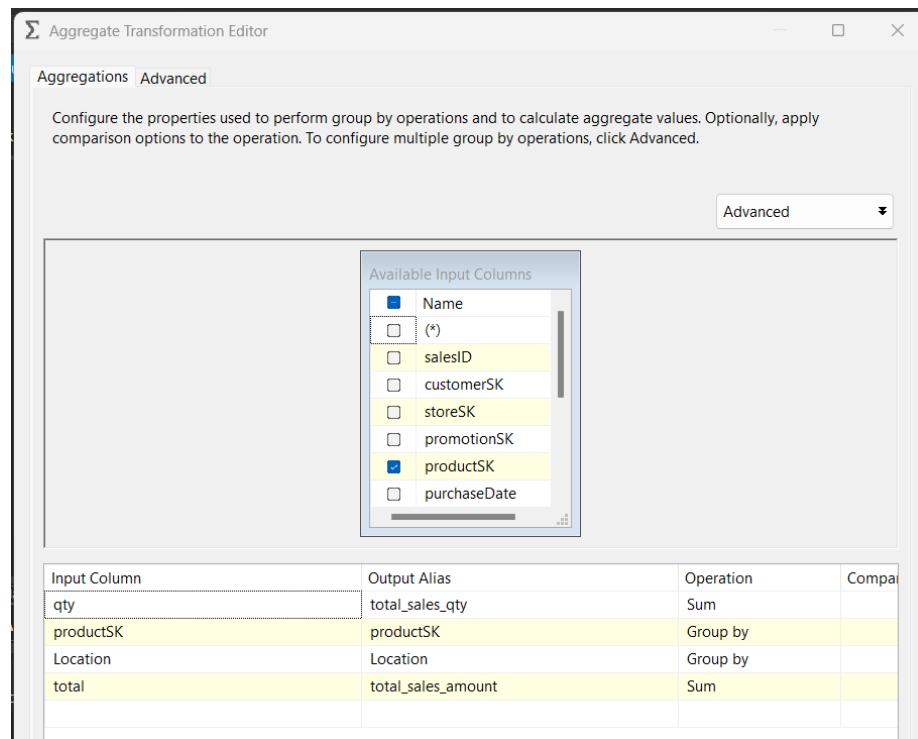


- Add source assistant and connect it with the Fact\_Sales

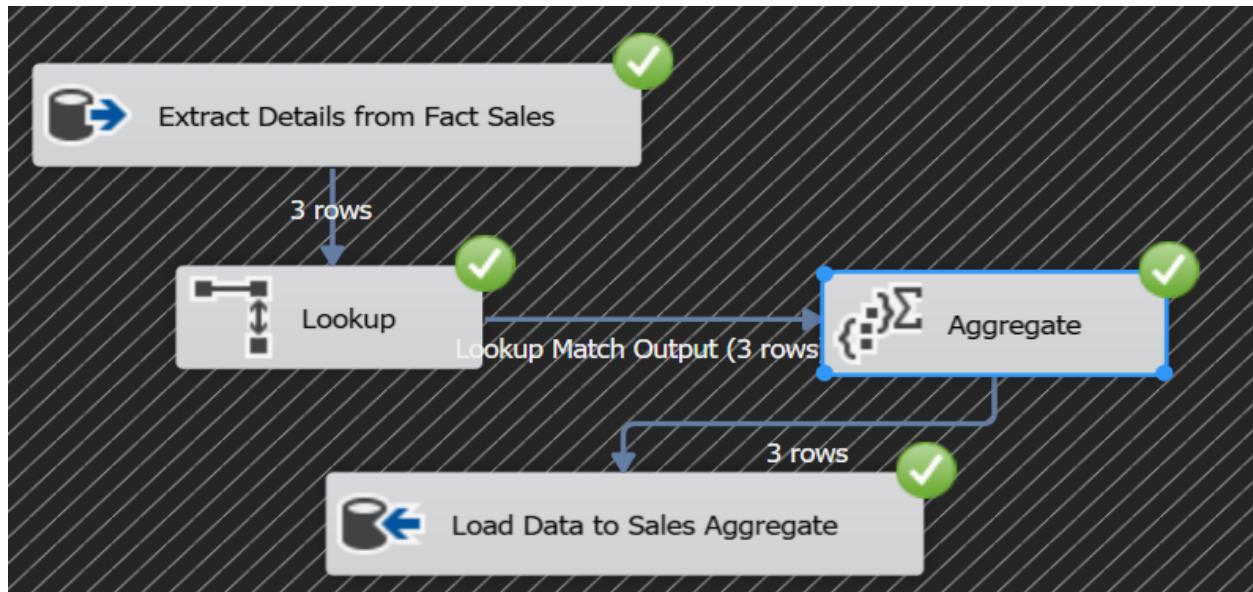
- A lookup is done with Dim\_Store to fetch the store location using storeSK and store ID



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



- Execute the package



- ❖ Verify the populated data

SELECT \* FROM SALES\_PROD\_LOC;

	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:

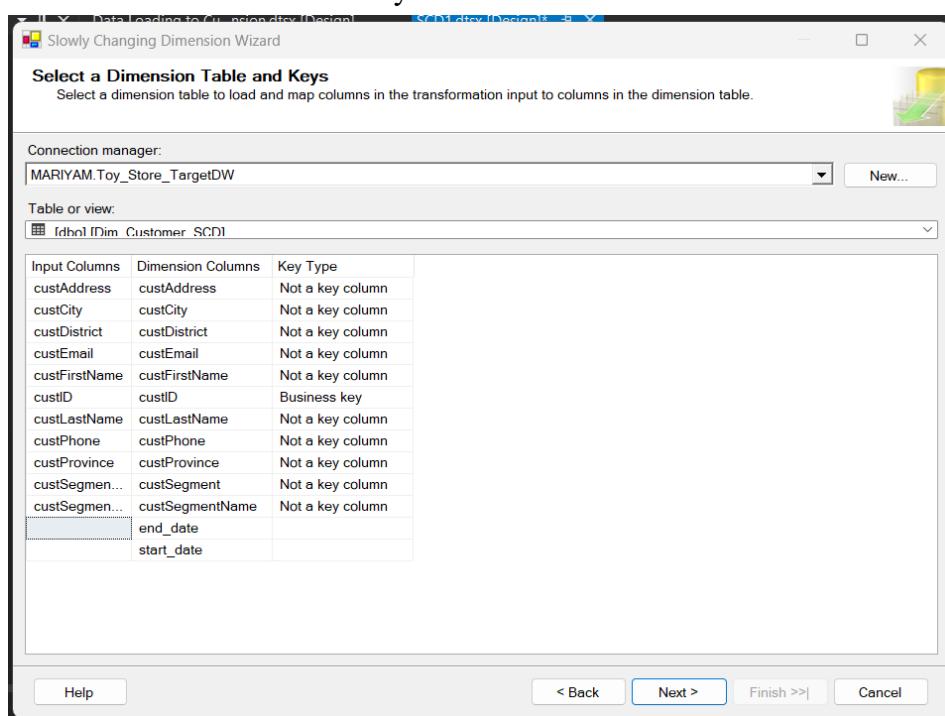
```

CREATE TABLE Dim_Customer_SCD(
    custSK int IDENTITY(1,1) NOT NULL,
    custID varchar(20),
    custFirstName varchar(20),
    custLastName varchar(20),
    custAddress varchar(100),
    custCity varchar(20),
    custDistrict varchar(20),
    custProvince varchar(20),
    custEmail varchar(50),
    custPhone nvarchar(20),
    custSegment varchar(20),
    custSegmentName varchar(50),
    start_date datetime,
    end_date datetime,
    CONSTRAINT [PK_Dim_Customer_SCD] PRIMARY KEY CLUSTERED
    (
        custSK ASC
    )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

```

Messages  
Command(s) completed successfully.

- ❖ 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

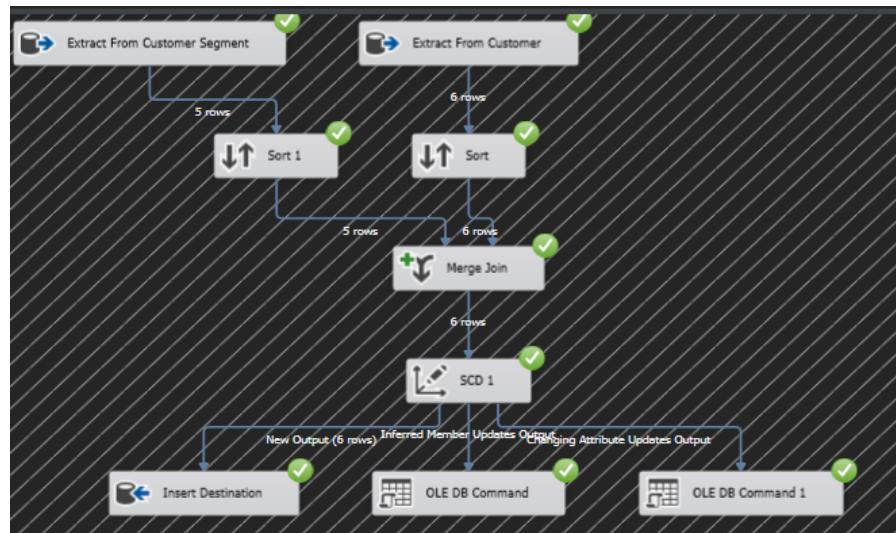
**Slowly Changing Dimension Wizard**

### Slowly Changing Dimension Columns

Manage the changes to column data in your slowly changing dimensions by setting the change type for different columns.

Dimension Columns	Change Type
custAddress	Changing attribute
custCity	Changing attribute
custDistrict	Changing attribute
custEmail	Changing attribute
custFirstName	Fixed attribute
custLastName	Fixed attribute
custPhone	Changing attribute
custProvince	Changing attribute
custSegment	Changing attribute
custSegmentName	Changing attribute

- Execute the package



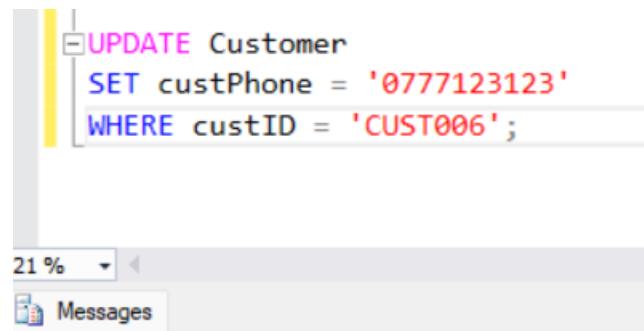
- View all inserted data

```

SELECT * FROM Dim_Customer_SCD;
  
```

custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName	start_date	end_date
1	CUST001	Malithi	Perera	123, Calvert Terrace	Dehiwala	Colombo	Western	malithi.perera@gmail.com	0773462834	SEG001	Regular Customer	NULL	NULL
2	CUST004	Migrangi	Kumar	76, Kiribathgoda	Gampaha	Gampaha	Western	migrangi.kumar@gmail.com	0729673772	SEG001	Regular Customer	NULL	NULL
3	CUST006	Mahesh	Peiris	8, Page Lane	Wellawatte	Colombo	Western	maheh.peiris@gmail.com	0773462834	SEG002	Direct Customer	NULL	NULL
4	CUST003	Ruwan	Perera	53, Ebenezer Place	Dehiwala	Colombo	Western	ruwan.perera@gmail.com	0763889111	SEG003	Online Customer	NULL	NULL
5	CUST002	Shehani	Gunatilake	29/A, Teldeniya Rd	Madawala	Kandy	Central	shehani.gk@gmail.com	0775612345	SEG004	Inactive Customer	NULL	NULL
6	CUST005	Anirudh	Ravichandar	10, Peradeniya rd	Peradeniya	Kandy	Central	aniravi@gmail.com	0777861448	SEG005	Dealer Customer	NULL	NULL

- 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@gmail.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';

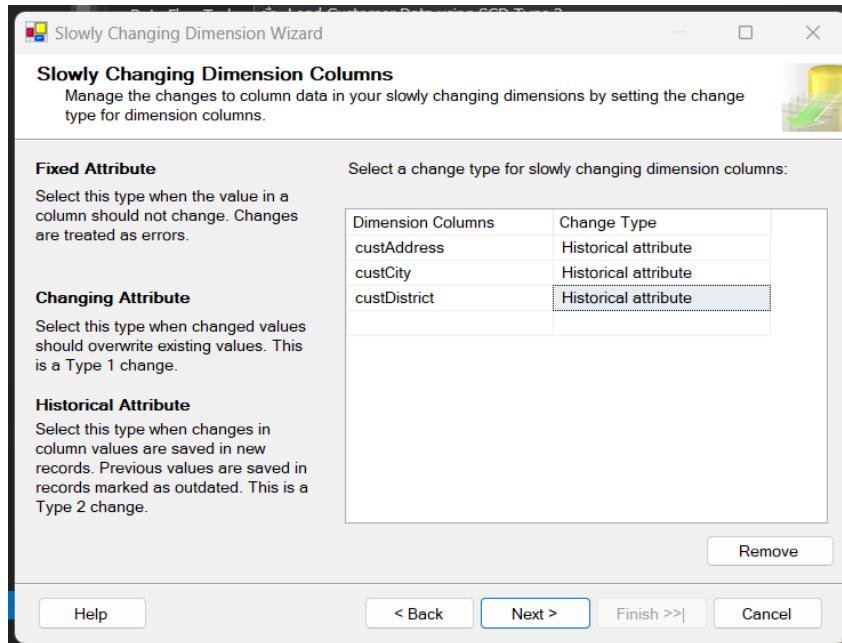
```

21 % ▾

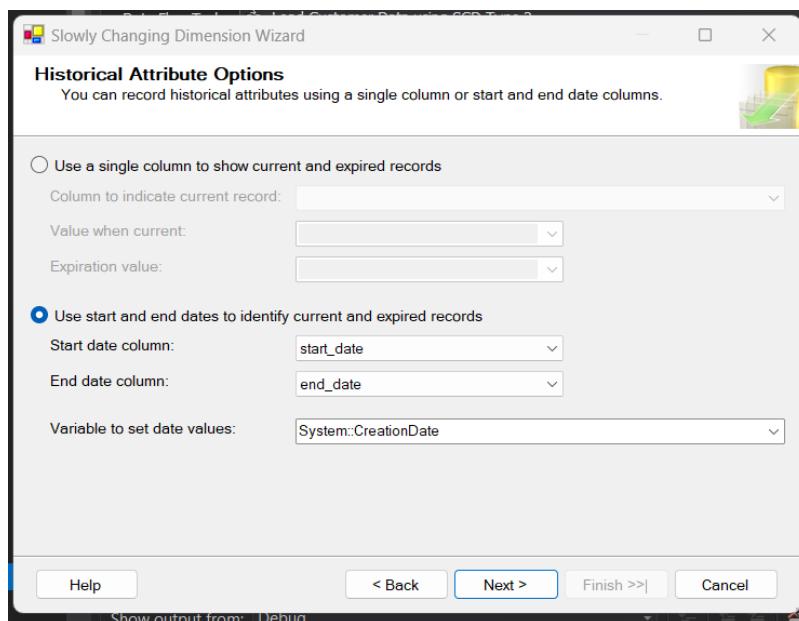
Results Messages

	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@gmail.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;																		
Results		Messages																
1	CUST001	Malithi	Perera	132, Calvert Terrace	Dehiwala	Colombo	Western	malithi.perera@gmail.com	0773462834	SEG001	Regular Customer	2023-12-19 02:30:16.000	NULL					
2	CUST004	Mihirangi	Kumari	76, Kiribathgoda	Gampaha	Gampaha	Western	mihirangi.kumari@gmail.com	0729873872	SEG001	Regular Customer	2023-12-19 02:30:16.000	NULL					
3	CUST006	Maheesh	Peiris	8, Page Lane	Wellawatte	Colombo	Western	maheesh.peiris@gmail.com	0777123123	SEG002	Direct Customer	2023-12-19 02:30:16.000	NULL					
4	CUST003	Ruwan	Perera	53, Ebenezer Place	Dehiwala	Colombo	Western	ruwan.perera@gmail.com	0763888911	SEG003	Online Customer	2023-12-19 02:30:16.000	NULL					
5	CUST002	Shehan	Gunatilake	29/A, Teldeniya Rd	Madawala	Kandy	Central	gshehan.gtk@gmail.com	0775612345	SEG004	Inactive Customer	2023-12-19 02:30:16.000	NULL					
6	CUST005	Anirudh	Ravichandar	10, Peradeniya rd	Peradeniya	Kandy	Central	ani.ravi@gmail.com	0777861448	SEG005	Dealer Customer	2023-12-19 02:30:16.000	NULL					

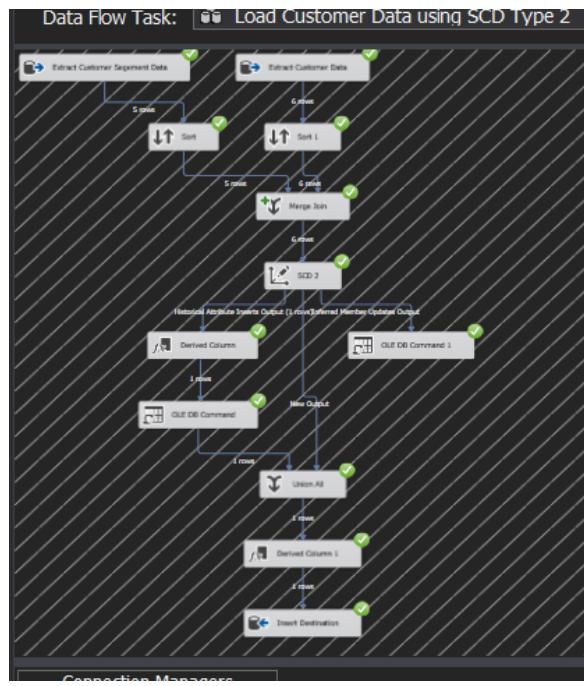
- Update the customer address in the source table

```

121 %
UPDATE Customer
SET custAddress = '18, Page Street', custCity = 'Mahaiyawa', custDistrict = 'Colombo'
WHERE custID = 'CUST006';
(1 row(s) affected)

```

- Re-execute the package



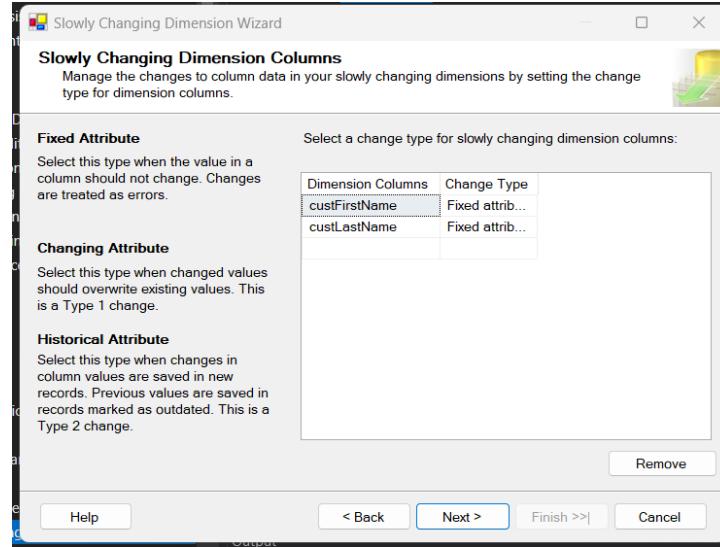
- Check for the particular customer data. Two records will be available which means the Slowly changing dimension is implemented using type 2

```

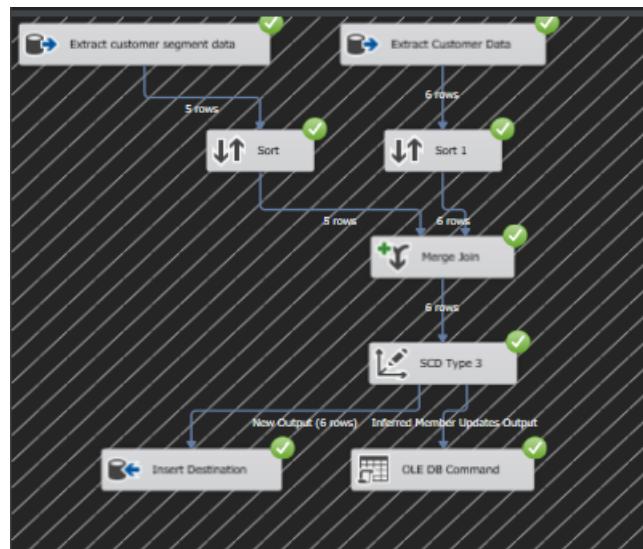
SELECT * FROM Dim_Customer_SCD
WHERE custID = 'CUST006'
  
```

	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@gmail.com	0777123123	SEG002	Direct Customer	2023-12-19 02:30:16.000	2023-12-19 02:
2	7	CUST006	Mahesh	Peiris	18, Page Street	Mahiyawa	Colombo	Western	mahesh.peiris@gmail.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

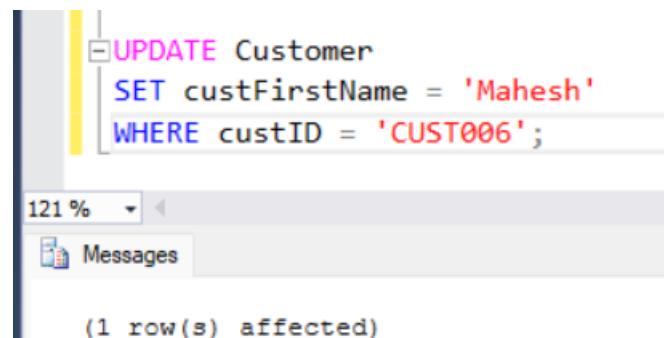


- Check the populated data

121 %

Results																	Messages	
	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	custProvince	custEmail	custPhone	custSegment	custSegmentName	start_date	end_date				
1	1	CUST001	Malith	Perera	132, Calvert Terrace	Deliwela	Colombo	Western	malith.perera@gamil.com	0773462834	SEG001	Regular Customer	NULL	NULL				
2	2	CUST004	Mihirangi	Kumari	76, Kiribathgoda	Gampaha	Gampaha	Western	mihirangi.kumari@gamil.com	0729873872	SEG001	Regular Customer	NULL	NULL				
3	3	CUST006	Maheeshika	Peiris	18, Page Street	Mahiyawa	Colombo	Western	maheesh.peiris@gamil.com	0777123123	SEG002	Direct Customer	NULL	NULL				
4	4	CUST003	Ruwan	Perera	53, Ebenezer Place	Deliwela	Colombo	Western	ruwan.perera@gamil.com	0763888911	SEG003	Online Customer	NULL	NULL				
5	5	CUST002	Shehani	Gunatilake	29/A, Teldeniya Rd	Madawala	Kandy	Central	gehshan.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';

(1 row(s) affected)
```

The screenshot shows a database query window with an UPDATE statement. The statement updates the 'Customer' table, setting 'custFirstName' to 'Mahesh' for the row where 'custID' is 'CUST006'. A message at the bottom indicates that 1 row was 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';

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	Mahaiawa	Colombo	Western	mahesh.peiris@gmail.com	0777123123	SEG002	Direct Customer	NULL	NULL

The screenshot shows a database query window with a SELECT statement. The statement retrieves all columns from the 'Dim\_Customer\_SCD' table where 'custID' is 'CUST006'. The results are displayed in a grid, showing one row of data. The columns include custSK, custID, custFirstName, custLastName, custAddress, custCity, custDistrict, custProvince, custEmail, custPhone, custSegment, custSegmentName, start\_date, and end\_date. The data for the row with custID 'CUST006' is: custSK=3, custID='CUST006', custFirstName='Maheshika', custLastName='Peiris', custAddress='18, Page Street', custCity='Mahaiawa', custDistrict='Colombo', custProvince='Western', custEmail='mahesh.peiris@gmail.com', custPhone='0777123123', custSegment='SEG002', custSegmentName='Direct Customer', start\_date=NULL, end\_date=NULL.

## SECTION D

### • PART 1

#### ❖ Enabling Change Data Capture on Database

The screenshot shows two separate queries in the SQL Server Management Studio (SSMS) Object Explorer window for the database 'Toy\_Store\_TargetDW'.

The first query (top) is:

```
USE Toy_Store_TargetDW;
--- Changing permissions since the role does not allow enabling of CDC on Database---
EXEC sp_changedbowner 'sa'
--- Enable CDC on the DB ---
EXEC sys.sp_cdc_enable_db;
```

The second query (bottom) is:

```
EXEC sys.sp_cdc_enable_db;

-- Verifying the CDC ---
SELECT name, is_cdc_enabled
FROM sys.databases
WHERE name='Toy_Store_TargetDW';
```

The results of the verification query are displayed in the 'Results' tab:

name	is_cdc_enabled	
1	Toy_Store_TargetDW	1

#### ❖ Enabling Change Data Capture on Customer Dimension

The screenshot shows a query in the SQL Server Management Studio (SSMS) Object Explorer window for the database 'MARIYAM'.

The query is:

```
-- Enable CDC on the table --
EXEC sys.sp_cdc_enable_table
@source_schema = N'dbo',
@source_name = N'DIM_CUSTOMER',
@role_name = NULL,
@supports_net_changes = 1;
```

The messages at the bottom indicate the success of the enable operation:

```
Job 'cdc.Toy_Store_TargetDW_capture' started successfully.
Job 'cdc.Toy_Store_TargetDW_cleanup' started successfully.
```

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, a database named 'MARIYAM' is selected. In the center pane, a query window titled 'changeDataCapture...RIYAM\Shahd (51)\*' is open, displaying the following T-SQL code:

```

@supports_net_changes = 1;

--- Verifying if CDC is enabled on table --
select name,type,type_desc,is_tracked_by_cdc
from sys.tables
where name = 'DIM_CUSTOMER';

```

The results pane shows a single row of data from the sys.tables system catalog:

	name	type	type_desc	is_tracked_by_cdc
1	Dim_Customer	U	USER_TABLE	1

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

The screenshot shows the SQL Server Management Studio interface. In the Object Explorer, a database named 'MARIYAM' is selected. In the center pane, a query window titled 'changeDataCapture...RIYAM\Shahd (51)\*' is open, displaying the following T-SQL code:

```

SELECT * FROM dbo.DIM_CUSTOMER;

INSERT INTO dbo.DIM_CUSTOMER VALUES('CUST001','Malithi','Perera','132, Calvert Terrace', 'Dehiwela', 'Colombo', 'Western', 'malithi.perera@gmail.com', '0773462834', 'SEG001', 'Regular Customer');

UPDATE Dim_Customer
SET custCity='Colombo 06'
WHERE custID='CUST001';

DELETE FROM Dim_Customer
WHERE custID='CUST001';

SELECT * FROM [cdc].[dbo_DIM_CUSTOMER_CT];

```

The results pane shows the data captured by the Change Data Capture (CDC) trigger:

_Start_Jrn	_Send_Jrn	_Seqval	_Operation	_Update_mask	custSK	custID	custFirstName	custLastName	custAddress	custCity	custDistrict	
1	0x0000002C00005B51001C	NULL	0x0000002C00005B51001B	2	0xFFFF	1	CUST001	Malithi	Perera	132, Calvert Terrace	Dehiwela	Colombo
2	0x0000002C00005B570003	NULL	0x0000002C00005B570002	3	0x0020	1	CUST001	Malithi	Perera	132, Calvert Terrace	Dehiwela	Colombo
3	0x0000002C00005B570003	NULL	0x0000002C00005B570002	4	0x0020	1	CUST001	Malithi	Perera	132, Calvert Terrace	Colombo 06	Colombo
4	0x0000002C00005B590004	NULL	0x0000002C00005B590002	1	0xFFFF	1	CUST001	Malithi	Perera	132, Calvert Terrace	Colombo 06	Colombo

*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

The screenshot shows the Object Explorer on the left with the 'MARIYAM' database selected. The 'Toolbox' tab is active. In the center, a query window displays T-SQL code to create a database and a table.

```

USE Toy_Store_TargetDW;
GO
CREATE TABLE [dbo].[DIM_PRODUCT_CT]
(
    [recordID] [int] IDENTITY(1,1) NOT NULL,
    [table_name] NVARCHAR(100),
    [change_type] NVARCHAR(10),
    [change_time] DATETIME,
    [changed_data] XML
)
CONSTRAINT [PK_DIM_PRODUCT_CT] PRIMARY KEY CLUSTERED
(
    [recordID] ASC
)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO

```

The 'Messages' pane at the bottom right shows the message: "Command(s) completed successfully."

- ❖ Creating a trigger to capture the insert operations

The screenshot shows the Object Explorer on the left with the 'MARIYAM' database selected. The 'Toolbox' tab is active. In the center, a query window displays T-SQL code to create a trigger.

```

GO
CREATE TRIGGER tr_insert_table
ON DIM_PRODUCT
AFTER INSERT
AS
BEGIN
    INSERT INTO DIM_PRODUCT_CT (table_name, change_type, change_time, changed_data)
    SELECT 'DIM_PRODUCT', 'INSERT', GETDATE(), (SELECT * FROM INSERTED FOR XML AUTO);
END;

```

The 'Messages' pane at the bottom right shows the message: "Command(s) completed successfully."

The screenshot shows the Object Explorer on the left with the 'MARIYAM' database selected. The 'Toolbox' tab is active. In the center, a query window displays T-SQL code to create a trigger and perform an insert operation.

```

----- CREATE TRIGGER FOR INSERTS
CREATE TRIGGER tr_insert_table
ON DIM_PRODUCT
AFTER INSERT
AS
BEGIN
    INSERT INTO DIM_PRODUCT_CT (table_name, change_type, change_time, changed_data)
    SELECT 'DIM_PRODUCT', 'INSERT', GETDATE(), (SELECT * FROM INSERTED FOR XML AUTO);
END;

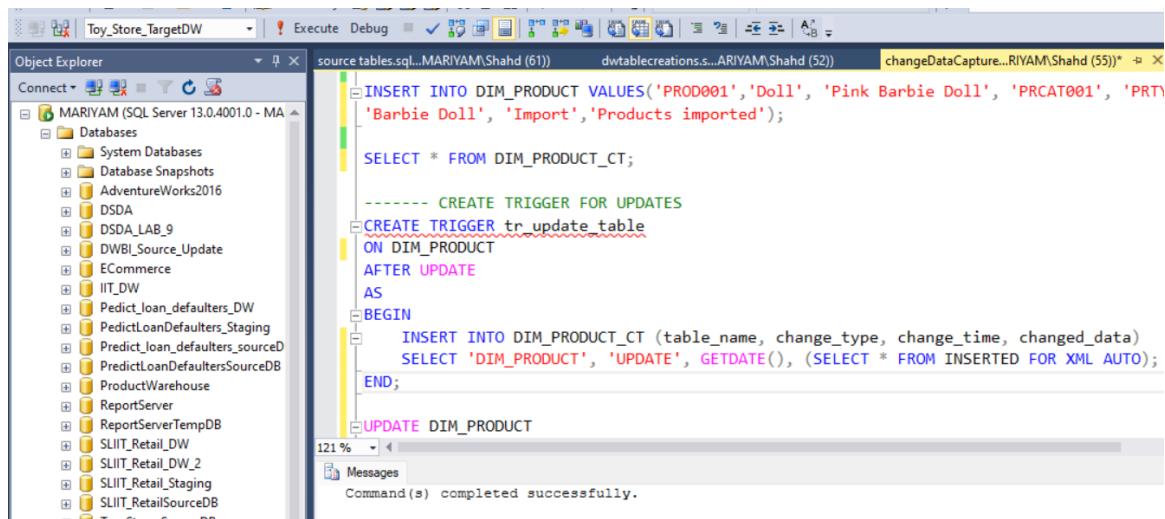
----- CREATE TRIGGER FOR UPDATES
CREATE TRIGGER tr_update_table

```

The 'Results' pane at the bottom right shows the output of the insert operation:

	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='Barbie Doll' changeType='Import' prodCategory='PRCAT001' prodSubCategory='PRTY002'>

❖ Creating trigger for update operations



```

Object Explorer   Toy_Store_TargetDW
Connect  Databases  System Databases  Database Snapshots  AdventureWorks2016  DSDA  DSDA_LAB_9  DWBI_Source_Update  ECommerce  IIT_DW  Predict_loan_defaulters_DW  PredictLoanDefaulters_Staging  Predict_loan_defaulters_sourceD  PredictLoanDefaultersSourceDB  ProductWarehouse  ReportServer  ReportServerTempDB  SLIIT_Retail_DW  SLIIT_Retail_DW_2  SLIIT_Retail_Staging  SLIIT_RetailSourceDB
source tables.sql...MARIYAM\Shahd (61)  dwtablecreations.s...ARIYAM\Shahd (52)  changeDataCapture...RIYAM\Shahd (55)*  X
----- INSERT INTO DIM_PRODUCT VALUES('PROD001', 'Doll', 'Pink Barbie Doll', 'PRCAT001', 'PRT')
----- 'Barbie Doll', 'Import', 'Products imported');

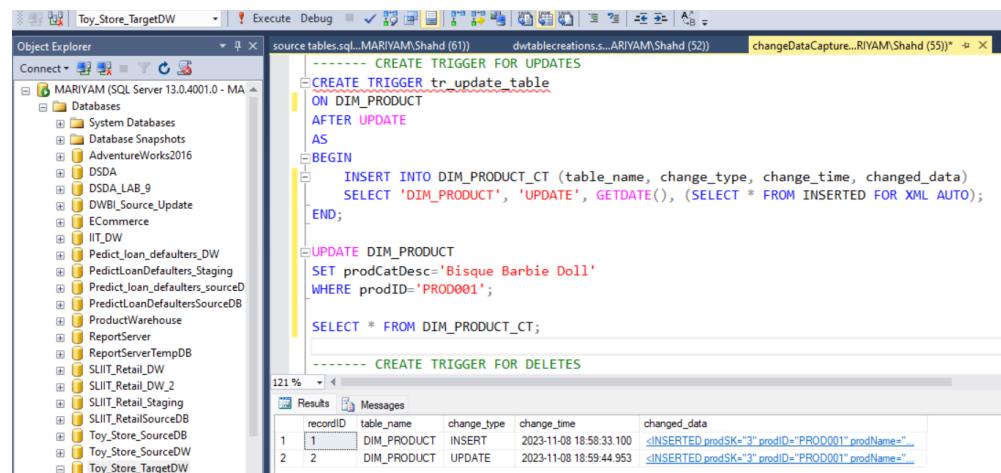
----- SELECT * FROM DIM_PRODUCT_CT;

----- CREATE TRIGGER tr_update_table
CREATE TRIGGER tr_update_table
ON DIM_PRODUCT
AFTER UPDATE
AS
BEGIN
    INSERT INTO DIM_PRODUCT_CT (table_name, change_type, change_time, changed_data)
    SELECT 'DIM_PRODUCT', 'UPDATE', GETDATE(), (SELECT * FROM INSERTED FOR XML AUTO);
END;

----- UPDATE DIM_PRODUCT
UPDATE DIM_PRODUCT
SET prodCatDesc='Bisque Barbie Doll'
WHERE prodID='PROD001';

----- SELECT * FROM DIM_PRODUCT_CT;
----- CREATE TRIGGER FOR DELETES
----- 
```

Messages  
Command(s) completed successfully.



```

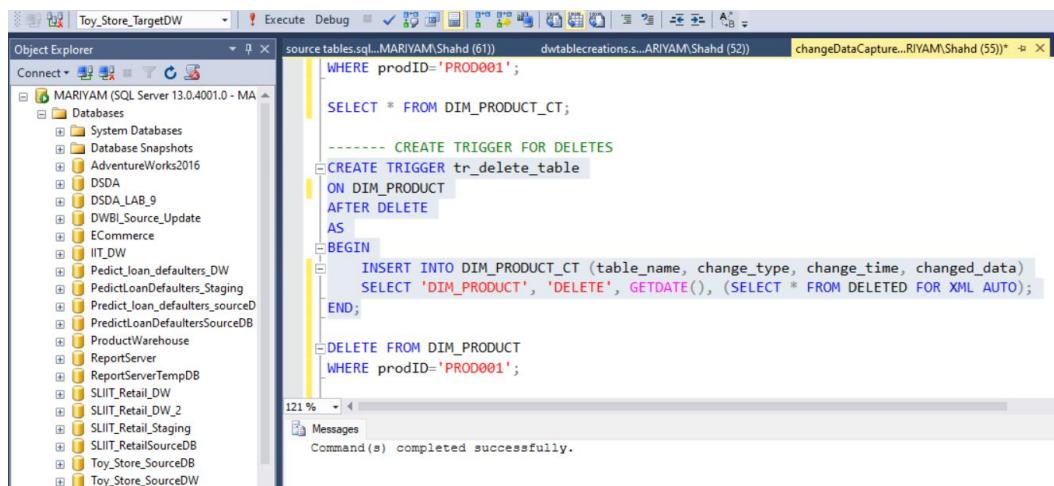
Object Explorer   Toy_Store_TargetDW
Connect  Databases  System Databases  Database Snapshots  AdventureWorks2016  DSDA  DSDA_LAB_9  DWBI_Source_Update  ECommerce  IIT_DW  Predict_loan_defaulters_DW  PredictLoanDefaulters_Staging  Predict_loan_defaulters_sourceD  PredictLoanDefaultersSourceDB  ProductWarehouse  ReportServer  ReportServerTempDB  SLIIT_Retail_DW  SLIIT_Retail_DW_2  SLIIT_Retail_Staging  SLIIT_RetailSourceDB  Toy_Store_SourceDB  Toy_Store_SourceDW  Toy_Store_TargetDW
source tables.sql...MARIYAM\Shahd (61)  dwtablecreations.s...ARIYAM\Shahd (52)  changeDataCapture...RIYAM\Shahd (55)*  X
----- CREATE TRIGGER FOR UPDATES
CREATE TRIGGER tr_update_table
ON DIM_PRODUCT
AFTER UPDATE
AS
BEGIN
    INSERT INTO DIM_PRODUCT_CT (table_name, change_type, change_time, changed_data)
    SELECT 'DIM_PRODUCT', 'UPDATE', GETDATE(), (SELECT * FROM INSERTED FOR XML AUTO);
END;

----- UPDATE DIM_PRODUCT
UPDATE DIM_PRODUCT
SET prodCatDesc='Bisque Barbie Doll'
WHERE prodID='PROD001';

----- SELECT * FROM DIM_PRODUCT_CT;
----- CREATE TRIGGER FOR DELETES
----- 
```

Results  
recordID table\_name change\_type change\_time changed\_data  
1 DIM\_PRODUCT INSERT 2023-11-08 18:58:33.100 <INSERTED prodSK='3' prodID='PROD001' prodName='<br/>'>  
2 DIM\_PRODUCT UPDATE 2023-11-08 18:59:44.953 <INSERTED prodSK='3' prodID='PROD001' prodName='<br/>'>

❖ Creating trigger for delete operations



```

Object Explorer   Toy_Store_TargetDW
Connect  Databases  System Databases  Database Snapshots  AdventureWorks2016  DSDA  DSDA_LAB_9  DWBI_Source_Update  ECommerce  IIT_DW  Predict_loan_defaulters_DW  PredictLoanDefaulters_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
source tables.sql...MARIYAM\Shahd (61)  dwtablecreations.s...ARIYAM\Shahd (52)  changeDataCapture...RIYAM\Shahd (55)*  X
----- WHERE prodID='PROD001';

----- SELECT * FROM DIM_PRODUCT_CT;

----- CREATE TRIGGER FOR DELETES
CREATE TRIGGER tr_delete_table
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
DELETE FROM DIM_PRODUCT
WHERE prodID='PROD001';

----- 
```

Messages  
Command(s) completed successfully.

Object Explorer

```

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;

```

Results

recordID	table_name	change_type	change_time	changed_data
1	DIM_PRODUCT	INSERT	2023-11-08 18:58:33.100	<INSERTED prodSK="3" prodID="PROD001" prodName="...">
2	DIM_PRODUCT	UPDATE	2023-11-08 18:59:44.953	<UPDATED prodSK="3" prodID="PROD001" prodName="...">
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

```
1 --CREATE THE SALES TABLE
2 v CREATE TABLE PRODUCT_SALES(
3     PRODUCT VARCHAR(20),
4     YEAR INT,
5     SALE NUMBER(20,2)
6 );
7
8 -- Insert Data to PRODUCT_SALES table
9 INSERT INTO PRODUCT_SALES VALUES('Prod1',2020,45000);
10 INSERT INTO PRODUCT_SALES VALUES('Prod1',2021,999999);
11 INSERT INTO PRODUCT_SALES VALUES('Prod1',2022,155000);
12 INSERT INTO PRODUCT_SALES VALUES('Prod2',2020,9999);
13 INSERT INTO PRODUCT_SALES VALUES('Prod2',2021,50000);
14 INSERT INTO PRODUCT_SALES VALUES('Prod2',2022,100000);
15 TNSERT TNTO PRODUCT_SALES VAU IES('Prod3'.2020.99999):
```

1 row(s) inserted.

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

```
18
19 -- CREATE A TABLE TO STORE THE YEARS
20 v CREATE TABLE YEAR_TAB(
21     YEARS clob
22 );
```

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.

```

24  -- CREATE VIEW
25  v  DECLARE
26      dynamic_years CLOB;
27      sql_query CLOB;
28  v  BEGIN
29      -- Clear existing data in YEAR_TAB
30      DELETE FROM YEAR_TAB;
31
32      -- Insert unique years from PRODUCT_SALES into YEAR_TAB
33      INSERT INTO YEAR_TAB (YEARS)
34          SELECT DISTINCT TO_CHAR(YEAR) FROM PRODUCT_SALES;
35
36      -- Get the dynamic years from YEAR_TAB
37      v  SELECT LISTAGG(YEARS, ',') WITHIN GROUP (ORDER BY YEARS DESC)
38          INTO dynamic_years
39          FROM (
40              SELECT TO_CHAR(YEARS) AS YEARS
41                  FROM YEAR_TAB
42                  ORDER BY YEARS DESC
43          );
44
45      EXECUTE IMMEDIATE 'DROP MATERIALIZED VIEW MV_VIEW1';
46
47      -- Build the dynamic SQL query
48      v  sql_query := 'CREATE MATERIALIZED VIEW mv_view1
49          BUILD IMMEDIATE
50          REFRESH FORCE
51          ON DEMAND
52          ENABLE QUERY REWRITE
53          AS
54          SELECT * FROM (
55              SELECT ps.PRODUCT,
56                  TO_CHAR(ps.YEAR) AS YEAR,
57                  COALESCE(ps.SALE, 0) AS SALE
58                  FROM YEAR_TAB yt
59                  LEFT JOIN PRODUCT_SALES ps ON TO_CHAR(yt.YEARS) = TO_CHAR(ps.YEAR)
60          )
61          PIVOT (
62              MAX(SALE) FOR YEAR IN (' || dynamic_years || ')
63          )
64          ORDER BY PRODUCT';
65
66      -- Execute the dynamic SQL query
67      EXECUTE IMMEDIATE sql_query;
68
69      -- Refresh the materialized view
70      DBMS_MVIEW.REFRESH('MV_VIEW1');
71 END;
~~

```

Statement processed.

- The view output

```

69  -- View the output
70  SELECT * FROM MV_VIEW1;
71

```

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;

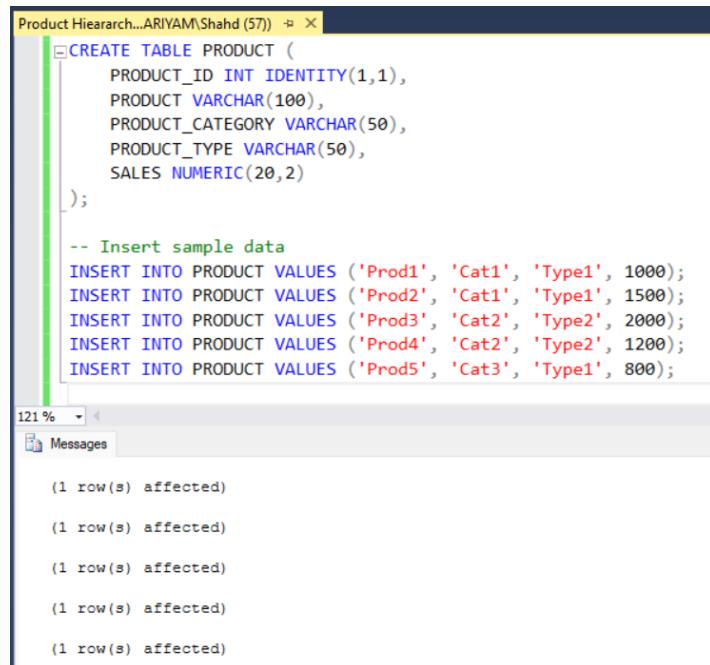
```

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



```

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);

```

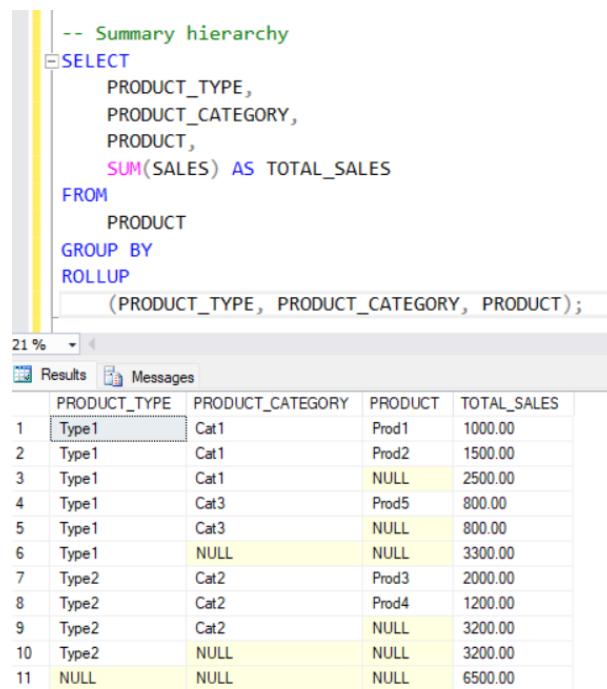
Messages

```

(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);

```

Results

	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

The screenshot shows a SQL query window and a results grid. The query is:

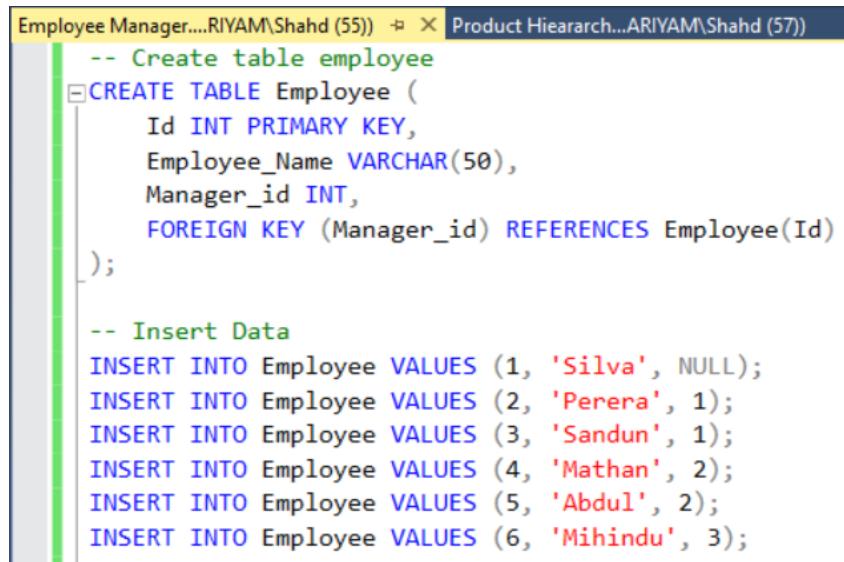
```
-- Hierarchy for each combination
SELECT
    PRODUCT_TYPE,
    PRODUCT_CATEGORY,
    PRODUCT,
    SUM(SALES) AS TOTAL_SALES
FROM
    PRODUCT
GROUP BY
    CUBE
    (PRODUCT_TYPE, PRODUCT_CATEGORY, PRODUCT);
```

The results grid displays 15 rows of data:

	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**

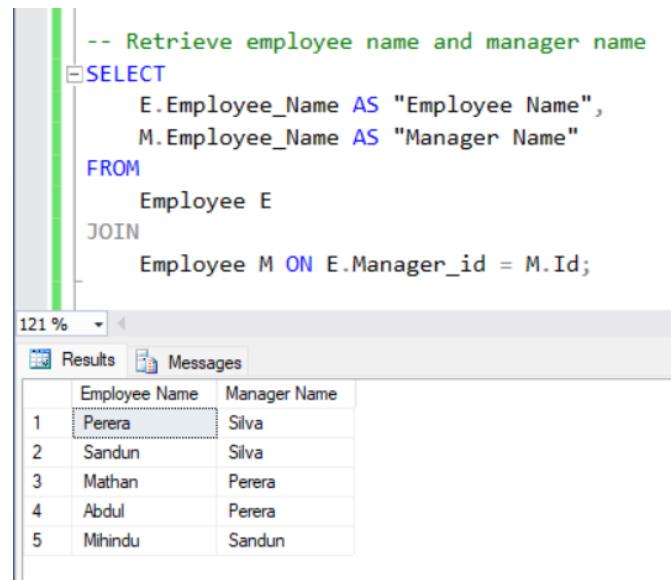
Create the employee table and insert sample data



```
-- Create table employee
CREATE TABLE Employee (
    Id INT PRIMARY KEY,
    Employee_Name VARCHAR(50),
    Manager_id INT,
    FOREIGN KEY (Manager_id) REFERENCES Employee(Id)
);

-- Insert Data
INSERT INTO Employee VALUES (1, 'Silva', NULL);
INSERT INTO Employee VALUES (2, 'Perera', 1);
INSERT INTO Employee VALUES (3, 'Sandun', 1);
INSERT INTO Employee VALUES (4, 'Mathan', 2);
INSERT INTO Employee VALUES (5, 'Abdul', 2);
INSERT INTO Employee VALUES (6, 'Mihindu', 3);
```

Query and Output are as follows:



```
-- Retrieve employee name and manager name
SELECT
    E.Employee_Name AS "Employee Name",
    M.Employee_Name AS "Manager Name"
FROM
    Employee E
JOIN
    Employee M ON E.Manager_id = M.Id;
```

121 %

	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.

Home > SQL databases >

### Create SQL Database

Microsoft

**⚠️** Changing Basic options may reset selections you have made. Review all options prior to creating the resource.

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* ⓘ

Resource group \* ⓘ

**Database details**

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name \*

Server \* ⓘ

Want to use SQL elastic pool? ⓘ  Yes  No

Workload environment  Development  Production

All resources ⚡ ...  
Robert Gordon University (live.rgu.ac.uk)

+ Create  Refresh

Filter for any field... Subscription equals all Resource group equals all Type equals all Location equals all Add filter

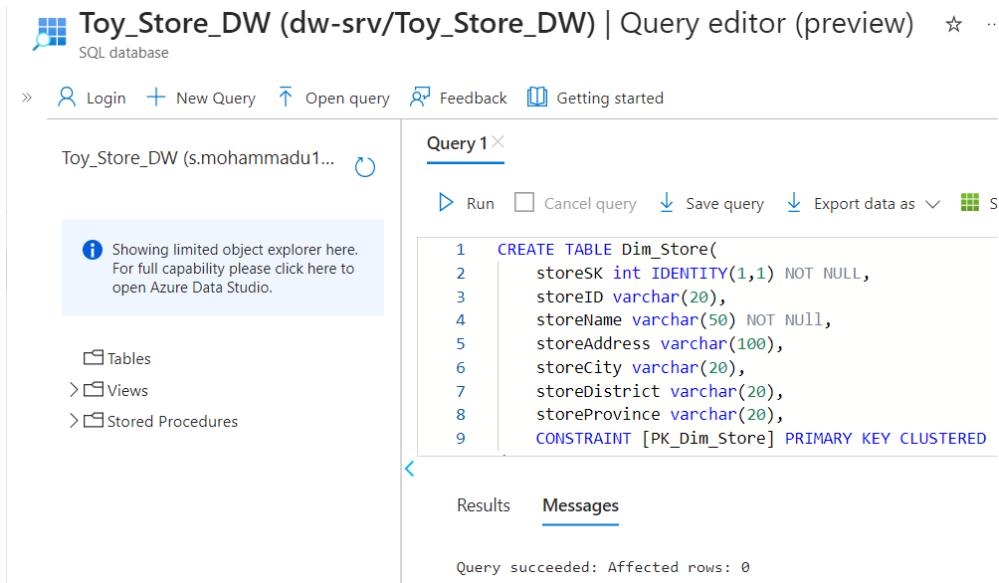
Name	Type	Resource group	Location	Subscription	...
dw-srv	SQL server	dw-rg	East US	Azure for Students	...
Toy_Store_DW (dw-srv/Toy_Store_DW)	SQL database	dw-rg	East US	Azure for Students	...

**Firewall rules**  
Allow certain public internet IP addresses to access your resource. Learn more

Rule name	Start IPv4 address	End IPv4 address
ClientIPAddress_2023-12-21-7-26-23	92.0.33.49	92.0.33.49

**Exceptions**  
 Allow Azure services and resources to access this server

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



The screenshot shows the Azure Data Studio interface for the 'Toy\_Store\_DW' database. The left sidebar displays the schema with 'Tables', 'Views', and 'Stored Procedures'. The main area is titled 'Query 1' and contains the following SQL code:

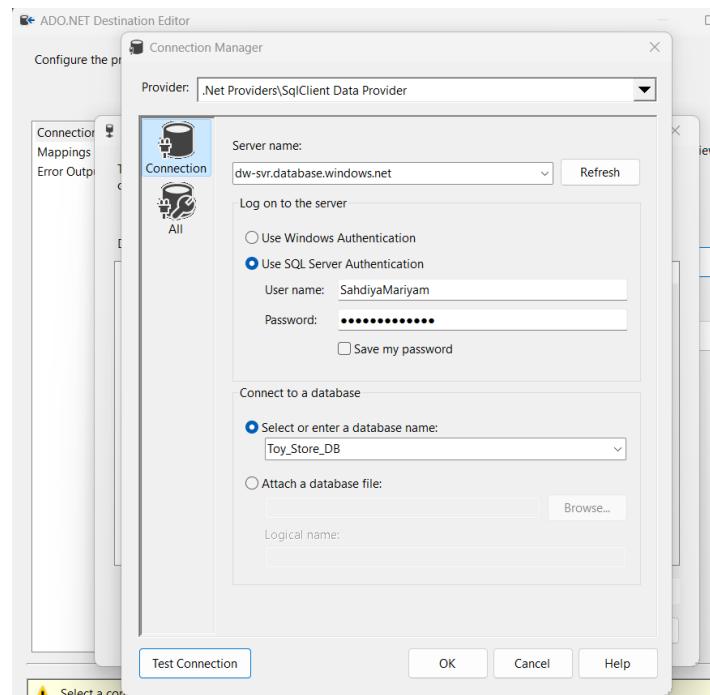
```

1 CREATE TABLE Dim_Store(
2     storeSK int IDENTITY(1,1) NOT NULL,
3     storeID varchar(20),
4     storeName varchar(50) NOT NULL,
5     storeAddress varchar(100),
6     storeCity varchar(20),
7     storeDistrict varchar(20),
8     storeProvince varchar(20),
9     CONSTRAINT [PK_Dim_Store] PRIMARY KEY CLUSTERED

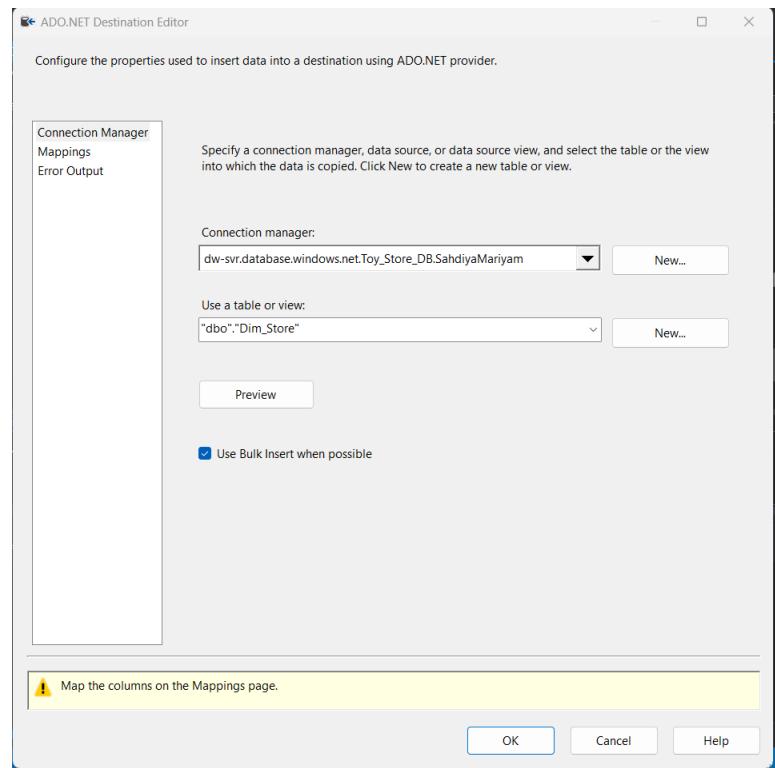
```

Below the code, the status message 'Query succeeded: Affected rows: 0' is displayed.

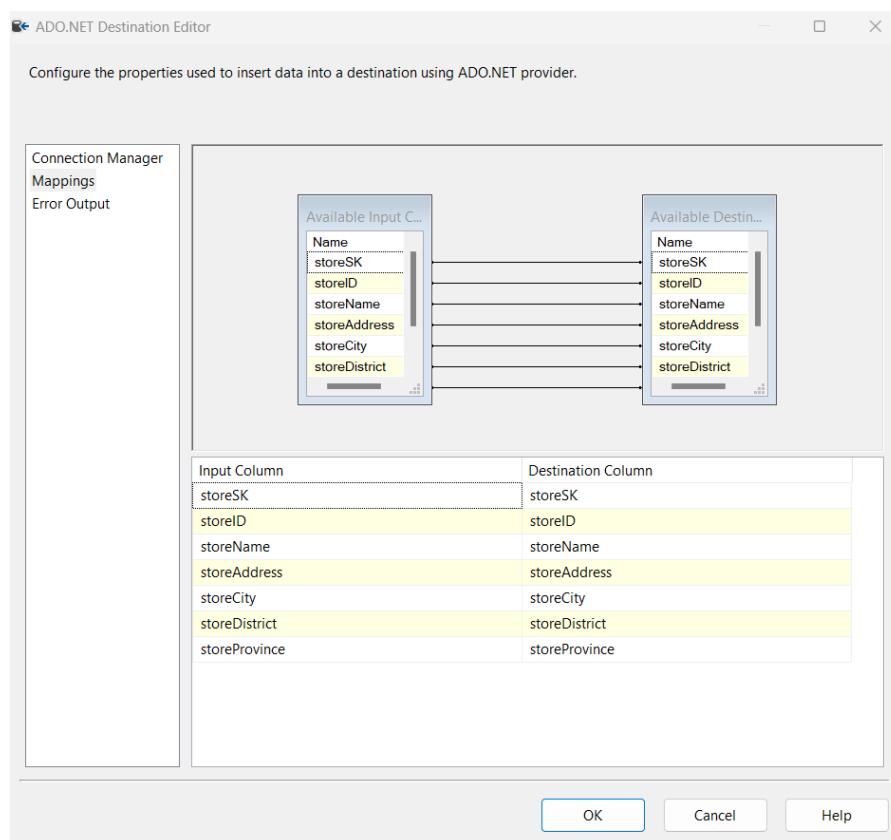
- Open SSIS and create a new integration project
- Add a Data Flow task in Control Area
- Add a source assistant/OLE DB Source and connect it to the Store dimension
- Add 'ADO.NET' destination task and connect to the server created in Azure Cloud.



- 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

Query 1 ×    Query 2 ×

Run    Cancel query    Save query    Export data as    Show only Editor

```
1   SELECT TOP (1000) * FROM [dbo].[Dim_Store]
```

Results    Messages

Search to filter items...

storeSK	storeId	storeName	storeAddress
1	ST001	Colombo Branch 1	475, Union Place
2	ST002	Kandy Branch 1	12, D.S Senanayake Rd
3	ST003	Kandy Branch 2	22/A, Peradeniya Rd
4	ST004	Colombo Branch 2	151, Ebenezer Place

## PART 2

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

```
1  -- CREATE TABLE EMPLOYEE
2  CREATE TABLE Employee(
3    emp_id NVARCHAR(20) PRIMARY KEY NOT NULL,
4    emp_name NVARCHAR(20),
5    designation NVARCHAR(20),
6  );
7
8  -- CREATE TABLE EXECUTIVE EMPLOYEE
9  CREATE TABLE ExecutiveEmployee(
10   emp_id NVARCHAR(20) PRIMARY KEY NOT NULL,
11   emp_name NVARCHAR(20),
12   designation NVARCHAR(20),
13 );
14
15 -- CREATE TABLE NON-EXECUTIVE EMPLOYEE
16 CREATE TABLE NonExecutiveEmployee(
17   emp_id NVARCHAR(20) PRIMARY KEY NOT NULL,
18   emp_name NVARCHAR(20),
19   designation NVARCHAR(20),
20 );
21
22  -- INSERT DATA
23 INSERT INTO Employee VALUES ('EMP001', 'Murugan', 'Executive');
24 INSERT INTO Employee VALUES ('EMP002', 'Arunan', 'Executive');
25 INSERT INTO Employee VALUES ('EMP003', 'Perera', 'Non-Executive');
26 INSERT INTO Employee VALUES ('EMP004', 'Steve', 'Executive');
27 INSERT INTO Employee VALUES ('EMP005', 'Ayeisha', 'Non-Executive');
28 INSERT INTO Employee VALUES ('EMP006', 'Nimalka', 'Non-Executive');
29 INSERT INTO Employee VALUES ('EMP007', 'Jacob', 'Non-Executive');
30 INSERT INTO Employee VALUES ('EMP008', 'Sachini', 'Executive');
31 INSERT INTO Employee VALUES ('EMP009', 'Shalini', 'Non-Executive');
```

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

Home > Data factories >

### Create Data Factory

Basics    Git configuration    Networking    Advanced    Tags    Review + create

One-click to create data factory with sample pipeline and datasets. [Try it](#)

**Project details**

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* ⓘ    Azure for Students

Resource group \* ⓘ    datawarehouse-rg

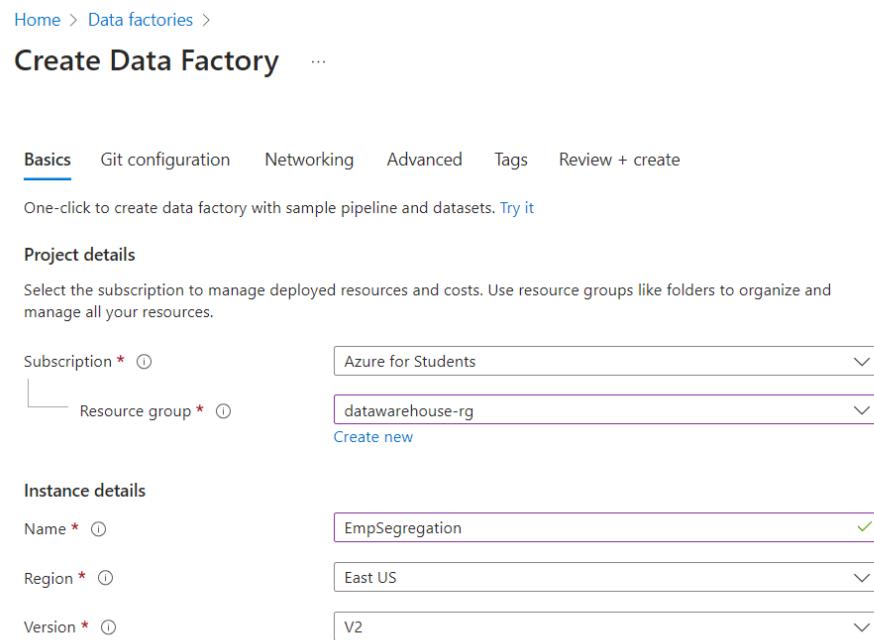
[Create new](#)

**Instance details**

Name \* ⓘ    EmpSegregation

Region \* ⓘ    East US

Version \* ⓘ    V2



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

The screenshot shows the Azure Data Factory Studio interface. On the left, a sidebar lists various activities: Move and transform, Copy data, Data flow, Synapse, Azure Data Explorer, Azure Function, Batch Service, Databricks, Data Lake Analytics, General, HDInsight, Iteration & conditionals, Machine Learning, and Power Query. The main area displays two parallel configurations:

**Data Pipeline Configuration:**

- General Tab:** Name: EmpSeggregationDFT, Description: The data flow task to categorize employee as executive and non executive.
- Settings Tab:** Data flow: dataflow1, Run on (Azure IR): AutoResolveIntegrationRuntime, Compute size: Small, Compute type: General purpose, Core count: 4 (+ 4 Driver cores), Logging level: Verbose.

**Data Flow Task Configuration:**

- General Tab:** Name: EmpSeggregationDFT, Description: EmpSeggregationDFT.
- Settings Tab:** Data flow: EmpSeggregationDFT, Run on (Azure IR): AutoResolveIntegrationRuntime, Compute size: Small, Compute type: General purpose, Core count: 4 (+ 4 Driver cores), Logging level: Verbose.

**New linked service configuration:**

**Connection string tab:** Selected. Account selection method: From Azure subscription. Azure subscription: Azure for Students (d0a6d27f-89bc-474b-bf05-8c803ec6a95e). Server name: dw-svr. Database name: Toy\_Store\_DB. Authentication type: SQL authentication. User name: SahdiyaMariyam. Password: (redacted). Always encrypted: (unchecked).

**Test connection results:** Connection successful (green checkmark). Test connection button.

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

**Set properties**

Name  
AzureSqlTable1

Linked service \*  
AzureSqlDatabase1

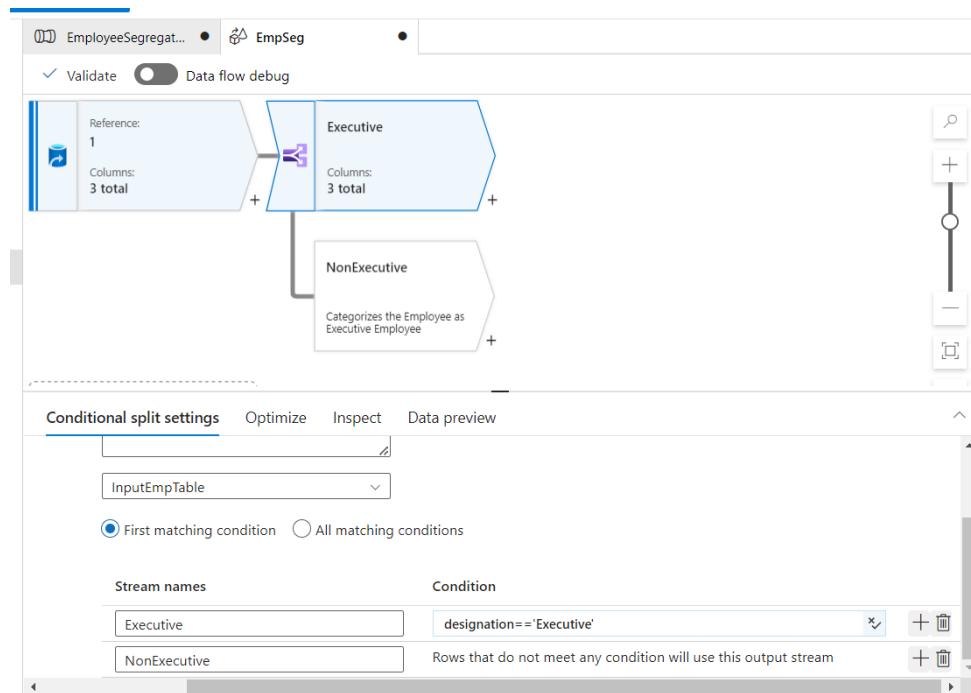
Table name  
dbo.Employee

Enter manually

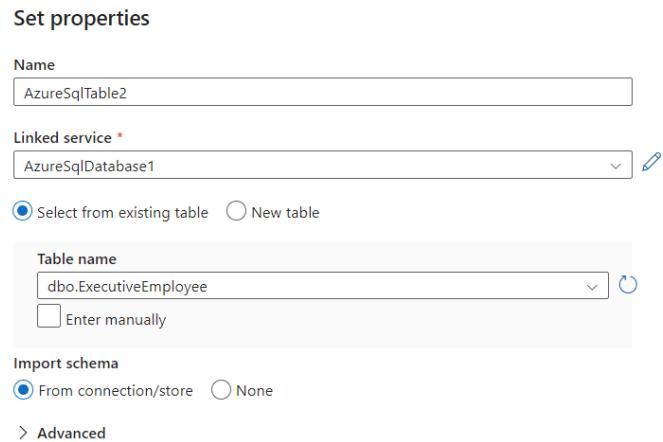
Import schema  
 From connection/store  None

> Advanced

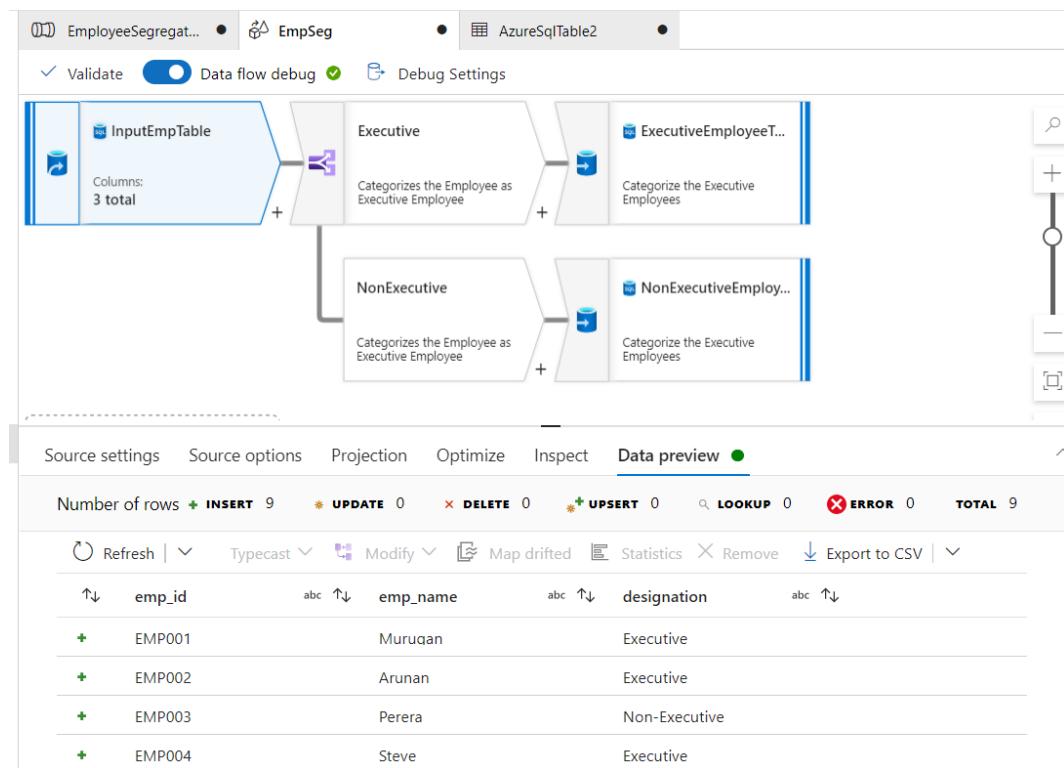
- After adding the source, add a conditional split and configure as follows to segregate the employees



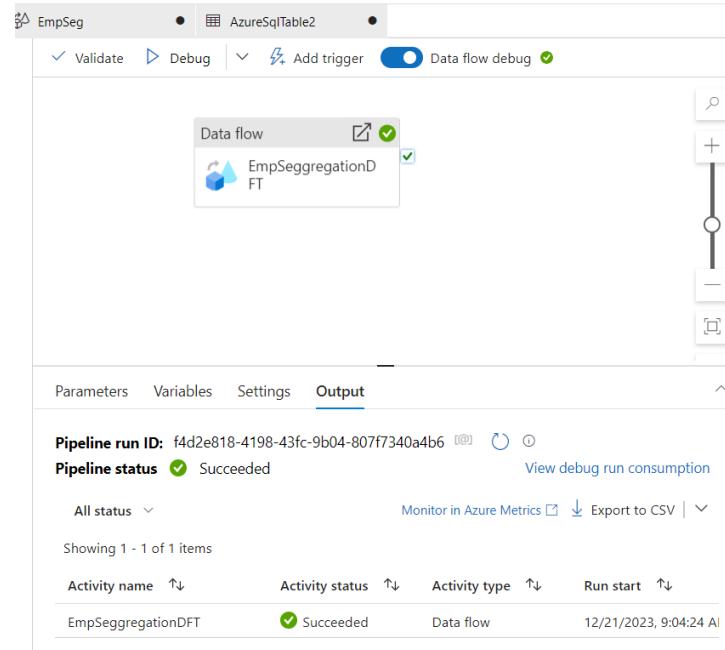
- Now add destination to both the splits and configure it to connect to the Executive and Non-Executive Employee Tables



- 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

### Create Synapse workspace ...

#### Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all of your resources.

Subscription *	Azure subscription 1
Resource group *	sm-rg
Managed resource group	Enter managed resource group name

#### Workspace details

Name your workspace, select a location, and choose a primary Data Lake Storage Gen2 file system to serve as the default location for logs and job output.

Workspace name *	emp-dl-workspace
Region *	East US
Select Data Lake Storage Gen2 *	<input checked="" type="radio"/> From subscription <input type="radio"/> Manually via URL
Account name *	(New) empstracc
File system name *	(New) empstrfs

[Review + create](#)

< Previous

Next: Security >

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

The screenshot shows the Azure Storage Accounts blade for the 'empstracc' account. On the left, there's a list of storage accounts: 'empstracc', 'shahimariaacc', and 'shahimatestacc'. The 'Containers' section is selected under 'Data storage'. On the right, a 'New container' dialog is open, asking for a name ('employee') and setting the 'Anonymous access level' to 'Private (no anonymous access)'. A note in the dialog states: 'The access level is set to private because anonymous access is disabled on this storage account.'

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

The screenshot shows the Azure Storage Explorer interface. On the left, there's a sidebar with options like Overview, Diagnose and solve problems, Access Control (IAM), Settings, Shared access tokens, Manage ACL, Access policy, Properties, and Metadata. The main area displays a list of blobs under the 'employee' container. One blob, 'Employee.csv', is listed with its name, modified date (12/22/2023, 9:59:01 ...), and a 'Hot' access tier indicator.

Name	Modified	Access Tier
Employee.csv	12/22/2023, 9:59:01 ...	Hot

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

This screenshot shows the Microsoft Azure Synapse Analytics workspace. The left sidebar has sections for Data, Workspace, and Linked. Under Data, there's a 'Lake database' section containing 'EmployeeDB'. A 'Get started' dialog is open in the center, prompting the user to choose a name for the database ('EmployeeDB') and select storage settings for the database. It also lists the linked service ('emp-dl-workspace-WorkspaceC...', Input folder ('employee/'), and Data format ('Delimited Text')).

This screenshot shows the Microsoft Azure Synapse Analytics workspace. The left sidebar has sections for Data, Workspace, and Linked. Under Data, there's a 'Lake database' section containing 'EmployeeDB'. Below it is a 'Tables' section. The top right corner shows a 'Validate all' button.

- 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.

The screenshot shows the 'Create external table from data lake' wizard in the Microsoft Azure Synapse Analytics interface. The 'External table details' section is displayed, where the 'External table name' is set to 'Employee', the 'Linked service' is 'emp-dl-workspace-WorkspaceDefaultStorage(empitracc)', and the 'Input file or folder' is 'employee/Employee.csv'. The 'Continue' and 'Cancel' buttons are at the bottom of the wizard.

- Now publish all changes to create the table

The screenshot shows the 'Employee' table properties in the Microsoft Azure Synapse Analytics interface. The 'General' tab is selected, displaying the table name 'Employee' and a description field. The 'Columns' tab lists columns: empID, ename, address, phone, age, and salary. The 'Properties' pane on the right indicates the table has been successfully published. The 'Publish' and 'Cancel' buttons are at the bottom of the properties pane.

## 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

```
1 -- Create a table with range partitioning for sales data, partitioned by year
2 v CREATE TABLE sales_data_range_partition (
3   sale_id INT PRIMARY KEY,
4   product_id INT,
5   year INT,
6   sale_amount DECIMAL(10, 2),
7   location VARCHAR(50)
8 )
9 PARTITION BY RANGE (year) (
10   PARTITION p2016 VALUES LESS THAN (2017),
11   PARTITION p2017 VALUES LESS THAN (2018),
12   PARTITION p2018 VALUES LESS THAN (2019),
13   PARTITION p2019 VALUES LESS THAN (2020),
14   PARTITION p2020 VALUES LESS THAN (2021),
15   PARTITION p2021 VALUES LESS THAN (2022),
16   PARTITION p2022 VALUES LESS THAN (2023),
17   PARTITION p2023 VALUES LESS THAN (MAXVALUE)
18 );
```

Table created.

Inserting dummy data to the table

```
20 -- Insert sample data into the sales_data table
21 INSERT INTO sales_data_range_partition VALUES (1, 101, 2017, 500.00, 'Colombo');
22 INSERT INTO sales_data_range_partition VALUES (2, 102, 2018, 700.50, 'Gampaha');
23 INSERT INTO sales_data_range_partition VALUES (3, 101, 2019, 450.25, 'Kandy');
24 INSERT INTO sales_data_range_partition VALUES (4, 103, 2020, 900.75, 'Colombo');
25 INSERT INTO sales_data_range_partition VALUES (5, 104, 2021, 400.00, 'Anuradhapura');
26 INSERT INTO sales_data_range_partition VALUES (6, 105, 2022, 350.85, 'Galle');
27 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

```
29
30 -- View Partition
31 v SELECT
32   TABLE_NAME,
33   PARTITION_NAME,
34   HIGH_VALUE
35   FROM
36   USER_TAB_PARTITIONS
37   WHERE
38     TABLE_NAME = 'SALES_DATA_RANGE_PARTITION';
39
40
```

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

```
42 -- Create a table with list partitioning for sales
43 v CREATE TABLE sales_data_list_partition (
44     sale_id INT PRIMARY KEY,
45     product_id INT,
46     year INT,
47     sale_amount DECIMAL(10, 2),
48     location VARCHAR(50)
49 )
50 PARTITION BY LIST (location) (
51     PARTITION p_colombo VALUES ('Colombo'),
52     PARTITION p_kandy VALUES ('Kandy'),
53     PARTITION p_matara VALUES ('Matara'),
54     PARTITION p_galle VALUES ('Galle'),
55     PARTITION p_other VALUES (DEFAULT)
56 );
```

Table created.

Inserting dummy data

```
59 -- Insert sample data into the sales_data_list_partitioned table
60 INSERT INTO sales_data_list_partition VALUES (1, 101, 2020, 500.00, 'Colombo');
61 INSERT INTO sales_data_list_partition VALUES (2, 102, 2020, 700.50, 'Kandy');
62 INSERT INTO sales_data_list_partition VALUES (3, 201, 2021, 450.25, 'Matara');
63 INSERT INTO sales_data_list_partition VALUES (4, 301, 2021, 900.75, 'Galle');
64
```

1 row(s) inserted.

1 row(s) inserted.

Viewing the partitions

```
65 -- View Partitions
66 v SELECT
67     TABLE_NAME,
68     PARTITION_NAME,
69     HIGH_VALUE
70 FROM
71     USER_TAB_PARTITIONS
72 WHERE
73     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 data
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 Partition
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  v 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  v 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.

The screenshot shows the SSMS interface. The Object Explorer on the left lists various databases and objects in the 'MARIYAM' database, including 'AdventureWorks2016', 'DSDA', 'DSDA\_LAB\_9', 'DWBI\_Source\_Update', 'ECommerce', 'IIT\_DW', 'Predict\_loan\_defaulters\_DW', 'PredictLoanDefaulters\_Staging', 'Predict\_loan\_defaulters\_sourceD', 'PredictLoanDefaultersSourceDB', 'ProductWarehouse', 'ReportServer', 'ReportServerTempDB', 'SLIIT\_Retail\_DW', 'SLIIT\_Retail\_DW\_2', 'SLIIT\_Retail\_Staging', 'SLIIT\_RetailSourceDB', 'TestDB', 'Toy\_Store\_SourceDB', 'Toy\_Store\_SourceDW', 'Toy\_Store\_TargetDW', and 'DataQuality'. The query editor window on the right contains a SQL script named 'Data Cleansing.sql'. The script creates two tables: 'CUSTOMER' and 'CORRECT\_NAMES'. It then inserts several rows of data into the 'CORRECT\_NAMES' table, mapping misspellings to their correct forms. The 'Messages' pane at the bottom shows the results of the insert operations.

```
-- Customer Table
CREATE TABLE CUSTOMER(
    Customer_ID INT IDENTITY(1,1) NOT NULL,
    Customer_name varchar(20)
);

-- Create a table to capture the correct spellings
CREATE TABLE CORRECT_NAMES (
    Correct_ID INT IDENTITY(1,1) NOT NULL,
    Incorrect_name VARCHAR(20),
    Correct_name VARCHAR(20)
);

-- Insert values to the correction table
INSERT INTO CORRECT_NAMES VALUES ('John Doh', 'John Doe');
INSERT INTO CORRECT_NAMES VALUES ('Sahidiya', 'Sahdiya');
INSERT INTO CORRECT_NAMES VALUES ('Shadiya', 'Sahdiya');
INSERT INTO CORRECT_NAMES VALUES ('Jananie', 'Janani');
INSERT INTO CORRECT_NAMES VALUES ('Jananee', 'Janani');
INSERT INTO CORRECT_NAMES VALUES ('Janany', 'Janani');
INSERT INTO CORRECT_NAMES VALUES ('Maheshie', 'Maheshi');
```

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

- 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

```
-- 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 %

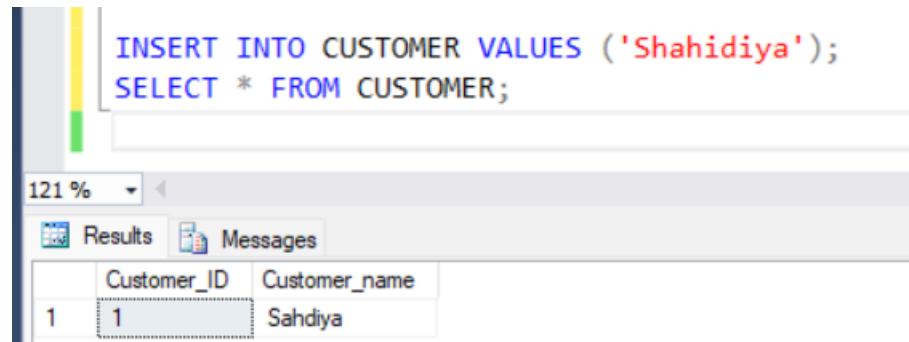
Results

	Customer_ID	Customer_name
1	1	Sahdiya

Messages

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.



The screenshot shows a SQL query window with the following code:

```
INSERT INTO CUSTOMER VALUES ('Shahidiya');
SELECT * FROM CUSTOMER;
```

The results pane shows a table with two columns: Customer\_ID and Customer\_name. There is one row with Customer\_ID 1 and Customer\_name Sahdiya. The 'S' in 'Sahdiya' is highlighted with a red box, indicating it was misspelled in the original query.

	Customer_ID	Customer_name
1	1	Sahdiya

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